OPTIMAL CONTROLS FOR MARKOV CHAINS WITH RESTRICTED STATE OBSERVATIONS

by

Christopher Robert John Butlin, B.Sc.

June 1982

A thesis submitted for the degree of Doctor of Philosophy of the University of London, and for the Diploma of Membership of the Imperial College.

Department of Electrical Engineering,
Imperial College of Science and Technology,
London, SW7.

In memory of my Father.

ABSTRACT

For systems modelled by discrete-time Markov decision processes with large, but finite numbers of states, optimal control policies (stationary of otherwise) can in principle be determined through the use of existing algorithms, which compute the optimal control action for each state. These methods often run into computational difficulties and the resulting control policies can be excessively complicated. The theme of this thesis is to present methods for determining optimal controls for such models, where control laws are defined as functions of pre-determined groups of states, thus allowing much simpler control structures, and reducing the computational burden.

The thesis is effectively split into two parts. The first deals with analytical solution methods which are found to be of practical use only where the model has relatively few (e.g. several hundred) states. It is shown that the problem can be solved using a dynamic programming algorithm. Another approach is to consider stationary probabilistic controls, using a non-linear programming algorithm to search the control space for the optimal point. It turns out that the properties of the Markov model make this approach attractive for smaller problems.

The second part deals with experimental (Monte Carlo) methods which are aimed at problems having very large numbers of states. The gradient of process cost with respect to the control parameters can be expressed as a function of the state distribution and adjoint vector sequences, and it is shown that these sequences can be estimated in several ways using Monte Carlo methods. It is therefore possible to define a number of cost gradient estimators, which can be effectively employed in a recursive algorithm of the stochastic approximation type, which solves

the optimal control problem. Further developments of this approach includes a discussion of variance reduction techniques.

Throughout the thesis, small Markov models are used to illustrate each newly-developed technique, and a larger problem involving industrial transfer-lines is solved by a stochastic approximation algorithm.

ACKNOWLEDGEMENTS

First and foremost, I wish to thank my supervisor, Professor D.Q. Mayne, for his significant contribution in guiding this course of work. In our frequent discussions over the past four years, his constructive criticism and many valuable suggestions have represented a major stimulus to research.

This work has benefited from the financial support of Metal Box p.l.c., and I am grateful to that company, and in particular to my former manager, Dr. C.W. Green, who originally negotiated the sponsorship on my behalf. Thanks are also due to the staff of the Typing Centre at Metal Box p.l.c., Wantage, who painstakingly produced the typescript, and to Mr. L. Rishton, also at Wantage, who provided the art work.

Finally, I wish to express special thanks to my wife and son, who patiently put up with the difficulties associated with the writing of the manuscript, and the years of hard work which went before.

C.R.J. Butlin.

CONTENTS

		Page
TERMINOLOGY, NOTATION AND SYMBOLS		19
CHAPTER 1:	INTRODUCTION	24
1.1	Formulating and solving optimal decision problems	24
1.2	Literature survey	30
	1.2.1 Analytic solution methods	31
	1.2.2 Experimental methods - Simulation and Monte Carlo methods	34
	1.2.3 Studies of industrial transfer-lines	37
1.3	Outline of the thesis	38
1.4	Contributions of the thesis	47
CHAPTER 2:	MARKOV CHAINS - NOTATION, BASIC CONCEPTS, AND APPLICATION TO INDUSTRIAL TRANSFER-LINE PROBLEMS	52
2.1	Introduction	52
2.2	Basic concepts	52
	2.2.1 Properties of a Markov chain	52
	2.2.2 Classification of states and chains	55
	2.2.3 Steady-state properties of regular Markov chains	57
	2.2.4 Regular Markov chains with costs	59
	2.2.5 Controlled Markov chains: the Markov decision problem	61
2.3	Problems involving industrial transfer-lines	66
	2.3.1 Definitions and description of transfer-line operation	67

Page

2.3.2 Identification of operating and repair time distributions 68
2.3.3 Transitions of a two-stage line or line-section 69
2.3.4 The transition matrix and cost vector for a two-stage line 73
2.3.5 Controlled transfer-lines 77
2.4 Summary 80

CHAPTER 3: ANALYTIC SOLUTION METHODS 82
3.1 Introduction 82
3.2 Dynamic programming methods 82
3.2.1 Conventional dynamic programming solution of a classical Markov decision problem 82
3.2.2 Use of dynamic programming to determine optimal open-loop controls 85
3.2.3 Discussion 88
3.2.4 A three-state example 90
3.3 Linear programming solutions 93
3.3.1 Introduction 93
3.3.2 Solution of the classical problem by linear programming 94
3.3.3 Reformulation of the open-loop control problem 98
3.4 Approximation in policy space 101
3.4.1 Howard's policy iteration algorithm 101
3.4.2 Howard's algorithm and the open-loop control problem 102

Page

3.4.3 Approximating the continuous function g(.) near any feasible point, $\hat{\alpha}$ 105

3.5 Solving Markov decision problems using cost function approximations 111

3.5.1 A procedure solving a general non-linear programming problem having linear constraints 111

3.5.2 Armijo step-length subalgorithm 114

3.5.3 Solving Markov decision problems using the linear approximation algorithm 116

3.6 Solving large Markov decision problems using the linear approximation algorithm 129

3.7 Summary 131

CHAPTER 4: MONTE CARLO ESTIMATION OF COST GRADIENTS 133

4.1 Introduction 133

4.2 Reformulation of the Markov decision problem 135

4.3 State probability distributions of the chain 136

4.4 Adjoint vectors of the chain 136

4.5 Statement of the optimal control problem and derivation of the cost gradient 137

4.6 Statistical estimation of state distributions and adjoint vectors 139

4.7 Estimation of state distributions 142

4.7.1 Obtaining a sequence of degenerate row matrices 142

4.7.2 Estimating the sequence $\{\pi_t\}$ 142

4.8 Estimation of the adjoint sequence $\{\lambda_t\}$ 144

Page

4.8.1 Obtaining degenerate estimates of a vector $c \in R^S$ 145

4.8.2 Obtaining a sequence of degenerate column matrices 147

4.8.3 Reverse-time estimation of the adjoint vector sequence - Method 1 148

4.8.4 A simple illustration of forward- and reverse-time estimation methods 150

4.8.5 Some comments on gradient estimation 154

4.8.6 Reverse-time estimation of the adjoint vector sequence - Method 2 155

4.8.7 Forward-time estimation of the adjoint sequence 159

4.8.8 Practical forward-time estimators 162

4.9 Efficient gradient estimation using forward-time simulations 172

4.10 Application of each estimation method to a simple three-state problem 174

4.11 Summary 179

CHAPTER 5: SIMULATION DETAILS FOR THE TRANSFER-LINE PROBLEM AND SOLUTION USING A STOCHASTIC APPROXIMATION ALGORITHM 181

5.1 Introduction 181

5.2 Simulation details for the transfer-line problem 182

5.2.1 Simplifying simulations by identifying subsystems 182

5.2.2 Forward-time simulation of cross-linked transfer-lines 185

Page

5.2.3 Reverse-time simulation of cross-linked transfer-lines by Method 1 191

5.2.4 Reverse-time simulation of cross-linked transfer-lines by Method 2 202

5.3 Gradient estimates for the standard transfer-line examples 203

5.4 Solution of Markov decision problems using cost gradient estimates 206

5.4.1 Discussion 206

5.4.2 Solution of the 64-state transfer-line problem 209

5.4.3 Solution of a problem having two control parameters 216

5.4.4 Choosing algorithm parameters 217

5.5 Summary 221

CHAPTER 6: VARIANCE REDUCTION METHODS 222

6.1 Introduction 222

6.2 Variance reduction : a review of standard techniques taken from the literature 223

6.2.1 Use of Expected Values 223

6.2.2 Antithetic Variates 224

6.2.3 Use of Common Random Numbers 225

6.2.4 Control Variates, or Regression Sampling 227

6.3 Application of standard techniques to the transfer-line problem 228

6.3.1 A necessary feature of the simulation 228

6.3.2 Use of expected values 229

6.3.3 Antithetic Variates 230

Page

6.3.4 Common Random Numbers 233

6.3.5 Control Variate Methods 234

6.4 Variance reduction by refinement of an approximation 238

6.4.1 Origins of the method 238

6.4.2 Use of the technique to reduce variance of gradient estimates 238

6.4.3 A simple example 241

6.5 Application of the refinement technique to the transfer-line problem 244

6.5.1 Use of Norman's reduced models 244

6.5.2 Model reduction by 'State Grouping' 244

6.5.3 Use of independent sub-systems 245

6.5.4 Some comments on the refinement method 251

6.6 Summary 252

CHAPTER 7: CONCLUDING REMARKS 254

7.1 Introduction 254

7.2 Assessing the usefulness of the recommended methods for solving problems of a realistic size 255

7.2.1 Discussion 255

7.2.2 Analytic Methods 257

7.2.3 Experimental Methods 259

7.3 Assessment of contributions in relation to the objectives set in Chapter 1 262

7.4 Suggestions for further work 266

7.4.1 Analytic methods using quadratic approximations to the cost function 266

Page

7.4.2 Adaptive controllers for large Markov chains 267

APPENDIX I: STANDARD 64-STATE AND 144-STATE TRANSFER-LINE PROBLEMS 269

AI.1 The 64-state problem 269

AI.2 The 144-state problem 272

AI.3 Solution of the classical 64-state Markov decision problem using Howard's policy iteration algorithm 274

APPENDIX II: PROOF OF EXISTENCE AND CONTINUITY OF THE DERIVATIVES $g_\alpha(.)$ AND $g_{\alpha\alpha}(.)$ 275

APPENDIX III: EXPERIMENTAL RESULTS FOR THE 3-STATE EXAMPLE DESCRIBED IN CHAPTER 4 279

APPENDIX IV: REDUCED MARKOV MODELS 292

AIV.1 Independent sub-systems 293

AIV.2 State Grouping 295

REFERENCES 298

LIST OF TABLES

		Page
Table 2.1 :	Toymaker example : process transition details.	62
Table 3.1 :	Effect of control over state transitions.	91
Table 3.2 :	Evaluation of $h_2^o(.)$.	92
Table 3.3 :	Optimal feedback control policy.	92
Table 3.4 :	Open-loop analysis.	93
Table 3.5 :	Table of coefficients.	97
Table 3.6 :	Use of the linear approximation algorithm to solve the toymaker example. $\alpha_0 = 0.0$.	121
Table 3.7 :	Use of the linear approximation algorithm to solve the toymaker example. $\alpha_0 = 1.0$	122
Table 3.8 :	Solving a two-parameter problem using the linear approximation algorithm. $\alpha_0 = [0.0\ 0.0]^T$.	127
Table 3.9 :	Solving a two-parameter problem using the linear approximation algorithm. $\alpha_0 = [0.0\ 1.0]^T$.	128
Table 4.1 :	Comparison of gradient estimators.	178
Table 5.1 :	Calculating buffer store contents as function of machine states at (t-1).	187
Table 5.2 :	Simulating independent state transitions for machine i.	191
Table 5.3 :	Calculating buffer store contents as a function of machine states at t.	197
Table 5.4 :	Simulating independent state transitions for machine i in reverse-time.	197

Page

Table 5.5 : Experimental results from program G1. 204

Table 5.6 : Experimental results from program G2. 204

Table 5.7 : Experimental results from program G3b. 205

Table 5.8 : Experimental results from program G3c. 205

Table 5.9 : 500 iterations of projection algorithm solving 64-state problem. 212

Table 5.10: 50 iterations of projection algorithm solving 64-state problem. 213

Table 5.11: 50 iterations of projection algorithm solving 64-state problem. 214

Table 5.12: 100 iterations of projection algorithm solving 64-state problem. 215

Table 6.1 : Estimated variance reduction through use of expected values. 231

Table 6.2 : Estimated gradients and variance reduction using the antithetic variates estimator. 231

Table 6.3 : Use of control variates to improve adjoint estimates. 237

Table 6.4 - 6.7: 2-state example - illustrating benefits due to the use of estimate refinement with a range of prior estimates. 243

Table 6.8 : Variance reduction using the refinement method with 'state grouping' model reduction. 246

Table 6.9 : Estimated gradients for 144-state problem without use of variance reduction methods. 246

Table 7.1 : Estimated storage and computing time for three relevant subroutines. 258

Page

Table AI.1 : Control structure for 64-state problem. 271

Table AI.2 : Control structure for 144-state problem. 273

Table AIII.1 : Standard (calculated) sequence of state distribution vectors. 280

Table AIII.2 : Standard (calculated) sequence of adjoint vectors. 280

Tables AIII.3 - AIII.6 : Estimates of the state distribution sequence using a range of ensemble sizes. 281 - 282

Tables AIII.7 - AIII.10: Estimates of the adjoint vector sequence using Method 1 with a range of ensemble sizes. 283 - 284

Tables AIII.11 - AIII.14: Estimates of the adjoint vector sequence using Method 2 with a range of ensemble sizes. 285 - 286

Table AIII.15: Method 1 adjoint estimator - analysis of variance of component estimates from a single run. 287

Table AIII.16: Method 2 adjoint estimator - analysis of variance of component estimates from a single run. 287

Tables AIII.17 - AIII.19: Method 3a - estimates of $\{\langle e_1, \lambda_t \rangle, \langle e_3, \lambda_t \rangle;\ t = 0, 1, \ldots, 10\}$ using different ensemble sizes. 288 - 289

Table AIII.20: Method 3a - estimates of $\{\langle e_1, \lambda_t \rangle, \langle e_3, \lambda_t \rangle;\ t = 0, 1, \ldots, 10\}$ from 10,000 runs. Single run variance analysis. 290

Table AIII.21: Method 3c - estimating second and third elements of $\bar{\lambda}$ without application of 'Russian Roulette'. 291

LIST OF FIGURES

		Page
Fig. 2.1 :	Two-stage line configuration.	70
Fig. 2.2 :	Two-stage line - transitions from states (D,U,O) and (U,D,Z).	71
Fig. 2.3 :	Two-stage line - transitions from all states except (D,U,O) and (U,D,Z).	72
Fig. 2.4 :	Defining transition matrix for a two-stage line with Z = 2.	75
Fig. 2.5 :	Schematic diagram showing two cross-linked canmaking lines.	78
Fig. 3.1 :	A sequence $\{ x_k \}$ generated by the linear approximation algorithm.	113
Fig. 3.2 :	Line search subproblem solved by Armijo subalgorithm with $\beta = 0.5$, $\lambda = 0.5$.	115
Fig. 3.3 :	Toymaker example with open-loop controls - graph of $g(\alpha)$ vs. α.	120
Fig. 3.4 :	Cost contours and solutions for the two-parameter problem.	126
Fig. 4.1 :	Graphical illustration of the forward-time estimation technique.	173
Fig. 5.1 :	Two-stage line configuration.	186
Fig. 5.2 :	Forward-time simulation rules for a two-stage transfer-line - transitions from force-down states.	188

Page

Fig. 5.3 : Forward-time simulation rules for a two-stage transfer-line - transitions from non-force-down states. 189

Fig. 5.4 : Flow diagram showing fundamental rules for simulating a process cycle in forward-time. 190

Fig. 5.5 : Reverse-time simulation rules for a two-stage transfer-line - transitions from states with $0 < X_3 < M$ 196

Fig. 5.6 : Reverse-time simulation rules for a two-stage transfer-line - transitions from states with $X_3 = 0$. 198

Fig. 5.7 : Reverse-time simulation rules for a two-stage transfer-line - transitions from states with $X_3 = M$ 199

Fig. 5.8 : Flow diagram showing fundamental rules for simulating a process cycle in reverse-time. 200

Fig. 5.9 : Conditions governing transitions from (D,U,X_3) for $0 < X_3 < M$. 202

Fig. 5.10: Performance characteristic for standard 64-state transfer-line example. 210

Fig. 5.11: Calculating $\pi_G(\tilde{x}_{n+1})$ and $\pi_G(\tilde{y}_{n+1})$ for two-parameter problem. 217

Fig. 5.12: Solution of problem having two parameters using projection algorithm. 218

Fig. 5.13: Use of 100-cycle estimates in projection algorithm. 219

Fig. 6.1 : Variance reduction through independent sub-system analysis. 249

Page

Fig. 6.2 : Relationship between β and α. 251

Fig. AIV.1 : Independent sub-systems analysis. 294

Fig. AIV.2 : State grouping model reduction method. 295

TERMINOLOGY, NOTATION AND SYMBOLS

Chapters are divided into sections which are numbered 1.1, 1.2, etc. Sections may be divided into subsections which are numbered 1.1.1, 1.1.2, etc. Tables, figures and equations are numbered consecutively within each chapter, prefixed by the chapter number.

Vectors, matrices and commonly used operators

Vectors are denoted by lowercase letters, e.g. c and q. The i'th component of an s-vector c is denoted by c_i, $i \in \{1, \ldots, s\}$, and c is said to be composed of elements $\{c_i\}$, meaning the set of scalars $\{c_i, i = 1, \ldots, s\}$. The s-vector e_i denotes the vector having (s-1) zeros and a '1' in the i'th position. The symbol '1' is used to denote both the real number one and the vector having all components equal to one. Similarly, '0' denotes both the real number zero and the zero vector.

More often than not in the text, a vector will be referred to quite simply by its symbol, e.g. c, q, etc., but when it is necessary to indicate row or column vectors, we use the notation $<q$ for the row vector q, and $q>$ for the column vector q. The inner product of two vectors is written $<p, q>$ $(= \sum_i p_i q_i)$; thus $<e_i, q> = q_i$, the i'th component of q. Similarly, if A is an s x n matrix, the n-vector $<e_i A$ denotes its i'th row. It is also possible to write an outer product, e.g. $q><1$, which denotes the matrix have $q>$ as each of its columns. The superscript 'T' is sometimes used to denote a transposed vector, e.g. the row vector c^T is the transposed column vector c.

Matrices are denoted by uppercase letters, e.g. A and B. The j'th entry in the i'th row of an s x n matrix A is written a_{ij}, or $[A]_{ij}$, and

A is said to be composed of elements $\{a_{ij}\}$, meaning the set of scalars $\{a_{ij},\ i = 1, \ldots, s;\ j = 1, \ldots, n\}$. I denotes the identity matrix of appropriate size.

R^n, $n = 1, 2, \ldots$, denotes n-dimensional Euclidean space. If $f : R^n \to R$ is a differentiable function, the derivative of f at $\hat{\alpha}(\in R^n)$ is the row vector $[(\partial f/\partial\alpha_1)(\hat{\alpha}), \ldots, (\partial f/\partial\alpha_n)(\hat{\alpha})]$, which is denoted by $(\partial f/\partial\alpha)(\hat{\alpha})$ or $f_\alpha(\hat{\alpha})$. The transposed derivative, $[f_\alpha(\hat{\alpha})]^T$, is a column vector denoted by $\nabla_\alpha f(\hat{\alpha})$ and is called the gradient of f at $\hat{\alpha}$.

If $f : R^n \to R^m$ is a differentiable function, then for each $i = 1, \ldots, m$, $\langle e_i, f\rangle : R^n \to R$ is also differentiable, and the derivative of f at $\hat{\alpha}$ is an m x n matrix given by

$$\frac{\partial f}{\partial\alpha}(\hat{\alpha}) = f_\alpha(\hat{\alpha}) = \begin{bmatrix} (\partial/\partial\alpha)\,[\langle e_1, f\rangle]\,(\hat{\alpha}) \\ \vdots \\ (\partial/\partial\alpha)\,[\langle e_m, f\rangle]\,(\hat{\alpha}) \end{bmatrix}$$

If $f : R^n \to R$ is twice differentiable, its second derivative at $\hat{\alpha}$ is the n x n matrix $(\partial^2 f/\partial\alpha \,.\, \partial\alpha)(\hat{\alpha}) = f_{\alpha\alpha}(\hat{\alpha})$ having elements defined by

$$[f_{\alpha\alpha}(\hat{\alpha})]_{ij} = (\partial^2 f/\partial\alpha_i \,.\, \partial\alpha_j)(\hat{\alpha});\quad i, j \in \{1, \ldots, n\}$$

For a matrix valued function $\psi : R^n \to R^s \times R^s$, $\psi^{-1}(\hat{\alpha})$ represents the inverse of the s x s matrix $\psi(\hat{\alpha})$ and $\psi^k(\hat{\alpha})$ represents the matrix $\psi(\hat{\alpha})$ raised to the k'th power (i.e. $\psi^k(\hat{\alpha}) = \underbrace{\psi(\hat{\alpha}) \times \ldots \times \psi(\hat{\alpha})}_{k \text{ terms}}$).

A^T and det(A) denote the transpose and determinant of a matrix A, respectively.

$\lim_{t\to\infty} \{\pi_t\} = \pi$ confirms that the sequence $\{\pi_t\}$ converges to π as t tends to ∞.

The closed interval [a, b] is taken to mean the set of all real numbers, x, such that $a \leq x \leq b$. Half-open intervals are also defined, e.g. x's such that $a \leq x < b$ are represented by [a, b).

$\max_{\alpha} \{J(\alpha)\}$ or $\text{maximize}_{\alpha} \{J(\alpha)\}$ means "find the value (or values) for α which maximizes the function J".

$\min_{\alpha} \{J(\alpha)\}$ is defined in a similar way.

The operator $\prod_{i=1}^{n} a_i$ represents the product of the n values $\{a_i, i = 1, \ldots, n\}$.

Notation relating to random variables

A sample space Ω might consist of m points or elementary events, $\{\omega_1, \ldots, \omega_m\}$; thus $\Omega = \{\omega_1, \ldots, \omega_m\}$. The sample space is sometimes written $\Omega = \{\omega\}$.

The probability of occurrence of some event A (which represents a subset of the sample space) is given by Prob [A], $\in [0,1]$. A random variable, X, is a function such that for each $\omega \in \Omega$, $X(\omega)$ is a real number. The expectation of $X(\omega)$ with respect to all $\omega \in \Omega$ is denoted by

$$E_{\omega\in\Omega}\{X(\omega)\} = \sum_{\text{all } x} x \cdot p_x$$

where X takes discrete values and p_x = Prob $[X(\omega) = x]$.

The notation is sometimes reduced to $E\{X(\omega)\}$ or simply $E\{x\}$.

The variance of X is written var(X) ($= E\{[X - E\{X\}]^2\}$) and the standard deviation (S.D.) of X is the square root of its variance. For two random variables X and Y, we write the covariance of X and Y as cov(X,Y) ($= E\{(X - E\{X\})(Y - E\{Y\})\}$).

Control schedules and policies

Consider a control system for which a control action u_t has been specified at each time, t = 0, 1, ..., N-1. If the controls were specified without reference to the state of the system at each t, the sequence $\{ u_0, \ldots, u_{N-1} \}$ is referred to as an open-loop schedule of control actions.

If u_t is specified by

$$u_t = h_t(x_t), \quad t = 0, 1, \ldots, N-1$$

where x_t is the system state at t, then $\{ h_0(.), \ldots, h_{N-1}(.) \}$ is a sequence of control laws representing a feedback policy.

When calculating the optimal (feedback) control policy we should (strictly speaking) calculate the complete sequence $\{ h_0(.), \ldots, h_{N-1}(.)\}$ which minimizes the system cost function. However, for systems described by an ergodic set of states (see Chapter 2 for

definitions) then as $t \to \infty$, $h_t(.) \to h(.)$, and we can calculate the control law $h(.)$, used at each $t = 0, \ldots, N-1$, which minimizes the function defining expected process cost per unit time. In such cases it is convenient to refer to the law $h(.)$ as the (feedback) control policy, or the stationary feedback policy. This definition coincides with that used by Howard (7) and others, when discussing approximation in the space of feasible stationary policies.

CHAPTER 1

INTRODUCTION

1.1 Formulating and solving optimal decision problems

Engineers (amongst others) are frequently required to solve problems involving systems whose behaviour, though subject to some random disturbance, can be influenced by actions which they may decide to take.

As an example, we can consider a simplified form of an engineering problem involving machine maintenance. A machine processes workpieces at a constant rate of one per minute, and is observed to be unreliable, i.e. it fails after processing a (random) number of workpieces. A machine failure is invariably found to be due to the failure of one specific part which can be replaced, restoring the machine to working order. Such repairs are observed to take a random number of minutes. Replacement parts can be obtained from two manufacturers, and it is found that one manufacturer supplies parts which tend to last longer (allow the machine to process more workpieces), but cost more than those from the alternative supplier. The engineer is therefore faced with the problem of deciding from which manufacturer to buy his parts in order to maximize the profit earned. Profit is calculated by subtracting costs (due to the replacement of parts) from returns (proportional to the number of workpieces processed).

It is important that the quantity to be maximized is given a more precise definition. Firstly, the engineer should decide which time-planning horizon to use. If he decides to maximize the total profit realized over a pre-determined period like N minutes, then the problem

is referred to as a "finite-horizon problem". Alternatively, he could consider the "infinite-horizon problem", for which he would seek to maximize the average profit per minute (or some other period) of operation. Secondly, because profit is influenced by the decision taken, the quantity to be maximized is referred to as the "objective function" and the optimal decision problem is thus a function maximization problem. Also, as the behaviour of the system is subject to random disturbances, the returns and costs are random variables, and it is therefore sensible to maximize the expected profit earned in N minutes, or the expected profit earned per minute of operation.

It should be noted that in describing machine operation over a period of time, we have failed to attach any importance to the state in which the system starts. In fact, it is reasonable to treat the machine maintenance problem in this way because after some period of operation, it is possible for the system to assume either of its two states (machine working or broken down), regardless of its initial state. A system which is able to "forget" its early history in this way is said to be described by a single ergodic set of states, and we can define an infinite-horizon problem or a problem involving a long but finite planning horizon without explicitly mentioning an initial state. However, there are systems whose future behaviour depends on the choice of initial state, and for which an optimal decision problem would only have meaning if an initial state was specified. In this thesis, we only consider problems described by a single ergodic set of states (for reasons given in Chapter 2).

One systematic approach for solving the machine maintenance problem comprises the following tasks:

(i) Gather data from the process, i.e. observe the machine over a

long period, during which some machine repairs should be carried out using parts supplied by one manufacturer, and others should be carried out using those from the alternative supplier. In each case, note the time between breakdowns, and the time taken for repairs. Also, list all costs and returns relevant to the problem.

(ii) Using the data previously collected, formulate a mathematical model of the system, thereby providing a precise statement of the relationships which exist between the components of the model.

(iii) Ensure that the model is valid by comparing output (or some other characteristic) predicted by the model with that observed from the real system.

(iv) Formulate an optimal decision problem, using the mathematical model.

(v) Solve the mathematical problem choosing whichever solution method is appropriate, bearing in mind the computing resources available and the type of mathematical model describing the system.

(vi) Implement in real-time the optimal policy (i.e. the optimal action to be taken in each state) obtained in (v). The optimal policy might be: "Whenever the machine fails, effect a repair using replacement parts supplied by manufacturer no.1".

In this thesis we effectively by-pass the first three tasks by

restricting our attention to systems which have been modelled by Markov decision processes. These simple mathematical models regularly appear in the literature representing a wide range of real systems. Although we use some abstract Markov models to illustrate certain aspects of the theory, we also study a practical engineering problem involving industrial transfer-lines, which previous studies have shown to be accurately modelled by Markov decision processes.

We now consider tasks (iv) and (v) in a more general context. If a physical system can exist in any one of s distinct states, then its Markov model represents the system dynamics by a linear s-dimensional vector equation. In the real world, systems of only a modest size can give rise to Markov models having very high dimensions, e.g. an industrial transfer-line comprising four machines linked together by three buffer stores, each having a capacity of 10 workpieces, can exist in any one of 21,296 states. Throughout the thesis, we will use the term "classical Markov decision problem" to describe a problem using a Markov model whose decision structure allows the use of a different action in each state for which alternative actions exist. The set comprising actions defined as functions of states is called a (control) policy. It can be seen that if m alternative actions exist for each state in an s-state model, then the total number of feasible policies is m^s. Given any feasible policy, the objective function is evaluated by solving s simultaneous equations, or by using one long system simulation. Thus it is generally not possible to determine the optimal policy (the policy yielding the maximum value of the objective function) by evaluating the objective function for each feasible policy, and we turn to mathematical programming and optimal control theory for more efficient solution methods.

Large classical Markov decision problems can theoretically be

solved by using one of the many solution methods which have appeared in the literature, but in practice, we face three major difficulties:

a) Most computers solving a 10,000-state problem using analytic methods would have insufficient central memory to allow manipulation of 10,000 x 10,000 matrices and 10,000-element vectors. Disc storage could be used, but then a solution would require much more computing time.

Solution by 'brute-force' simulation (using one long simulation to assess each system variant) has previously been the only practical option, but makes inefficient use of computing resources.

b) Though the existing analytic methods are efficient, in the sense that they solve a problem using much less effort than would be required to evaluate the objective function for each feasible policy, it seems likely that more efficient methods might be developed for problems where the number of feasible policies has been restricted in some way.

c) The solution is expressed in the form of a function, defining the (optimal) action to be taken for each of a large number of states. If the actions represent a control policy which is to be implemented using computer-controlled equipment forming part of (say) a transfer-line described by 10,000 states, then the computer is required to store a function having 10,000 values (unless it can be simplified in some way). After identifying the system state, the computer can use the stored function to determine which control action to take. Unfortunately, computers operating as real-time

controllers rarely have a great amount of central memory available for storing such a complex function, and in any case, the policy might be considered unnecessarily complicated. Also, there exist situations where it might be impossible, or too costly to identify the exact state of the underlying process, but where it is certain that one of a group of states is occupied.

We are able to make a contribution here by developing new and efficient solution methods for problems modelled by Markov decision processes having many states and actions. Our chief concern is that the proposed methods should be of practical use, and should to some extent overcome the difficulties which have been identified for existing methods. Two of the three difficulties mentioned above imply that control policies for the classical Markov decision problem are more complicated than they need be. We therefore begin by redefining the decision structure so that the choice of action in one state restricts the choice in another. One convenient and effective way of doing this is to group together suitable states and to determine the optimal action for each group (thus for each state of a given group, we take the same action). In order to differentiate between problems having each type of decision structure, we use the term "grouped-state Markov decision problem" to identify a problem modelled by a Markov process having the simplified decision structure described above. The optimal decision problem for either type of mathematical description is to "maximize the expected rate of return (or equivalently to minimize the expected rate of cost)", where the maximization should be over all feasible policies.

Two benefits of the grouped-state formulation immediately become apparent. The first and most obvious benefit is that the optimal grouped-state policy can be far less complex than the optimal policy for

the classical problem. When specifying state groups, the problem solver is only constrained by characteristics of the underlying process, i.e. it must be possible to take the same action in each state of a group. We can conceive of processes for which it is possible to group together all the process states. Any action specified for this group of states is effectively an open-loop control action, so called because the real-time controller would require no knowledge of the system state.

A second point is that whereas policies for the classical problem are functions a large number of states, policies for the grouped-state problem are functions of a smaller number of state-groups, and it follows that there must be fewer feasible policies for the grouped-state problem than for the classical problem. We would expect that an optimizing algorithm searching over a smaller number of policies would be more efficient.

Formal mathematical definitions of classical and grouped-state Markov decision problems are given in Chapter 2.

1.2 Literature Survey

As stated earlier, this thesis treats a problem in which the number of feasible control policies is deliberately restricted in an attempt to reduce the computing time required to calculate the optimal policy, and to ensure that the resulting policy is not over-complex (especially important where the system is to be controlled automatically by computer, logic controller, etc. in real-time). It appears that this "grouped-state" problem has not been considered previously in the literature (though a similar problem is posed and solved by Hastings and Sadjadi (1) - see below) and it is natural, therefore, to begin this study with a review of the more popular solution methods for the

classical problem.

1.2.1 Analytic Solution Methods

Solution methods for the classical Markov decision problem fall into four basic categories. We identify algorithms from each category as dynamic programming algorithms, linear programming algorithms and algorithms similar to Howard's policy iteration algorithm, and White's successive approximation algorithm.

Dynamic programming was one of the first solution methods to be suggested and appears in Bellman (2), Bellman and Dreyfus (3), and many more recent studies. The algorithm can be used in principle to solve classical Markov decision problems of any size, but in practice it is very difficult to obtain numerical solutions to large problems, because of the vast amount of computer storage required. Many authors have addressed themselves to the practical aspects of dynamic programming, and some interesting methods for reducing storage requirements are the polynomial approximation method due to Bellman and Dreyfus (3), the diagonal decomposition method developed by Collins (4), Collins and Lew (5), and the differential dynamic programming approach due to Jacobson and Mayne (6).

When the Markov chain is regular (see definitions in Chapter 2) it is normal to calculate the optimal stationary control policy, i.e. the stationary policy yielding the minimum mean process cost per cycle. The dynamic programming algorithm yields a sequence of N control laws (each law defining the action to be taken in each state) minimizing the total process cost over N cycles, but Howard (7) has shown that the optimal stationary policy defined above can be found using value iteration dynamic programming, i.e. iterating over a number of cycles, N, large

enough for the process to reach steady state conditions. All other solution methods mentioned in this section are designed to calculate the optimal stationary policy directly.

A second approach using the theory of linear programming was suggested by Manne (8) and later studied by Wolfe and Dantzig (9). In order to use a linear programming algorithm, it is necessary to reformulate the problem, transforming the finite set of control alternatives available in each state, into a continuous space of probabilistic controls. If k control alternatives, labelled 1,...,k exist for a given state, then the probabilistic control action to be taken in that state is denoted by $\{\alpha_i, i = 1, \ldots, k\}$ where $\alpha_i \in [0, 1]$ is the probability of adopting control alternative i and $\sum_{i=1}^{k} \alpha_i = 1$. Probabilistic controls prove to be particularly useful when solving grouped-state control problems.

By far the most popular of the analytic solution methods is the policy iteration algorithm due to Howard (7). Many versions of the algorithm have appeared since its introduction in 1960, but Howard's original version is still used in many studies. Mine and Osaki (10) compared the three analytic methods mentioned above, and show that the linear programming, and policy iteration algorithms are equivalent, i.e. the policy iteration algorithm is a special extension of linear programming such that pivot operations for many (at most s, for an s-state problem) variables are performed simultaneously.

A fourth category of solution methods arose out of a need to overcome a major computational drawback of the policy iteration algorithm. At each iteration, the algorithm requires the solution of s simultaneous equations (for an s-state problem), and when s is a large number, this feature of the policy iteration algorithm makes it

unwieldy. White (11) used a modification of Bellman's method of "successive approximation" which offers the considerable computational advantage over Howard's algorithm that only pointwise (decoupled) calculations are required at each iteration. Several algorithms resembling White's original have appeared in the literature. Varaiya (12) has recently proposed a similar algorithm which successively generates dual variables using easily computable bounds on the mean process cost corresponding to any feasible policy.

Whereas all the aforementioned analytic solution methods were designed to solve classical Markov decision problems, i.e. where control actions can be chosen independently for each state, and are defined as functions of (certain) state observations, there exists a number of studies (notably Åström (13), Smallwood and Sondik (14) and Sondik (15)) dealing with Markovian problems where controls are to be defined as functions of state estimates. Here, the controller observes system outputs which are probabilistically related to the true system states, and uses present and past observations to construct state estimates.

Studies by Riordon (16) and more recently by Borkar and Varaiya (17) are concerned with the definition of adaptive control schemes for systems modelled by Markov chains. Rather than struggle using methods like those described above to solve decision problems modelled by very large Markov chains, it might be possible to formulate a reduced Markov model, identifying its transition matrix and optimal control policy in real time, using an adaptive control scheme. This approach has not been investigated in this thesis, but is certainly worthy of consideration. Some further comments are made in Chapter 7 when adaptive control schemes are suggested as an area for further research.

To complete this review of analytical solution methods, we discuss

a recent paper by Hastings and Sadjadi (1), which poses and solves a problem resembling the one considered in this thesis. Considering a process modelled by a finite-state, discrete-time Markov chain, and defining a finite (and possibly different) set of control alternatives for each state, the authors describe a method for determining the maximal gain (equivalently, minimal cost) policy in the "policy constrained" case. They define a "policy constrained" Markov decision process to be one where selecting a given action in one state restricts the choice of action in another. The set of feasible control policies for this problem is any pre-determined set of policies forming a subset of the feasible policies for the classical problem. Thus the control structure has a more general form than that described in this thesis. It follows that the solution method described by Hastings and Sadjadi could be used to solve the grouped-state Markov decision problem.

Unfortunately, this procedure requires significantly more computation than would be required to solve the classical problem. In fact, the first step in the procedure is to determine the optimal stationary policy for the unconstrained problem (the classical Markov decision problem), which immediately dictates against its use for solving very large constrained problems.

1.2.2 Experimental Methods - Simulation and Monte Carlo Methods

We use this section to identify references dealing with the basic principles of experiment design and analysis. The literature abounds with reports of studies using experimental methods, but more often than not, these studies deal with the application of the well known basic principles of experimental work to some specific problem, and the resulting method is too problem-oriented to be of general use. This comment is particularly true of simulation studies appearing in the

operations research literature. O.R. problems are frequently of such a size that an analytical solution would be out of the question and the usual approach is to develop a simulation model and to carry out direct "brute-force" simulation observing the required system characteristics from simulated histories. Two references offering useful advice on the development of simulation models are Tocher (18) and Emshoff and Sisson (19).

The book by Kleijnen (20) is an excellent reference covering all aspects of simulation, concentrating more on basic principles (e.g. development and validation of models, experiment design, analyses of system variants, steady state responses, etc.) than on the modelling of specific systems.

We are concerned in this thesis with choosing the best of several system variants (more precisely, choosing the parameters of a system model to yield the 'best' response) and it is interesting to see that until quite recently, the accepted procedure for solving this type of problem was to use brute-force simulation, i.e. generating long simulated histories for each system variant and using the observed system responses to select the best variant. Because an observed response is a random variable, whose accuracy depends on the length of the simulation run, another problem is to decide how much computing effort is necessary to produce an accurate ranking of the system variants from worst to best. "Multiple ranking procedures" exist for this purpose and are a popular area of research in their own right. Some recent work on the topic is given by Iglehart (21) and by Dudewicz and Zaino (22). Also useful are papers by Bechhofer (23), Robbins, Sobel & Starr (24), Dudewicz and Zaino (25), and by Dudewicz and Dalal (26). Though "multiple ranking" represents a valid solution method for grouped-state Markov decision problems, it makes inefficient use of

computer resources, and we seek a more systematic technique.

In 1979, as an alternative to the brute-force approach, Ho (27) described an efficient experimental method for evaluating parametric sensitivity. Here, only a single statistical experiment (plus some computational overhead) need be implemented in order to estimate sensitivities of the system response to the various parameters. The sensitivities are then used to indicate a direction in which to search for a better response. In a sense, this approach is similar to another response analysis method known as "response surface methodology (RSM)" which is discussed in the literature (e.g. see Burdick and Naylor (28)). RSM requires the fitting of a regression equation to a set of observed responses (resulting in a "response surface"). The response surface is then used in a hill-climbing procedure which eventually identifies the system parameters yielding the 'best' response. However, RSM requires multiple system simulations to construct the response surface and we conclude that Ho's approach represents a significant step forward. He has applied this method to a transfer-line design problem which we shall discuss in the next section.

Monte Carlo methods are taken by many authors to mean sophisticated techniques which call for the alteration of certain fundamental laws (usually probability distributions) governing a statistical experiment so that, compared with the naïve approach, we obtain an unbiased estimate having lower variance for the same amount of work. Others use the term "Monte Carlo method" to mean any technique which uses random numbers to produce (unbiased) estimated solutions to abstract mathematical problems. In this thesis, the term "Monte Carlo" refers to both variance reduction and abstract sampling methods.

The early work on Monte Carlo methods has been found most useful;

in particular we use methods previously applied to atomic physics problems, described by Kahn (29), and methods designed to solve large systems of linear equations, reviewed by Halton (30). The use of variance reduction techniques for solving abstract problems is discussed by Hammersley and Handscomb (31) and Kleijnen (20) reports on their application in simulation. Kleijnen concludes that variance reducing techniques had hardly been applied in simulation studies up to 1974. Since then the situation does not appear to have changed much, and it is usually accepted that variance reducing techniques are difficult to apply to "complex" simulations like those of queuing systems.

1.2.3 Studies of Industrial Transfer-lines

A resumé of research results up to 1972 is given by Wild (32). He states that at the time of writing, "..... comparatively little research has been directed to this problem, and there is a shortage of definitive information to assist the transfer-line designer". Some relevant studies are by Hunt (33), who uses queuing theory to formulate an analytical model for small production lines whose stages have variable service times; Hillier and Boling (34), who extend Hunt's approach by improving the numerical procedure to allow analysis of much larger systems with exponential or Erlang service times; and Freeman (35), who uses brute-force simulations of a three-stage line to draw-up a list of observations which are offered as a rough guide to transfer-line designers. Buzacott (36),(37) derives a Markov model of a two-stage transfer line with constant service times. The model is used to calculate buffer effectiveness, to indicate the best line configuration (optimum buffer sizes), and to solve a problem involving real-time line control.

More recently, Gershwin and Schick (38) have used Buzacott's Markov

model for an n-stage line and have shown how the nested block tri-diagonal structure of the transition matrix can be exploited in analytic procedures for calculating matrix inverses and the stationary state distribution for the system.

The efficient simulation approach mentioned in the last section has been applied by Ho, et al in two papers (39), (40). The problem treated by both papers is the well known buffer store design problem for n-stage transfer lines. Ho's approach provides estimates of the parameter (buffer size) sensitivities, which are later used in a hill- climbing routine which determines the configuration maximizing line throughout. The main contribution of these papers is to show how the parameter sensitivities can be estimated from one simulation run using a very efficient procedure. The effectiveness of this procedure stems from the authors' insight of the fundamental role played by buffers in contributing to the production increase of the transfer line. Unfortunately, the technique can not be used directly to solve the cross-linked transfer-line problem studied in this thesis, because cross-linked lines behave differently when parameters are adjusted.

The problem involving real-time control of cross-conveyors linking two or more transfer lines is based on line configurations suggested by Brilman (41). The paper is primarily concerned with practical aspects of designing can handling systems (mass conveyors, elevators, etc.) for two-piece tinplate cans, and provides some useful background information for the study.

1.3 Outline of the Thesis

The aim of the thesis is to develop new and efficient methods for solving optimal control problems modelled by large Markov chains.

Chapter 2 begins with a review of some Markov chain terminology, also introducing notation which will be used in later chapters and some well known results for controlled chains operating in steady-state conditions. The remainder of Chapter 2 is devoted to a discussion of decision problems involving industrial transfer-lines, which are well suited to modelling by Markov chains. We give an account of a buffer store design problem which has appeared previously in the literature, but the principle aim of the discussion is to introduce a problem involving the real-time control of two transfer-lines using cross-conveyors. Cross-linked systems of a realistic size are accurately modelled by Markov decision processes having a large number of states, and it was precisely for this type of problem that the solution methods in this thesis were developed. We therefore choose two problems involving cross-linked lines described by relatively few states (referring to them as "standard" 64-state and 144-state transfer-line problems) and use them to compare the efficiency of each solution method.

The practical aspects of cross-conveying in can-making factories have been investigated by Brilman (41) and Chapter 2 uses this and other transfer-line studies to set up a suitable Markov model to describe the process. The "standard" transfer-line problems (which are based on Brilman's suggestions) are defined in Appendix I.

The purpose of Chapter 3 is to develop analytic methods for solving grouped-state Markov decision problems. We begin by reviewing three widely used analytic methods for solving classical Markov decision problems, and attempt to solve grouped-state problems by direct application of these methods. Throughout the chapter, we use small problems with open-loop controls to illustrate new solution methods.

It is shown that the grouped-state problem can be solved by applying the theory of dynamic programming in the continuous space of state probability distributions. However, we choose not to take the discussion further as the difficulties involved in applying dynamic programming in a continuous (hyper-) state space have been studied extensively in the literature. To illustrate the method, and to compare different control structures, we solve simple problems having classical (full state feedback) and grouped-state (open-loop, in the simplest case) control structures using dynamic programming algorithms.

It is shown that neither linear programming, nor the policy iteration algorithm can be applied directly to grouped-state problems, as each algorithm at some stage requires that control actions can be chosen independently in each system state. The objective function (mean process cost per cycle) for a Markov decision process is normally defined on a discrete space of controls. Reformulating the problem so that the objective function is defined on a continuous space of control parameters, we can consider the use of the objective function gradient (with respect to the parameters) in existing non-linear programming algorithms. A continuous parameter space results from the use of probabilistic controls (described earlier), and Appendix II proves the existence and continuity of the first and second derivatives of the objective function with respect to the parameters. It is shown that an equation similar to that appearing in the policy iteration algorithm describes a first order approximation to the objective function near any given point in parameter space. This function can be used in a non-linear programming algorithm described by Zangwill (42), to solve the grouped-state problem. This "linear approximation" algorithm generates a sequence of points in parameter space, each point yielding a lower value of the objective function than its predecessor. We illustrate the use of the algorithm by solving two problems, the first

being an adapted version of Howard's toymaker example, having open-loop controls and a single control parameter, and the second having a slightly different cost structure, using open-loop controls but having two control parameters. The examples show that regardless of the choice of algorithm parameters, the algorithm is slow to converge in a neighbourhood of the solution point. Quadratic approximations to the objective function are likely to be useful here and, as a pointer to further work, we derive an expression for a quadratic approximation to the objective function near any given point which, for an s-state chain, requires the inversion of an s x s matrix.

The two methods described in Chapter 3 can theoretically be used to solve grouped-state Markov decision problems described by any number of states. However, for a system described by s states, the linear approximation algorithm requires, at each iteration, the solution of no less than two sets of s simultaneous equations, or possibly the inversion of an s x s matrix. When s is a large number (=10,000, say), limited computing resources (central memory and computing time) make the algorithm impractical. The same restrictions also apply to the dynamic programming algorithm, which requires large amounts of storage and computing time to manipulate 10,000 element vectors over a large number of process cycles. We devote the remaining chapters of the thesis to a study of solution methods which search a continuous parameter space, as does the linear approximation algorithm, but use noisy (statistical) estimates of the objective function gradient, obtained from system simulations in forward and reverse-time.

The purpose of Chapter 4 is to develop methods for obtaining unbiased estimates of the objective function gradient at any feasible point in parameter space. Such estimates can later be used in an algorithm of the stochastic approximation or non-linear programming type

to solve the grouped-state decision problem. We consider a general form of the grouped-state Markov decision process using probabilistic controls, for which the objective function is taken to be the total cost incurred by the process over N cycles, and give an expression for the objective function gradient in terms of the state distribution and adjoint vector sequences. As we endeavour to show in Chapter 4, it is possible to obtain independent, unbiased estimates of each sequence which can then be substituted into the aforementioned expression to provide an unbiased estimate of the gradient.

The state distribution sequence of a controlled Markov chain using a given control parameter is very simply defined in terms of the state transition matrix for the chain, and it is shown that an unbiased estimate of the sequence can be obtained in a straightforward way by simulating the system in forward time. Each N-cycle simulation requires at least N random numbers (i.e. N numbers independently drawn from a uniform distribution over the interval $[0,1]$). The procedure for estimating state distributions is basically no more than a formal statement of the rules for simulating any abstract chain, but it is discussed in some detail because the recursive equations defining the state distribution and adjoint vector sequences are very similar, and it therefore seems likely that adjoint vectors and state distributions can be estimated using similar techniques. For the following discussion, it is convenient to define a "degenerate s-vector" as a vector of the form e_i, $i \in \{1, \ldots, s\}$, and a "degenerate s x s row (column) matrix" as a matrix comprising s degenerate row (column) vectors.

It is interesting to view a conventional system simulation over N cycles as a set of rules for reducing N system transition matrices to N degenerate row matrix estimates using (at least) N independent random numbers. A naïve approach to adjoint estimation is to simply calculate

estimates of the adjoint vectors by substituting N degenerate row matrix estimates into the equation defining the adjoint vector sequence. This method is impractical for two reasons. Firstly, an adjoint vector estimated in this way would consist mostly of non-zero elements. For a large, s-state chain, the s elements of each adjoint vector (or for the complete adjoint sequence, N x s elements) would have to be stored on disc, and computing time would therefore be increased significantly. Secondly, conventional forward-time simulation produces a sequence of degenerate vectors, which means that only one row of each of the N degenerate row matrices needs to be calculated (a total of N degenerate vectors). The naïve adjoint estimation method described above requires the calculation of N x s degenerate vectors per simulation run. Thus both storage and computing time are increased by the factor s compared with conventional simulation. We are therefore interested in developing estimation methods which produce a sequence of degenerate adjoint vectors.

Chapter 4 gives details of two methods (labelled Methods 1 and 2) for generating degenerate adjoint vector sequences in reverse-time. Method 1 uses weighting techniques to obtain degenerate column matrix estimates of the system transition matrix. In fact, Method 1 mimics forward-time simulation in that the resulting degenerate vectors (of the form $[z].\ e_i$, $i \in \{1, \ldots, s\}$, where z is a weighting factor) require the identification of only one column of each degenerate column matrix for each simulated cycle. The computing resources required for a complete simulation run of N cycles in reverse-time are therefore approximately equal to those required for a run in forward-time. Unbiased estimates of the adjoint vectors are shown to be obtained by forming partial summations of degenerate vectors generated using Method 1.

Method 2 is probably the simplest of the adjoint estimation

methods. It is based directly on the naïve approach mentioned earlier, but makes use of weighting methods to reduce vectors with many non-zero elements to degenerate vector estimates.

Methods 1 and 2 are reasonably efficient techniques providing unbiased estimates of the adjoint vector sequence. However, our main concern is to obtain estimates of the objective function gradient. Direct substitution of the estimated state distribution and adjoint vector sequences (the latter obtained using either reverse-time method) in the expression for the objective function gradient, results in a gradient estimate having very high variance. This is primarily due to the lack of correlation between forward and reverse-time simulations. We define "lack of correlation" by the following comments. Each simulation produces, at each cycle, an estimate of a complete s-vector, whereas it can be shown that having determined a degenerate estimate of the distribution vector π_t at each $t \in \{0, \ldots, N-1\}$, we only require estimates of a small number (two for each partial derivative) of elements of the adjoint vector λ_{t+1} in order to estimate the gradient. Accordingly, we show in Chapter 4 that it is possible to use a forward-time simulation to estimate any given element of any adjoint vector (one simulation run is required for each estimated element). It follows that the gradient estimate can be determined in principle by estimating a distribution vector sequence from a forward-time simulation, using the distribution vector estimate at each cycle to determine which adjoint vector elements are required, at the same time initiating other forward-time simulations to estimate those elements. To summarize, this naïve approach provides an unbiased estimate of the objective function gradient by generating a set of forward-time simulations. The number of simulations in the set is determined at run time by the main simulation, and one obvious drawback of the approach is that a prohibitively large number of extra simulations would be initiated by a main simulation of

reasonable length (say $\geq$ 1000 cycles).

In Chapter 4 we develop practical "forward-time" gradient estimators (labelled Methods 3a, 3b and 3c) which improve the naïve approach described above. It is shown that a Monte Carlo method known as "Russian Roulette" can be used to reduce the number of forward-time simulations required to estimate the gradient, thereby making the method feasible for all sizes of Markov model. Also, as we are usually concerned with steady state properties of regular Markov chains (i.e. we require the gradient of mean process cost per cycle, with respect to the parameters), we show that forward-time simulations can be most efficiently used to provide estimates of selected elements of the "relative cost vector" (a stationary vector of costs for the chain), thus allowing very efficient estimation of the "mean cost gradient".

Throughout Chapter 4, we have discussed estimation methods with reference to small, abstract Markov chains, and at the end of the chapter, we use a simple three-state problem to compare the accuracy and efficiency of each method when used to estimate adjoint vectors, or to estimate the mean cost gradient directly. Most of these results appear in Appendix III.

Our objective in developing these gradient estimation procedures is to provide a means for solving problems modelled by very large Markov models which cannot be solved efficiently using existing methods. In Chapter 5, we demonstrate the effectiveness of the gradient estimators discussed in the previous chapter when applied to the standard 64-state transfer-line problem. Before discussing and comparing results, we define rules for forward-time simulation of the system and show that reverse-time simulation can be viewed as a dual of the forward-time process. Forward-time simulation can be simplified by breaking down the

complete model into a number of subsystems which can be simulated using independent streams of random numbers. It is interesting to see that reverse-time simulation can also be simplified by manipulating the same subsystem models. An analysis of variance for each estimation method shows that the most efficient gradient estimators are those which generate a set of forward-time simulations, i.e. programs G3b and G3c.

Having described efficient gradient estimation methods, we proceed to use them in a stochastic approximation algorithm which is described in Chapter 5. The algorithm (with gradient estimates provided by program G3c) is used to solve two problems, of which one is the standard 64-state transfer-line problem which has one control parameter, and the other is a two-parameter problem which was previously solved in Chapter 3 using an analytical method. Chapter 5 concludes with some recommendations on the choice of algorithm parameters.

The use of a forward-time gradient estimator in a stochastic approximation algorithm represents an acceptable solution method for grouped-state Markov decision problems, but the computing time required to solve a problem described by several thousand states could still be considered excessive. Hence in Chapter 6, we apply a selection of well-known variance reduction techniques to gradient estimators using reverse- and forward-time simulations in an attempt to increase efficiency still further. Four standard techniques are discussed, and applied during simulations of the 64-state transfer-line problem. Modest improvements are recorded which compare favourably with results reported in the literature. We conclude that the standard techniques do not offer significant variance reductions in the simulation of complex systems.

We would ideally like to have at our disposal some technique which

would allow the partial solution of the problem using an analytic method, leaving the remainder of the problem to be solved by experimental methods. Such a technique has been used by numerical analysts to refine approximate solutions to linear vector equations. Chapter 6 shows that this method can be effective in reducing the variance of gradient estimates, by using pre-calculated approximations to the "relative cost vector". It is important to calculate the best possible approximation prior to carrying out experimental work and a section of Chapter 6 (and Appendix IV) provides details of suitable techniques for calculating approximate vectors, which use reduced Markov models originally discussed by Norman (43). Gradient estimates are obtained for the standard 64-state and 144-state transfer-line problems using approximate vectors calculated using two types of reduced model. The results show that very large reductions (by up to 98%) in variance are possible, even when simulating complex systems like the transfer-line problems.

In Chapter 7 we calculate some approximate figures for storage and computing time which would be required by the analytic and experimental methods when solving large problems. Hence it is possible to give a rough idea of the limitations of the proposed methods. We also give a short summary of the objectives of the research and discuss the effectiveness of the proposed methods in meeting those objectives. Chapter 7 concludes with some suggestions for further research.

1.4 Contributions of the Thesis

The literature provides details of a wide variety of practical problems which are modelled as finite-state, discrete-time Markov decision processes having very large numbers of states. Previously, attempts have been made to solve these problems using existing analytic

methods, or by direct, 'brute-force' simulation. Solving the problems using existing analytic methods invariably places heavy demands on computing resources, sometimes to such an extent that solution is impossible. Solution using the brute-force simulation approach is tedious and inefficient. Accordingly, the thesis offers as its main contribution a number of new and efficient methods (of both an analytic and experimental nature) for solving large Markov decision problems.

In developing these methods, the thesis makes the following specific contributions:

(i) Conventional (referred to here as 'classical') Markov decision problems define a control structure in which control actions can be chosen independently in each state for which alternative actions exist. The thesis gives details of a new Markov model whose control structure is less complex in that control actions are defined as functions of certain pre-specified groups of states, i.e. the same control action is taken for each of a given group of states. The modified control structure is important because the space of state groups can be made arbitrarily small down to a lower limit of one group (effectively specifying open-loop controls). This 'grouped-state' problem does not appear to have been studied previously, and it is shown that existing algorithms for the classical problem can not be applied directly.

Analytic Solution Methods:

(ii) It is shown that the grouped-state problem can be solved using a dynamic programming algorithm.

(iii) By specifying probabilistic controls (thus defining a continuous space of control parameters), it is shown that there exist convenient expressions for linear and quadratic approximations to the objective function near any given point in parameter space. Some grouped-state problems are solved using the linear approximation in a non-linear programming algorithm.

Experimental Methods:

Our general approach is to use statistical estimates of the objective function gradient at any point in parameter space to indicate a direction in which to search for 'better' control parameters. The estimates are used in an iterative algorithm of the stochastic approximation type which yields the optimal parameter vector. This approach is significantly more efficient than brute-force simulation methods which 'search' the parameter space in a less organised way by estimating system responses at many parameter values, using long simulations to determine each estimate. A more detailed comparison of new and existing experimental methods appears in Chapter 7.

Specific contributions include:

(iv) Expressing the objective function gradient in terms of the state distribution and adjoint vector sequences, it follows that independent estimates of each vector sequence will provide an independent estimate of the gradient. It is shown that a forward-time simulation of an abstract Markov chain provides an estimate of the state distribution sequence.

Furthermore, by using Monte Carlo weighting techniques and simulating the process in reverse-time, it is possible to estimate the adjoint vector sequence by two different methods. Two different gradient estimators follow directly.

(v) The thesis develops methods for efficiently estimating distribution and adjoint vector sequences using a number of forward-time simulations running simultaneously.

For large Markov chain models, the thesis proposes a Monte Carlo method which allows a reduction of the computing burden to match available resources without biasing the results.

Three (forward-time) gradient estimators are explicitly mentioned, and are compared using a simple abstract problem.

(vi) A (seemingly) new problem involving the real-time control of industrial transfer-lines using cross-linking conveyors is posed and solved (using analytic and experimental methods).

Of particular interest is the development of rules for reverse-time simulation. It turns out that a system simulation in reverse-time can be viewed as the dual of a forward-time simulation, i.e. it can be broken down into a number of subsystem simulations, each obeying a dual set of simulation rules.

(vii) The use of gradient estimates in a stochastic approximation algorithm is demonstrated by solving two grouped-state Markov decision problems.

(viii) A selection of well-known techniques have been used in a attempt to reduce the variance of gradient estimates. The resulting variance reductions are found to be disappointing, but it is shown that a refinement technique can be used which has the potential for reducing variance by up to 98% (for the example studied). The method requires a certain amount of prior calculation using one of a selection of 'reduced' Markov models which are also discussed in the thesis.

The combination of the most efficient of the forward-time gradient estimators, together with the refinement technique for variance reduction, represents a highly efficient procedure for estimating gradients. This procedure is attractive not only because it offers a means for reducing variance (statistical error) to very low levels, but also because the gradients are estimated directly, thus completely eliminating the deterministic error which has resulted from previous efforts to estimate gradients by generating extra simulations using perturbed parameters, and calculating 'noisy' finite-difference approximations to each partial derivative.

CHAPTER 2

MARKOV CHAINS - NOTATION, BASIC CONCEPTS AND APPLICATION TO INDUSTRIAL TRANSFER-LINE PROBLEMS

2.1 Introduction

The theme of this thesis is the solution of optimal control problems modelled by large Markov chains and it is therefore essential that we spend some time establishing the Markov chain terminology and notation which will be used later. This is especially important since a unique set of Markov chain definitions has not yet emerged in the literature, and properties such as ergodicity have different meanings for different authors. For the most part, we refer to Heathcote (44) and Kemeny and Snell (45) for basic Markov chain theory, and some results for controlled chains are taken from Howard (7).

We also use this chapter to discuss problems involving industrial transfer-lines, which are well suited to modelling by Markov chains.

2.2 Basic concepts

2.2.1 Properties of a Markov chain

We begin by giving two of the axioms of probability theory which will be used later. For two events, X and Y, the probability of the event (X or Y), sometimes written (X+Y), is given by

$$\text{Prob}\,[X+Y] = \text{Prob}\,[X] + \text{Prob}\,[Y] - \text{Prob}\,[XY] \qquad 2.1$$

If X and Y are disjoint events, i.e. they cannot occur simultaneously, then the probability of the compound event [XY] is zero,

$$\text{Prob } [XY] = 0 \qquad 2.2$$

If X and Y are not disjoint, and the occurrence of Y affects the probability of X, we define the probability of the compound event [XY] by

$$\text{Prob } [XY] = \text{Prob } [X|Y] \,.\, \text{Prob } [Y] \qquad 2.3$$

where Prob [X|Y] is the conditional probability of X given Y. Also, if X and Y are independent, i.e. they can occur simultaneously but do not influence each other, then

$$\text{Prob } [X|Y] = \text{Prob } [X] \qquad 2.4$$

Throughout the thesis, we consider a discrete-time stochastic process having states x_t, $t = 0,1, \ldots,$ where t is the time index. For any t, the random vector x_t is one of a finite set of s-vectors e_k, $k = 1,\ldots, s$ referred to as the state-space of the chain. When x_t depends only on the preceding member of the sequence, x_{t-1}, and not on the process history up to time t, i.e.

$$\text{Prob } [x_t = e_j | x_0 = e_m, x_1, \ldots, x_{t-1} = e_i]$$

$$= \text{Prob } [x_t = e_j | x_{t-1} = e_i] \qquad 2.5$$

$$\text{for } i, j, m \in \{1, \ldots, s\}$$

the process is referred to as a discrete-time Markov process or chain. Furthermore, if the conditional probability given by equation 2.5 does not depend on the time index, then the state sequence $\{x_t\}$ is said to be a homogeneous Markov chain, and we shall restrict our discussion to such chains for the remainder of the thesis.

We define the constants

$$p_{ij} \triangleq \text{Prob}\,[x_t = e_j | x_{t-1} = e_i] \qquad 2.6$$

$$\text{for } i = 1, \ldots, s$$

$$j = 1, \ldots, s$$

and for each t, a probability distribution π_t which is an s-vector having elements

$$\{\text{Prob}\,[x_t = e_k],\ k = 1, \ldots, s\}.$$

If the initial condition is given by the state $x_0 = e_k$, $k \in \{1, \ldots, s\}$ it follows that $\pi_0 = x_0$. The s x s matrix P, having elements $\{p_{ij}\}$, is called the transition (probability) matrix of the Markov chain because it defines the probability of a transition from any state x_t to any other state x_{t+1}. It can be seen that P is a probability matrix having the following properties.

$$P \cdot 1\rangle = 1\rangle \qquad 2.7$$

$$0 \leq p_{ij} \leq 1, \quad i, j \in \{1, \ldots, s\} \qquad 2.8$$

Using the notation above, and equations 2.1 to 2.4, we see that

$$\text{Prob}\,[x_t = e_j] = \sum_{i=1}^{s} \text{Prob}\,[x_{t-1} = e_i] \cdot p_{ij} \qquad 2.9$$

Hence we can re-write equation 2.9 in the following form

$$<\pi_t = <\pi_{t-1} \,.\, P, \quad t = 1, 2, \ldots \qquad 2.10$$

Successive application of equation 2.10 provides us with

$$<\pi_t = <\pi_0 \,.\, P^t, \quad t = 1, 2, \ldots \qquad 2.11$$

and it can be seen that for a homogeneous chain, the probabilistic development of the sequence $\{x_t\}$ is completely known once the values of π_0 and P have been given.

2.2.2 Classification of states and chains

It is helpful to classify states according to whether it is possible to go from a given state to another given state (not necessarily in one transition). We initially give the definitions and offer an illustration later.

(i) States are divided into equivalence classes

Two states are in the same equivalence class if they communicate i.e. if the process can go from either state to the other one.

(ii) Equivalence classes are referred to as ergodic sets (having elements called ergodic states) or transient sets (having

elements called transient states).

There is at least one ergodic set for each Markov chain, but there may be no transient set. A process leaving a transient set can never return to this set, and once having entered an ergodic set, can never leave it. If an ergodic set contains only one state, then this is called an absorbing state.

(iii) We further subdivide an equivalence class into cyclic classes. If there are exactly d cyclic classes, then the process moves cyclically from class to class, with a cycle of length d.

(iv) The cycle degenerates when d = 1 and a single 'cyclic class' implies that after a sufficiently large number of transitions the process can be in any state of the class. Thus for sufficiently high values of i, P^i has only positive entries. A chain with such an equivalence class is said to be regular, otherwise it is said to be cyclic.

If a chain has more than one ergodic set, then there is no interaction between these sets. We thus have two or more unrelated Markov chains lumped together and these chains can be studied separately. Thus we lose no generality by assuming any chain to be a single ergodic set (possibly having additional transient states). A detailed description of these classifications can be found in Kemeny and Snell (45 ,pp.35-37).

To illustrate these classifications, we consider the following transition matrix:

$$P_1 = \begin{bmatrix} 0.3 & 0.7 & 0.0 & 0.0 \\ 0.2 & 0.8 & 0.0 & 0.0 \\ 0.2 & 0.1 & 0.2 & 0.5 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

For this chain, we can identify two ergodic sets and one transient set. State 3 is a transient state, and the process starting in state 3 will eventually make a transition to the ergodic set having elements {state 1 and state 2}, or to the ergodic set consisting of the single absorbing state 4. The process will remain forever in whichever ergodic set it reaches first.

The matrix

$$P_2 = \begin{bmatrix} 0.0 & 0.0 & 0.4 & 0.6 \\ 0.0 & 0.0 & 0.7 & 0.3 \\ 0.2 & 0.8 & 0.0 & 0.0 \\ 0.9 & 0.1 & 0.0 & 0.0 \end{bmatrix}$$

describes a cyclic chain which has two cyclic classes. The process beginning at t = 0 in state 1 or 2 will return to state 1 or 2 at t = 2, 4, 6, ... and will occupy either state 3 or 4 at t = 1, 3, 5, ... Riordon (16) has shown that a cyclic chain can be analysed by treating each cyclic class separately. We therefore devote the whole of this thesis to the study of regular Markov chains, i.e. chains having no transient sets and having a single ergodic set with only one cyclic class.

2.2.3 Steady-state properties of regular Markov chains

The transition matrix for a regular Markov chain is called a

regular transition matrix. Kemeny and Snell (45 ,p.70) prove that if P is a regular transition matrix then

$$\lim_{t\to\infty} \{P^t\} = A \qquad 2.12$$

where A is a probability matrix of the form

$$A = 1 >< \pi \qquad 2.13$$

and π is a unique probability (s-) vector for the chain, satisfying

$$<\pi, 1> = 1 \qquad 2.14$$

$$0 \leq (<\pi, e_j>) \leq 1, \quad j = 1, \ldots, s \qquad 2.15$$

From equations 2.11 and 2.12 we see that

$$\lim_{t\to\infty} \{\pi_t\} = [<\pi_0, 1>]<\pi$$

$$= <\pi \quad \text{for any initial distribution} \qquad 2.16$$

and π gives the probability distribution of the states of the chain in the steady state. It follows that:-

$$<\pi = <\pi P \qquad 2.17$$

and the steady-state distribution is calculated by solving for the s unknown elements of π using the (s+1) equations of 2.14 and 2.17. Further comments on calculation methods are given in Chapter 3.

2.2.4 Regular Markov chains with costs

In the general case we define process costs in the form of an s x s matrix B, having elements $\{ b_{ij} \}$, where

$$b_{ij} \triangleq \left\{ \begin{array}{l} \text{cost incurred by a transition} \\ \text{from state } x_t = e_i \text{ to} \\ \text{state } x_{t+1} = e_j \text{ at } t = 0, 1, \ldots \end{array} \right\}$$

Some examples, e.g. the transfer-line problem described in Section 2.3, have a simpler cost structure defined by a vector c, having elements $\{ c_j \}$, $j = 1, \ldots, s$; where

$$c_j \triangleq \left\{ \begin{array}{l} \text{cost incurred by a transition } \underline{\text{to}} \text{ state} \\ x_{t+1} = e_j \text{ from any state } x_t \text{ at } t = 0, 1 \ldots. \end{array} \right\}$$

For the regular transition matrix P, with elements $\{ p_{ij} \}$, and process cost matrix B, we can calculate q_i, the expected cost incurred by the process in making a transition from state $x_t = e_i$.

$$q_i = \sum_{j=1}^{s} p_{ij} \, b_{ij}, \quad i = 1, \ldots, s \qquad 2.18$$

The s-vector q, having elements $\{ q_i \}$ is referred to as the one-stage expected cost vector for the chain. It can immediately be seen that the mean process cost per transition is given by g, where

$$g = \langle \pi, q \rangle \qquad 2.19$$

and π is calculated using equations 2.14 and 2.17.

If we now assume that the process is to make N transitions from the initial state x_o, and we define element i of the s-vector v_t to be the expected total cost incurred by the process in making [N-t] transitions from state $x_t = e_i$, then we have

$$v_t> = q> + Pv_{t+1}> \quad , t = 0, \ldots, N - 1 \qquad 2.20a$$

and we shall always take

$$v_N> = 0 \qquad 2.20b$$

The vectors $\{ v_t, t = N, N-1, \ldots, 0 \}$ can be calculated using equations 2.20a and 2.20b in reverse time, and Howard (7) has shown by Z-transform analysis that v_t assumes the following asymptotic form for large [N-t],

$$v_t> = [N - t]g\ 1> + v> \qquad 2.21$$

where once again g is the mean process cost per transition, and the s-vector v denotes the asymptotic intercepts of v_t.* The importance of the vector v will become apparent later when controlled chains are discussed. Substituting the asymptotic form of v_t into equation 2.20a we obtain

$$[N - t]g\ 1> + v> = q> + P([N - t - 1]g\ 1> + v>) \qquad 2.22$$

Using equation 2.7, and rearranging, we obtain

$$g\ 1> + v> = q> + Pv> \qquad 2.23$$

---

* i.e. v gives the intercepts of the asymptotes with the [N-t] = 0 axis

The vector equation 2.23 represents s simultaneous equations in (s+1) unknowns (g and the s elements of v), and we cannot find a unique solution. A straightforward substitution and use of equation 2.7 shows that if $(g, v>)$ solves equation 2.23, then so does $(g, \bar{v}>)$ where $\bar{v}> = v> + a1>$ and a is any scalar. However, it will become clear that we only require an indication of relative costs and so we can force the s'th element to be zero by setting $a = -\langle e_s, v\rangle$ and then solving for s unknowns to obtain $(g, [v> - \langle e_s, v\rangle .1>])$. We will refer to $\bar{v}$ as the 'relative cost vector' for the chain because its i'th element gives the marginal cost involved in allowing the process to make a large number of transitions from state $x_t = e_i$. Further comments on the calculation of vectors π and $\bar{v}$ are found later in the thesis.

2.2.5 Controlled Markov chains : the Markov decision problem

We shall use as an illustration a simple two-state example adapted from one used by Howard (7).

We assume that a toymaker's business can exist in one of two (process-) states:

1. It can have a successful toy
2. It can have an unsuccessful toy.

The effects of random fluctuations in the toy market are modelled by changes in the state of business at the end of each accounting period. Probabilities and costs of state transitions are given by the transition matrix P' and cost matrix B' i.e.

$$P' = \begin{bmatrix} .5 & .5 \\ .4 & .6 \end{bmatrix} \qquad B' = \begin{bmatrix} -9 & -3 \\ -3 & 7 \end{bmatrix}$$

where negative costs are interpreted as profit.

Using equation 2.18, we obtain the one-stage expected cost vector q', i.e.

$$q' = \begin{bmatrix} -6 \\ 3 \end{bmatrix}.$$

We now allow the toymaker a measure of control over his business by introducing advertising as an option which can be exercised for the duration of an accounting period. The process is best described using notation found in Riordon's Ph.D thesis (16) where each accounting period (or "process cycle") is imagined to be split into two phases. In the first phase the process makes a control transition from the present process state to a control state identified by a process state/control alternative combination. In the second phase, the process makes a process transition from the control state to the next process state. Control alternatives have the effect of specifying both probabilities and costs for transitions from the present process state.

Process details for the toymaker example are summarised in Table 2.1

i	Control state defined by: Process state	Control state defined by: Control alternative	Probability of transition from control state e_i to each process state, i.e. $<e_i P$	Transition costs i.e. $<e_i . B$	One-stage expected cost i.e. $<e_i, q>$
1	Successful toy	No advertising	[.5 .5]	[-9 -3]	-6
2	Unsuccessful toy	No advertising	[.4 .6]	[-3 +7]	+3
3	Successful toy	Advertising	[.8 .2]	[-3 -3]	-3
4	Unsuccessful toy	Advertising	[.7 .3]	[-1 +19]	+5

Table 2.1 : Toymaker example : process transition details

It can be seen that the toymaker example has two process states and four control states, though control transitions are allowed between the initial process state and only two of the control states. The two phases of the process cycle can be represented by two probability matrices $D(\alpha)$ and P, with the 2 x 2 matrix $P(\alpha)$ representing the system transition matrix given by the matrix product,

$$P(\alpha) = D(\alpha) \cdot P \qquad 2.24$$

In the general case with s process states and n control states, (the next section will show that control and process states can be identical with s = n), α is a parameter vector, $D(\alpha)$ is an s x n matrix, B and P are constant n x s matrices and the n-vector q is the one-stage expected cost vector resulting from the use of P and B in equation 2.18. For the toymaker example, these quantities are given by

$$D(\alpha) = \begin{bmatrix} \alpha_1 & 0 & (1-\alpha_1) & 0 \\ 0 & \alpha_2 & 0 & (1-\alpha_2) \end{bmatrix}$$

$$P = \begin{bmatrix} .5 & .5 \\ .4 & .6 \\ .8 & .2 \\ .7 & .3 \end{bmatrix}, \quad B = \begin{bmatrix} -9 & -3 \\ -3 & +7 \\ -3 & -3 \\ -1 & +19 \end{bmatrix} \quad q = \begin{bmatrix} -6 \\ 3 \\ -3 \\ 5 \end{bmatrix}$$

with $\alpha = [\alpha_1 \ \alpha_2]^T$

Here α_1 represents the probability of selecting control state 1 when in process state 1. Hence when the toymaker has a successful toy at the start of an accounting period, $D(\alpha)$ indicates that with probability α_1, he fails to advertise, and consequently that he advertises with probability $(1 - \alpha_1)$. The parameter α_2 is defined in a similar way.

In the general case, where k_i control options exist for process state e_i, the i'th row of $D(\alpha)$ has (k_i-1) parameters and we retain the condition $D(\alpha)1\rangle = 1\rangle$. Furthermore it can easily be verified that $P(\alpha)$ is a probability matrix, i.e.

$$P(\alpha)\ 1\rangle = 1\rangle$$

$$0 \leq (\langle e_i, P(\alpha)e_j\rangle) \leq 1 \qquad 2.25$$

for $i, j \in \{1,\ldots,s\}$ and all feasible α's

Using a stationary parameter vector α, the one-cycle expected cost vector is defined by

$$q(\alpha)\rangle \triangleq D(\alpha)q\rangle \qquad 2.26$$

and so if $\{x_t\}$ is the sequence of process states, the long run expected cost per unit time is

$$J(\alpha) = \lim_{T\to\infty} \frac{1}{T+1} E\{\sum_{t=0}^{T} \langle x_t, q(\alpha)\rangle\} \qquad 2.27$$

and the classical problem is

$$\underset{\alpha}{\text{minimize}} \quad \{J(\alpha)\} \tag{2.28}$$

where we minimize over the range of feasible α's.

An equivalent formulation of the problem is

$$\underset{\alpha}{\text{minimize}} \quad \{g(\alpha)\} \tag{2.29}$$

where $g(\alpha)$ is the mean process cost per transition given by

$$g(\alpha) = \langle \pi(\alpha), q(\alpha) \rangle \tag{2.30}$$

and $\pi(\alpha)$ is the solution of

$$\left.\begin{aligned} \langle \pi(\alpha) &= \langle \pi(\alpha) \cdot P(\alpha) \\ \langle \pi(\alpha), 1 \rangle &= 1 \end{aligned}\right\} \tag{2.31}$$

From the 2 x 4 matrix $D(\alpha)$ given above, we see that controls for the toymaker example are defined such that actions can be chosen independently for each process state. The resulting problem is therefore a 'classical' Markov decision problem according to the definition given in Chapter 1. We can formulate a 'grouped-state' decision problem for the same process by insisting on the use of an identical (probabilistic) control action in each of the two process states. A mathematical description of the 'grouped-state' control structure is provided by the revised form of $D(\alpha)$ below,

$$D(\alpha) = \begin{bmatrix} \alpha & 0 & (1-\alpha) & 0 \\ 0 & \alpha & 0 & (1-\alpha) \end{bmatrix}$$

where α is a scalar parameter defining the probability of not advertising during the accounting period, a probabilistic action to be applied in each process state. Substituting this control matrix into equations 2.24 and 2.26 leads directly to a statement of a grouped-state Markov decision problem using equations 2.27 to 2.31.

A feature distinguishing 'classical' from 'grouped-state' problems is that even though probabilistic control policies are feasible, the optimal policy for the classical problem is always deterministic, i.e. rows of $D(\alpha)$ are of the form e_i, $i \in \{1, \ldots, n\}$. This is not always the case for grouped-state control structures : in fact we solve the toymaker example above with grouped-state controls in Chapter 3 and the resulting optimal control strategy is probabilistic.

2.3 Problems involving industrial transfer-lines

We have established in the previous sections that the Markov decision process is a simple model possessing elegant results. The model is therefore of great interest to operations researchers, applied mathematicians, statisticians, economists and engineers. Probably the major area of application is to operations research (O.R.) problems, but unfortunately even systems of a modest size give rise to Markov models having very large numbers of states. Solving large classical decision problems by existing analytic methods requires vast amounts of computer storage, and the alternative approach using multiple system simulations makes inefficient use of computing resources.

The aim of this thesis is to present new and efficient methods for determining optimal controls for any operational system modelled by a Markov decision process. Though several small (2 and 3-state) Markovian models are used throughout the text as simple illustrations of the new solution methods, we also indicate the potential of the new methods for solving larger problems by considering an O.R. problem involving two cross-linked transfer-lines.

A transfer-line problem was chosen for a number of reasons, not least of which was that transfer-line problems have been studied by many researchers in the past, and solved using conventional methods. Also, the size of the resulting Markov model can be varied between 64 and a more realistic number of states e.g. 10,000, without changing the basic structure of the problem. If follows that we can use the proposed methods to solve a 64-state problem and accurately assess their performance when used to solve larger problems.

2.3.1 Definitions and description of transfer-line operation

Buzacott's studies of transfer-lines (36 and 37) provide an excellent introduction to the topic, and his definitions and mathematical model need only be extended slightly to accommodate the cross-linked transfer-line problem which is discussed throughout this thesis. We define the transfer-line in this case to be an industrial production line whose stages (machines) operate with constant service times.

The stages of the line are arranged serially so that each workpiece is initially processed at the same stage and then transfers via some form of buffer store to the next stage, continuing in this manner until the piece passes through the final stage, at which point one unit of

production is realised for the line. Though finite capacity interstage buffers are used, we always assume that an infinite stock of pieces is available before the first stage of the line and that infinite capacity exists at the end of the line to accommodate finished pieces. We assume that the line is balanced, implying that all pieces begin to transfer from one stage to the next at the same time instant, the time between successive transfers being referred to as the "cycle time". Thus during a cycle when consecutive stages are working, the contents of their interstage buffer remains constant. Also we assume that the time taken to transfer pieces between stages is negligible compared with the cycle time.

We identify states of each stage as up if the stage is in working order or down if it is broken down. The operation of a stage is characterised by two parameters. The operating time is defined to be the number of pieces processed by the stage between successive breakdowns. Similarly the repair time is taken to mean the number of cycles required to repair the stage once it has broken down. We take all time measurements for the line in units of process cycles; thus operating and repair times are integer valued variables. In fact, these quantities are random variables and in order to describe the process mathematically we are required to specify suitable probability distribution functions, using observed data from a real transfer line to determine distribution parameters.

2.3.2 Identification of operating and repair time distributions

For the general case, Buzacott (37) assumes that operating time has a geometric distribution i.e. the probability of a stage breaking down in a cycle is a constant, regardless of the time since the last breakdown. He comments that although tool wear implies an increasing

probability of stoppage with time, it would be normal to carry out a programme of tool replacement between shifts so that tool wear would not be a major cause of breakdown. The paper describes as an example an automatic transfer line machining cylinder blocks where the time between transfers is taken to be the process cycle time. Buzacott observes that both operating and repair time distributions are geometric for this example.

As an example in this thesis, we consider small sections of canmaking lines, which operate at a much faster rate than Buzacott's machining process. In this case therefore, it is convenient to define the cycle time not as the time between transfers but as a predetermined period of production during which a number of pieces is processed, or not, according to whether stages are in up or down states. A work study report (46) analysing can line performance over a number of years in U.K. factories confirms that with a cycle defined in this way, repair and operating times for individual machines or line sections are invariably found to have geometric distributions.

Insufficient information has prevented precise identification of the parameters relating to the metal-forming machines in question but the work study report (46) has been used to obtain rough estimates of the mean operating time (10 cycles) and mean repair time (4 cycles) for a can-making machine which we shall use to represent each machine in the study. These estimates correspond to constant probabilities of breakdown, (B = 0.1) and of repair, (REP = 0.25) within the next process cycle.

2.3.3 Transitions of a two-stage line or line section

By a two-stage line we mean the configuration shown in Fig. 2.1.

The interstage buffer store can hold sufficient pieces to maintain operation of machine 2 for Z cycles and is therefore said to have a capacity of Z (production) units.

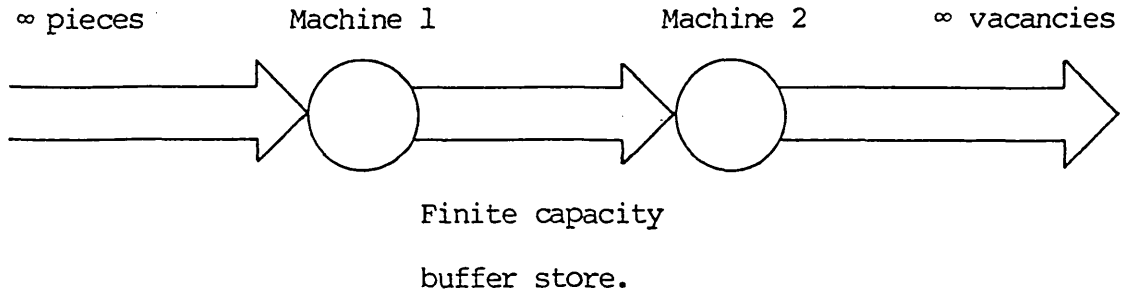


Fig. 2.1 : Two-stage line configuration

We assume that the system state can only be observed just after the transfer of the last piece processed in each cycle. We further assume that the observed state has been maintained since the last observation instant, to be consistent with our earlier definition of 'cycle time'. The system state can be uniquely represented by the vector (X_1,X_2,X_3), where X_1 represents the state ϵ {Up, Down}, of machine 1, X_2 represents the state ϵ {Up, Down}, of Machine 2, and X_3 represents the state $\epsilon\{0,1,\ldots,Z\}$ of the buffer store. The total number of states required to model the line is therefore given by the number of possible combinations of the variables X_1, X_2, and X_3, i.e. 2 x 2 x (Z+1). We shall always consider geometric operating and repair time distributions, and we can show graphically the conditions governing state transitions in Figs. 2.2 and 2.3. It can be seen that transitions from states (D,U,O) and (U,D,Z) are the only exceptions to the general transition rules shown in Fig. 2.3. States (D,U,O) and (U,D,Z) are referred to as 'force-down' states, so called because they describe situations where either machine 2 or 1 is forced to stop operation (even though it has not broken down) because of the state of the buffer store. Force-down states are of particular interest in the simulation of production line

State at time (t-1) transition probability State at time t

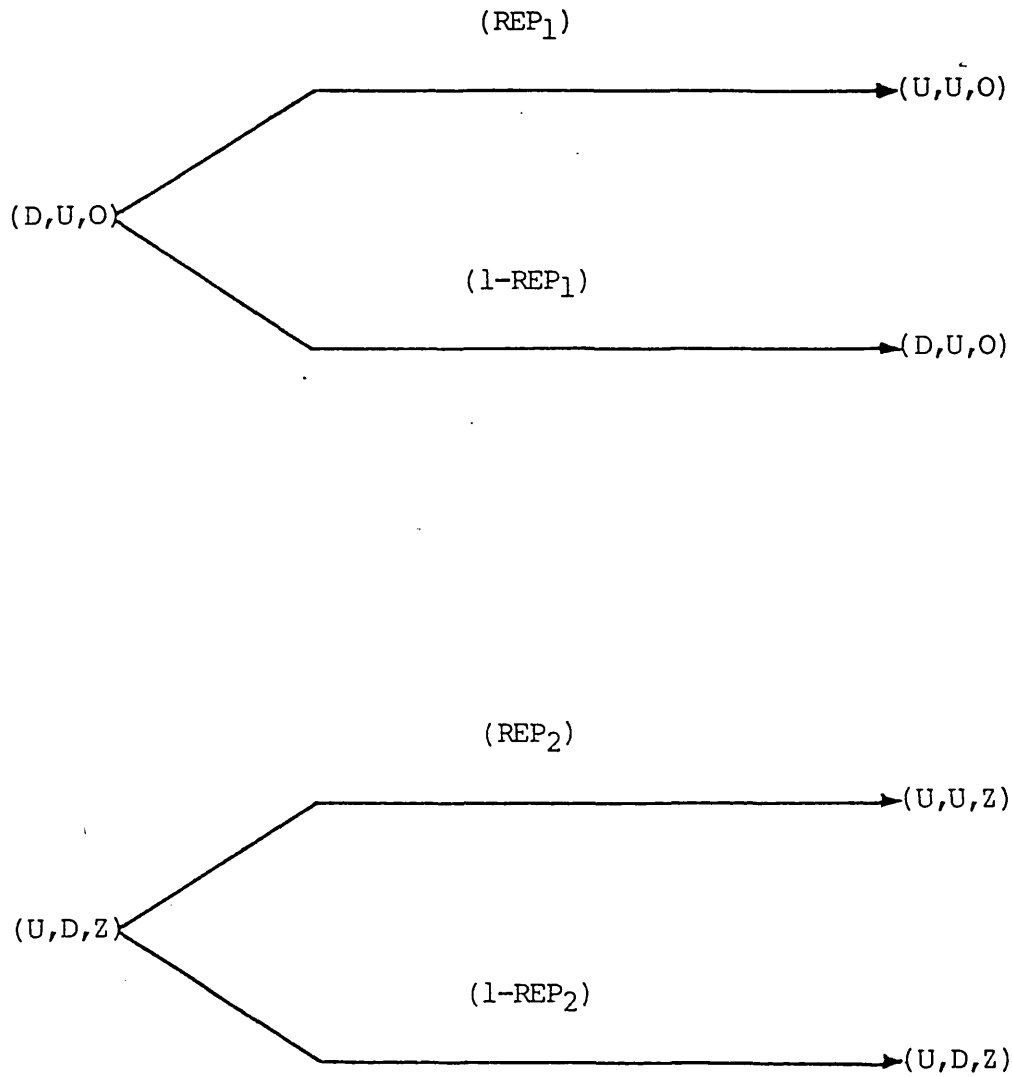


Fig. 2.2: Two-stage line; transitions from states (D,U,O) and (U,D,Z)

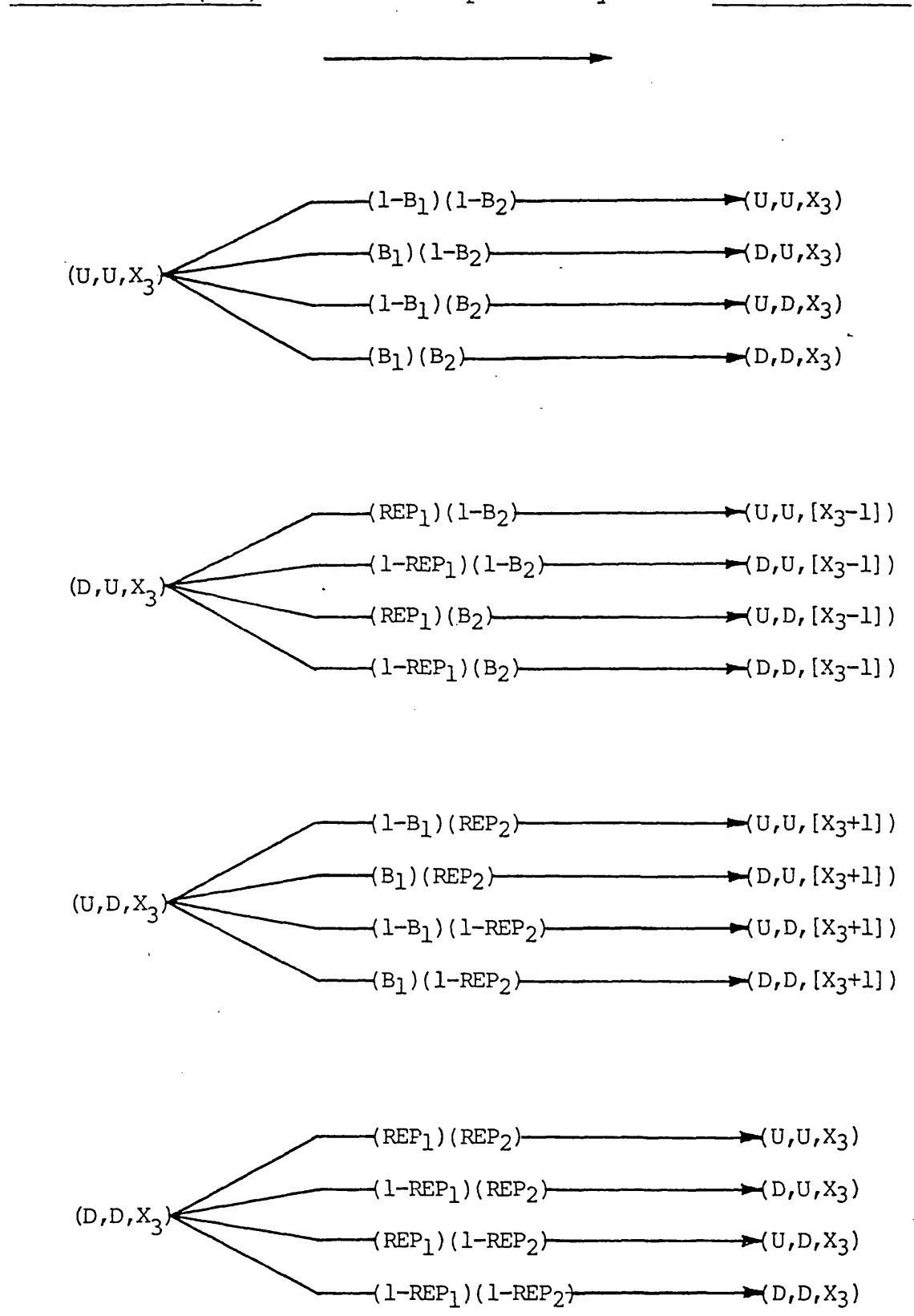


Fig. 2.3: Two-stage line; transitions from all states except (D,U,O) and (U,D,Z)

operation and are therefore discussed fully in Chapter 5. In Figs. 2.2 and 2.3, probabilities of transition are expressed in terms of geometric distribution parameters, B_i and REP_i, which represent the probabilities of breakdown and repair of machine i within the next process cycle.

From the definitions above, we see that if machine 2 is not forced down, then its state at the observation instant, (Up or Down), determines the output of the line, (one production unit or none) since the latest observation instant.

2.3.4 The transition matrix and cost vector for a two-stage line

We would like to describe transfer-line problems using the standard Markov chain notation given in Section 2.2, and the first step is to express each state in the form e_i, where i is an integer. The one condition which must be satisfied is that if $i \triangleq f(X_1, X_2, X_3)$, where X_1 and X_2 are redefined by equation 2.32,

$$\left.\begin{array}{l} \text{for } j = 1,2; \quad X_j = 2 \text{ , if machine j is up} \\ \qquad\qquad\quad = 1 \text{ , if machine j is down} \end{array}\right\} \qquad 2.32$$

then the function f should be one-to-one. There is an infinite number of functions which would satisfy this condition, and f is therefore chosen to produce a form of the transition matrix which is 'best' in some sense. Our recommended function is defined by equation 2.33

$$i = f(X_1, X_2, X_3) \triangleq (X_3).2^2 + (2-X_2).2^1 + (X_1).2^0 \qquad 2.33$$

In defining this function our aim was to produce a transition matrix whose form closely resembles a (block-) diagonal matrix. For a

two-stage line, with Z = 2, application of equation 2.33 with transition probabilities defined by Figures 2.2 and 2.3 results in the block tri-diagonal matrix given in Fig. 2.4.

Partitioning the matrix into 4x4 submatrices, denoted by A, B, C, D and E gives

$$P_2 \triangleq \begin{bmatrix} A & D & O \\ E & B & D \\ O & E & C \end{bmatrix} \qquad 2.34$$

Extending the notation for a transfer-line with general Z, we have

$$P_Z \triangleq \begin{bmatrix} A & D & & & \\ E & B & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & B & D \\ & & & E & C \end{bmatrix} \qquad 2.35$$

Z+1 submatrices

Once the parameters B_j, REP_j; j = 1, 2 have been identified, the constant submatrices A, B, C, D, and E can be evaluated and equation 2.35 used to model transfer-lines having buffers of any size. Though they do not explicitly state the function f above, Gershwin and Schick (38) assume the same form of transition matrix in their study, and give an efficient procedure for inverting a block tri-diagonal matrix by exploiting this structure. Further comments on this paper appear in Chapter 3 where we are concerned with computational methods for solving Markov decision problems.

$$
P_2 \triangleq
\begin{bmatrix}
(1-REP_1) & (REP_1) & & & & & & & & & & & & & & \\
(B_1)(1-B_2) & (1-B_1)(1-B_2) & (B_1)(B_2) & (1-B_1)(B_2) & & & & & & & & & & & & \\
(1-REP_1)(REP_2) & (REP_1)(REP_2) & (1-REP_1)(1-REP_2) & (REP_1)(1-REP_2) & & & & & & & & & & & & \\
 & & & & (B_1)(REP_2) & (1-B_1)(REP_2) & (B_1)(1-REP_2) & (1-B_1)(1-REP_2) & & & & & & & & \\
(1-REP_1)(1-B_2) & (REP_1)(1-B_2) & (1-REP_1)(B_2) & (REP_1)(B_2) & & & & & & & & & & & & \\
 & & & & (B_1)(1-B2) & (1-B_1)(1-B_2) & (B_1)(B_2) & (1-B_1)(B_2) & & & & & & & & \\
 & & & & (1-REP_1)(REP_2) & (REP_1)(REP_2) & (1-REP_1)(1-REP_2) & (REP_1)(1-REP_2) & & & & & & & & \\
 & & & & & & & & (B_1)(REP_2) & (1-B_1)(REP_2) & (B_1)(1-REP_2) & (1-B_1)(1-REP_2) & & & & \\
 & & & & (1-REP_1)(1-B_2) & (REP_1)(1-B_2) & (1-REP_1)(B_2) & (REP_1)(B_2) & & & & & & & & \\
 & & & & & & & & (B_1)(1-B_2) & (1-B_1)(1-B_2) & (B_1)(B_2) & (1-B_1)(B_2) & & & & \\
 & & & & & & & & (1-REP_1)(REP_2) & (REP_1)(REP_2) & (1-REP_1)(1-REP_2) & (REP_1)(1-REP_2) & & & & \\
 & & & & & & & & & (REP_2) & & (1-REP_2) & & & &
\end{bmatrix}
$$

NOTE: Only non-zero entries are shown.

Fig. 2.4: The transition matrix for a two-stage line with $Z = 2$. States are identified using equation 2.33

Referring back to Sections 2.2.4 and 2.3.3, it can be seen that transfer-lines have a cost structure described by a vector c, where the cost incurred during a transition to state e_j is given by $\langle e_j, c \rangle$. Rather than write c as a vector of negative costs as discussed in Section 2.2.4, with elements 0 or -1 measured in production units, we use conventional transfer-line notation and interpret c to be a vector of returns realised by the line at the end of each transition. Here, c has non-negative elements (0 or +1), and we are generally required to maximize a function of c by specifying some control policy. The change from function minimization to function maximization does not require significant changes to any of the theory discussed later.

From Section 2.3.3 the vector c_Z^* for a two-stage line with a buffer of capacity Z is given by

$$c_Z = [0100|\underbrace{1100|1100|.....|1100}_{Z \times [1100]}] \qquad 2.36$$

It should be noted that state e_1 identifies machine 2 as being in a working state, but no pieces are available from the buffer for processing, resulting in zero output.

The definition of "regular chain" given in Section 2.2.2 requires that a process starting in any state x_o should be able to reach any other state of the chain in a finite number of transitions. Equivalently, raising the transition matrix to a sufficiently high power should result in a matrix having no zero entries. Applying either criterion to the chain having transition matrix P_Z (given by equation 2.35), confirms that any two-stage transfer-line model is a regular chain.

---

* Referring to equation 2.33 we see that the i'th element is '1' or '0' according to whether or not the downline machine can process a piece. Rather than using the vector c as on P.59, we use c_Z to indicate the general form of the vector for a two-stage line with buffer capacity Z.

2.3.5 Controlled transfer-lines

Previous studies of transfer-lines have to a great extent concentrated on two areas i.e. design problems (specification of position and size of buffer stores), and the calculation of operating efficiency of existing lines. Examples are the studies by Freeman (35), Buzacott (36 and 37), Gershwin and Schick (38) and the simulation approach due to Ho et al. (39), (40). The transfer-line design problem is significant because Markov models of lines with a large number of stages connected by interstage buffers have enormous numbers of states, e.g. a four-stage line with interstage buffers of capacity 10 gives rise to 21,296 states. Conceptually, the allocation of buffer capacity to optimize line efficiency is a straightforward function maximization problem, but the size of the model leads to major complications. Arguably more complex are problems involving real-time control of transfer-lines. Buzacott (37) has posed and solved two and three-stage transfer-line problems where machines capable of working are deliberately stopped in order to adjust buffer stock levels. Gershwin and Schick (38) also mention the possibility of real-time control by adjusting machine speeds.

The real-time control problem studied in this thesis is one involving sections of two can-making lines linked by cross-conveyors. The use of cross-conveyors in two-piece can (i.e. a type of beverage can) manufacture was suggested by Brilman (41) in his study of conveying methods for tinplate cans. In a two-piece canmaking factory, different types of can (usually the same size but decorated differently) are produced by independent multi-stage lines. Though the end product of each line may be different, a number of operations near the start of each line are identical and Brilman indicates at least four opportunities for using cross-conveyors to link a pair of lines. The

fact that many of these canmaking lines are now controlled by central computer means that we can consider controlling the movement of pieces between two or more lines by detecting the states of each line in real-time, and controlling the cross-conveyors in an attempt to increase the combined output of the lines. For the purposes of this study we will consider a simplified version (see Fig. 2.5) of Brilman's line configuration involving two identical line sections (each having two machines and an interstage buffer) and a cross-conveyor.

Fig. 2.5 indicates that our control options include running the two lines independently or moving any number of pieces in either direction between lines A and B. Brilman's paper discusses a range of equipment (mass conveying systems, etc.) which might be used to implement such a scheme.

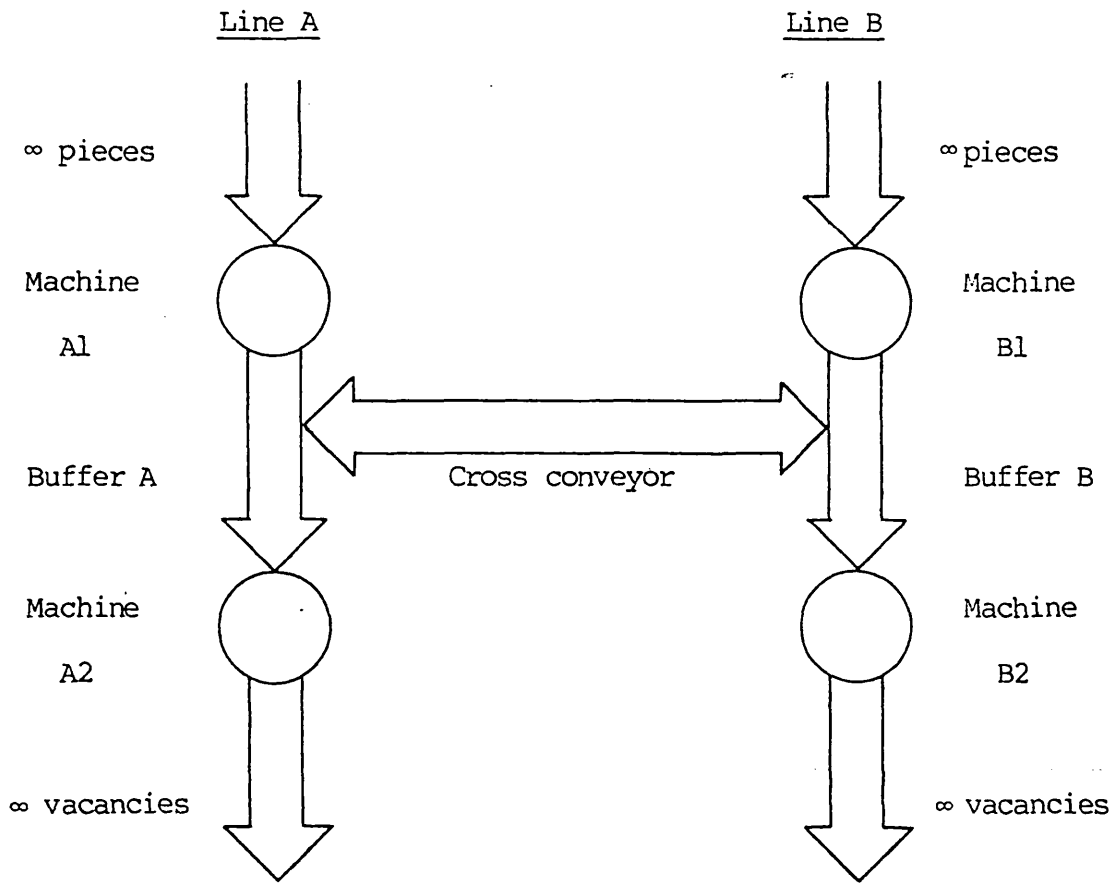


Fig. 2.5: Schematic diagram showing two cross-linked canmaking lines.

If each line section is described by one of a number, $\sqrt{s}$, of states, then the total number of states describing the cross-linked system is s. We observe that when the buffer store of each line section has a capacity of Z units, then

$$s = [2 \times 2 \times (Z+1)]^2 \qquad 2.37$$

When states of lines A and B are identified by ($\sqrt{s}$-) vectors e_m and e_n respectively, (where m and n are determined using equation 2.33), then the corresponding state of the cross-linked system can be uniquely represented by the (s-) vector e_i, where

$$i \triangleq [(m-1) \times \sqrt{s}] + n \qquad 2.38$$

with $m, n \in \{1, \ldots, \sqrt{s}\}$ and $i \in \{1, \ldots, s\}$

It is worth noting that in reality, the sequence of events in each cycle would be

(i) to observe the state of the system

(ii) to determine the number of pieces (possibly none) to be transferred between the two lines basing our decision on (i), and implementing the transfer.

(iii) to observe the new state of the system after completion of machine operations during the cycle.

Also bearing in mind our assumption that transport time for the pieces is negligible compared to the cycle time, we see that (i), (ii) and (iii) above are accurately described using Riordon's notation (see

Section 2.2.5) where the space of control states is identical to the space of process states, and both $D(\alpha)$ and P are s x s transition matrices. The crosslinking feature is conveniently separated from the operation of two independent lines (each having s states) by the use of this notation. An interesting point is that the system transition matrix P, describing transitions of the independent lines, has a similar structure to the one which describes a three-stage line with two inter-stage buffers. Both matrices have diagonal blocks which are themselves block tri-diagonal and the matrices are described as having a nested block tri-diagonal structure. This property allows the use of the efficient matrix inversion algorithm given by Gershwin and Schick (38), but unfortunately the introduction of crosslinking control schemes destroys this simple structure. The process transition matrix $P(\alpha)$ $(= D(\alpha)P)$ is also far less sparse than P.

2.4 Summary

The chapter begins with a review of Markov chain notation and terminology, leading to a discussion of controlled chains and Markov decision problems. Small examples illustrate results which are established for use in later chapters of the thesis and special attention is paid to the steady-state properties of regular Markov chains.

A class of operations research problems which has received much attention in the literature involves industrial transfer-lines (i.e. production lines having deterministic cycle times). This type of problem is well suited to modelling by Markov decision processes but even relatively small systems require models with large numbers of states, making solution by conventional methods tedious and in some cases impossible. The aim of the thesis is to present new solution

methods for large problems like these and the remainder of Chapter 2 is therefore devoted to a review of existing terminology for transfer-line problems and a description of a new problem involving cross-linked transfer-lines with real-time controls.

CHAPTER 3

ANALYTIC SOLUTION METHODS

3.1 Introduction

The optimal control problem referred to in Chapter 1 as the 'classical' Markov decision problem has been a popular research topic for the last 25 years. Three alternative optimization techniques for solving this problem, i.e. dynamic programming, linear programming, and Howard's policy iteration algorithm, emerged within the first 3-4 years, and most of the solution methods proposed since then have been variants of these three methods. It is natural therefore, to attempt to solve grouped-state Markov decision problems using these algorithms.

In this chapter we take the simplest form of grouped-state problem, i.e. one having an open-loop control structure, and attempt to formulate it in such a way that it can be solved by dynamic programming, linear programming, and policy iteration algorithms. Though the linear programming and conventional policy iteration algorithms cannot be applied directly, it is possible to use a non-linear programming algorithm which employs a linear program as a subroutine, and resembles Howard's algorithm in that it searches the space of probabilistic control policies for the optimal point.

3.2 Dynamic Programming Methods

3.2.1 Conventional dynamic programming solution of a classical Markov decision problem

We consider N transitions of a finite-state Markov chain x_t,

$t = 0, \ldots, N$ with state space $\{e_1, e_2, \ldots, e_s\}$, whose transition probability at time t depends on the control u_t chosen at t. Thus Prob $[x_{t+1}|x_t]$ = Prob $[x_{t+1}|x_t,u_t]$. For each $x_t = e_i$, $i = 1, \ldots, s$, we have a finite set, U_i, of control alternatives, labelled by the integers $\{1, 2, \ldots, k_i\}$. State transitions can be defined by:

$$x_{t+1} = f(x_t, u_t, w_t), \quad t = 0, 1, \ldots, N-1 \qquad 3.1$$

where $\{ w_t, t = 0, 1, \ldots, N \}$ is a sequence of independent random variables with appropriately defined distributions. The sequence can be regarded as an element, ω, of the sample space denoted by Ω. We assume that the initial state, x_o, is given and that controls are specified as a feedback policy, ψ, being a sequence of control laws, $\{h_0(\cdot), \ldots, h_{N-1}(\cdot)\}$. The control law at t defines the control action u_t as a function of x_t,

$$u_t = h_t(x_t), \quad t = 0, \ldots, N-1 \qquad 3.2$$

The system transition costs are denoted by the scalar functions $L_t(x_t, u_t, w_t)$, $t = 0, \ldots, N-1$, where the sequences $\{x_t\}$ and $\{u_t\}$ satisfy equations 3.1 and 3.2 respectively.

We define the trajectory cost by:

$$\tilde{V}_0(x_0, \psi, \omega) \triangleq \sum_{t=0}^{N-1} L_t(x_t, u_t, w_t) \qquad 3.3$$

We see from equation 3.1 that the sequence $\{x_t\}$ depends on ω and is more accurately written $\{x_t(\omega)\}$. Similarly, from equation 3.2, $\{u_t\}$ is actually a sequence of random variables, $\{u_t(\omega)\}$. The trajectory cost

is therefore a random variable whose expected value is given by

$$V_0(x_0, \psi) = \underset{\omega \in \Omega}{E} \{\tilde{V}_0(x_0, \psi, \omega)\} \qquad 3.4$$

Optimal Control Problem

We are required to find an admissible policy, ψ , such that $V_0(x_0, \psi)$ is minimized.

In order to solve the problem using dynamic programming we initially define the functions

$$\tilde{V}_t(x_t, \psi, \omega) \triangleq \sum_{i=t}^{N-1} L_i(x_i, u_i, w_i) \qquad 3.5a$$

for $t = 1, \ldots, N-1$

and $\tilde{V}_N(x_N, \psi, \omega) \triangleq 0$ 3.5b

The sequences $\{x_i, u_i;\ i = t, \ldots, N-1\}$ above, satisfy equations 3.1 and 3.2. It follows that

$$\tilde{V}_t(x_t, \psi, \omega) = L_t(x_t, u_t, w_t) + \tilde{V}_{t+1}(x_{t+1}, \psi, \omega) \qquad 3.6$$

Considering all sequences ω including the state x_t, we obtain the expectation of equation 3.6:

$$\underset{\omega|x_t}{E} \{\tilde{V}_t(x_t, \psi, \omega)\} = \underset{\omega|x_t}{E} \{L_t(x_t, u_t, w_t) + \tilde{V}_{t+1}(x_{t+1}, \psi, \omega)\} \qquad 3.7$$

By extending the notation of equation 3.4, equation 3.7 becomes

$$V_t(x_t, \psi) = \underset{w_t|x_t}{E} \{L_t(x_t, h_t(x_t), w_t) + V_{t+1}(x_{t+1}, \psi)\} \qquad 3.8$$

where $x_{t+1} = f(x_t, u_t, w_t)$

We use the superscript 'o' to denote optimal values, i.e. $h_t^o(\cdot)$ is the optimal control law at t, $\psi^o = \{h_0^o(\cdot), \ldots, h_{N-1}^o(\cdot)\}$ is the optimal feedback policy, and $V_t^o(x_t)$ denotes $V_t(x_t, \psi^o)$.

The backward recursion equation of dynamic programming is

$$V_t^o(x_t) = \min_{u_t} \{ \underset{w_t|x_t}{E} \{L_t(x_t, u_t, w_t) + V_{t+1}^o(x_{t+1})\}\} \qquad 3.9$$

where $x_{t+1} = f(x_t, u_t, w_t)$

which is used to determine $V_t^o(\cdot)$ in reverse time, i.e. we solve for $V_N^o(\cdot) = 0$, $V_{N-1}^o(\cdot)$, ..., $V_0^o(\cdot)$, at the same time providing the required optimal feedback control policy, ψ^o.

3.2.2 Use of dynamic programming to determine optimal open-loop controls

It can be seen from the previous section that the conditional expectation in equation 3.9 automatically makes u_t a function of x_t. Conventional dynamic programming is thus suited to the calculation of optimal feedback controls. When we consider open-loop control schemes (i.e. the simplest form of grouped-state control structure) for a Markov decision process, the concept of dynamic programming can be used if we

begin by defining the probability distributions of x_t, $t = 0, \ldots, N$ as the system hyperstates. Dynamic programming is used to calculate optimal control laws which are functions of the hyperstates, and the sequence of optimal open-loop control actions can then be determined in a straightforward manner.

The sequence of hyperstates is denoted by $\{ \pi_t, t = 0, \ldots, N \}$, each vector π_t having elements $\{\pi_t^i\}$ defined by

$$\pi_t^i \triangleq \text{Prob}\,[x_t = e_i], \quad i = 1, \ldots, s \qquad t = 0, \ldots, N \qquad 3.10$$

It is possible to mirror the discussion of the previous section, beginning with the transition rule for the hyperstates:

$$\pi_{t+1} = g(\pi_t, u_t), \quad t = 0, \ldots, N-1 \qquad 3.11$$

where in this case the control action u_t is one of a set U of alternatives labelled by the integers $\{1, 2, \ldots, k\}$.

We assume the initial condition, $\pi_0 (\equiv x_0)$ to be given and that a sequence γ, of control laws $\{d_0(\cdot), \ldots, d_{N-1}(\cdot)\}$ is to be calculated. Putting

$$u_t = d_t(\pi_t), \quad t = 0, \ldots, N-1 \qquad 3.12$$

we allocate costs to hyperstate transitions by the scalar functions $M_t(\pi_t, u_t)$, $t = 0, \ldots, N-1$, using $\{\pi_t\}$ as defined by equation 3.11. We define trajectory cost by

$$C_0(\pi_0, \gamma) \triangleq \sum_{i=0}^{N-1} M_i(\pi_i, u_i) \qquad 3.13$$

Optimal Control Problem

We are required to find the sequence of control laws

$$\gamma = \{d_0(\cdot), \ldots, d_{N-1}(\cdot)\} \text{ minimizing } C_0(\pi_0, \gamma).$$

Defining the functions

$$C_t(\pi_t, \gamma) = \sum_{i=t}^{N-1} M_i(\pi_i, u_i) \qquad 3.14a$$

$$\text{for } t = 1, \ldots, N-1$$

$$\text{and} \quad C_N(\pi_N, \gamma) = 0 \qquad 3.14b$$

where $\{\pi_i, u_i;\ i = t, \ldots, N-1\}$ satisfy equations 3.11 and 3.12. It follows that

$$C_t(\pi_t, \gamma) = M_t(\pi_t, u_t) + C_{t+1}(\pi_{t+1}, \gamma) \qquad 3.15$$

where $\pi_{t+1} = g(\pi_t, u_t)$.

If the optimal γ-sequence is denoted by $\gamma^o = \{d_0^o(\cdot), \ldots, d_{N-1}^o(\cdot)\}$ and $C_t^o(\pi_t)$ denotes $C_t(\pi_t, \gamma^o)$ then the backward recursion equation of dynamic programming is, in this case,

$$C_t^o(\pi_t) = \min_{u_t} \{M_t(\pi_t, u_t) + C_{t+1}^o(\pi_{t+1})\} \qquad 3.16$$

where $\pi_{t+1} = g(\pi_t, u_t)$.

Solution of equation 3.16 in reverse-time yields the sequence of optimal costs $\{C_t^o(\cdot)\}$ together with the sequence of optimal control laws $\gamma^o = \{d_0^o(\cdot), \ldots, d_{N-1}^o(\cdot)\}$. Solving equation 3.11 in forward-time from the initial condition π_0 provides the hyperstate sequence $\{\pi_t\}$ which can then be used in conjunction with γ^o to provide the sequence of optimal open-loop control actions.

3.2.3 Discussion

In Section 3.2.1 we have established that when the system is in state $x_t = e_i$ at time t, we can select a control action u_t from U_i, a set of integers labelling the control alternatives. The choice of control action influences the distribution of the random variable w_t, so that in selecting $u_t \in U_i$, we are effectively specifying the conditional distribution of x_{t+1}, i.e. $\{p_{ij}^{u_t}, j = 1, \ldots, s\}$, where

$$p_{ij}^{u_t} \triangleq \text{Prob}\,[x_{t+1} = e_j | x_t = e_i, u_t] \qquad 3.17$$

The s rows of the transition probability matrix P^{u_t} are defined by

equation 3.17 with i = 1 ,...., s and equation 3.2 specifying the s control actions at time t.

From the open loop analysis of Section 3.2.2 we see that equation 3.11 can be written as:

$$\pi_{t+1} = \pi_t \cdot P^{u_t}, \quad t = 0, \ldots, N-1 \tag{3.18}$$

Because the control action at time t is specified as a function of the state distribution rather than the state, u_t in equation 3.18 indicates which of the alternative transition matrices should be used.

Transition costs for the chain are given in the form of an s x s matrix B with elements $\{b_{ij}\}$. b_{ij} is defined to be the cost incurred by the process in making a transition from state $x_t = e_i$ to state $x_{t+1} = e_j$. The cost function $M_t(\pi_t, u_t)$ is the average cost of a single transition of the system whose states have a distribution π_t at t. The function is defined by:

$$M_t(\pi_t, u_t) \triangleq \sum_{i=1}^{s} \sum_{j=1}^{s} \pi_t^i \cdot p_{ij}^{u_t} \cdot b_{ij} \tag{3.19}$$

It is important to note that the conventional dynamic programming formulation of Section 3.2.1 solves a discrete-time Markov decision problem with a discrete state-space, and a discrete action-space. In Section 3.2.2 on the other hand, we solve a discrete-time Markov decision problem with a discrete action space but a continuous space of hyperstates, i.e. π_t satisfies the conditions

$$\left.\begin{aligned} &\pi_t^i \in [0, 1], \quad i = 1, \ldots, s \\ &\langle \pi_t, 1 \rangle = 1 \end{aligned}\right\} \quad t = 0, \ldots, N-1 \tag{3.20}$$

so that whereas the value of $V_t^o(x_t)$ is obtained for each t = N, N-1, ..., 0 by solving equation 3.9, and setting $x_t = e_1, e_2, \ldots, e_s$, $C_t^o(\pi_t)$ theoretically has to be calculated for each possible vector π_t, at t = N, N-1, ..., 0. For this, and for other applications of dynamic programming where the state space is continuous, the algorithm can only be programmed for a computer if the optimal cost function is approximated in some way. Several methods have appeared in the literature, e.g. Bellman and Dreyfus (3) select a grid of parameter values, and use an interpolation technique to obtain other values when required. Another of Bellman's suggestions is to approximate the optimal cost function by curve fitting. The differential dynamic programming technique due to Jacobson and Mayne (6) is also applicable. As the application of dynamic programming in a continuous state space has been considered previously, we close the discussion here, using the next section to illustrate closed- and open-loop solutions of a simple problem.

3.2.4 A 3-state example

As an illustration of the dynamic programming procedures we can calculate optimal feedback and open-loop controls for a simple 3-state Markov decision process making three transitions from the initial state (or state distribution),

$$x_0 = [0 \quad 1 \quad 0]$$
$$= \pi_0$$

At the beginning of each transition, we choose a control action, u_t from two alternatives, {Up, Down}, the effect of which is to specify the probability distribution of the next system state, according to Table 3.1:

u_t	π_{t+1}
Up	[0.75 0.0 0.25]
Down	[0.25 0.0 0.75]

Table 3.1: Effect of control over state transitions.

For feedback controls, equation 3.2 implies that row i of the 3 x 3 transition matrix P^{u_t} will be specified by Table 3.1 according to whether $u_t (= h_t(e_i))$ is up or down.

Open-loop controls define alternative transition matrices as follows:

$$P^{u_t = up} = 1> \cdot [0.75 \quad 0.0 \quad 0.25]$$

$$P^{u_t = down} = 1> \cdot [0.25 \quad 0.0 \quad 0.75] \qquad 3.21$$

State transition costs, which are independent of control selections, are given by the matrix

$$B = \begin{bmatrix} 8 & 0 & 4 \\ 3 & 0 & 6 \\ 4 & 0 & 12 \end{bmatrix} \qquad 3.22$$

Optimal Feedback Controls

We evaluate $\psi^o = \{h_0^o(\cdot), h_1^o(\cdot), h_2^o(\cdot)\}$ by solving equation 3.9 in reverse-time, e.g. the calculations necessary to determine $h_2^o(.)$ are listed in Table 3.2.

x_2	$h_2(x_2)$ ($*h_2^o(x_2)$)	$V_2(x_2, \psi)$ ($*V_2^o(x_2)$)
e_1	Up	(.75 x 8) + (.25 x 4) = 7
e_1	Down *	(.25 x 8) + (.75 x 4) = 5 *
e_2	Up *	(.75 x 3) + (.25 x 6) = 3.75*
e_2	Down	(.25 x 3) + (.75 x 6) = 5.25
e_3	Up *	(.75 x 4) + (.25 x 12) = 6 *
e_3	Down	(.25 x 4) + (.75 x 12) = 10

Table 3.2: Evaluation of $h_2^o(\cdot)$.

The optimal feedback control policy ψ^o is summarised in Table 3.3.

x_t	$h_2^o(x_2)$	$V_2^o(x_2)$	$h_1^o(x_1)$	$V_1^o(x_1)$	$h_0^o(x_0)$	$V_0^o(x_0)$
e_1	Down	5	Down	10.75	Down	16.125
e_2	Up	3.75	Up	9	Up	14.625
e_3	Up	6	Up	11.25	Up	16.875

Table 3.3: Optimal feedback control policy.

Optimal Open-Loop Controls

We begin by calculating the average cost function $M_t(\pi_t, u_t)$ using equation 3.19:

$$\left.\begin{aligned} M_t(\pi_t, \text{Up}) &= <\pi_t \cdot \begin{bmatrix} 7 \\ 3.75 \\ 6 \end{bmatrix} \\ M_t(\pi_t, \text{Down}) &= <\pi_t \cdot \begin{bmatrix} 5 \\ 5.25 \\ 10 \end{bmatrix} \end{aligned}\right\} \qquad 3.23$$

In the general case, a dynamic programming solution involves the evaluation of $C_t^o(\cdot)$, a function of the continuous vector variable π_t, at each t = N, N-1, ..., 0. However for the 3-state example above, a few

simple calculations using equations 3.18 and 3.21 reveal that π_t, t = 1, ..., N can only assume values [.75 0 .25] or [.25 0 .75] according to the choice of u_t, and π_0 is [0 1 0] as previously stated. Thus it is only necessary to evaluate $C_t^o(\cdot)$ for two π_t values at each t except t = 0, when $C_0^o([0\ 1\ 0])$ is calculated.

Table 3.4 summarizes the results of the open loop analysis.

π_t, t = 2, 1, 0	$d_2^o(\pi_2)$	$C_2^o(\pi_2)$	$d_1^o(\pi_1)$	$C_1^o(\pi)_1$	$d_0^o(\pi_0)$	$C_0^o(\pi_0)$
[.25 0 .75]	Up	6.25	Up	12.5	-	-
[0 1 0]	-	-	-	-	Up	16.25
[.75 0 .25]	Down	6.25	Down	12.5	-	-

Table 3.4: Open-loop analysis.

A comparison of Tables 3.3 and 3.4 confirms that the optimal feedback solution yields a lower cost over 3 transitions than the optimal open-loop solution, as would be expected. The optimal open-loop control strategy can be found by calculating the sequence $\{\pi_t, t = 0, 1, 2\}$ starting at π_0 = [0 1 0] and using the control sequence $\gamma^o = \{d_0^o(\pi_0), d_1^o(\pi_1), d_2^o(\pi_2)\}$ which turns out to be {Up, Down, Up}.

3.3 Linear Programming Solutions

3.3.1 Introduction

In this section, and for the remainder of the chapter, we consider problems having an infinite time horizon, and seek to minimize the long-run expected cost per transition of the system. It will be shown in the following discussion that the technique known as linear programming can be used to solve classical Markov decision problems, but

that the definition of control actions as functions of state groups does not permit a suitable formulation for solution by linear program. However, the discussion is included here not only for completeness, but also because the linear programming approach initially requires the transformation of a discrete action space into a continuous one, and this proves useful in later work.

3.3.2 Solution of the classical problem by linear programming

The application of linear programming to infinite Markov decision processes was studied originally by Wolfe and Dantzig (9) and later by Mine and Osaki (10) where the relationship between linear programming and Howard's policy iteration algorithm was established.

As in Section 3.2.1 we assume that for a system presently in state e_i, $i \in \{1, \ldots, s\}$ we can choose from a set of control alternatives labelled by the integers $\{1, 2, \ldots, k_i\}$ and denote the chosen control by u_i. Hence we effectively specify alternative distribution vectors (rows of the transition matrix) for each of s states, and the resulting transition matrix defines conditional probabilities for state transitions at $t = 0, 1, \ldots$.

We illustrate this procedure by considering a three-state Markov chain where two control alternatives are allowed in each state. Thus for the system in state e_i, we can choose between two alternative row vectors $[p_{i1}^k \; p_{i2}^k \; p_{i3}^k]$, $k = 1, 2$ denoted by p_i^k.

The formulation of the process as a linear programming problem involves transforming the discrete action space into a continuous one by considering stationary probabilistic control policies.

Let α_i^k represent the probability of choosing control action k in state e_i, for i = 1, 2, 3; k = 1, 2 and the vector $\alpha_i = [\alpha_i^1 \ \alpha_i^2]$. Thus the distribution vector specified by the probabilistic control action α_i adopted in state e_i is given by:

$$p_i^{\alpha_i} \triangleq \alpha_i^1 \ p_i^1 + \alpha_i^2 \ p_i^2 = \sum_{k=1}^{2} \alpha_i^k \ p_i^k \qquad 3.24$$

and the system transition matrix is

$$P(\alpha_1, \alpha_2, \alpha_3) = \begin{bmatrix} \sum_{k=1}^{2} \alpha_1^k \ p_1^k \\ \sum_{k=1}^{2} \alpha_2^k \ p_2^k \\ \sum_{k=1}^{2} \alpha_3^k \ p_3^k \end{bmatrix} \qquad 3.25$$

and, letting α denote the stationary probabilistic policy $\{\alpha_1, \alpha_2, \alpha_3\}$, we calculate the steady state probability distribution from

$$<\pi(\alpha) = <\pi(\alpha) \cdot [P(\alpha)] = \sum_{i=1}^{3} \sum_{k=1}^{2} \pi_i(\alpha) \cdot \alpha_i^k \cdot p_i^k \qquad 3.26$$

so that, using a probabilistic policy, the constraints become

$$\left.\begin{aligned} &\pi_j(\alpha) - \sum_{i=1}^{3} \sum_{k=1}^{2} \pi_i(\alpha) \cdot \alpha_i^k \cdot p_{ij}^k = 0, \quad j = 1, 2, 3 \\ &\sum_{j=1}^{3} \pi_j(\alpha) = 1 \end{aligned}\right\} \qquad 3.27$$

If we assume the existence of a matrix B, with elements $\{b_{ij}\}$ where b_{ij} defines the cost of a transition between $x_t = e_i$ and $x_{t+1} = e_j$, then the expected cost of operating the process over one transition from state $x_t = e_i$, using control k is:

$$q_i^k = \sum_{j=1}^{3} p_{ij}^k \cdot b_{ij} \qquad 3.28$$

Thus the expected process cost per transition using the policy α is

$$\sum_{i=1}^{3} \sum_{k=1}^{2} \pi_i(\alpha) \cdot \alpha_i^k \cdot q_i^k \qquad 3.29$$

It is helpful to define the variable

$$y_i^k \triangleq \pi_i(\alpha) \cdot \alpha_i^k \qquad 3.30$$

and we see that

$$\sum_{k=1}^{2} y_i^k = \pi_i(\alpha) \sum_{k=1}^{2} \alpha_i^k = \pi_i(\alpha) \qquad 3.31$$

We can now transform the original Markov decision problem into a standard linear programming problem, i.e.

$$\left.\begin{aligned} &\text{minimize} \sum_{i=1}^{3}\sum_{k=1}^{2} y_i^k \cdot q_i^k \\ &\text{subject to} \sum_{k=1}^{2} y_j^k - \sum_{i=1}^{3}\sum_{k=1}^{2} p_{ij}^k \, y_i^k = 0 \\ &\qquad\qquad j = 1, 2, 3 \\ &\sum_{i=1}^{3}\sum_{k=1}^{2} y_i^k = 1 \\ &y_i^k \geq 0 \end{aligned}\right\} \quad 3.32$$

We have assumed a completely ergodic 3-state process for which Kemeny and Snell (45) prove that one of the constraints of equation 3.32 is redundant. Thus we are left with a linear programming problem in 6 variables with 3 constraints. The relevant table of coefficients, omitting the constraint for j = 3, is shown below:

y_1^1	y_1^2	y_2^1	y_2^2	y_3^1	y_3^2
$p_{11}^1 - 1$	$p_{11}^2 - 1$	p_{21}^1	p_{21}^2	p_{31}^1	p_{31}^2
p_{12}^1	p_{12}^2	$p_{22}^1 - 1$	$p_{22}^2 - 1$	p_{32}^1	p_{32}^2
1	1	1	1	1	1

Table 3.5: Table of coefficients

Wolfe and Dantzig (9) prove that a solution of equations 3.32 has the property that for each i there is at most one k for which $y_i^k > 0$, i.e.

the optimal stationary policy is not probabilistic. Thus from equations 3.30 and 3.31 it follows that for each i, if $y_i^k > 0$, then $y_i^k = \pi_i(\alpha^o)$, where α^o is the optimal policy. Also α^o must represent a non-probabilistic policy. General principles of linear programming will not be discussed here, and are probably best treated by Dantzig (47).

3.3.3 Reformulation of the open-loop control problem

For the Markov decision problem where we are required to choose an open-loop control action from a set of alternatives in order to minimize the expected long-run process cost per transition, we might seek a problem formulation allowing solution by linear program.

For the three-state process described previously, we assume the existence of two alternative open-loop controls. Thus by selecting control k, we are effectively specifying a 3 x 3 matrix P^k, having rows p_1^k, p_2^k, and p_3^k. We transform the discrete action space into a continuous one by defining α^k to be the probability of choosing control alternative k such that

$$\left.\begin{aligned} &\sum_{k=1}^{2} \alpha^k = 1 \\ &\alpha^k \geq 0, \quad k = 1, 2 \\ &\text{and} \quad \alpha \triangleq [\alpha^1 \;\; \alpha^2]^T \end{aligned}\right\} \qquad 3.33$$

For such a probabilistic control selection, the system transition matrix is

$$P(\alpha) \triangleq [\sum_{k=1}^{2} \alpha^k \cdot P^k] \qquad 3.34$$

and the stationary probability distribution for the states is

$$<\pi(\alpha) = <\pi(\alpha) [\sum_{k=1}^{2} \alpha^k \cdot P^k] = \sum_{i=1}^{3} \sum_{k=1}^{2} \pi_i(\alpha) \cdot \alpha^k \cdot p_i^k \qquad 3.35$$

Using the definition for the one-stage expected cost q_i^k, above, we find that the expected process cost per transition using probabilistic controls is

$$\sum_{i=1}^{3} \sum_{k=1}^{2} \pi_i(\alpha) \cdot \alpha^k \cdot q_i^k \qquad 3.36$$

Defining a variable z_i^k:

$$z_i^k \triangleq \pi_i(\alpha) \cdot \alpha^k \qquad 3.37$$

we have now formulated a problem which is suitable for solution by linear program.

Using the variable z_i^k the problem is

$$\left.\begin{aligned}
&\text{minimize } \{ \sum_{i=1}^{3} \sum_{k=1}^{2} z_i^k \cdot q_i^k \} \\
&\text{subject to constraints} \\
&\sum_{k=1}^{2} z_j^k - \sum_{i=1}^{3} \sum_{k=1}^{2} p_{ij}^k \cdot z_i^k = 0, \quad j = 1, 2, 3 \\
&\sum_{i=1}^{3} \sum_{k=1}^{2} z_i^k = 1 \\
&z_i^k \geq 0 \quad , \quad \text{all } i, k
\end{aligned}\right\} \qquad 3.38$$

The table of coefficients in this case is identical to Table 3.5, with y_i^k replaced by z_i^k. Obviously, the same linear programming solution cannot generally be correct for both full state feedback and open-loop cases.*

It is not difficult to find an example (see Section 3.5.3) for which the optimal open-loop control strategy is probabilistic, i.e. $0 < \alpha^k < 1$, for at least two values of k. Also for the regular Markov chain which we are considering, $\pi_i(.) > 0$, for each i, so that from equation 3.37, $z_i^k > 0$ for at least two values of k. Wolfe and Dantzig have shown that a linear programming solution produces, for each i, only one z_i^k which is non-zero and it follows that solving the linear programming problem (equations 3.38) will not provide the required optimal open-loop control strategy.

Though it has not been found possible to express the open-loop problem in a form allowing solution by linear program, the use of probabilistic control structures has led to the definition of analytic

* Note the difference between equations 3.30 and 3.37. Although both linear programming problems 3.32 and 3.38 can be solved for y_i^k and z_i^k respectively, it is not then possible to solve equation 3.37 for <u>all</u> cases. For grouped-state problems, α must appear explicitly in one of the constraints.

and Monte Carlo algorithms which successfully solve the problem and which are described later in the thesis.

3.4 Approximation in policy space

3.4.1 Howard's policy iteration algorithm

The policy iteration algorithm was first put forward by Howard (7) as a solution method for classical Markov decision problems. A proof of convergence for the algorithm as originally defined is provided, with examples, by Howard, and by Mine and Osaki (10), where its relationship to the linear programming solution is also discussed. We therefore summarize the algorithm as it is applied to regular Markov chains having s states.

Notation for the classical Markov decision problem is given in Section 3.3.2 and here we let f denote a stationary control policy, $[f_1, \ldots, f_s]^T$ where f_i is the label of the control alternative $\epsilon \{1, \ldots, k_i\}$ used when the system begins a transition from state e_i. The transition matrix P(f) has rows defined by equation 3.17, with $f_i = u_t = h_t(e_i)$, $t = 0, 1, \ldots$. Our problem, as before, is to minimize, with respect to all feasible policies, the mean process cost per transition, denoted by g(f), where

$$g(f) = \langle \pi(f), q(f) \rangle \qquad 3.39$$

q(f) has elements $\{q_i^{f_i}, i = 1, \ldots, s\}$ as defined by equation 3.28. Howard's algorithm has two parts: we begin by choosing any feasible stationary policy, f.

Value Determination Routine

For the stationary policy, f, solve the vector equation

$$g(f) \cdot 1\!\!> + v(f)\!\!> = q(f)\!\!> + P(f) \cdot v(f)\!\!> \qquad 3.40$$

where v(f) is the 'relative cost (s-) vector', discussed in Section 2.2.4.

Policy Improvement Scheme

Using the vector v(f) determined previously, we revise the stationary policy by defining f_i to be the control alternative $k \in \{1, \ldots, k_i\}$ satisfying

$$\min_k \{q_i^k + \sum_{j=1}^{s} p_{ij}^k \cdot v_j\} \qquad 3.41$$

for each i = 1, ..., s.

If any element of f is changed by the policy improvement routine, we return to the value determination routine and repeat the cycle. The algorithm stops when the policy improvement routine fails to yield a revised policy, at which point we are assured that the current policy f is optimal, incurring the minimum mean process cost g(f).

3.4.2 Howard's algorithm and the open-loop control problem

An important feature of Howard's algorithm is that the policy

improvement routine involves s pointwise or decoupled minimizations. Decoupling is possible only because feedback controls are allowed. When we are restricted to the same choice of control action for each of a group of states, as in the open-loop case, pointwise minimizations are effectively ruled out, and to implement a scheme similar to Howard's, we would require some knowledge of the stationary probability distribution of the states. As a first step towards solving the open-loop problem by a non-linear programming algorithm, we modify the problem slightly to allow probabilistic control strategies, making the transformation in much the same way as in Section 3.3.3.

Let a regular s-state Markov chain have open-loop control alternatives labelled $\{1, \ldots, k\}$. Each control alternative implies the use of an s x s transition matrix $\in \{P^i, i = 1, \ldots, k\}$ and a one-stage expected cost vector $\in \{q^i, i = 1, \ldots, k\}$, where q^i has elements $\{q^i_j, j = 1, \ldots, s\}$ defined by equation 3.28. We now define a matrix valued function $P(\alpha)$ and a vector valued function $q(\alpha)$ by:

$$P(\alpha) \triangleq \alpha_1 P^1 + \alpha_2 P^2 + \ldots + \alpha_{k-1} P^{k-1} + [1 - \sum_{i=1}^{k-1} \alpha_i] P^k \qquad 3.42$$

$$q(\alpha) \triangleq \alpha_1 q^1 + \alpha_2 q^2 + \ldots + \alpha_{k-1} q^{k-1} + [1 - \sum_{i=1}^{k-1} \alpha_i] q^k \qquad 3.43$$

The elements of the parameter vector, $\alpha \triangleq [\alpha_1, \ldots, \alpha_{k-1}]$ give the probability of selecting control alternative i, for each i = 1, ..., k-1 and it follows that

$$\left.\begin{aligned} & 0 \leq \alpha_i \leq 1, \quad i = 1, \ldots, k-1 \\ & \sum_{i=1}^{k-1} \alpha_i \leq 1 \end{aligned}\right\} \qquad 3.44$$

Given any feasible parameter vector α, the stationary distribution for the states is denoted by $\pi(\alpha)$, which satisfies

$$\left.\begin{aligned} & <\pi(\alpha) = <\pi(\alpha) \,.\, P(\alpha) \\ & <\pi(\alpha), 1> = 1 \\ & <\pi(\alpha), e_i> \leq 1, \quad i = 1, \ldots, s \end{aligned}\right\} \qquad 3.45$$

and we are left with the non-linear programming problem of finding a feasible α minimizing the cost function $g(.)$, where

$$g(\alpha) \triangleq <\pi(\alpha), q(\alpha)> \qquad 3.46$$

subject to the constraints given in equation 3.45.

As the space of parameters (probabilistic controls) is continuous, we can use the gradient $\nabla_\alpha g(.)$ in a standard non-linear program, if $g(.)$ can be proved to be continuously differentiable. Consider equation 3.40, where instead of a stationary control policy, f, we use a probabilistic control strategy, or parameter vector, α. Equation 3.40 becomes

$$g(\alpha) \,.\, 1> + v(\alpha)> = q(\alpha)> + P(\alpha) \,.\, v(\alpha)> \qquad 3.47$$

This is an s-vector equation which we solve for $g(\alpha)$ and $v(\alpha) = [v_1(\alpha), \ldots, v_s(\alpha)]^T$ by setting $<e_s, v(\alpha)> = 0$ and re-arranging to give

$$\psi(\alpha) \cdot z(\alpha)> = q(\alpha)> \qquad 3.48$$

where

$$z(\alpha)> \triangleq [v_1(\alpha) \ \ldots \ v_{s-1}(\alpha) \ g(\alpha)]^T \qquad 3.49$$

and

$$\psi(\alpha) \triangleq (I - P(\alpha))(I - e_s><e_s) + 1><e_s \qquad 3.50$$

In Appendix II, we prove that $g_\alpha(\cdot)$ and $g_{\alpha\alpha}(\cdot)$ exist and are continuous mappings, i.e. g(.) is twice continuously differentiable.

3.4.3 Approximating the continuous function g(.) near a feasible point, $\hat{\alpha}$.

It is proved in Appendix II that the function g(.) is twice continuously differentiable. It follows that by expanding g(.) in a Taylor series about any feasible point $\hat{\alpha}$, we can truncate the series at the third term to obtain a quadratic approximation to g(.), passing through the point $g(\hat{\alpha})$ at $\hat{\alpha}$, or at the second term to determine the tangent hyperplane to the surface g(.) at $\hat{\alpha}$. We define $h(., \hat{\alpha})$ and $f(., \hat{\alpha})$ to be the first and second order approximations respectively, i.e.

$$h(\alpha, \hat{\alpha}) \triangleq g(\hat{\alpha}) + <g_\alpha(\hat{\alpha}), \ (\alpha - \hat{\alpha})> \qquad 3.51$$

and

$$f(\alpha, \hat{\alpha}) \triangleq g(\hat{\alpha}) + <g_\alpha(\hat{\alpha}), \ (\alpha - \hat{\alpha})> + \tfrac{1}{2} \cdot <(\alpha - \hat{\alpha}), \ G(\hat{\alpha}) \cdot (\alpha - \hat{\alpha})> \qquad 3.52$$

where $G(\hat{\alpha})$ $(= g_{\alpha\alpha}(\hat{\alpha}))$ is the (k-1) x (k-1) matrix referred to as the Hessian of g at $\hat{\alpha}$.

There exist convenient expressions for these approximations which require little more than the calculation of the matrix inverse, $\psi^{-1}(\hat{\alpha})$. This calculation corresponds to the value determination step of Howard's algorithm, and is the feature which limits its use for large Markov decision problems. It is also a factor which limits the use of the non-linear programming algorithm to be discussed later.

Proposition 3.1

The function $h(., \hat{\alpha})$ can be evaluated at any feasible point α using equation 3.53.

$$h(\alpha, \hat{\alpha}) = g(\hat{\alpha}) + <\pi(\hat{\alpha}), [q(\alpha) - \psi(\alpha) \cdot z(\hat{\alpha})]> \qquad 3.53$$

where

$$<\pi(\hat{\alpha}) = <e_s \psi^{-1}(\hat{\alpha}) \qquad 3.54$$

As ψ and q are affine functions, we see that the RHS of equation 3.53 is also an affine function. Therefore in order to prove Proposition 3.1 we simply have to show that

(i) $h(\hat{\alpha}, \hat{\alpha}) = g(\hat{\alpha})$

and (ii) $h_\alpha(\hat{\alpha}, \hat{\alpha}) = g_\alpha(\hat{\alpha})$

Proof

From equations 3.42, 3.43 and 3.50 we deduce that ψ is a matrix-valued affine function and q is a vector-valued affine function. If α, having elements $\{\alpha_i, i = 1, \ldots, k-1\}$, is any feasible parameter vector, then we define

$$\psi(\alpha) \triangleq \alpha_1 S_1 + \alpha_2 S_2 + \ldots + \alpha_{k-1} S_{k-1} + T \qquad 3.55$$

and

$$q(\alpha) \triangleq \alpha_1 u_1 + \alpha_2 u_2 + \ldots + \alpha_{k-1} u_{k-1} + w \qquad 3.56$$

where $\{S_i, i = 1, \ldots, k-1\}$ and T are real s x s matrices

and $\{u_i, i = 1, \ldots, k-1\}$ and w are real s-vectors.

(i) From equation 3.48, we obtain

$$z(\alpha)> = \psi^{-1}(\alpha) \cdot q(\alpha)> \qquad 3.57$$

and using the definition of z(.),

$$g(\alpha) = <e_s, z(\alpha)> = <e_s, \psi^{-1}(\alpha) \cdot q(\alpha)> \qquad 3.58$$

Also, we have

$$g(\alpha) = <\pi(\alpha), q(\alpha)> \qquad 3.46$$

$$\therefore \quad <\pi(\alpha) = <e_s \cdot \psi^{-1}(\alpha) \qquad 3.59$$

Using equations 3.59 and 3.53

$$h(\hat{\alpha}, \hat{\alpha}) = g(\hat{\alpha}) + <e_s \cdot \psi^{-1}(\hat{\alpha}), [q(\hat{\alpha}) - \psi(\hat{\alpha}) \cdot z(\hat{\alpha})]>$$

$$h(\hat{\alpha}, \hat{\alpha}) = g(\hat{\alpha}) + g(\hat{\alpha}) - g(\hat{\alpha})$$

$$= g(\hat{\alpha})$$

(ii) Expanding equation 3.48 using the notation above,

$$\alpha_1 S_1\, z(\alpha) + \alpha_2 S_2 z(\alpha) + \ldots + \alpha_{k-1} S_{k-1} z(\alpha) + Tz(\alpha)$$

$$= \alpha_1 u_1 + \alpha_2 u_2 + \ldots + \alpha_{k-1} u_{k-1} + w \qquad 3.60$$

Differentiating both sides of equation 3.60 with respect to α,

$$[S_1 z(\alpha) \,\vdots\, S_2 z(\alpha) \,\vdots\, \cdots \,\vdots\, S_{k-1} z(\alpha)] + \psi(\alpha) z_\alpha(\alpha) = [u_1 \,\vdots\, \cdots \,\vdots\, u_{k-1}]$$

$$\therefore\ z_\alpha(\alpha) = \psi^{-1}(\alpha)\{[u_1 \,\vdots\, \cdots \,\vdots\, u_{k-1}] - [S_1 z(\alpha) \,\vdots\, \cdots \,\vdots\, S_{k-1} z(\alpha)]\} \qquad 3.61$$

$$\therefore\ g_\alpha(\hat{\alpha}) = \langle e_s,\ z_\alpha(\hat{\alpha})\rangle$$

$$= \langle \pi(\hat{\alpha})\ \{[u_1 \,\vdots\, \cdots \,\vdots\, u_{k-1}] - [S_1 z(\hat{\alpha}) \,\vdots\, \cdots \,\vdots\, S_{k-1} z(\hat{\alpha})]\} \qquad 3.62$$

Writing equation 3.53 using the same notation and differentiating gives

$$h_\alpha(\hat{\alpha},\ \hat{\alpha}) = \langle \pi(\hat{\alpha}),\ \{[u_1 \,\vdots\, \cdots \,\vdots\, u_{k-1}] - [S_1 z(\hat{\alpha}) \,\vdots\, \cdots \,\vdots\, S_{k-1} z(\hat{\alpha})]\}$$

$$= g_\alpha(\hat{\alpha}),\ \text{from equation 3.62}$$

It follows that equation 3.53 describes the tangent hyperplane to the function $g(\cdot)$ at $\hat{\alpha}$. Q.E.D.

Proposition 3.2

$f(\cdot,\ \hat{\alpha})$ can be calculated at any α using equation 3.63

$$f(\alpha, \hat{\alpha}) = g(\hat{\alpha}) + \langle \pi(\hat{\alpha}), [q(\alpha) - \psi(\alpha) \cdot z(\hat{\alpha})] \rangle$$

$$- \langle \pi(\hat{\alpha}) \cdot S(\alpha - \hat{\alpha}) \psi^{-1}(\hat{\alpha}), [q(\alpha) - \psi(\alpha) z(\hat{\alpha})] \rangle \qquad 3.63$$

where $S(.) \triangleq \psi(.) - T$ is a linear matrix-valued function.

Proof

Let $u(.)$ be the vector-valued function defined by $u(.) \triangleq q(.) - w$. As ψ and q are affine mappings we can write

$$\psi(\alpha) = \psi(\hat{\alpha}) + S(\alpha - \hat{\alpha}) \qquad 3.64$$

and

$$q(\alpha) \rangle = q(\hat{\alpha}) \rangle + u(\alpha - \hat{\alpha}) \rangle \qquad 3.65$$

Using equations 3.57, 3.64 and 3.65,

$$z(\alpha) \rangle = [\psi(\hat{\alpha}) + S(\alpha - \hat{\alpha})]^{-1} [q(\hat{\alpha}) + u(\alpha - \hat{\alpha})] \rangle$$

$$= \psi^{-1}(\hat{\alpha}) [I + S(\alpha - \hat{\alpha}) \psi^{-1}(\hat{\alpha})]^{-1} [q(\hat{\alpha}) + u(\alpha - \hat{\alpha})] \rangle \qquad 3.66$$

The matrix inversion lemma states that if $A_2 = A_1 + FGH$ then $A_2^{-1} = A_1^{-1} - A_1^{-1} F (HA_1^{-1} F + G^{-1})^{-1} HA_1^{-1}$

Therefore, setting $A_2 = [I + S(\alpha - \hat{\alpha}) \psi^{-1}(\hat{\alpha})]$

$$A_1 = I$$

$$F = S(\alpha - \hat{\alpha})$$

$$G = I$$

and $$H = \psi^{-1}(\hat{\alpha})$$

we obtain

$$[I + S(\alpha - \hat{\alpha})\psi^{-1}(\hat{\alpha})]^{-1} = I - S(\alpha - \hat{\alpha})[\psi^{-1}(\hat{\alpha})S(\alpha - \hat{\alpha}) + I]^{-1}\psi^{-1}(\hat{\alpha})$$

Repeated application of the matrix inversion lemma allows $z(\alpha)$ to be expanded as a Taylor series, i.e.

$$z(\alpha) = \psi^{-1}(\hat{\alpha})[I - S(\alpha - \hat{\alpha})\psi^{-1}(\hat{\alpha}) + S(\alpha - \hat{\alpha})\psi^{-1}(\hat{\alpha})S(\alpha - \hat{\alpha})\psi^{-1}(\hat{\alpha}) -$$

$$.....] . [q(\hat{\alpha}) + u(\alpha - \hat{\alpha})] \qquad 3.67$$

Expanding and using equation 3.57,

$$z(\alpha) = \psi^{-1}(\hat{\alpha})[q(\hat{\alpha}) - S(\alpha - \hat{\alpha})z(\hat{\alpha}) + S(\alpha - \hat{\alpha})\psi^{-1}(\hat{\alpha})S(\alpha - \hat{\alpha})z(\hat{\alpha})$$

$$+ u(\alpha - \hat{\alpha}) - S(\alpha - \hat{\alpha})\psi^{-1}(\hat{\alpha})u(\alpha - \hat{\alpha})$$

$$+ S(\alpha - \hat{\alpha})\psi^{-1}(\hat{\alpha})S(\alpha - \hat{\alpha})\psi^{-1}(\hat{\alpha})u(\alpha - \hat{\alpha}) -$$

$$.....]$$

Premultiplying each side by $<e_s$,

$$g(\alpha) = <\pi(\hat{\alpha}), \{q(\hat{\alpha})> + [u(\alpha - \hat{\alpha}) - S(\alpha - \hat{\alpha})z(\hat{\alpha})]>$$

$$- S(\alpha - \hat{\alpha})\psi^{-1}(\hat{\alpha})[u(\alpha - \hat{\alpha}) - S(\alpha - \hat{\alpha})z(\hat{\alpha})]>$$

$$+ ..... \} \qquad 3.68$$

Note that, using equations 3.57, 3.64 and 3.65,

$$[u(\alpha - \hat{\alpha}) - S(\alpha - \hat{\alpha})z(\hat{\alpha})]$$

$$= [u(\alpha - \hat{\alpha}) + q(\hat{\alpha}) - S(\alpha - \hat{\alpha})z(\hat{\alpha}) - \psi(\hat{\alpha})z(\hat{\alpha})]$$

$$= [q(\alpha) - \psi(\alpha)z(\hat{\alpha})]$$

$\therefore$ equation 3.68 becomes,

$$g(\alpha) = g(\hat{\alpha}) + \langle \pi(\hat{\alpha}), [q(\alpha) - \psi(\alpha) z(\hat{\alpha})] \rangle$$

$$- \langle \pi(\hat{\alpha}) S(\alpha - \hat{\alpha}) \psi^{-1}(\hat{\alpha}) [q(\alpha) - \psi(\alpha) z(\hat{\alpha})] \rangle$$

$$+ \ldots\ldots \qquad 3.69$$

Comparing equation 3.69 and 3.52 we see that a second order approximation to g(.) passing through $g(\hat{\alpha})$ at $\hat{\alpha}$ is obtained by truncating the series defined by equation 3.69, i.e. $f(., \hat{\alpha})$ is defined by equation 3.63.

It can be verified by straightforward differentiation of equations 3.63 and 3.61 that $f_{\alpha}(\hat{\alpha}, \hat{\alpha}) = g_{\alpha}(\hat{\alpha})$ and $f_{\alpha\alpha}(\hat{\alpha}, \hat{\alpha}) = g_{\alpha\alpha}(\hat{\alpha})$ which is the (k-1) x (k-1) Hessian matrix, $G(\hat{\alpha})$.

Q.E.D.

3.5 Solving Markov decision problems using cost function approximations

3.5.1 A procedure solving a general non-linear programming problem having linear constraints

Consider the following function minimization problem with linear constraints:

$$\left.\begin{array}{l} \min \{f(x)\} \\ \text{Subject to } Ax = b \\ \qquad\qquad x \geq 0 \end{array}\right\} \qquad 3.70$$

Zangwill (42, Chapter 8) has proposed a straightforward non-linear programming algorithm which solves equations 3.70 using linear approximations to the function f. The algorithm generates a sequence

of points $\{x_k,\ k = 0, 1, \ldots.\}$ which are approximate solutions to equations 3.70. Given a feasible point x_k, let y_k be the solution of the linear programming problem which uses the same constraints but whose cost function is a linear approximation to f at x_k. It is reasonable to assume that the direction $d_k = y_k - x_k$ is a good direction in which to seek a reduced value of f.

Let $f_L(.)$ denote the linear approximation to f(.) at x_k. $f_L(.)$ is defined by

$$f_L(y) \triangleq f(x_k) + \langle \nabla_x f(x_k), [y - x_k] \rangle \qquad 3.71$$

The algorithm, given f_L and x_k fixed, obtains a solution y_k to the subproblem:

$$\left.\begin{array}{l} \min \{f(x_k) + \langle \nabla_x f(x_k), [y - x_k] \rangle\} \\ \text{Subject to } Ay = b \\ \qquad\qquad y \geq 0 \end{array}\right\} \qquad 3.72$$

but since several of the terms of the cost function are constant, the algorithm actually solves the simpler but equivalent linear programming subproblem:

$$\left.\begin{array}{l} \min \{\langle \nabla_x f(x_k), y \rangle\} \\ \text{Subject to } Ay = b \\ \qquad\qquad y \geq 0 \end{array}\right\} \qquad 3.73$$

where x_k is a fixed feasible point, and y is a variable. If the solution to equation 3.73 is denoted by y_k, then we search for a reduced cost in the direction $d_k = y_k - x_k$, i.e. the next point in the sequence $\{x_k\}$ is given by the line search procedure:

Choose τ_k to be the value of τ satisfying

$$\min_{\tau} \left\{ f(x_k + \tau[y_k - x_k]) \right\} , \quad \tau \in [0, 1] \qquad 3.74$$

then set $$x_{k+1} = x_k + \tau_k[y_k - x_k]$$

Line searching is carried out in practice by direct methods like the Fibonacci and golden section methods where the cost function is evaluated and compared at different points along the line, or by indirect methods requiring curve fitting at a point or a number of points followed by the determination of the minimum point on the fitted curve. A full discussion of these procedures is given by Luenberger (48), Zahradnik (49) and Polak (50). Fig. 3.1 illustrates the solution of a two-dimensional problem using the algorithm. A general feature of the algorithm is that the generated sequence $\{x_k\}$ tends to zigzag back and forth, slowing down convergence near x_∞, the point giving the constrained minimum cost.

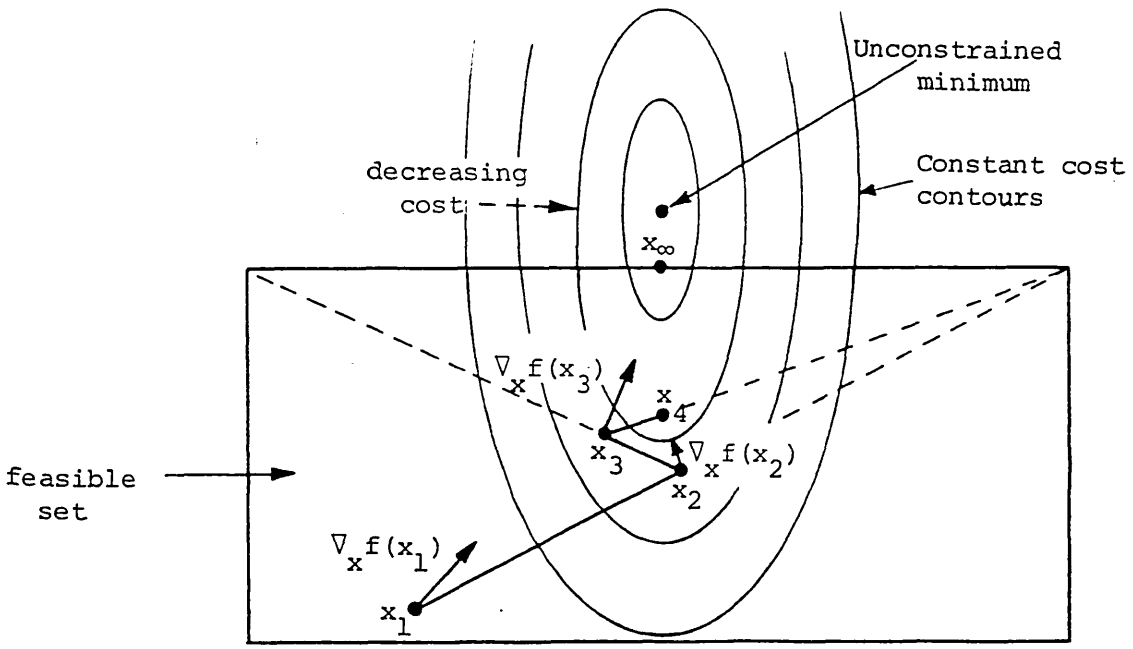


Fig. 3.1: A sequence $\{x_k\}$ generated by the linear approximation algorithm

We shall discuss a way of improving convergence near the solution point later. Zangwill (42) gives a proof of convergence for the linear approximation algorithm outlined above, and we shall give a formal statement of the algorithm after discussing a direct line-search subalgorithm due to Armijo, found in Polak (50, p36).

3.5.2 Armijo step-length subalgorithm

With reference to the line search subproblem (equation 3.74) and using the notation of the previous section, we define the functions:

$$\theta(\tau;\ x_k) \triangleq f(x_k + \tau d_k) - f(x_k) \qquad 3.75$$

$$\bar{\theta}(\tau;\ x_k) \triangleq [f(x_k + \tau d_k) - f(x_k)]$$

$$- \lambda\tau < \nabla_x f(x_k),\ d_k > \qquad 3.76$$

Having initially selected the parameters $\lambda \in (0,\ 1)$ and $\beta \in (0,1)$, (where normally $\lambda \simeq 0.5$) we state the step-length subalgorithm:

Step 0: Set $\mu = 1$

Step 1: Compute $\bar{\theta}(\mu;\ x_k)$

Step 2: If $\bar{\theta}(\mu;\ x_k) \leq 0$, set $\tau_k = \mu$ and stop,
else set $\mu = \beta\mu$ and go to Step 1.

Fig. 3.2 illustrates the solution method with $\beta = 0.5$ and $\lambda = 0.5$.

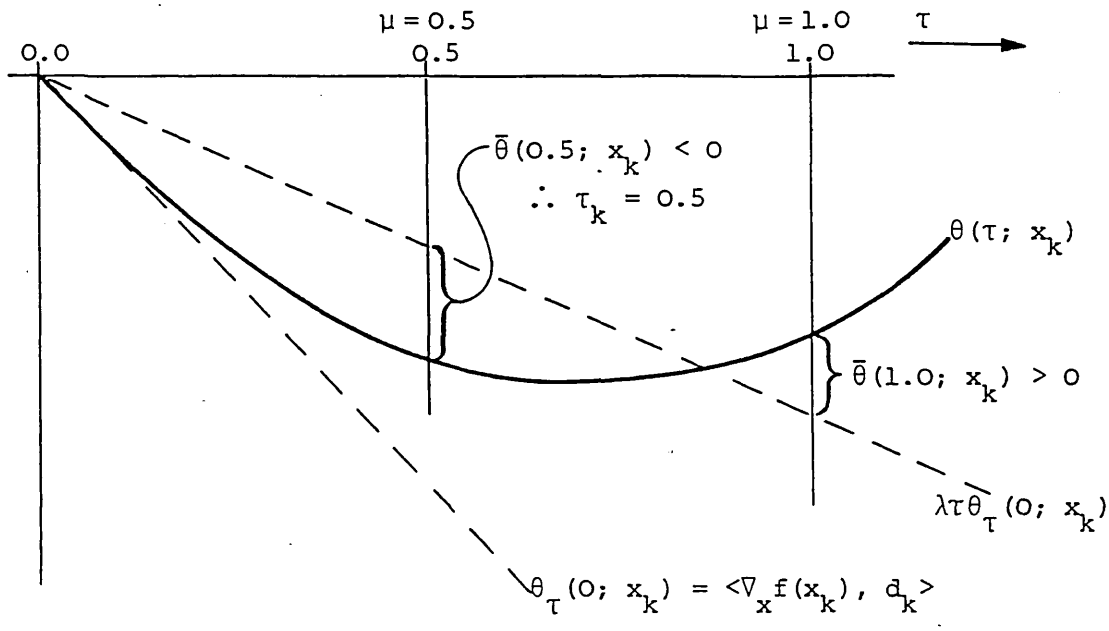


Fig. 3.2: Line search subproblem solved by Armijo subalgorithm with $\beta = 0.5$, $\lambda = 0.5$.

Although the subalgorithm is very simple to implement, it should be appreciated that the choice of values for the parameters λ and β directly affects the computing time required to solve the line search subproblem, or equivalently, the number of points at which f must be evaluated. With λ set to its recommended value of 0.5, it can be seen that the greater the value of β, the more accurate will be the line search.

If, as is generally the case, it is computationally expensive to calculate a search direction via solution of equation 3.73, we would prefer to make more effective use of this information by increasing the accuracy of the line search, and would then choose $\beta \in [0.5, 0.8]$, according to the time required to evaluate f at any point. As we shall see in the next section, evaluation of the function g for a Markov decision problem is very time consuming and in this case it is better to relax the parameter β, e.g. $\beta \in [0.1, 0.5]$, and make more efficient use of computing time by updating the search direction more frequently.

3.5.3 Solving Markov decision problems using the linear approximation algorithm

Following the discussion of the last two sections, we give a formal statement of the linear approximation algorithm applied to a Markov decision problem.

Step 0: Select a feasible parameter vector α_0 and $\beta \in [0.1, 0.5]$; set $\lambda = 0.5$, $k = 0$

Step 1: Calculate $\psi^{-1}(\alpha_k)$ and using equations 3.54, and 3.57, evaluate $\pi(\alpha_k)$, $z(\alpha_k)$. In practice, the solution of two sets of simultaneous equations:

$$<\pi(\alpha_k)\psi(\alpha_k) = <e_s$$
$$\psi(\alpha_k)z(\alpha_k)> = q(\alpha_k)>$$

may require less computation.

Step 2: Use a linear program to solve the problem:

$$\min_{\alpha} \{<\pi(\alpha_k), [q(\alpha) - \psi(\alpha)\, z(\alpha_k)]>\}$$

Subject to constraints

$$0 \le \alpha_i \le 1$$

$$\sum_i \alpha_i \le 1$$

where $\{\alpha_i\}$ are elements of the vector α, and set $\hat{\alpha}$ = minimizing value of α .

Stop Rule: If $[<\pi(\alpha_k), [q(\hat{\alpha}) - \psi(\hat{\alpha})\, z(\alpha_k)]>] \ge 0$

stop; else, go to Step 3.

Step 3: Redefining the functions θ and $\bar{\theta}$:

$$\theta(\tau;\ \alpha_k) \triangleq g(\alpha_k + \tau[\hat{\alpha} - \alpha_k]) - g(\alpha_k)$$

$$\bar{\theta}(\tau;\ \alpha_k) \triangleq \theta(\tau;\ \alpha_k) - \lambda\tau < \pi(\alpha_k),\ [q(\hat{\alpha}) - \psi(\hat{\alpha})z(\alpha_k)]>$$

where $g(\alpha) = <e_s,\ z(\alpha)>$

$$= <e_s \psi^{-1}(\alpha),\ q(\alpha)> \quad \text{for all feasible } \alpha\text{'s}$$

use Armijo subalgorithm of Section 3.5.2 to compute τ_k.

Step 4: Set $\alpha_{k+1} = \alpha_k + \tau_k[\hat{\alpha} - \alpha_k]$

Set $k = k + 1$, go to Step 1

To demonstrate the amount of computation required by this algorithm, we solve two small Markov decision problems; one having a single control parameter and the other having two.

(i) The toymaker example

A full description of this example has already been given in Section 2.2.5. In order to illustrate the use of the linear approximation algorithm for determining optimal grouped-state controls, the toymaker example is formulated as an 'open-loop' decision problem. Only one small change need be made to the classical problem description, with the original two-element parameter vector replaced by a scalar. Thus the problem is described by

$$P = \begin{bmatrix} .5 & .5 \\ .4 & .6 \\ .8 & .2 \\ .7 & .3 \end{bmatrix} \triangleq \begin{bmatrix} P^1 \\ \cdots\cdots \\ P^2 \end{bmatrix}, \quad B = \begin{bmatrix} -9 & -3 \\ -3 & +7 \\ -3 & -3 \\ -1 & +19 \end{bmatrix}$$

$$q = \begin{bmatrix} -6 \\ 3 \\ -3 \\ 5 \end{bmatrix} \triangleq \begin{bmatrix} q^1 \\ \cdots\cdots \\ q^2 \end{bmatrix}, \quad D(\alpha) = \begin{bmatrix} \alpha & 0 & \vdots & (1-\alpha) & 0 \\ 0 & \alpha & \vdots & 0 & (1-\alpha) \end{bmatrix}$$

for any $\alpha \in [0, 1]$

and using equations 2.24 and 2.26:

$$P(\alpha) = D(\alpha).P = \alpha P^1 + (1-\alpha)P^2$$

and

$$q(\alpha) = \alpha q^1 + (1-\alpha)q^2$$

The Markov decision problem in this case is:

$$\min_{\alpha \in [0, 1]} \{g(\alpha)\} \qquad 3.77$$

where $g(\alpha)$ is the mean process cost per cycle given by

$$g(\alpha) = \langle \pi(\alpha), q(\alpha) \rangle \qquad 3.78$$

and $\pi(\alpha)$ is the solution of

$$\left.\begin{aligned} \langle \pi(\alpha) &= \langle \pi(\alpha) . P(\alpha) \\ \langle \pi(\alpha), 1 \rangle &= 1 \end{aligned}\right\} \qquad 3.79$$

It is interesting to see whether the abstract problem above has any meaning when described in terms of the original toymaking business. Firstly it should be appreciated that the submatrices P^1 and P^2 give transition probabilities for the business conducted without and with advertising, respectively. Thus $P(\alpha)$ gives probabilities for business state transitions when advertising is used for $[(1-\alpha) \times 100]\%$ of the time. The decision to use advertising or not is taken without knowledge of the current business state (business having a popular or unpopular toy) which might occur if toys are sold through a wholesaler, and market response is subject to a long time lag. Business and advertising costs would be known, and the possible benefits due to advertising could be assessed by market research.

The optimizing value of α can be determined analytically, and turns out to be 1/6. Thus a strategy of advertising for 5 periods out of every 6, on average, would minimize costs (i.e. maximize profits).

Direct calculation of the function g(.) over the range of feasible α's results in the graph shown in Fig. 3.3, confirming the existence of only one minimum at $\alpha = 1/6$. Tables 3.6 and 3.7 show sequences $\{\alpha_k\}$ resulting from the use of the linear approximation algorithm to solve the toymaker example and the last point in sequence is the solution to an accuracy of $\pm$ 0.001. Initial points of sequences in Tables 3.6 and 3.7 are $\alpha_0 = 0.0$ and $\alpha_0 = 1.0$ respectively, each table showing a selection of parameter values, β, required by the Armijo subalgorithm. As specified in the formal statement of the algorithm earlier in this section, the parameter λ is held constant at 0.5 throughout. A feature of the Markov decision problem is that evaluation of the mean cost function g(.) at a given point requires the solution of s simultaneous equations (for an s-state process), which is a significant portion of the computation required for a new linear approximation to g(.) near

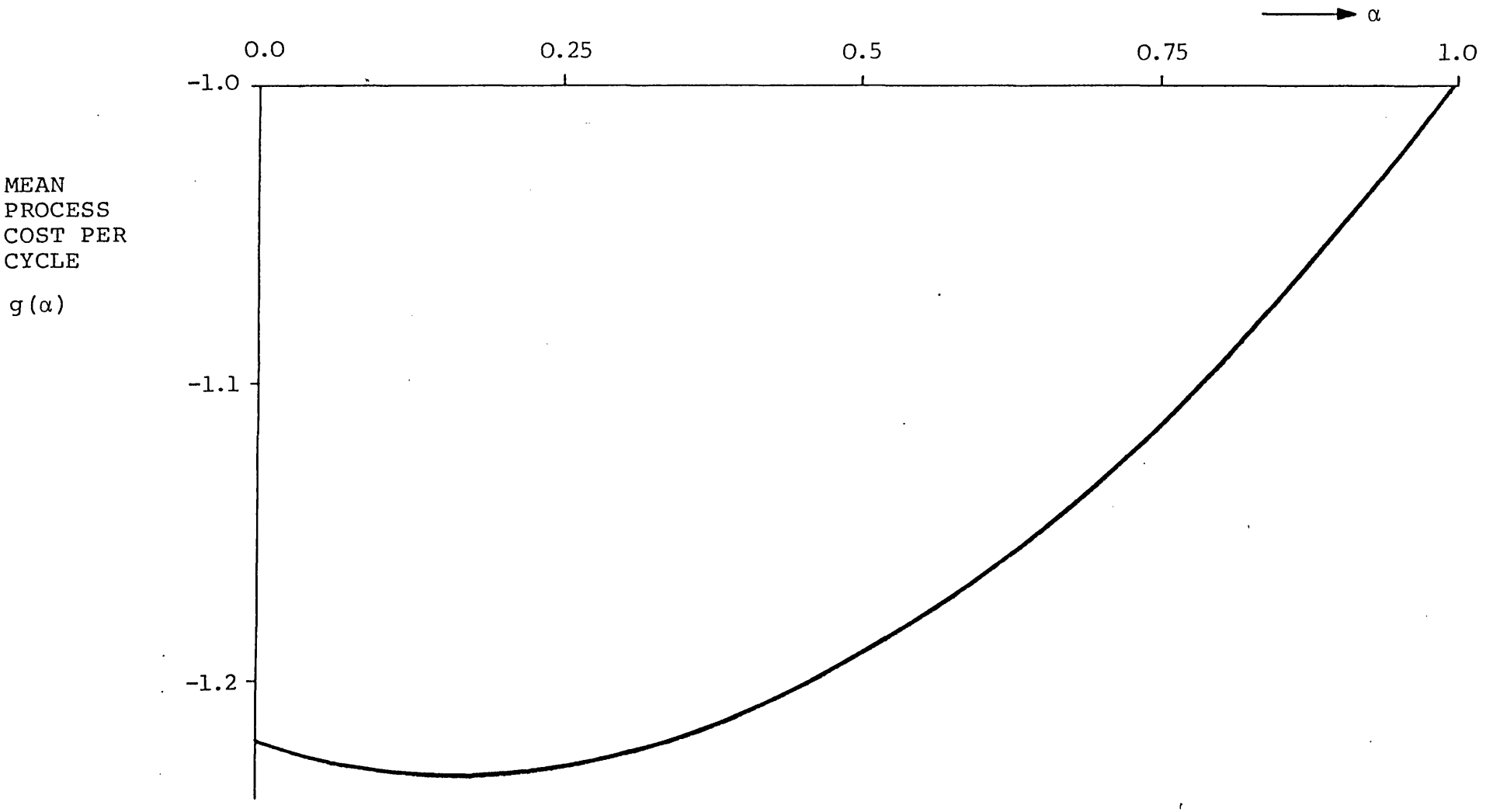


FIG. 3.3 : TOYMAKER EXAMPLE WITH OPEN-LOOP CONTROLS - GRAPH OF g(α) vs α

k	$\beta = 0.2$		$\beta = 0.3$		$\beta = 0.6$	
	α_k	n_k	α_k	n_k	α_k	n_k
0	0.0000	3	0.0000	3	0.0000	5
1	0.0400	3	0.0900	4	0.1296	8
2	0.0780	3	0.1146	4	0.1540	10
3	0.1153	3	0.1385	4	0.1625	12
4	0.1507	4	0.1617	6	0.1655	14
5	0.1574	4	0.1638	6	0.1666	-
6	0.1642	5	0.1658	-		
7	0.1655	6				
8	0.1658	-				
$\sum_k n_k$		31		27		49

Note: β and α_k are defined in the text. n_k denotes the number of function (g(.)) evaluations required by the Armijo subalgorithm to determine α_{k+1}.

Table 3.6: Use of the linear approximation algorithm to solve the toymaker example. $\alpha_0 = 0.0$

k	β = 0.2		β = 0.3		β = 0.6	
	α_k	n_k	α_k	n_k	α_k	n_k
0	1.0000	2	1.0000	2	1.0000	2
1	0.8000	2	0.7000	2	0.4000	3
2	0.6400	2	0.4900	2	0.2560	4
3	0.5120	2	0.3430	2	0.2007	5
4	0.4096	2	0.2401	2	0.1747	8
5	0.3277	2	0.1681	5	0.1698	9
6	0.2621	2	0.1667	-	0.1669	-
7	0.2097	2				
8	0.1678	5				
9	0.1675	-				
$\sum_k n_k$		21		15		31

Notation as for Table 3.6

Table 3.7: Use of the linear approximation algorithm to solve the toymaker example. $\alpha_0 = 1.0$.

that point. When the algorithm is searching for the optimal point in a multi-dimensional parameter space it is easier to appreciate that computing time is better spent obtaining directional information from more frequent revisions of the linear approximation than by carrying out accurate line searches, and we would therefore use a β of lower value. This single parameter search does not indicate such benefits but we nevertheless include a comparison of the number of function evaluations required using each value of β.

Comparing sequences for $\beta = 0.2$ and for $\beta = 0.6$ in both tables, we see that 50-60% more directional information is obtained using the lower β value, requiring 33% less function evaluations overall to obtain the solution. We conclude that the use of lower values for β offers significant benefits when using the algorithm to solve Markovian problems.

(ii) A problem involving two parameters

We consider the open-loop control of a two-state Markov decision process where now we are required to search a two-dimensional parameter space.

$P(.)$, $q(.)$, $g(.)$ etc. operate on a set of feasible parameters $\{\alpha = [\alpha_1 \quad \alpha_2]^T\}$ defined by

$$\left.\begin{aligned} & 0 \leq \alpha_i \leq 1, \quad i = 1, 2 \\ & \sum_{i=1}^{2} \alpha_i \leq 1 \end{aligned}\right\} \qquad 3.80$$

There is a difference between this problem and the previous one, in that controls are not costed, i.e. a constant cost is incurred for a transition between two given process states, regardless of the choice of control alternative. Thus process state transition costs are given by the 2 x 2 matrix B, i.e.

$$B^1 = \begin{bmatrix} 8 & 4 \\ 4 & 12 \end{bmatrix}$$

The effect of choosing a control alternative is simply to specify probabilities of transition to each process state, regardless of the present state. Hence we describe the effect of three alternative controls by three transition matrices.

$$P^1 = \begin{bmatrix} .75 & .25 \\ .75 & .25 \end{bmatrix}, \quad P^2 = \begin{bmatrix} .25 & .75 \\ .25 & .75 \end{bmatrix}, \quad P^3 = \begin{bmatrix} .1 & .9 \\ .1 & .9 \end{bmatrix}$$

Using Riordon's generalized Markov chain notation, the problem is described by

$$P = \left[\begin{array}{cc} .75 & .25 \\ .75 & .25 \\ \hdashline .25 & .75 \\ .25 & .75 \\ \hdashline .1 & .9 \\ .1 & .9 \end{array}\right], \qquad B = \left[\begin{array}{cc} 8 & 4 \\ 4 & 12 \\ \hdashline 8 & 4 \\ 4 & 12 \\ \hdashline 8 & 4 \\ 4 & 12 \end{array}\right]$$

$$q = \left[\begin{array}{c} 7 \\ 6 \\ \hdashline 5 \\ 10 \\ \hdashline 4.4 \\ 11.2 \end{array}\right], \qquad D(\alpha) = \left[\begin{array}{cc:cc:cc} \alpha_1 & 0 & \alpha_2 & 0 & 1-\alpha_1-\alpha_2 & 0 \\ 0 & \alpha_1 & 0 & \alpha_2 & 0 & 1-\alpha_1-\alpha_2 \end{array}\right]$$

for all feasible α's

Using equations 2.24 and 2.26

$$P(\alpha) = D(\alpha) \cdot P$$

$$= \alpha_1 \begin{bmatrix} .75 & .25 \\ .75 & .25 \end{bmatrix} + \alpha_2 \begin{bmatrix} .25 & .75 \\ .25 & .75 \end{bmatrix} + (1-\alpha_1-\alpha_2) \begin{bmatrix} .1 & .9 \\ .1 & .9 \end{bmatrix}$$

$$= \begin{bmatrix} (.65\alpha_1 + .15\alpha_2 + .1) & (-.65\alpha_1 - .15\alpha_2 + .9) \\ (.65\alpha_1 + .15\alpha_2 + .1) & (-.65\alpha_1 - .15\alpha_2 + .9) \end{bmatrix}$$

Similarly

$$q(\alpha) = \begin{bmatrix} (2.6\alpha_1 + .6\alpha_2 + 4.4) \\ (-5.2\alpha_1 - 1.2\alpha_2 + 11.2) \end{bmatrix}$$

and the Markov decision problem is

$$\min_{(\alpha\text{'s defined by equation 3.80})} \{g(\alpha)\} \qquad 3.81$$

with $g(\alpha)$ defined by equations 3.78 and 3.79 using vector valued α's.

Cost contours for this problem are straight lines of slope -0.23, and are shown with the solution in Fig. 3.4. It can be seen that vectors satisfying the decision problem 3.81 are those described by the line

$$\alpha_1 = 0.8717 - 0.23\alpha_2 \qquad 3.82$$

where it crosses the feasible region.

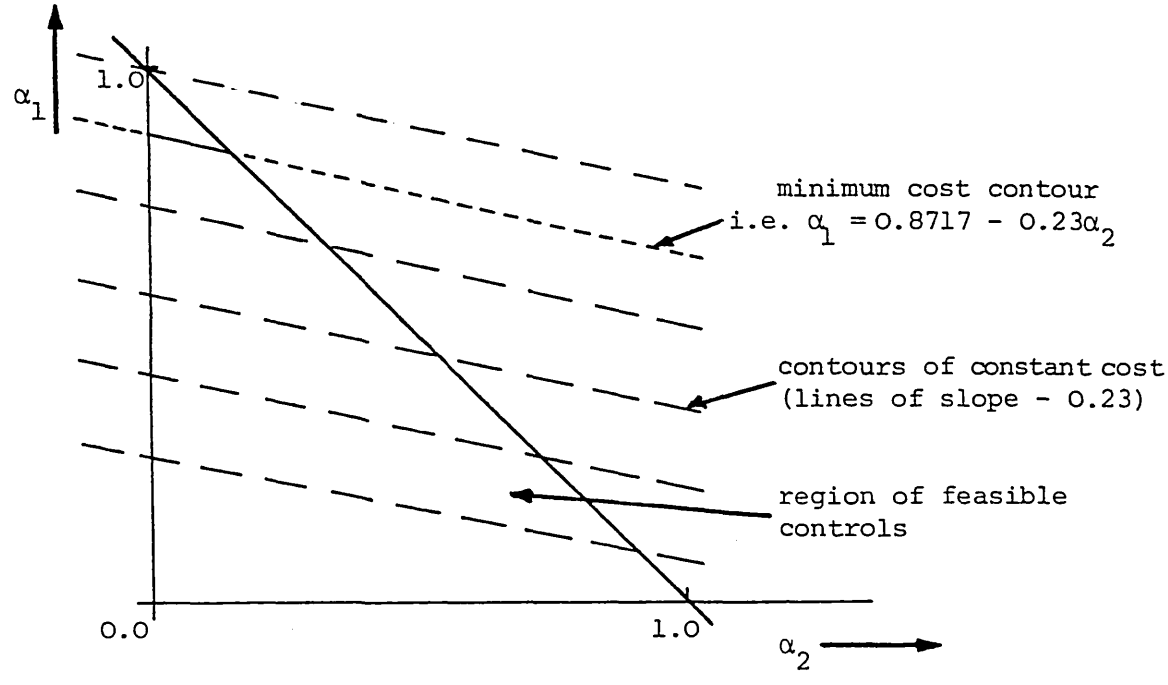


Fig 3.4: Cost contours and solutions for the two-parameter problem

Tables 3.8 and 3.9 show vector sequences $\{\alpha_k\}$ leading to a solution (accurate to ± 0.001) using different values for α_0 and β. Again we see that it is preferable to use a low value for β, lower values implying more frequent revisions of the search direction at little extra cost.

k	$\beta = 0.2$			$\beta = 0.6$		
	α_k^T		n_k	α_k^T		n_k
0	0.0000	0.0	2	0.0000	0.0	2
1	0.2000	0.0	2	0.6000	0.0	2
2	0.3600	0.0	2	0.8400	0.0	5
3	0.4880	0.0	2	0.8607	0.0	6
4	0.5904	0.0	2	0.8716	0.0	-
5	0.6723	0.0	2			
6	0.7378	0.0	2			
7	0.7903	0.0	2			
8	0.8322	0.0	2			
9	0.8658	0.0	3			
10	0.8712	0.0	-			
$\sum_k n_k$			21			15

Notation as for Table 3.5

Table 3.8: Solving a two-parameter problem using the linear approximation algorithm - $\alpha_0 = [0.0\ 0.0]^T$

k	β = 0.2			β = 0.6		
	α_k^T		n_k	α_k^T		n_k
0	0.0000	1.0000	2	0.0000	1.0000	2
1	0.2000	0.8000	2	0.6000	0.4000	3
2	0.3600	0.6400	2	0.7440	0.2560	4
3	0.4880	0.5120	2	0.7993	0.2007	5
4	0.5904	0.4096	2	0.8253	0.1747	8
5	0.6723	0.3277	2	0.8302	0.1698	9
6	0.7378	0.2621	2	0.8331	0.1669	-
7	0.7903	0.2097	2			
8	0.8322	0.1678	-			
$\sum_k n_k$			16			31

Notation as for Table 3.5

Table 3.9: Solving a two-parameter problem using the linear approximation algorithm - $\alpha_0 = [0.0\ 1.0]^T$

The two examples have established that the linear approximation algorithm is a valid solution method for Markov decision problems. However, its performance near the solution point leaves a lot to be desired. The generated sequence $\{\alpha_k\}$ not only tends to zig-zag (a feature of the routine determining search direction), but also requires evaluation of the cost function at an ever increasing number of points (see Tables 3.6 - 3.9) as the solution is approached (a feature of the Armijo step-length subroutine). It seems likely that both of these situations can be significantly improved by the use of second-order approximations to the cost function near any given point. For any Markov decision problem, Proposition 3.2 provides a quadratic function approximating the cost function locally and this function provides the

basis for some suggestions, given in Chapter 7, aimed at improving the convergence properties of the linear approximation algorithm.

3.6 Solving large Markov decision problems using the linear approximation algorithm

The linear approximation algorithm can be used in principle to solve Markov decision problems of any size but a factor limiting its practical use is the computer resource (computing time, or storage, or both) required to evaluate the functions $z(.)$ and $\pi(.)$ at any given point. Each function evaluation requires the solution of s simultaneous equations, where s is the number of states in the chain, and when s is very large (e.g. $s \geq 10{,}000$), solution by general purpose algorithms like Gaussian elimination or Gauss-Seidel becomes impractical.

A paper by Paige, Styan and Wachter (51) lists and compares eight algorithms for the computation of the stationary distribution π of a chain described by any transition matrix P. Each of the eight algorithms is designed to solve

$$\left.\begin{aligned} &<\pi P = <\pi \\ &<\pi, 1> = 1 \end{aligned}\right\} \qquad 3.83$$

for a general s x s matrix P without exploiting any sparsity or other structural feature that P might have. Computational comparisions are based on s-state chains where s takes the values 8, 20 and 40, and the conclusions may not therefore be relevant when much larger chains are considered. One of the features of their recommended algorithm is that it requires the solution of (s-1) equations in (s-1) unknowns (using Gaussian elimination), whereas all others need one more equation and one more unknown. This is unimportant when considering s of the order

10,000 or more. For unstructured problems in this thesis we use the same algorithm (algorithm 6 in the paper) to evaluate both π and z. Using Gaussian elimination, we solve the two sets of s simultaneous equations:

$$<\pi.\psi = <e_s \qquad 3.84$$

and

$$\psi.z > = q> \qquad 3.85$$

where q is the one-stage expected cost vector and

$$\psi \triangleq [(I - P)(I - e_s> <e_s) + 1> <e_s]$$

We see that if the problem structure allows straightforward computation of ψ^{-1} (see below) then π and z can be evaluated directly.

A paper by Gershwin and Schick (38) is concerned with exploiting the sparseness and nested block tri-diagonal structure of transition matrices describing transfer-line problems to calculate π and ψ^{-1}. An algorithm for computing ψ^{-1} breaks down the original problem into a sequence of subproblems involving the inversion of relatively small matrices. Unfortunately, the algorithm requires that these intermediate solutions are stored for later operations, leading to computer storage problems for very large Markov chains. From previous comments in Chapter 2, we note that the transition matrix loses its nested block tri-diagonal structure when cross-linking conveyors are used and the results of this paper cannot therefore be applied to transfer-line problems considered in this thesis.

We conclude that the linear approximation algorithm requires the solution of two sets of s simultaneous equations (equations 3.84 and 3.85) using a Gaussian elimination subalgorithm or (if the transition matrix is sparse) using an iterative method. Problems with very large numbers of states are solved more efficiently using one of the Monte Carlo methods described later in the thesis.

3.7. Summary

"Classical" Markov decision problems are frequently solved by one of three algorithms known as dynamic programming, linear programming, and policy iteration algorithms. Chapter 3 outlines the basic theory of each method, and shows that the principle of dynamic programming can be used to solve Markov decision problems having controls defined as functions of groups of states. The simplest problem of this type has an open-loop control structure and several small open-loop problems are used as illustrations throughout the chapter. Neither linear programming nor the policy iteration algorithm can be used to solve 'grouped-state' problems directly, but certain features of each can be used to formulate a non-linear programming problem which can be solved using a well-known algorithm.

Solving classical Markov problems using a linear program requires, as an essential first step, the transformation of the control space (a finite number of control alternatives) into a continuous space of probabilistic controls. The mean process cost per cycle is a scalar-valued function defined on a continuous parameter space and our task is to minimize this function by specifying a feasible parameter (or set of parameters). It is shown that the mean cost function can be approximated near any feasible point in parameter space using equations similar to those appearing in Howard's algorithm. The problem is now in

a form suitable for solution by a linear approximation algorithm, which resembles Howard's algorithm in that a sequence of points (feasible parameters) is generated, each point yielding a lower mean process cost than the preceding one. A full description of the algorithm is given, and it is used to solve two small problems.

A drawback of the linear approximation algorithm is that an excessive amount of computation (for an s-state chain, the solution of at least two sets of s simultaneous equations) is required at each iteration and this can prohibit its use if a large number of states is involved.

CHAPTER 4

MONTE CARLO ESTIMATION OF COST GRADIENTS

4.1 Introduction

In Chapter 3 we have shown that it is possible to devise novel analytic methods for solving grouped-state Markov decision problems. Unfortunately, the dynamic programming and linear approximation algorithms discussed in that chapter are only applicable to relatively small problems due to the limited computing resources (time and central memory) normally available. Solving a 200-state problem using the linear approximation algorithm would require the solution of several sets of 200 simultaneous equations at each iteration (probably using a Gaussian elimination subroutine). Sufficient central memory would thus be required to accommodate the 200 x 200 = 40,000 elements of the transition matrix, and extra storage would be needed for intermediate calculations, the control matrix, various 200-element vectors, etc. Of course, disc or magnetic tape storage could easily accommodate a model of this size, but then the access time for those facilities would greatly extend the time required to achieve an overall solution to the optimal control problem. In fact, 200 states is a practical upper limit to the size of Markov decision problem which can be solved by the C.D.C. computer installation at Imperial College using analytical methods, and bearing in mind the maximum time and storage available to users at the time of writing. In many instances, the transition matrix is sparse (though it may become less so when the chain is controlled - see Section 3.6) and an iterative method like the Gauss-Seidel algorithm would be a more efficient way of solving the simultaneous equations. However, although we would benefit from the reduction in storage requirements, we would still have to contend with inaccuracies inherent

in such iterative methods. Furthermore, the linear approximation algorithm would make frequent use of the subroutine solving simultaneous equations and the computing time required by an iterative method to achieve a reasonably accurate solution would prohibit its use. When the size of the chain is increased to (say) 10,000 states, which is by no means uncommon for many O.R. problems, methods requiring the storage of many 10,000-element vectors (like dynamic programming) also become impractical.

It is the purpose of this chapter to develop solution methods for very large Markov decision problems. Specifically, we discuss experimental methods for estimating partial derivatives of process cost with respect to each of a finite set of control parameters. An estimate of the cost gradient is therefore available for use in a stochastic approximation algorithm which solves the optimal control problem. A suitable algorithm is described in Chapter 5.

Working from the problem formulation (involving a set of continuous control parameters) given in Section 3.4.2, we develop an expression for the cost gradient in terms of the state distribution vectors and the co-state or adjoint vectors. Estimates of the state distribution vectors are obtained by a straightforward Monte Carlo method which can be applied to any abstract Markov chain description. The similarity between equations defining the state distribution and adjoint vector sequences implies that it should be possible to devise related experimental methods which yield estimates of the adjoint vectors. Two of these methods involve simulating the Markov process over N cycles in reverse-time, and the resulting adjoint vector estimates can be used with an estimated sequence of state distribution vectors to obtain unbiased cost gradient estimates. Later in the chapter, we describe more efficient Monte Carlo methods which generate sets of forward-time

simulations, simultaneously estimating state distribution and adjoint vector sequences to provide relatively accurate gradient estimates. An abstract, three-state Markov decision problem is used to demonstrate and compare all experimental methods described in this chapter.

4.2 Reformulation of the Markov decision problem

Some of the equations and definitions from earlier chapters are repeated here for the sake of completeness.

We consider N transitions of a finite-state Markov chain x_t, $t = 0, 1, \ldots, N$ with state space $\{e_1, e_2, \ldots, e_s\}$ whose transition probability at time t depends on the control u_t chosen at t. Thus Prob $[x_{t+1} | x_t]$ = Prob $[x_{t+1} | x_t, u_t]$. The state x_t is observed at each t, and u_t depends on which pre-determined state group x_t falls into. If x_t falls into group i, then a control $u_t \in U(i)$ may be used, where $U(i)$ is a pre-specified set. For r distinct state groups, we consider a stationary control policy to be any element of the product set $U(1) \times U(2) \times \ldots \times U(r)$.

In fact we can simplify the problem statement by using probabilistic controls with Riordon's notation (see Chapter 2). We assume a pre-determined finite number of control alternatives for each of r state groups which gives us the structure of the decision matrix $D(\alpha)$, with the stationary parameter vector α specifying the probability that a given control action will be taken. It follows that rather than discussing control constraint sets $U(i)$, $i = 1, \ldots, r$, we have parameter constraint sets $A(i)$, $i = 1, \ldots, r$, where a feasible α is a member of the product set $A(1) \times \ldots \times A(r)$. Thus each cycle consists of a 'control transition' from the previous process state to a control state determined by matrix $D(\alpha)$ followed by a 'process transition' from

the control state to the next process state determined by the constant matrix P. It has been established in Chapter 2 that for all problems so defined, the matrix product $[D(\alpha).P] = P(\alpha)$ is an s x s probability matrix, i.e. for all i,j

$$[P(\alpha)]_{ij} \geq 0, \quad \sum_{j} [P(\alpha)]_{ij} = 1$$

At the end of each cycle, the process incurs a cost $\langle x_t, c \rangle$ where c is a constant s-vector.

4.3 State probability distributions of the chain

The sequence of state probability distributions $\{\pi_t(\alpha), t = 0, 1, \ldots, N\}$ satisfies the following equation:-

$$\langle \pi_{t+1}(\alpha) = \langle \pi_t(\alpha) . P(\alpha), \quad t = 0, 1, \ldots, N - 1 \qquad 4.1$$

with given $\pi_o(\alpha)$.

The row-vector $\pi_t(\alpha)$ has s components defined by:

$$\langle \pi_t(\alpha), e_i \rangle \triangleq \text{Prob } [x_t = e_i] \qquad 4.2$$

for $t = 0, 1, \ldots, N$; $i = 1, 2, \ldots, s$.

4.4 Adjoint vectors of the chain

The expected cost of each transition is given by $\langle \pi_t(\alpha).P(\alpha), c \rangle$, $t = 0, 1, \ldots, N-1$, so we define the Hamiltonian function H by

$$H(t, \pi_t(\alpha), \alpha, \lambda_{t+1}) \triangleq \langle \pi_t(\alpha) P(\alpha), c \rangle + \langle \pi_t(\alpha) P(\alpha), \lambda_{t+1} \rangle \qquad 4.3$$

The adjoint equation is:-

$$\lambda_t> = \left[\frac{\partial}{\partial \pi_t} \cdot H(t, \pi_t(\alpha), \alpha, \lambda_{t+1}) \right]^T$$

$$= P(\alpha) \cdot [c> + \lambda_{t+1}>], \quad t = 0, \ldots, N-1 \qquad 4.4a$$

and

$$\lambda_N> = 0> \qquad 4.4b$$

and $\{\lambda_t\}$ is the sequence of adjoint (s-) vectors or co-states for the process. It is worth noting that the adjoint equation is a form of the well-known 'state value' equation (with 'one-stage expected cost vector' $q(\alpha)> = P(\alpha)c>$) used in many studies of Markov processes including Howard (7), Kushner (52), and Varaiya (12).

4.5 Statement of the optimal control problem and derivation of the cost gradient

Over N transitions of the chain, the total expected cost given a control policy α and an initial state x_o ($\equiv \pi_o$) is:

$$L(\alpha, \pi_0, 0) = \sum_{t=0}^{N-1} <\pi_t(\alpha) \cdot P(\alpha), c> \qquad 4.5$$

where $\pi_o(\alpha) \equiv \pi_o$

The optimal control problem is:-

$$\left.\begin{array}{l} \text{minimize } \{L(\alpha, \pi_0, 0)\} \\ \\ \text{Subject to } \alpha \in A(1) \times \ldots \times A(r) \end{array}\right\} \qquad 4.6$$

and the gradient of process cost with respect to α is given by

$$\frac{\partial}{\partial\alpha} \cdot L(\alpha, \pi_0(\alpha), 0) = \sum_{t=0}^{N-1} \frac{\partial}{\partial\alpha} \cdot H(t, \pi_t(\alpha), \alpha, \lambda_{t+1})$$

$$= \sum_{t=0}^{N-1} \langle \pi_t(\alpha) \cdot \left[\frac{\partial}{\partial\alpha} \cdot D(\alpha) \right] \cdot P, [c\rangle + \lambda_{t+1}\rangle] \qquad 4.7$$

The following is an important feature of the above formulation. If α_i is the i'th component of α, then each row of the matrix $\frac{\partial}{\partial\alpha_i} \cdot D(\alpha)$ has all zero components, or has all zero components but two, which are -1 and +1. Taking a simple example from Chapter 3, if the decision matrix defines two alternative control actions, i.e. has a two dimensional parameter vector, then

$$P(\alpha) = \alpha_1 \begin{bmatrix} .75 & .25 \\ .75 & .25 \end{bmatrix} + \alpha_2 \begin{bmatrix} .25 & .75 \\ .25 & .75 \end{bmatrix} + (1-\alpha_1-\alpha_2) \begin{bmatrix} .1 & .9 \\ .1 & .9 \end{bmatrix}$$

$$= D(\alpha) \cdot P$$

where

$$D(\alpha) = \begin{bmatrix} \alpha_1 & 0 & \alpha_2 & 0 & 1-\alpha_1-\alpha_2 & 0 \\ \vdots & & \vdots & & \vdots & \\ 0 & \alpha_1 & 0 & \alpha_2 & 0 & 1-\alpha_1-\alpha_2 \end{bmatrix},$$

for all feasible α's

$$\text{and } P = \begin{bmatrix} .75 & .25 \\ .75 & .25 \\ \hdashline .25 & .75 \\ .25 & .75 \\ \hdashline .1 & .9 \\ .1 & .9 \end{bmatrix}$$

$$\text{Hence } \frac{\partial}{\partial \alpha_1} \cdot D(\alpha) = \left[\begin{array}{cc:cc:cc} 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \end{array}\right]$$

$$\text{and } \frac{\partial}{\partial \alpha_2} \cdot D(\alpha) = \left[\begin{array}{cc:cc:cc} 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 \end{array}\right]$$

This feature makes the estimation of gradients using simulations easier to understand and to program, as will be seen later.

4.6 Statistical estimation of state distributions and adjoint vectors

The following sections show how statistical estimates of the s-vector sequences $\{\pi_t,\ t = 0,\ldots,N\}$ and $\{\lambda_t,\ t = 0,\ldots,N\}$ can be obtained. In order to simplify the discussion, N transitions of an uncontrolled process are considered, so that with an s x s transition matrix P, equations 4.1 and 4.4 become:-

$$<\pi_{t+1} = <\pi_t . P, \quad t = 0, \ldots, N - 1 \qquad 4.8$$

with π_o given, and

$$\lambda_t> = P . [c> + \lambda_{t+1}>], \quad t = 0, \ldots, N - 1 \qquad 4.9a$$

with $\lambda_N> = 0>$ 4.9b

For the remainder of the chapter, numbers which are drawn independently from uniform distributions over the interval [0,1] are referred to simply as 'random numbers'.

It is as well to point out at this stage why we find it necessary to estimate the required vectors. As the properties of the Markov model ensure that there is a linear relationship between successive members of the state distribution sequence, it would seem obvious that the N state distribution vectors could be obtained by straightforward matrix multiplication. Thus for a dense transition matrix P, s^2 calculations (products and sums) would be required to calculate π_1, using π_0 and equation 4.8. The remaining members of the sequence, i.e. $\{\pi_2, \ldots, \pi_N\}$would be calculated in the same way, and a total of Ns^2 products and sums would be required. A similar argument implies that the adjoint vector sequence could be calculated using equation 4.9a, starting with $\lambda_N> = 0>$ and ending the sequence at λ_0.

This method is uncomplicated and acceptable for chains having a small number of states, but soon becomes impractical as the value of N is increased. As an example, consider a 100-state chain with N = 100: storage requirements would be (100 x 100 =) 10,000 words for the transition matrix, plus (100 x 100 =) 10,000 words for each of the sequences $\{\pi_t\}$ and $\{\lambda_t\}$. About 10^6 products and sums would be required for the

sequence $\{\pi_t\}$ and slightly more for the sequence $\{\lambda_t\}$. When thousands of states are involved, and N assumes large values, it becomes necessary to calculate (or estimate) each vector sequence using methods which require less computation and less computer memory.

Later we show that the state distributions can be estimated by a straightforward Monte Carlo technique which uses a random number, w, to transform any probability vector into a 'degenerate vector' of the form e_i, $i \in \{1, \ldots, s\}$, which is an unbiased estimate of the original. As this technique can be applied to <u>any</u> probability vector, it follows that the matrix P (whose rows are probability vectors) can be transformed into a 'degenerate matrix' F(w), (whose rows are degenerate vectors), using a single random number, w. By generating a sequence of N random numbers, it is possible to obtain estimates of each member of the sequence $\{\pi_t\}$. It can be seen that the vector and matrix estimates have few non-zero elements, and the estimated sequence can therefore be computed using much less storage and time. It follows that the method would be applicable to much larger chains having, e.g. 10,000 states.

We would like to use the same, or similar techniques to estimate the adjoint vector sequence, and one possibility is to calculate a sequence of vector estimates by substituting $F(w_t)$ for P in equation 4.9a, where $\{w_t, t = 0, \ldots, N-1\}$ is a sequence of random numbers. However, it turns out that the vectors thus obtained are not degenerate vectors (giving rise to computational difficulties once again), and it is necessary to use additional random numbers to make further transformations, finally obtaining a degenerate vector sequence. This chapter describes several methods for estimating the adjoint vector sequence, leading to the definition of several cost gradient estimators. The most efficient of these is identified and used extensively throughout the thesis.

4.7 Estimation of state distributions

4.7.1 Obtaining a sequence of degenerate row matrices

By generating a sequence of independent random numbers $\{ w_t, t = 0, 1, \ldots, N-1\}$ denoted by ω, we can produce N unbiased estimates $\{F(w_t), t = 0, 1, \ldots, N-1\}$ of the matrix P. Specifically, considering a certain row i and defining an integer random variable $n(w_t)$ by

$$n(w_t) \triangleq k, \quad \text{when} \quad \sum_{j=1}^{k-1} p_{ij} < w_t \leq \sum_{j=1}^{k} p_{ij} \quad * \qquad k \in \{1, \ldots, s\} \qquad 4.10$$

then row i of $F(w_t)$ is given by $e_{n(w_t)}$, and each row of $F(w_t)$ for $t = 0,1,\ldots, N-1$ can be determined in this way. It follows from equation 4.10 that the i'th row of $F(w_t)$ is $e_{n(w_t)}$ with probability $p_{in(w_t)}$ and that $F(w_t)$ is a degenerate (row) probability matrix, i.e. having only elements 0 and 1, with expectation

$$E_{w_t} \{F(w_t)\} = P, \quad t = 0, 1, \ldots, N - 1 \qquad 4.11$$

Also, it can be seen that any probability matrix (e.g. the 2 x 6 matrix $D(\alpha)$ and the 6 x 2 matrix P in the example above) can be reduced to such a degenerate form and in an N-cycle simulation of a controlled process, at least 2N random numbers would be required.

4.7.2 Estimating the sequence $\{\pi_t\}$

Proposition 4.1

$$E_{\omega} \{x_t(\omega)\} = \pi_t \qquad 4.12$$

---

* Note that Prob $[n(w_t) = k] = p_{ik}$, as w_t is a random number.

where $\{x_t(\omega)\}$ is the sequence generated by

$$<x_{t+1} = <x_t \cdot F(w_t), \quad t = 0, 1, \ldots, N-1 \qquad 4.13$$

with given initial state $x_o = e_i$, $i \in \{1, \ldots, s\}$

Proof

Obviously, for given initial state x_o, $\pi_o = \underset{\omega}{E}\{x_o\} = x_o$

Expanding equation 4.8,

$$<\pi_{t+1} = <\pi_0 \prod_{i=0}^{t} P, \quad t = 0, 1, \ldots, N-1 \qquad 4.14$$

and similarly from equation 4.13

$$<x_{t+1} = <x_t \prod_{i=0}^{t} F(w_i), \quad t = 0, 1, \ldots, N-1 \qquad 4.15$$

$\therefore$ as the sequence ω consists of independent random numbers,

$$\underset{\omega}{E}\{<x_{t+1}(\omega)\} = \underset{\omega}{E}\{<x_0 \prod_{i=0}^{t} F(w_i)\}$$

$$= <x_0 \prod_{i=0}^{t} \underset{w_i}{E}\{F(w_i)\}$$

$$= <\pi_0 \prod_{i=0}^{t} P, \quad \text{using equation 4.11}$$

$$= <\pi_{t+1}, \quad \text{from equation 4.14}$$

The sequence $\{x_t(\omega)\}$ is therefore an unbiased estimate of the sequence of state distributions

4.8 Estimation of the adjoint sequence $\{\lambda_t\}$

We seek methods, analogous to those outlined in Section 4.7, for generating a sequence of s-vectors $\{y_t,\ t = 0, 1, \ldots, N\}$, which is an unbiased estimate of the adjoint sequence $\{\lambda_t,\ t = 0, \ldots, N\}$. Whereas the method described in Section 4.7.2 produces estimates of the state distributions in the order $x_0, x_1, \ldots, x_N$, the similarity between equations 4.8 and 4.9a, and the definition of boundary conditions at opposite ends of the time scale, imply that a similar method might be devised for estimating the adjoint vectors, $\lambda_N, \lambda_{N-1}, \ldots, \lambda_o$, in that order. As the time index for one sequence increases, and that for the other decreases, we use the terms 'forward-time' and 'reverse-time' to identify estimation methods in the thesis.

Weighting methods similar to those mentioned by Hammersley and Handscomb (31) form the basis of the adjoint estimation methods to be described. Ideally, we would like to be able to generate one sequence of random numbers, $\{w_t,\ t = 0, \ldots, N-1\}$, which could be used to estimate the sequence $\{\pi_t\}$ from a forward-time simulation and the sequence $\{\lambda_t\}$ from a reverse-time simulation. As $\{\pi_t\}$ can be estimated by solving equation 4.13 (obtained by substituting the degenerate matrix $F(w_t)$ for P in equation 4.8), we surmise that $\{\lambda_t\}$ might be estimated by solving an equation similar to equation 4.9a but with P replaced by a degenerate matrix of some description. Because $\{\pi_t\}$ and $\{\lambda_t\}$ are sequences of row and column vectors respectively, it comes as no surprise that degenerate <u>column</u> matrices (having columns of the form $[z].e_i,\ i \in \{1, \ldots, s\}$, where z is a weighting factor) are required here, rather than degenerate row matrices like $F(w_t)$. Section 4.8.2

describes how these matrices are obtained, and Sections 4.8.3 and 4.8.6 give details of two reverse-time methods for estimating the adjoint vector sequence.

Straightforward substitution of the estimated vector sequences into equation 4.7 provides us with two forms of cost gradient estimator. Unfortunately, the resulting gradient estimates tend to be inaccurate (i.e. have a high variance), due to the fact that reverse-time methods yield estimates of each complete adjoint vector in the sequence, whereas in fact we require estimates of relatively few components from some (but not all) adjoint vectors. We begin a discussion of forward-time estimation methods in Section 4.8.7 by describing how individual components of any adjoint vector can be estimated using the random number sequence $\{ w_t, t = 0, \ldots, N-1 \}$. Later we discuss methods which are more closely related to the gradient estimation problem. Thus prepared, we are able to move on to Section 4.9, which deals with efficient gradient estimation methods.

4.8.1 Obtaining degenerate estimates of a vector $c \in R^S$

By degenerate estimate, we mean an s-vector y_c of the form $[z_c].e_i$ where z_c is a scalar.

If $\{c_i\}$ are the elements of c, then in general $\sum_i c_i \neq 1$, i.e. c is not normally a probability vector, so we define a scalar

$$z_c \triangleq \sum_i c_i \qquad 4.16$$

and a probability vector

$$q_c > \triangleq \left[\frac{1}{z_c}\right] . c> \qquad 4.17$$

Thus the i'th component of q_c is c_i/z_c

Following the reasoning of Section 4.7.1, an integer random variable $n(v_c)$ is defined by:-

$$n(v_c) \triangleq k, \quad \text{when} \sum_{i=1}^{k-1} \frac{c_i}{z_c} < v_c \leq \sum_{i=1}^{k} \frac{c_i}{z_c} \qquad 4.18$$

$$k \in \{1, \ldots, s\}$$

where v_c is a random number $\in [0,1]$. Thus letting $y_c(v_c) = [z_c].e_{n(v_c)}$, we have

Proposition 4.2

$$E_{v_c} \{y_c(v_c)\} = c>$$

Proof

From equation 4.18, for each $k \in \{1, \ldots, s\}$,

$$\text{Prob}\,[y_c(v_c) = [z_c] \cdot e_k] = \frac{c_k}{z_c}$$

$$\therefore E_{v_c} \{y_c(v_c)>\} = \sum_{k=1}^{s} \frac{c_k}{z_c} \cdot [z_c] \cdot e_k> = c> \qquad 4.19$$

Corollary

It follows from Proposition 4.2 that the matrix P (the transpose of which is not, in general, a probability matrix) can be expressed in the form

$$P = Q^T \cdot Z \qquad 4.20$$

where Q is an s x s probability matrix with elements $\{q_{ij}\}$, and Z is a diagonal matrix of transition weights, i.e. Z assumes the form

$$Z = \begin{bmatrix} z_1 & & \bigcirc \\ & \ddots & \\ \bigcirc & & z_s \end{bmatrix}$$

Thus if

$$z_j \triangleq \sum_{i=1}^{s} p_{ij} \qquad 4.21$$

then

$$q_{ji} = p_{ij}/z_j, \quad i, j \in \{1, \ldots, s\} \qquad 4.22$$

4.8.2 Obtaining a sequence of degenerate column matrices

Following the argument of Section 4.7.1, we can generate a sequence of random numbers $\{v_t, t = 1, \ldots, N\}$, and produce N unbiased estimates $\{G(v_t), t = 1, \ldots, N\}$ of the matrix $P = Q^T Z$. The j'th column of $G(v_t)$ is obtained by the following method:

Defining the integer random variable $n(v_t)$ by

$$n(v_t) \triangleq k, \quad \text{when } \sum_{i=1}^{k-1} q_{ji} < v_t \leq \sum_{i=1}^{k} q_{ji} \qquad 4.23$$

$$\text{for } k \in \{1, \ldots, s\}$$

$$\text{and } t \in \{1, \ldots, N\}$$

then the j'th column of $G(v_t)$ is given by $[z_j].e_{n(v_t)}$, $j = 1, \ldots, s$.

Again, we see that

$$\underset{v_t}{E}\{G(v_t)\} = Q^T . Z$$
$$= P, \quad t = 1, \ldots, N \qquad 4.24$$

and $G(v_t)$ has the property that each of its columns has only one non-zero element.

4.8.3 Reverse-time estimation of the adjoint vector sequence - Method 1

From equation 4.9a we see that

$$\left.\begin{aligned} \lambda_{N-1}> &= Pc> \\ \lambda_{N-2}> &= Pc> + P^2c> \\ &\;\;\vdots \\ \lambda_{N-n}> &= Pc> + P^2c> + \ldots + P^nc> \end{aligned}\right\} \qquad 4.25$$

Members of the sequence $\{Pc>, P^2c>, \ldots., P^nc>\}$ can be estimated using the techniques given in Sections 4.8.1 and 4.8.2. Let γ denote a sample of (N+1) random numbers $\{v_c; v_t, t = 1, \ldots., N\}$.

Proposition 4.3

$$\underset{\gamma}{E}\{\sum_{t=1}^{n} y_t(\gamma)\} = \lambda_{N-n} \qquad 4.26$$

where $y_t(\gamma)$ is the s-vector sequence generated by

$$y_{t+1}> = G(v_{t+1}) \cdot y_t>, \quad t = 1, \ldots, N-1 \qquad 4.27a$$

with $$y_1> = G(v_1) \cdot y_c(v_c)> \qquad 4.27b$$

Proof

From equations 4.27a and 4.27b,

$$y_t(\gamma)> = \prod_{i=1}^{t} G(v_i) \cdot y_c(v_c)> \qquad 4.28$$

and as the members of γ are independent, then

$$E_{\gamma} \{y_t(\gamma)>\} = \prod_{i=1}^{t} E_{v_i} \{G(v_i)\} E_{v_c} \{y_c(v_c)>\}$$

$$= P^t \cdot c> \text{ from equations 4.19 and 4.24}$$

Hence

$$E_{\gamma} \{ \sum_{t=1}^{n} y_t(\gamma)\} = \lambda_{N-n} \qquad 4.29$$

from equations 4.25.

Thus by calculating partial summations from one sequence of degenerate vectors $\{y_c;\ y_t,\ t = 1, \ldots, N\}$, it is possible to obtain unbiased estimates of the adjoint vectors $\{\lambda_t,\ t = 0, \ldots, N-1\}$.

4.8.4 A simple illustration of forward- and reverse-time estimation methods

From a practical point of view it should be noted that there are few differences between the procedures for estimating the sequences $\{\pi_t\}$ and $\{\lambda_t\}$. We have already established that s, the dimension of the state space, can be a very large number, e.g. 10,000. Both of the estimation techniques require only the storage of a number (or unique set of conditions) identifying the non-zero component of each distribution or adjoint vector. In the case of the $\{y_t\}$ sequence, we not only have to identify the non-zero element, but also have to store a value corresponding to the trajectory 'score', which is revised with every transition simulated. In neither case is it necessary to calculate and store the s^2 elements of each $F(w_t)$ and $G(v_t)$ matrix; the appropriate row or column being identified as required and calculated.

To give an idea of the simplicity of the techniques, consider the following 2-state, uncontrolled Markov chain described by:

$$P = \begin{bmatrix} .5 & .5 \\ .6 & .4 \end{bmatrix}, \quad c = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad x_0 = [1 \quad 0]$$

Assuming that we are to use one forward-time simulation and one reverse-time simulation to obtain estimates of the vector sequences $\{\pi_0, \pi_1, \pi_2, \pi_3\}$ and $\{\lambda_0, \lambda_1, \lambda_2, \lambda_3\}$ i.e. 3 transitions from each boundary state, Sections 4.7.2 and 4.8.3 tell us that no less than 3 random numbers are required for each forward-time run and 4 for each reverse-time run.

Estimating $\{\pi_t\}$

Taking the sequence ω from a table of random numbers,

$$w_0 = .9969, \quad w_1 = .7920, \quad w_2 = .0881$$

we can calculate $\{F(w_t)\}$:

$$F(w_0) = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}, \; F(w_1) = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}, \; F(w_2) = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

and use equation 4.13 to calculate the sequence $\{x_t(\omega)\}$

i.e. $x_0 = [1 \;\; 0]$, $x_1 = [0 \;\; 1]$, $x_2 = [0 \;\; 1]$, $x_3 = [1 \;\; 0]$

and these vectors are unbiased estimates of $\{\pi_t, \; t = 0, 1, 2, 3\}$.

Note, however, that because only one component (the first) of x_0 is non-zero, only the first row of $F(w_0)$ is required for calculation of x_1. Similarly only one row of each degenerate matrix is required for each transition in the simulation. Thus we identify and determine the required row of each degenerate matrix at the end of every transition.

Estimating $\{\lambda_t\}$

Again, using a random number table to determine γ,

$$v_c = .8268, \quad v_1 = .0786, \quad v_2 = .6976, \quad v_3 = .7401$$

Using equations 4.16 and 4.17, c is written as $c> = [z_c].q_c>$, where $q_c = [1/3 \quad 2/3]^T$, and $z_c = 3$. Also, from equation 4.18,

$$y_c(v_c) = 3 . \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Using equations 4.21 and 4.22, P can be replaced by the matrix product Q^TZ, where

$$Q = \begin{bmatrix} .4545 & .5455 \\ .5556 & .4444 \end{bmatrix}, \quad Z = \begin{bmatrix} 1.1 & 0 \\ 0 & 0.9 \end{bmatrix}$$

Application of equation 4.23 gives the sequence $\{G(v_t)\}$, i.e.

$$G(v_1) = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} . \begin{bmatrix} 1.1 & 0 \\ 0 & 0.9 \end{bmatrix},$$

$$G(v_2) = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} . \begin{bmatrix} 1.1 & 0 \\ 0 & 0.9 \end{bmatrix}$$

$$G(v_3) = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} . \begin{bmatrix} 1.1 & 0 \\ 0 & 0.9 \end{bmatrix}$$

and calculating the sequence $\{y_t(\gamma)\}$ using equation 4.27a,

$$y_1> = (0.9 \times 3) \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$y_2> = (1.1 \times 0.9 \times 3) \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$y_3> = (0.9 \times 1.1 \times 0.9 \times 3) \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

i.e.

$$y_1> = \begin{bmatrix} 2.7 \\ 0 \end{bmatrix}, \quad y_2> = \begin{bmatrix} 0 \\ 2.97 \end{bmatrix}, \quad y_3> = \begin{bmatrix} 0 \\ 2.673 \end{bmatrix}$$

We know that $\lambda_3> = 0>$, and from Proposition 4.3,

$$(\text{estimate of } \lambda_2) = y_1 = \begin{bmatrix} 2.7 \\ 0 \end{bmatrix}$$

$$(\text{estimate of } \lambda_1) = y_1 + y_2 = \begin{bmatrix} 2.7 \\ 2.97 \end{bmatrix}$$

$$(\text{estimate of } \lambda_0) = y_1 + y_2 + y_3 = \begin{bmatrix} 2.7 \\ 5.643 \end{bmatrix}$$

From the example, it can be seen that each y_t requires identification of the non-zero element and storage of a 'trajectory score' or 'weighting factor', which is modified with each transition. Again it should be apparent that only one column and weighting factor of $G(v_t)$ need be calculated at each transition.

4.8.5 Some comments on gradient estimation

So far we have defined methods for estimating $\{\pi_t\}$ from a forward-time simulation and $\{\lambda_t\}$ from a reverse-time simulation. By using different sequences of independent random numbers in each simulation, it is apparent that the sequence estimates are independent of each other. It follows that these estimates can be substituted for $\{\pi_t\}$ and $\{\lambda_t\}$ in equation 4.7 to provide an unbiased estimate of the gradient of total process cost over N cycles with respect to the control parameters. Multiplying this quantity by the factor (1/N) yields the gradient of mean process cost per cycle, which is used in later chapters in preference to the gradient of total cost. It is worth noting that the two sequences of degenerate matrices could be obtained by using the same sequence of random numbers, $\{w_t,\ t = 0, \ldots, N-1\}$, with each of the functions F(.) and G(.) defined earlier. To do this, we retain an independent random number, v_c, and then put $v_1 = w_{N-1}$, $v_2 = w_{N-2}$, etc. Because the estimate $x_t(\omega)$ is a function of $\{w_i,\ i = 0, \ldots, t-1\}$ and the estimate $\sum_{i=1}^{N-t-1} y_i(\omega)$ is a function of $\{w_i,\ i = t, \ldots, N-1\}$, it can be seen that the estimates are independent of each other. It follows therefore, that each of the N components

$$<x_t(\omega) \cdot \frac{\partial}{\partial \alpha} \cdot D(\alpha) \cdot P \cdot \sum_{i=1}^{N-t-1} y_i(\omega)>, \quad t = 0, \ldots, N-1$$

is an unbiased estimate of

$$<\pi_t \cdot \frac{\partial}{\partial \alpha} \cdot D(\alpha) \cdot P \cdot \lambda_{t+1}>$$

Because the sum of unbiased estimates is an unbiased estimate of the sum, we ultimately obtain an unbiased cost gradient estimate. For methods which will be described later, the use of common random number sequences is shown to decrease the variance of gradient estimates but it seems unlikely that such benefits would be realised for the gradient estimator considered in this section, due to the partial summations required for estimates of $\{\lambda_t\}$. We cannot guarantee a reduction in variance, but we certainly reduce computing time by using common random number sequences, so the examples which appear later employ the technique for that reason.

4.8.6 Reverse-time estimation of the adjoint vector sequence - Method 2

Consider the following, conceptual approach involving analytical manipulation of the degenerate row matrices $\{F(w_t)\}$ in reverse-time.

Proposition 4.4

$$E_{\omega} \{z_t(\omega)\} = \lambda_t, \quad t = 0, \ldots, N$$

where the sequence $\{z_t(\omega)\}$ is given by

$$z_t> = F(w_t) \cdot [c> + z_{t+1}>], \quad t = 0, \ldots, N-1 \qquad 4.30a$$

with $z_N> = 0>$ 4.30b

Proof

Follows from equations 4.4 and 4.11

Of course, it would be an extremely laborious exercise to calculate each of the s rows of $F(w_t)$ for $t = 0, 1, \ldots, N-1$. We therefore use the methods described in Sections 4.8.1 and 4.8.2 to obtain unbiased estimates of $\{z_t(\omega)\}$ which we denote by $\{\bar{z}_t(\omega, \tau)\}$. τ represents a sample of independent random numbers needed both to reduce vectors $\{c, (c + \bar{z}_{N-1}(\omega, \tau)), \ldots, (c + \bar{z}_1(\omega, \tau))\}$ to degenerate estimates, and to express $F(w_t)$ in a form similar to equation 4.20. Though $F(w_t)$ has degenerate rows, its columns might consist of all zero elements or might include several (up to s) ones.

In practice, it is not necessary to store the N matrices $\{F(w_t), t = 0, \ldots, N-1\}$. By storing the sequence ω, any columns of F(.) can be directly calculated as required, using the same rules as those applied during forward-time estimation of $\{\pi_t, t = 0, \ldots, N\}$, e.g. each of the s elements of column j of $F(w_t)$ is evaluated using the rule:

$$\text{element } i = 1, \quad \text{if } \sum_{k=1}^{j-1} p_{ik} < w_t \leq \sum_{k=1}^{j} p_{ik}$$

$$= 0, \quad \text{otherwise}$$

To illustrate the technique, we again consider the 2-state example of the previous section. As before, the ω-sequence is

$$w_0 = .9969, \quad w_1 = .7920, \quad w_2 = 0881,$$

from which we obtain,

$$F(w_0) = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}, \quad F(w_1) = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}, \quad F(w_2) = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

Now to implement Method 2, we require the following sequence, τ, of random numbers, again taken from a table: {.9464, .4034, .7434, .6142, .5820 }

We have:

$$\bar{z}_3(\omega, \tau) = 0,$$

$$\therefore [c + \bar{z}_3(\omega, \tau)] = \begin{bmatrix} 1 \\ 2 \end{bmatrix} = 3 \begin{bmatrix} 1/3 \\ 2/3 \end{bmatrix}$$

and we use the random number .9464 to rewrite this in the degenerate form, $[3].[0 \quad 1]^T$, i.e. identifying state e_2 with weighting factor 3.

Possible transitions from state e_2 are determined by the positions of ones (if any) in column 2 of $F(w_2)$. We see that in this case there is no transition to be calculated and our estimate of λ_2, i.e. $\bar{z}_2(\omega, \tau)$ is simply the zero vector, $[0 \quad 0]^T$. To prepare for the next transition, the vector c is added to $\bar{z}_2(\omega, \tau)$ and another random number, .4034, used to provide the degenerate form $[3].[0 \quad 1]^T$. Looking at column 2 of $F(w_1)$, the presence of two 1's in the column means that another random number (.7434) is required to reduce the matrix to the degenerate form

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 \\ 0 & 2 \end{bmatrix},$$

and the transition is calculated, giving an estimate of λ_1, i.e.

$$\bar{z}_1(\omega, \tau) = [6] \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Preparing for the next transition,

$$[c + \bar{z}_1(\omega, \tau)] = \begin{bmatrix} 1 \\ 8 \end{bmatrix}$$

and .6142 provides the degenerate form $[9].[0 \quad 1]^T$.

The final number in the τ-sequence, .5820, reduces $F(w_0)$ to

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 \\ 0 & 2 \end{bmatrix}$$

and calculating the final transition provides the λ_0 estimate,

$$\bar{z}_0(\omega, \tau) = [18] \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The estimates obtained by this method are apparently worse than those provided by Method 1, but this method offers the considerable advantage that the adjoint vector estimates are stored in a degenerate form, whereas Method 1 yields a sequence of degenerate vectors, but requires summations (sometimes very time consuming if the simulation is

long) to provide adjoint estimates. Partial summations cannot be stored during the execution of Method 1 because the resulting vector could easily have as many non-zero elements as the number of simulated transitions. The relative efficiencies of Methods 1 and 2 with respect to run time and variance are well illustrated by later examples.

4.8.7 Forward-time estimation of the adjoint sequence

This approach is recommended for cases where estimates of only one or two elements of each adjoint vector are required, and is especially effective when used as part of a gradient estimation routine, where the sequence $\{x_t(\omega)\}$ effectively determines which elements of the vectors $\{\lambda_t,\ t = 0, \ldots, N\}$ should be estimated.

From equation 4.4, we have

$$\lambda_t > = Pc> + P^2c> + \ldots + P^{N-t}c> \qquad 4.31$$

Suppose we wish to estimate element i of the vector λ_t, i.e. we require an estimate of

$$<e_i, \lambda_t> = <e_i, [Pc> + P^2c> + \ldots + P^{N-t}c>] \qquad 4.32$$

Section 4.7 describes a method for estimating the vectors $\{<e_iP, <e_iP^2, \ldots, <e_iP^{N-t}\}$, using the random number sequence ω. Applying equation 4.13, and defining $x_t = e_i$, the resulting degenerate row vector estimates are

$$\{<e_iF(w_t), <e_i \prod_{k=t}^{t+1} F(w_k), \ldots, <e_i \prod_{k=t}^{N-1} F(w_k)\}$$

which can be seen to have the required expected values from equation 4.11. Thus, at least in principle, we could obtain an unbiased estimate of $\langle e_i, \lambda_t \rangle$ from

$$\langle e_i, \lambda_t \rangle = E_\omega \{ \sum_{j=t}^{N-1} \langle e_i \prod_{k=t}^{j} F(w_k), c \rangle \} \qquad 4.33$$

If we define a 'state trajectory' by equation 4.13, repeated below:

$$\langle x_{t+1} = \langle x_t . F(w_t), \quad t = 0, 1, \ldots, N-1 \qquad 4.13$$

with given initial state x_0, then it follows from equation 4.33 that by generating a state trajectory $\{x_0 = e_i, x_1 = e_j, \ldots, x_{N-1} = e_k, x_N\}$, we can obtain unbiased estimates of $\langle e_i, \lambda_0 \rangle, \langle e_j, \lambda_1 \rangle, \ldots, \langle e_k, \lambda_{N-1} \rangle$.

Suppose, assuming the worst case, that we wish to estimate all s elements of each co-state vector, i.e. $\{\langle e_i, \lambda_t \rangle, i = 1, \ldots, s; t = 0, 1, \ldots, N\}$.

Using the random number sequence ω, and equation 4.13, we can obtain estimates of the s elements $\{\langle e_i, \lambda_0 \rangle, i = 1, \ldots, s\}$ by generating s state trajectories with initial states $\{x_0 = e_i, \; i = 1, \ldots, s\}$, and using the estimator defined below:

$$\lambda_0 \rangle = \begin{bmatrix} \langle e_1, \lambda_0 \rangle \\ \vdots \\ \langle e_s, \lambda_0 \rangle \end{bmatrix} = E_\omega \left\{ \begin{bmatrix} \sum_{j=0}^{N-1} \langle e_1 \prod_{k=0}^{j} F(w_k), c \rangle \\ \vdots \\ \sum_{j=0}^{N-1} \langle e_s \prod_{k=0}^{j} F(w_k), c \rangle \end{bmatrix} \right\} \qquad 4.34$$

If, and only if, the Markov chain is regular (as defined in Kemeny and Snell (45, p.35)) and has no transient states, then there is a finite probability that at each $t \in \{1, \ldots, N-1\}$ at least one of the s trajectories above will progress from each state $\{x_t = e_j, j = 1, \ldots, s\}$.

In this case we could, in principle, generate all possible random number sequences ω and not only estimate λ_0, but also use the same state trajectories and equation 4.35 to estimate $\{\lambda_t, t = 1, \ldots, N-1\}$:

$$\lambda_t> = \begin{bmatrix} <e_1, \lambda_t> \\ \vdots \\ <e_s, \lambda_t> \end{bmatrix} = E_\omega \left\{ \begin{bmatrix} \sum_{j=t}^{N-1} <e_1 \prod_{k=t}^{j} F(w_k), c> \\ \vdots \\ \sum_{j=t}^{N-1} <e_s \prod_{k=t}^{j} F(w_k), c> \end{bmatrix} \right\} \qquad 4.35$$

Unfortunately, this approach is impractical, not only because it applies only to regular Markov chains, but also because it would be necessary to generate enough state trajectories to ensure that every state $\{x_t = e_i, i = 1, \ldots, s\}$ is included in at least one state trajectory at each $t = 0, \ldots, N-1$. The probability that a given state is included in some trajectory may be so low that the variance of any estimate based on transitions from that state would be unacceptably high. Also we consider the estimation of all elements of each adjoint vector to be an extreme requirement, many cases calling for estimation of no more than one or two elements of each vector, so that whilst this approach efficiently estimates λ_0, its efficiency in estimating λ_t, $t = 1, \ldots, N-1$ can be low. We therefore seek a method which allows us to concentrate our efforts upon estimating just those elements required from each vector.

4.8.8 Practical forward-time estimators

Method 3a

To illustrate a practical solution, we again consider the problem of estimating $\{\langle e_i, \lambda_t \rangle, i = 1, \ldots, s; t = 0, \ldots, N-1\}$. A brute force approach might be to use the random number sequence ω and equation 4.13 to generate s state trajectories over N transitions, with initial states $\{x_0 = e_i, i = 1, \ldots, s\}$, defining the estimator for λ_0 by equation 4.34. Forgetting obvious redundancies temporarily, we can generate another s state trajectories over N-1 transitions from t = 1 with initial states $\{x_1 = e_i, i = 1, \ldots, s\}$, defining the estimator for λ_1 by equation 4.35. Continuing this procedure up to t = N-1 gives us a large number of state trajectories to follow and for N and s of only modest values, (e.g. N = 1000 and s of order several hundred) the computational burden becomes intolerable. Nevertheless we include results which give an indication of the computing time and accuracy of this approach in Section 4.9 and continue by considering a weighting method similar to von Neuman's 'Russian Roulette', which reduces the number of trajectories to a manageable level.

Method 3b

Let ρ denote another sequence of N random numbers $\{r_i, i = 0, \ldots, N-1\}$. For each $t \in \{0, \ldots, N-1\}$ we define an integer random variable $i(r_t)$,

$$i(r_t) \triangleq k, \quad \text{when } \frac{k-1}{s} < r_t \leq \frac{k}{s} \qquad 4.36$$

$$k \in \{1, \ldots, s\}$$

from which we obtain a sequence of (row-) vectors

$$\bar{x}_t = e_{i(r_t)}, \quad t = 0, \ldots, N-1 \qquad 4.37$$

By setting $x_t = \bar{x}_t$ and applying equation 4.13, we generate one state transition from each x_t, t = 0, ..., N-1 and record the following vector sequence:

$$\ell_t^1(\omega, \rho) \triangleq [\,[s] < x_t \,.\, F(w_t),\, c>]\, x_t> \qquad 4.38$$

where the superscript signifies that only one transition from state $\bar{x}_t$ selected at t is considered.

Because there exists a finite (and equal) probability that any one of the s possible states of the chain is selected at each t = 0, ..., N-1, then there is also a finite probability that a trajectory (as defined by the use of equation 4.13 with ω) is followed for (N-t) transitions from a given state $\bar{x}_t$. The probability of following a trajectory for n transitions is $1/s^n$, and so, at the end of the n'th transition from state $x_t = \bar{x}_t$, we define another vector

$$\ell_t^n(\omega, \rho) \triangleq [\,[s^n] < x_t \prod_{j=t}^{n+t-1} F(w_j),\, c>]\, x_t> \qquad 4.39$$

Any $\ell_t^n(\omega, \rho)$ not defined by equation 4.38 or 4.39 is set equal to zero.

Let

$$\ell_t(\omega, \rho) \triangleq \sum_{k=1}^{N-t} \ell_t^k(\omega, \rho) \qquad 4.40$$

for t = 0, 1, ..., N - 1

Proposition 4.5

$$E_\omega \{ E_\rho \{\ell_t(\omega, \rho)\} \} = \lambda_t, \quad t = 0, 1, \ldots, N-1$$

Proof

From equation 4.40

$$E_\omega \{ E_\rho \{\ell_t(\omega, \rho)\} \}$$

$$* = E_\omega \left\{ \sum_{k=1}^{N-t} \frac{1}{s^k} \sum_{i(r_t)=1}^{s} [\, [s^k] < e_{i(r_t)} \cdot \prod_{j=t}^{k+t-1} F(w_j), c> \,] \cdot e_{i(r_t)}> \right\}$$

$$= E_\omega \left\{ \sum_{k=1}^{N-t} \sum_{i(r_t)=1}^{s} [<e_{i(r_t)} \prod_{j=t}^{k+t-1} F(w_j), c> \,] \cdot e_{i(r_t)}> \right\}$$

$$= E_\omega \left\{ \sum_{k=1}^{N-t} \begin{bmatrix} <e_1 \prod_{j=t}^{k+t-1} F(w_j), c> \\ \vdots \\ <e_s \prod_{j=t}^{k+t-1} F(w_j), c> \end{bmatrix} \right\}$$

$$= \lambda_t, \qquad \text{from equations 4.11 and 4.35}$$

It should be apparent that Methods 3a and 3b as described are two extremes, i.e. Method 3a provides estimates of the s components of each adjoint vector while Method 3b makes use of weighting methods to reduce the computing burden in each run by estimating only one component of each vector. The price paid for this reduction in work-load is an increase in the variance of the estimated values. The ideal method would allow estimation of as many vector components as computer

* From the previous page and equation 4.39, it is apparent that the k'th transition of the trajectory starting from state $<x_t = <e_{i(r_t)}$ is generated with probability $1/s^k$, and this holds for $i(r_t) = 1, \ldots, s$.

resources allow, leaving weighting methods to account for the remainder. A single run would thus be forced to provide the maximum amount of information without becoming impractical. It is not difficult to see that the weighting method employed above can be modified in a number of ways to achieve this objective.

As a simple illustration which will be relevant to our discussion of gradient estimation, we consider the estimation of one component of each of N adjoint vectors, where N is a very large number. When using estimation method 3a, an additional trajectory is required at each t = 0, 1, ..., N-1, so that as the simulation progresses, we are faced with the impossible task of following up to N trajectories. Method 3b can be modified to solve this problem. Assuming that the computer can generate a maximum of m trajectories simultaneously, we can begin estimating the first m components by Method 3a. Thus the trajectory beginning at state $x_0 = e_i$ adds a 'contribution' $\langle e_i \prod_{j=0}^{t} F(w_j), c\rangle$ to the estimate of the component $\langle e_i, \lambda_0\rangle$ at each $t \in \{0, ..., m\}$.

By the end of the m'th transition, a total of m trajectories are being generated, and the requirement for another component estimate (of $\langle x_m, \lambda_m\rangle$) calls for initiation of another trajectory, which we assume to be beyond the computer's capability. A variant of Method 3b allows the elimination of one trajectory from the (m+1) without introducing bias. Using the former sequence ρ, of random numbers, we redefine the integer random variable:

$$i(r_t) \triangleq k, \quad \text{when} \quad \frac{(k-1)}{(m+1)} < r_t \leq \frac{k}{(m+1)}, \qquad 4.41$$

$$k \in \{1, 2, ..., m, (m+1)\}$$

$$\text{for } t = (m+1), (m+2), ..., (N-1)$$

After labelling the possible trajectories $\{1, 2, \ldots, (m+1)\}$, then at $t = m+1$ we discard (i.e. generate no more transitions or contributions for) the one having $i(r_{m+1})$ as its label. The probability that each of the $m+1$ trajectories is terminated in this way is $1/(m+1)$, and therefore each survives with probability $m/(m+1)$.

Thus in order to avoid bias we weight further contributions from the m survivors by the factor $(m+1)/m$. Because of the structure of the problem, it turns out that by following this procedure at each $t \in \{(m+2), (m+3), \ldots, (N-1)\}$, we always have to discard one from $m+1$ trajectories at the beginning of each transition. If a trajectory 'survives' n of these trials of 'Russian Roulette', then the contribution from its next transition will be weighted by the factor $[(m+1)/m]^n$. It follows that the trajectory starting at state $x_0 = e_i$ and surviving until $t = m + n$ will provide a contribution

$$[(m+1)/m]^n \cdot \langle e_i \prod_{t=0}^{m+n} F(w_t), c \rangle \quad *$$

to the estimate of $\langle e_i, \lambda_0 \rangle$ at its next transition.

If it is possible to classify trajectories as more, or less interesting by some criterion, it may be desirable to increase the probability of discarding less interesting trajectories and therefore increase the probability of keeping the 'more interesting' ones, adjusting the weighting factors accordingly. Using this, and other variations on the Russian Roulette weighting technique, we can exercise complete control over the size of the computing problem, and we can make the most effective use of the resources available.

---

*The probability that the trajectory 'survives' each trial of 'Russian Roulette' is $(m/(m+1))$, and it therefore survives n trials with probability $(m/(m+1))^n$. The expected value of the 'contribution' from the n th transition is therefore $\langle e_i \prod_{t=0}^{n+0} F(w_t), c \rangle$, as required.

Method 3c

The work of Howard (7) and others has shown that the adjoint vector sequence for a regular Markov chain approaches the following asymptotic form:

i.e. for large n ($\leq$ N)

$$\lambda_{N-n}> \cong ng\ 1> + \lambda> \qquad 4.42$$

where g is a scalar representing the mean process cost per cycle and λ is a constant s-vector for the chain quantifying the relative merits of commencing a trajectory in each of the s states.

Whereas g indicates the steady-state performance of the process, the vector λ represents the transient costs associated with each state. We have established that in order to estimate gradients, we require estimates of both the state distribution vector sequence and the adjoint vector sequence and we see from equation 4.7 that the formulation requires that adjoint vectors be pre-multiplied by the matrix product $[\frac{\partial}{\partial \alpha} D(\alpha)\ ]\ P$. The assumption of long simulation runs effectively allows the use of the assymptotic adjoint equation (4.42), and because P is a probability matrix, it can be seen that the following equation holds:

$$\begin{aligned} P\lambda_{N-n}> &\simeq n g P\ 1> + P\lambda> \\ &= n g\ 1> + P\lambda> \end{aligned} \qquad 4.43$$

From Section 4.5, we see that rows of $[\frac{\partial}{\partial \alpha} D(\alpha)]$ are either zero, or of the form [0....0 1 0....0 -1 0....0], so that

$$[\frac{\partial}{\partial \alpha} D(\alpha)\ ]P\lambda_{N-n}> \simeq [\frac{\partial}{\partial \alpha} D(\alpha)\ ]\ P\lambda> \qquad 4.44$$

i.e. for the purposes of gradient estimation, we are not concerned with the steady-state costs of the chain, and the adjoint estimation routine is reduced to providing an estimate of the constant vector λ . The fact that λ represents a transient feature of the chain is highly significant. The estimation of λ theoretically requires simulation of an infinite number of cycles, but because $[\lambda_{N-n}> - \lambda>] \to 0$, we can in practice abandon the simulation after a finite number of cycles (either fixed beforehand, or decided by some stochastic technique - see later discussion of non-terminating sequences in Section 6.3.4; also Kahn (29)).

As the components of λ represent the relative costs associated with the s states of the chain, we are justified in setting the s'th component of λ equal to zero and estimating the remaining (s-1) components relative to that. This method therefore estimates the 'relative cost vector' defined by

$$\bar{\lambda}> \triangleq \lambda> - [<e_s, \lambda>] \, . \, 1> \qquad 4.45$$

Method 3c differs from those previously described in that components of the relative cost vector $\bar{\lambda}$ are estimated from simulated trajectories which have a stochastic run length. Also it follows that the number of trajectories being generated at any $t \in \{0, 1, \ldots, N-1\}$ is a random variable which is influenced by the transient response of the system.

If all s elements of $\bar{\lambda}$ are to be estimated, we simultaneously generate s trajectories starting from the states:

$$\{x_0^1 = e_1, x_0^2 = e_2, \ldots, x_0^s = e_s\}$$

and using a sequence of degenerate row matrices with transitions defined by:

$$<x_{t+1} = <x_t \cdot F(w_t), \quad t = 0, 1, \ldots, \quad j = 1, 2, \ldots, s \qquad 4.46$$

with $F(w_t)$ defined as in Section 4.7.1, and with the sequence of random numbers redefined by

$$\omega \triangleq \{w_0, w_1, \ldots\ldots\ldots\}$$

After generating the n'th transition (n = 1, 2, ...), of each trajectory we add the following 'contribution' to the estimate of $\bar{\lambda}$:

$$\begin{bmatrix} <x_0^1 \prod_{j=0}^{n-1} F(w_j), c> - <x_0^s \prod_{j=0}^{n-1} F(w_j), c> \\ \vdots \\ <x_0^{s-1} \prod_{j=0}^{n-1} F(w_j), c> - <x_0^s \prod_{j=0}^{n-1} F(w_j), c> \\ 0 \end{bmatrix} \qquad 4.47$$

We terminate the k'th trajectory (i.e. we generate no more transitions or contributions) after m transitions, when

$$<x_0^k \prod_{j=0}^{m-1} F(w_j) = <x_0^s \prod_{j=0}^{m-1} F(w_j), \quad k = 1, \ldots, s-1 \qquad 4.48$$

as it can be seen from equation 4.47 that further transitions would make no contribution to the estimate of $<x_0^k, \bar{\lambda}>$. It follows therefore that the run length of each trajectory is a random variable whose mean is a function of the eigenvalues of the chain's transition matrix.

When all trajectories have been terminated, we are left with an estimate of $\bar{\lambda}$ which consists of the sum of a random number of contributions of the form shown in equation 4.47. Proposition 4.6 establishes that the estimator is unbiased.

Proposition 4.6

$$E_{\omega}\left\{\sum_{n=0}^{\infty}\begin{bmatrix} \langle x_0^1 \prod_{j=0}^{n} F(w_j), c\rangle - \langle x_0^s \prod_{j=0}^{n} F(w_j), c\rangle \\ \vdots \\ \langle x_0^{s-1} \prod_{j=0}^{n} F(w_j), c\rangle - \langle x_0^s \prod_{j=0}^{n} F(w_j), c\rangle \\ 0 \end{bmatrix}\right\} = \bar{\lambda} \rangle \quad 4.49$$

where $x_0^k \triangleq e_k, \quad k = 1, \ldots, s.$

Proof

If N is large, then using equations 4.33 and 4.42,

$$E_{\omega}\left\{\sum_{n=0}^{N-1} \langle x_0^k \prod_{j=0}^{n} F(w_j), c\rangle\right\}$$

$$= \langle e_k, \lambda_0 \rangle$$

$$= \langle e_k, [N g\, 1 + \lambda] \rangle$$

$$= N g + \langle e_k, \lambda \rangle, \quad k = 1, \ldots, s \qquad 4.50$$

Hence

$$E_{\omega}\left\{\sum_{n=0}^{\infty}\begin{bmatrix} \langle x_0^1 \prod_{j=0}^{n} F(w_j), c\rangle - \langle x_0^s \prod_{j=0}^{n} F(w_j), c\rangle \\ \vdots \\ \langle x_0^{s-1} \prod_{j=0}^{n} F(w_j), c\rangle - \langle x_0^s \prod_{j=0}^{n} F(w_j), c\rangle \\ 0 \end{bmatrix}\right\}$$

$$= \begin{bmatrix} \langle e_1, \lambda\rangle - \langle e_s, \lambda\rangle \\ \vdots \\ \langle e_{s-1}, \lambda\rangle - \langle e_s, \lambda\rangle \\ 0 \end{bmatrix}$$

$$= \lambda\rangle - [\langle e_s, \lambda\rangle] \cdot 1\rangle$$

$$= \bar{\lambda}\rangle \qquad 4.51$$

The important feature of the forward-time estimators is that adjoint vectors and state distributions can be estimated simultaneously. Moreover, these adjoint estimators exhibit a significant improvement in efficiency compared with the alternatives, when estimates of only a relatively small number of elements of each vector are required. The advantages become apparent in the next section.

4.9 Efficient gradient estimation using forward-time simulations

Consider a state trajectory $\{x_t(\omega)\}$ generated by a controlled Markov chain over N cycles (where N is large) using the random number sequence ω, consisting of 2N numbers (see note on simulating controlled chains, Section 4.7.1). To simplify the discussion we also assume that only two control alternatives exist, giving rise to a probabilistic controller with only one parameter, α.

It is convenient to repeat below the formulation of the cost gradient using the state distribution and adjoint vector sequences:

$$\frac{\partial}{\partial\alpha}.L(\alpha, \pi_0(\alpha), 0) = \sum_{t=0}^{N-1} <\pi_t(\alpha) \left[\frac{\partial}{\partial\alpha}.D(\alpha)\right] P, [c> + \lambda_{t+1}>] \quad 4.7$$

Proposition 4.1 tells us that the degenerate vector sequence $\{x_t(\omega)\}$ is an unbiased estimate of $\{\pi_t\}$, and so substituting $\{x_t(\omega)\}$ for $\{\pi_t\}$ in equation 4.7 above provides an expression for an unbiased gradient estimator. We see now that very few adjoint vector components are required to provide an estimate of the cost gradient. The sequence of row vectors $\{<x_t.[\frac{\partial}{\partial\alpha}.D(\alpha)], t = 0, 1, ..., N-1\}$ consists of either zero vectors or vectors of the form [0....0 1 0....0 -1 0....0] so that each degenerate vector x_t, $t \in \{0, ..., N-1\}$ effectively specifies which components (at most two) of the vector $P\lambda>$ (λ rather than λ_t because a long simulation run is assumed) are required . The estimation of components of $P\lambda>$ is carried out by an obvious extension of Method 3c, using the same random number sequence ω , as that used to generate $\{x_t(\omega)\}$. Thus a small (and random) number of 'adjoint' simulation runs is initiated directly by the sequence $\{x_t(\omega)\}$ which, also bearing in mind that the same random numbers are used for each run, results in a very efficient and accurate gradient estimation method. Moreover, a point which should be emphasised is that the 'main' simulation providing

$\{x_t(\omega)\}$ is also providing one of the components needed for the calculation of $[0....0\ 1\ 0....0\ -1\ 0....0].P\lambda>$. This can be seen by reference to Fig. 4.1, which is a graphical illustration of the forward-time estimation processes. Although $D(\alpha)$ and P show the matrix operations resulting in $\{\pi_t\}$, we realise that the simulations create sequences of degenerate estimates of these quantities.

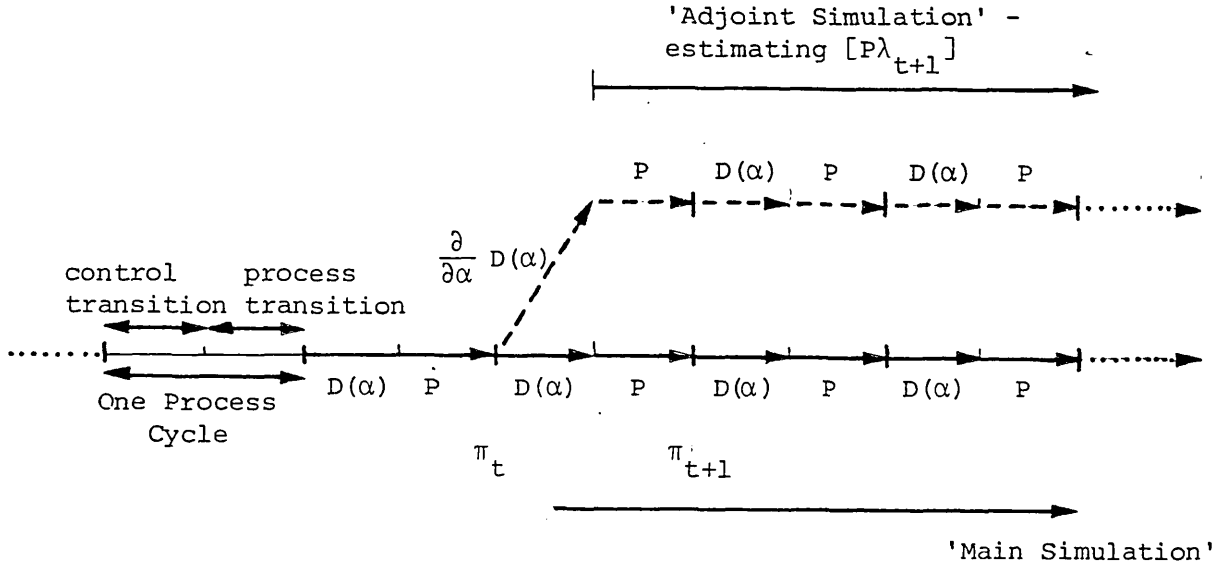


Fig. 4.1: Graphical illustration of the forward-time estimation technique.

Referring to Fig. 4.1, it can be seen that at time t, the main simulation has provided the degenerate estimate x_t of π_t, and we assume that $<x_t.[\frac{\partial}{\partial\alpha}.D(\alpha)]$ is a non-zero vector, i.e. a vector of the form $[0....0\ 1\ 0....0\ -1\ 0....0]$. Note however that the 'main simulation' progresses from the simulated state x_t with a degenerate estimate of the vector $<x_t.D(\alpha)$ of the form $[0....0\ \alpha\ 0....0\ (1-\alpha)\ 0....0]$, such that whichever control state is assumed by this trajectory, it must be one of the two initial states of the 'adjoint simulations' required to estimate components of $P\lambda>$. Because the main and adjoint simulations use the same random number sequence from that point forward, we can estimate one of the necessary components of $P\lambda>$ using the main simulation, thus making the estimation process even more efficient.

It can be seen that by using a sequence ω of random numbers to generate a 'main simulation run' which in turn initiates a number of 'adjoint simulations', we can obtain an unbiased estimate of the gradient of cost over N cycles, or, multiplying by the factor 1/N, the gradient of mean cost per cycle. The computing burden at any given point in the main simulation depends upon the run-length of the adjoint simulations (a stochastic variable related to the eigenvalues of the system), and upon the rate at which new adjoint runs are initiated (dependent upon the relationship between the steady-state probability distribution of the process states and the way the state groups are specified for control purposes). The weighting method described in conjunction with adjoint estimation method 3b can be used to overcome problems of excessive work-load, but it is often found (certainly in all cases considered in this thesis) that Method 3c can be applied to relatively large problems without major difficulty.

4.10 Application of each estimation method to a simple three-state problem

To illustrate and compare the estimation techniques described in this chapter, we consider a three-state controlled Markov chain defined by the following:

$$\text{Process transition matrix, } P = \begin{bmatrix} .7 & .2 & .1 \\ .1 & .5 & .4 \\ .1 & .6 & .3 \end{bmatrix}$$

$$\text{Control matrix, } D(\alpha) = \begin{bmatrix} (1-\alpha) & 0 & \alpha \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\therefore \frac{\partial}{\partial \alpha} \cdot D(\alpha) = \begin{bmatrix} -1 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\text{Cost vector, } c\rangle = \begin{bmatrix} 1.0 \\ 2.0 \\ 4.0 \end{bmatrix}$$

$$\text{Expected one-stage cost vector, } q = Pc\rangle = \begin{bmatrix} 1.5 \\ 2.7 \\ 2.5 \end{bmatrix}$$

Mean process cost per cycle = 2.3409

$$\text{Relative cost vector, } \bar{\lambda}\rangle = \lambda\rangle - \langle e_1, \lambda\rangle \cdot 1\rangle = \begin{bmatrix} 0 \\ 2.8636 \\ 2.6818 \end{bmatrix}$$

(Note that this definition differs slightly from that given earlier)

We consider 10 cycles of the process, operating with nominal control parameter $\bar{\alpha} = 0.0$ (thus $D(\bar{\alpha}) = I$, the 3 x 3 identity matrix, and $P(\bar{\alpha}) = P$), and initial state $x_0 = [0\ 0\ 1]$. As a standard for comparison, we solve equations 4.8 and 4.9 to obtain the accurate vector sequences $\{\pi_t,\ t = 0, \ldots, 10\}$ and $\{\lambda_t,\ t = 0, \ldots, 10\}$. Substitution of these vectors into equation 4.7 provides the exact value of the gradient of total process cost over ten cycles with respect to α, which turns out to be 4.039989. Each of the methods previously described for estimating the state distribution and adjoint vectors has been applied to this example and full details can be found in Appendix III. The appendix lists results in the form of twenty-one tables and includes the standard (calculated) sequences of state distributions and adjoint vectors, which can be compared with other tables listing estimates of the same sequences.

Results appearing in Chapter 6 indicate that when using estimated state distribution and adjoint vector sequences to obtain cost gradient estimates, most of the variance (as much as 98% for one example) can be attributed to the adjoint estimator, and comparatively little to the state distribution estimator. It is therefore interesting to analyse and compare variance for each adjoint estimator. The variances relating to components of the ten adjoint vector estimates were determined experimentally for each estimator, and are included with a number of typical sequence estimates in Appendix III.

Methods 1 and 2 are similar in that they both generate 'state'-trajectories in reverse-time, and both are designed to estimate the <u>complete</u> vector λ_t, for each t (i.e. it is not possible to concentrate computational effort on estimating several vector components of particular interest). In one respect, however, Method 1 differs significantly from Method 2. Method 1 obtains its adjoint vector

estimates by summing selected estimates of the vectors $\{P(\alpha)c\ ,\ P^2(\alpha)c,\ \ldots.\}$, and therefore does not estimate the adjoint vectors directly. On the other hand, Method 2 directly obtains degenerate vector estimates of the adjoints, and this fact is significant when considering long simulations of large Markov chains. In such cases, the use of Method 1 requires the storage of many degenerate vector estimates until it is possible to form the vector summations. This represents inefficient use of computer memory and time, and makes Method 2 the better choice for long simulations. However, one should also bear in mind the relative accuracy of the methods, which is indicated by results in Appendix III. Method 1 appears to be more accurate (for this example) by a factor of up to 200.*

Turning to the forward-time estimators: results in Appendix III show clearly that their main advantage over the reverse-time methods is that they only estimate those adjoint vectors (or components of vectors) actually required. Method 3a is shown to be more accurate than Method 1 by a factor of about 10.** The appendix also indicates a particular feature of Method 3c. Estimation of $\bar{\lambda}$ by this method requires little computing time even though the estimates are very accurate. It is clear that forward-time methods are generally much more efficient than reverse-time methods.

Experimental results relating to the estimators of the vector sequences have been given over to an appendix because our main concern in this thesis is to develop estimators of the cost gradient. It is important to determine the relative efficiencies of the different cost gradient estimators which emerge from the theory given in this chapter. To this end, we consider Table 4.1, which lists experimental results obtained from the application of four different gradient estimators to the three-state problem described above. As the same state distribution

* See Tables AIII.15, AIII.16, page 287.

** See Tables AIII.15, AIII.19, pp. 287, 289.

Gradient estimator	Description of adjoint sequence estimator used	Number of runs	Gradient estimate (ensemble average)	(estimated) single-run variance	(estimated) single-run S.D.	Computing time for experiment (secs)
program G1	Method 1: Indirect adjoint estimation in reverse-time	1 20 1000 10,000	-11.9700 -6.4640 +5.0577 +4.4404	2090.00	45.72	1.233 1.304 4.607 35.164
program G2	Method 2: Direct adjoint estimation in reverse-time	1 20 1000 10,000	+56.0000 +20.3000 +15.0080 +5.6105	21,700.00	147.64	1.244 1.371 5.783 47.055
program G3a	Method 3a: Forward-time method generating separate and complete state trajectories for each component estimate	1 20 1000 10,000	+2.0000 +5.7000 +4.1730 +3.9623	47.92	6.92	0.946 1.028 3.860 29.415
program G3b	Method 3b: As for Method 3a but 'Russian Roulette' used to reduce no. of adjoint transitions to one per cycle	1 20 1000 10,000	+2.0000 +8.3500 +3.9310 +3.9996	78.54	8.86	1.076 1.039 2.445 16.179

Notes: 1. Each gradient estimator obtains estimates of the state distribution sequence by the method described in Section 4.7.

2. Accurate gradient of total cost over 10 transitions = +4.039989.

Table 4.1: Comparison of gradient estimators

estimates are used in each case, it is convenient to characterize each gradient estimator according to the type of adjoint estimator used: two reverse-time methods - the 'indirect' Method 1 and the 'direct' Method 2, and two forward-time methods - Method 3a and Method 3b, (the latter using 'Russian Roulette'), give rise to the gradient estimator labels: programs G1, G2, G3a and G3b respectively. For program G3b, we assume a highly improbable situation where computing resources can only cope with the generation and storage of the 'main trajectory' plus a single adjoint transition at each t = 0, ..., 9.

Comparing the estimated variances, and bearing in mind the computing times for each estimator we note that program G1 is more efficient than program G2 by a factor of 14*, but that neither can compete with the remaining two estimators which exclusively use forward-time simulations. Gains in efficiency for both program G3a and G3b are of the order of 50 compared with program G1. A more detailed analysis of relative accuracy and a discussion of variance reduction methods is given in Chapter 6.

4.11 Summary

The aim of the chapter is to develop procedures for determining gradients of cost with respect to control parameters for very large Markov chains. Because the transition matrix and vectors associated with such models are so large, and therefore not amenable to storage and processing by computer, we propose to estimate the gradients using Monte Carlo methods.

The chapter gives details of a method for estimating the sequence of state distribution vectors, together with two Monte Carlo methods for estimating the sequence of adjoint vectors in reverse-time. It turns

* Calculated using Moy's definition of "normalized variance reduction" appearing on page 223.

out, however, that forward-time simulations can more accurately provide the required adjoint vector estimates using less computing time, and offering the advantage that state distributions and adjoints can be estimated from a single set of forward-time simulations, using one random number sequence. Three variations of the forward-time adjoint estimator are presented, together with an application of the 'Russian Roulette' weighting method, used in this case to reduce the computing burden without biasing the result. Gradient estimates are obtained by direct calculation using the distribution and adjoint vector estimates, or, if estimating the adjoints in forward-time, by simultaneous application of the distribution and adjoint estimators during each simulation run. Finally, a three-state Markov decision problem is used to demonstrate and compare the distribution, adjoint, and gradient estimation methods.

CHAPTER 5

SIMULATION DETAILS FOR THE TRANSFER-LINE PROBLEM AND SOLUTION USING A STOCHASTIC APPROXIMATION ALGORITHM

5.1 Introduction

Having given the theoretical background to several Monte Carlo estimation techniques in Chapter 4, we use this chapter to assess their efficiency when applied to larger operations research problems.

We begin by discussing in detail the simulation of cross-linked transfer-line examples in forward- and reverse-time. Attempting to generate state transitions by applying the theory of Section 4.7.1 to the complete system model would present computational problems. In the case of forward-time simulation, it is perhaps obvious that a complex system can be regarded as a number of smaller subsystems which can each be simulated using independent streams of random numbers, thus reducing program complexity. It may not be so easy to see how a reverse-time simulation used for adjoint vector estimation can be viewed as a dual of the forward-time process. It is shown that there exists a direct correspondence between subsystem equations used in both simulations.

The chapter continues with a comparison of the gradient estimation methods as applied to the standard 64-state transfer-line example of Appendix I. Detailed results with an analysis of variance in each case indicate the relative efficiencies of each method.

A further section shows, using two examples, how a stochastic approximation algorithm solves the optimal control problem using

gradient estimates, and comments on the selection of algorithm parameters.

5.2 Simulation details for the transfer-line problem

5.2.1 Simplifying simulations by identifying subsystems

Consider any system for which we can identify two subsystems. The complete system description uniquely identifies each combination of the states assumed by each of the two subsystems, A and B. Thus, at time t, if the state of subsystem A is $x_t^A \in \{e_{n_A}, n_A = 1, 2\}$, and the state of subsystem B is $x_t^B \in \{e_{n_B}, n_B = 1, 2\}$, then a state x_t in the complete system is defined to be $x_t = e_m$, where $m = h(n_A, n_B)$ and h is any integer-valued, one-to-one function. One definition of h is given by equation 5.1:

$$m = h(n_A, n_B) \triangleq (n_A - 1) \times 2 + n_B \qquad 5.1$$

It is convenient to repeat equations 2.3 and 2.4, which define probabilities for combinations of two events, X and Y. If Y affects the probability of X, then the probability of the compound event [XY] is given by:

$$\text{Prob}\,[XY] = \text{Prob}\,[X|Y] \,.\, \text{Prob}\,[Y] \qquad 2.3$$

where Prob [X|Y] is the conditional probability of X given Y. If X and Y are independent,

$$\text{Prob}\,[X|Y] = \text{Prob}\,[X] \qquad 2.4$$

Here, we define X to be the event $[x_t^A = e_{n_A}]$ and Y to be the event $[x_t^B = e_{n_B}]$. The compound event is thus $[x_t = e_m]$, and if the subsystems are independent,

$$\text{Prob}\,[x_t = e_m] = \text{Prob}\,[x_t^A = e_{n_A}] \,.\, \text{Prob}\,[x_t^B = e_{n_B}] \qquad 5.2$$

Also, it can be seen that the equation holds for conditional probabilities:

$$\text{Prob}\,[x_t = e_m | x_{t-1}]$$

$$= \text{Prob}\,[x_t^A = e_{n_A} | x_{t-1}^A] \,.\, \text{Prob}\,[x_t^B = e_{n_B} | x_{t-1}^B] \qquad 5.3$$

If the 4 x 4 transition probability matrix for the complete system is P, then by definition, its elements $\{p_{im}\}$ are given by

$$p_{im} \triangleq \text{Prob}\,[x_t = e_m | x_{t-1} = e_i] \qquad 5.4$$

and it follows that the conditional distribution $(<x_{t-1} P)$ is obtained through use of equation 5.3, setting $n_A = 1,2$; $n_B = 1,2$. Thus the transition probabilities for the complete system can be defined in terms of subsystem transition probabilities.

Section 4.7.1 shows that any sequence of state vectors, $\{x_t, t = 0, 1, \ldots\}$ obtained through the use of a single stream of random numbers, is an unbiased estimate of the sequence of probability distributions, $\{\pi_t, t = 0, 1, \ldots\}$. However, for the example above, it is clear that generation of the sequence $\{x_t, t = 0,1,\ldots\}$ can be carried out much more easily by using independent streams of random numbers to simulate state sequences for each subsystem, i.e. $\{x_t^A, x_t^B; t = 0, 1,\ldots\}$ and then applying equation 5.1.

Of course, this approach can still be used when the subsystems are dependent. Let the conditional probability distribution for x_t^A be conditioned not only by x_{t-1}^A but also by x_t^B, i.e. in this case equation 5.3 becomes

$$\text{Prob}\,[x_t = e_m | x_{t-1}]$$

$$= \text{Prob}\,[x_t^A = e_{n_A} | x_{t-1}^A, x_t^B] \,.\, \text{Prob}\,[x_t^B = e_{n_B} | x_{t-1}^B] \qquad 5.5$$

$$\text{for } n_A = 1, 2;\ n_B = 1, 2.$$

Initially, a random number would be used to obtain x_t^B, which would then indicate which probability distribution to use in the generation of x_t^A, finally allowing calculation of x_t.

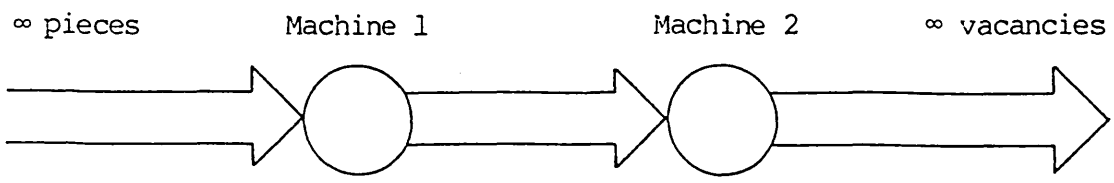
When considering systems of a more realistic size, the breaking down of a complete system description into subsystem models, whether independent or not, is an important feature of experiment design. We could not, for example, consider the direct application of equation 4.10 to a system having several thousand states, but simulation would be made possible by taking account of system structure and identifying a number of subsystems.

Models of physical systems usually divide quite naturally into subsystems, and the points raised by the preceeding discussion may be considered obvious. However, rather less obvious is the way in which a reverse-time system simulation can be reduced to a number of subsystem simulations in reverse-time.

5.2.2 Forward-time simulation of cross-linked transfer-lines

Because the control actions for this example act in such a way that the system is driven from one process state to another at the beginning of each system cycle, the process lends itself quite naturally to description by Riordon's notation. This notation has been discussed generally in Section 2.2.5, and with reference to system simulations in Chapter 4. In the transfer-line example, it can be seen that the space of control states is the same as the space of process states, so that for an s-state chain, the control transition matrix $D(\alpha)$ (for any feasible control parameter, α), and the process transition matrix P, are both s x s matrices. In using this description we have effectively separated the cross-linking feature of the system (represented by $D(\alpha)$) from the operation of the two independent lines (represented by P).

Section 5.2.1 shows that these phases of the system cycle can be simulated separately, and we use a single random number, and the theory of Section 4.7.1 to obtain a degenerate form of $D(\alpha)$. It would then be possible to use another random number in the same way to obtain a degenerate form of the s x s matrix, P, but this would not be an easy task. Most rows of P have 16 non-zero entries, and implementation of equation 4.10 would require an excessive amount of calculation during the simulation. The first, and most obvious simplification, is to recognize that P gives transition probabilities for two independent lines, which can be considered separately (see equation 5.3), effectively transforming one simulation in s-space into two simulations in $\sqrt{s}$-space. Figure 5.1 shows the components of a two-stage transfer-line.



Finite capacity

buffer store.

Fig. 5.1: Two-stage line configuration

The simulation of each line in $\sqrt{s}$-space can be further simplified by recognizing certain system charateristics. We begin by identifying two of the states as "N.I. and F.O. force-down states" according to the following definitions:

(i) A no input (N.I.) force-down state is characterised by an empty buffer, with machine 2 up and machine 1 down. Thus during the next system transition, machine 2, though able to process a piece, cannot do so because the buffer is unable to supply one. Operation of machine 2 is therefore suspended (i.e. it is "forced-down") until machine 1 is repaired and able to supply pieces once again.

(ii) A full output (F.O.) force-down state can be regarded as the dual of a N.I. state and is characterised by a full buffer, with machine 1 up and machine 2 down. Because machine 2 is down, it cannot accept pieces from the buffer for processing with the result that no vacancy is created in the buffer store. Machine 1 therefore cannot pass on its processed pieces and is suspended from operation until machine 2 is repaired.

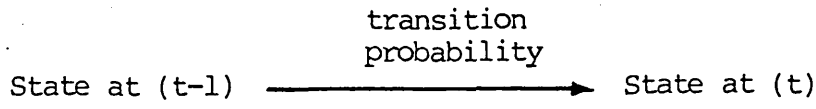
It can be seen that a force-down state implies that one machine becomes dependent upon the other for at least one system transition.

The state transition diagrams of Fig. 5.2 give forward-time simulation rules for transitions from the two force-down states, where the appearance of 1's as factors of the transition probabilities indicate the suspension of operation of machine 1(2) in a F.O.(N.I.) state. Probabilities of machine state transitions are given in terms of the constants B_i and REP_i, which are the breakdown and repair probabilities for machine i, assumed to have been previously identified. For reasons which will become clear later (in Section 6.3.1) we put $\{R_i = (1-REP_i), i = 1,2\}$ and generate transitions from the force-down states by the following method. Generating random numbers, w_i, we take the next state of machine i to be up if $R_i \leq w_i \leq 1$ and down if $0 \leq w_i < R_i$, starting in a F.O.(N.I.) state with i = 2(1).

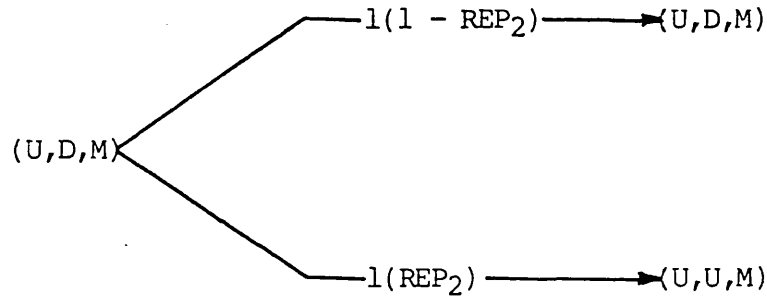
Transitions from all single line states except the force-down states can be simulated by generating independent state transitions for machines 1 and 2 and directly calculating buffer store contents from the machine state histories. Fig. 5.3 gives forward-time simulation rules for transitions from non-force-down states and it is apparent that the change in buffer store contents at t is calculated from a simple function of the machine states at (t-1). This function is defined by Table 5.1. Table 5.2 shows how state transitions for each independent machine are generated.

(X_1,X_2) at (t-1)	Change in buffer store contents at t
U,U	no change
U,D	add 1 piece
D,U	subtract 1 piece
D,D	no change

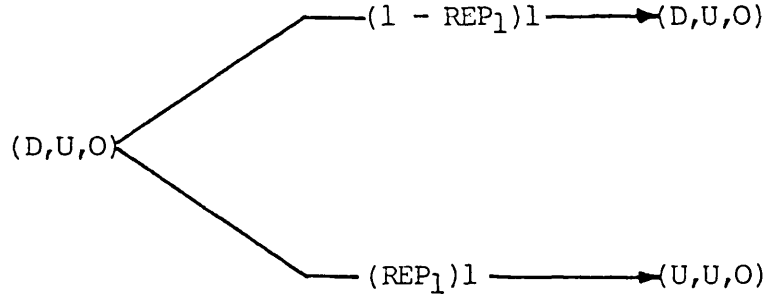
Table 5.1: Calculating buffer store contents as function of machine states at (t-1)



(i) F.O. force-down state



(ii) N.I. force-down state



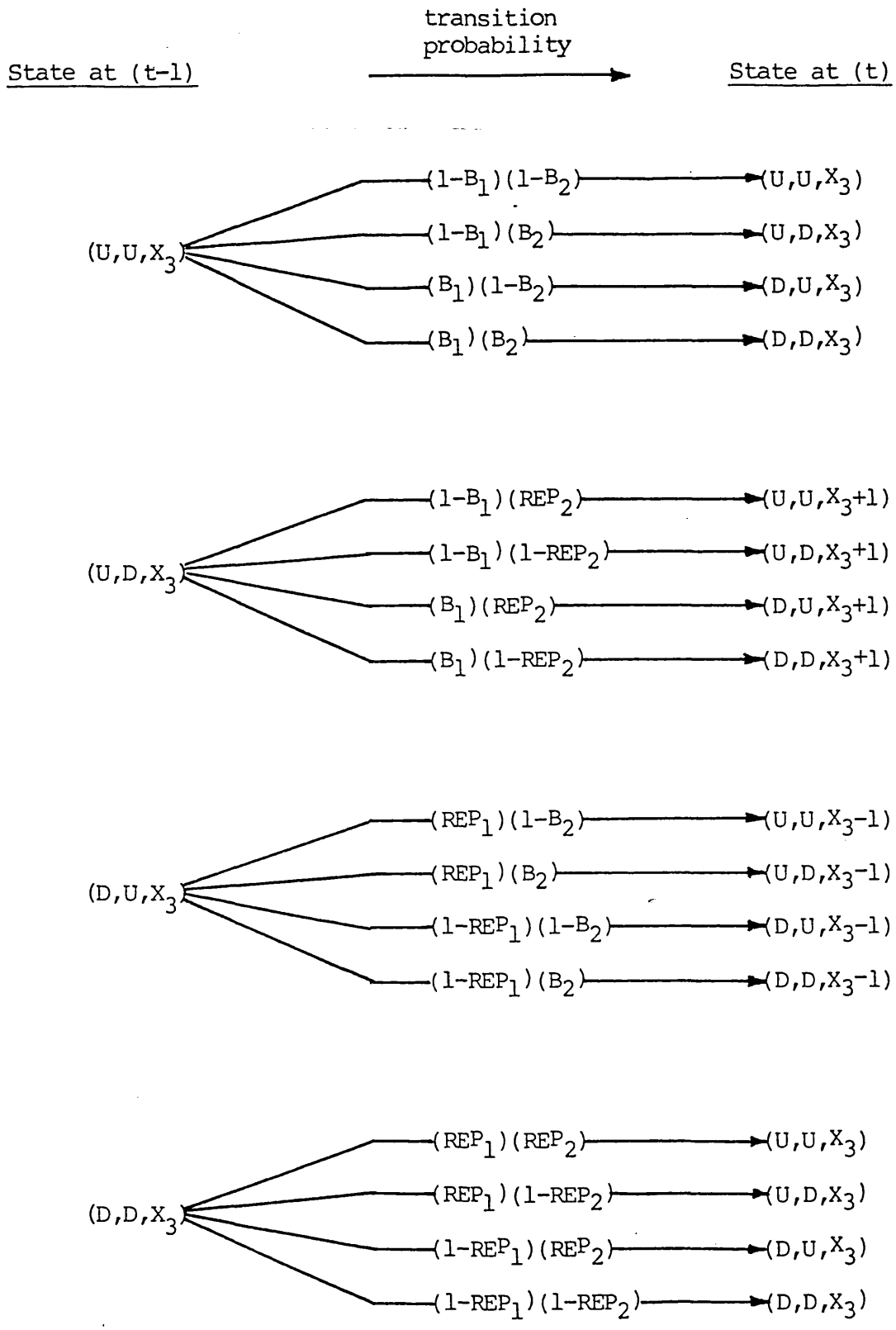
Note: States uniquely identified by (X_1, X_2, X_3) where

$X_1 \in \{\underline{Up}, \underline{Down}\}$ = State of machine 1

$X_2 \in \{\underline{Up}, \underline{Down}\}$ = State of machine 2

$X_3 \in \{0, 1, \ldots, M\}$ = Contents of buffer store

Fig. 5.2: Forward-time simulation rules for a two-stage transfer-line - transitions from force-down states



Note: Notation as in Fig. 5.2

Fig 5.3: Forward-time simulation rules for a two-stage transfer-line - transitions from non-force-down states.

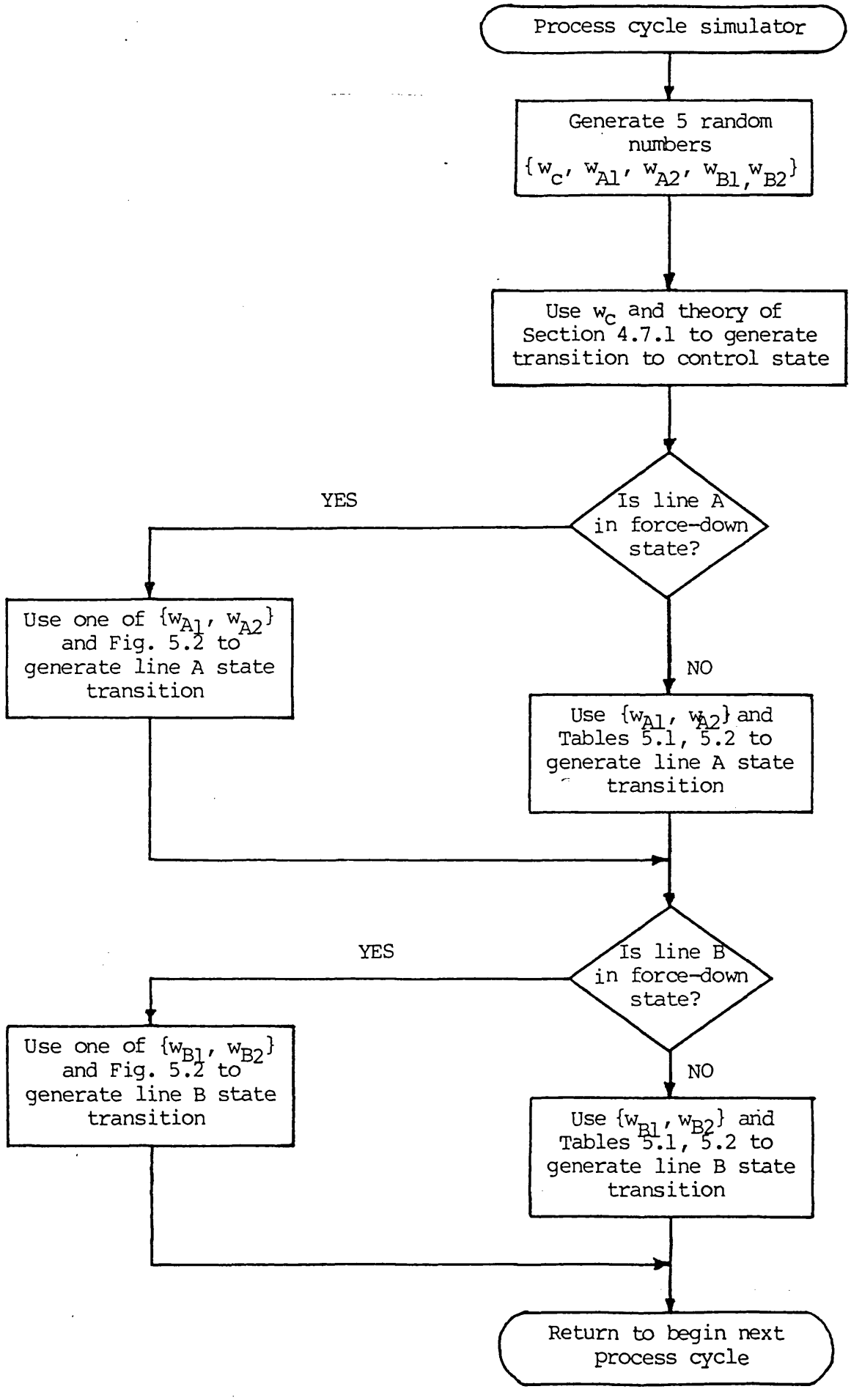


Fig. 5.4: Flow diagram showing fundamental rules for simulating a process cycle in forward-time.

State of machine i at (t-1)	Conditions to be satisfied for state of machine i at t to be:	
	D	U
D	$0 \leq w_i < R_i$	$R_i \leq w_i \leq 1$
U	$0 \leq w_i < B_i$	$B_i \leq w_i \leq 1$

Table 5.2: Simulating independent state transitions for machine i.

Note that two random numbers, w_1 and w_2, are required to generate a state transition for each two-stage line operating independently. Two numbers are generated, but only one is required for each transition from a force-down state. This procedure is necessary to synchronize simulations using common random numbers - see Kleijnen (20) and Section 6.2.3. We therefore require five random numbers in order to simulate each process cycle; one for the transition to a control state and two for state transitions of each two-stage line. Fig. 5.4 is a flow diagram which summarizes the forward-time simulation method for a single process cycle. Extra computation results from straightforward application of the theory of Chapter 4 to obtain forward-time estimates of state distribution vectors, adjoint vectors and gradients.

5.2.3 Reverse-time simulation of cross-linked transfer-lines by Method 1

Generating a degenerate estimate of the cost vector c

We see from Section 4.8.3 that in order to calculate the adjoint vector sequence, we require degenerate estimates of the vectors $\{P(\alpha)c, P^2(\alpha)c, \ldots P^n(\alpha)c\}$ and that we begin the simulation by reducing c to a degenerate form.

The simple structure of c for the cross-linked transfer-line example makes its reduction to a degenerate form an easy task. The system scores '1' (signifying one production unit processed by upline and downline machines) for each downline machine ending a process cycle in an up state. Thus elements of c can be 0, 1 or 2 depending whether neither downline machine is up, only one downline machine is up, or both downline machines are up, at the end of the process cycle. Using the theory of Section 4.8.1 to reduce c to a degenerate form is rather laborious, and due to the fact that each element of c is an integer, the task can be more easily performed by sampling from uniform distributions of random integers. The smallest cross-linked transfer-line model has a 64-element c vector and it is difficult to illustrate this sampling technique for such a large vector. However, it should be possible to convey the general idea by considering a hypothetical c of similar form.

If $c = [0\ 2\ 0\ 0\ 1\ 1\ 0\ 0]^T$, then we begin by calculating the weighting factor, $z_c = 2 + 1 + 1 = 4$. We then sample from a uniform distribution of random integers $\{1, 2, \ldots, z_c\}$ to obtain an integer, n, and define a degenerate form of c by

$$y_c = [z_c] \cdot q_c \qquad 5.6$$

where $q_c = e_2$, if n = 1 or 2

$= e_5$, if n = 3

$= e_6$, if n = 4

Thus Prob $[q_c = e_2] = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$, Prob $[q_c = e_5] = \frac{1}{4}$, and Prob $[q_c = e_6] = \frac{1}{4}$,

$$\text{giving } E\{y_c\} = \begin{bmatrix} 0 \\ \frac{1}{2} \\ 0 \\ 0 \\ \frac{1}{4} \\ \frac{1}{4} \\ 0 \\ 0 \end{bmatrix} \times 4 = \begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \text{ as required.}$$

For c vectors of a realistic size, the 'decoding' of q_c from n turns out to be an uncomplicated procedure, and q_c is identified by four machine states and two buffer store values which effectively define i, when $q_c = e_i$. Decoding into this form is convenient for the next phase of the simulation process.

Simulation of a process cycle

Having derived y_c, we proceed to obtain a denerate estimate of the vector $P(\alpha).c$. This can be done in principle by pre-multiplying Y_c by a degenerate column matrix form of $P(\alpha)$, obtained through application of the theory given in Section 4.8.2. However, by using Riordon's notation to re-write $P(\alpha)c\rangle$ as $D(\alpha)P.c\rangle$, we see that the simulation is simplified by splitting the cycle into control transition and process transition phases. For simulation purposes, these phases can be considered independently, and two random numbers could be used to reduce matrices $D(\alpha)$ and P to degenerate forms. We can further simplify the problem by recalling that P gives transition probabilities for two independent subsystems. It is reasonable to assume that it should be possible to perform independent reverse-time simulations of those subsystems which were simulated independently in forward-time. It turns out that it is

indeed possible to do this and that forward and reverse-time simulations are dual processes.

We use a small example to illustrate reverse-time simulation of two identical subsystems A and B. Suppose A and B are two-state systems with transition probabilities given by a 2 x 2 matrix P_{AB}, then the transition matrix for the complete system is a 4 x 4 matrix P:

$$\text{If } P_{AB} = \begin{bmatrix} .75 & .25 \\ .1 & .9 \end{bmatrix}$$

then

$$P = \begin{bmatrix} .75 \times \begin{bmatrix} .75 & .25 \\ .1 & .9 \end{bmatrix} & .25 \times \begin{bmatrix} .75 & .25 \\ .1 & .9 \end{bmatrix} \\ .1 \times \begin{bmatrix} .75 & .25 \\ .1 & .9 \end{bmatrix} & .9 \times \begin{bmatrix} .75 & .25 \\ .1 & .9 \end{bmatrix} \end{bmatrix}$$

Let states of the subsystems A and B be denoted by $[X_A, X_B, \epsilon \{1,2\}]$, so that (X_A, X_B) uniquely identifies states of the complete system. Also assume that the degenerate form of c, $y_C = [z_C].[0\ 1\ 0\ 0]^T$ or defined in terms of the subsystem states, $[z_C, (X_A, X_B)] = [z_C, (1,2)]$. In simulating the system transition in reverse-time, we are required to find a degenerate form of the vector $P.y_C$. Using the theory of Section 4.8.2, we calculate the transition weighting factor,

$$z = \sum_{i=1}^{4} [P]_{i2} = .75(.25+.9) + .1(.25+.9)$$

$$= (.75+.1)(.25+.9)$$

and then, using a random number, sample from the probability vector

$$q = \begin{bmatrix} (.75 \times .25)/z \\ (.75 \times .9)/z \\ (.1 \times .25)/z \\ (.1 \times .9)/z \end{bmatrix}$$

Alternatively, it can be seen that by considering subsystems A and B independently and defining transition weighting factors for each,

$$\text{i.e.} \quad z_A \triangleq \sum_{i=1}^{2} [P_{AB}]_{i1}, \quad z_B \triangleq \sum_{i=1}^{2} [P_{AB}]_{i2}$$

we can use two random numbers to sample from probability vectors

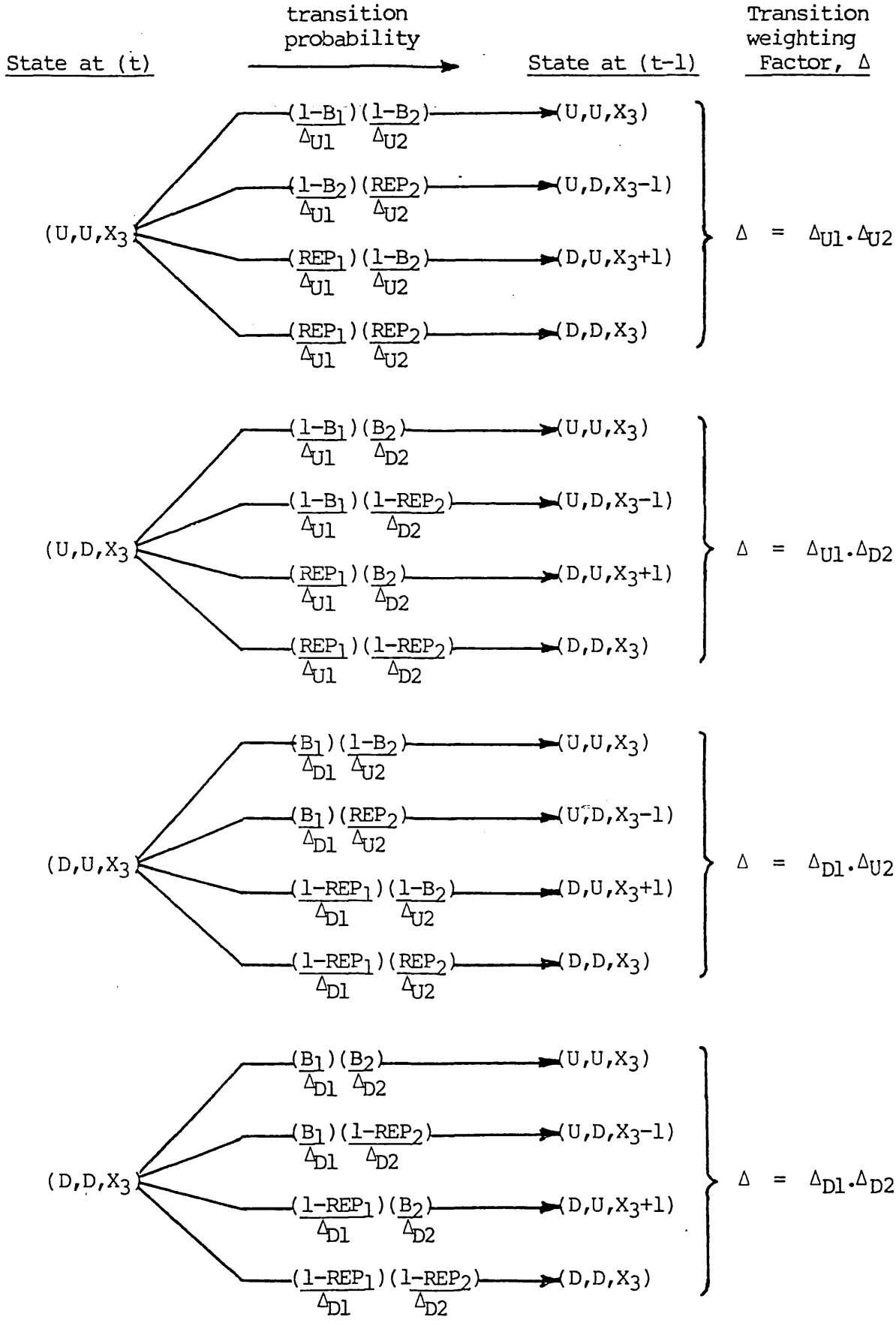
$$q_A = \begin{bmatrix} .75/z_A \\ .1/z_A \end{bmatrix}, \quad \text{and } q_B = \begin{bmatrix} .25/z_B \\ .9/z_B \end{bmatrix}$$

for each subsystem, and the overall effect is the same, e.g. probability of transition to (2,2) is

$$\begin{bmatrix} \frac{.1}{z_A} \times \frac{.9}{z_b} \end{bmatrix} = \begin{bmatrix} \frac{.1 \times .9}{z} \end{bmatrix}$$

with overall transition weight $z_A.z_B = z$, as required.

Considering the transfer-line example once again, it is possible to reduce (in the sense of reducing program complexity) the process transition phase to independent line simulations, and also to reduce two independent line simulations to four machine simulations, calculating buffer store contents as a function of machine states at the end of each transition.



Note: $\Delta_{Di} \triangleq B_i + (1-REP_i)$ for $i \in \{1,2\}$

$\Delta_{Ui} \triangleq REP_i + (1-B_i)$ for $i \in \{1,2\}$

other notation as in Fig. 5.2

Fig. 5.5: Reverse-time simulation rules for a two-stage transfer-line - transitions from states with $0 < X_3 < M$.

Comparing Figs. 5.3 and 5.5, we detect similarities between forward- and reverse-time simulation rules. The transition probabilities shown in Fig. 5.5 are factored so that transition probabilities applying to each subsystem (machine) become apparent.

Tables 5.3 and 5.4 (equivalent to Tables 5.1 and 5.2 for forward-time simulation) can be drawn up for reverse-time transitions from states with $0 < X_3 < M$:

(X_1,X_2) at t	Change in buffer store contents at t
U,U	no change
U,D	subtract 1 piece
D,U	add 1 piece
D,D	no change

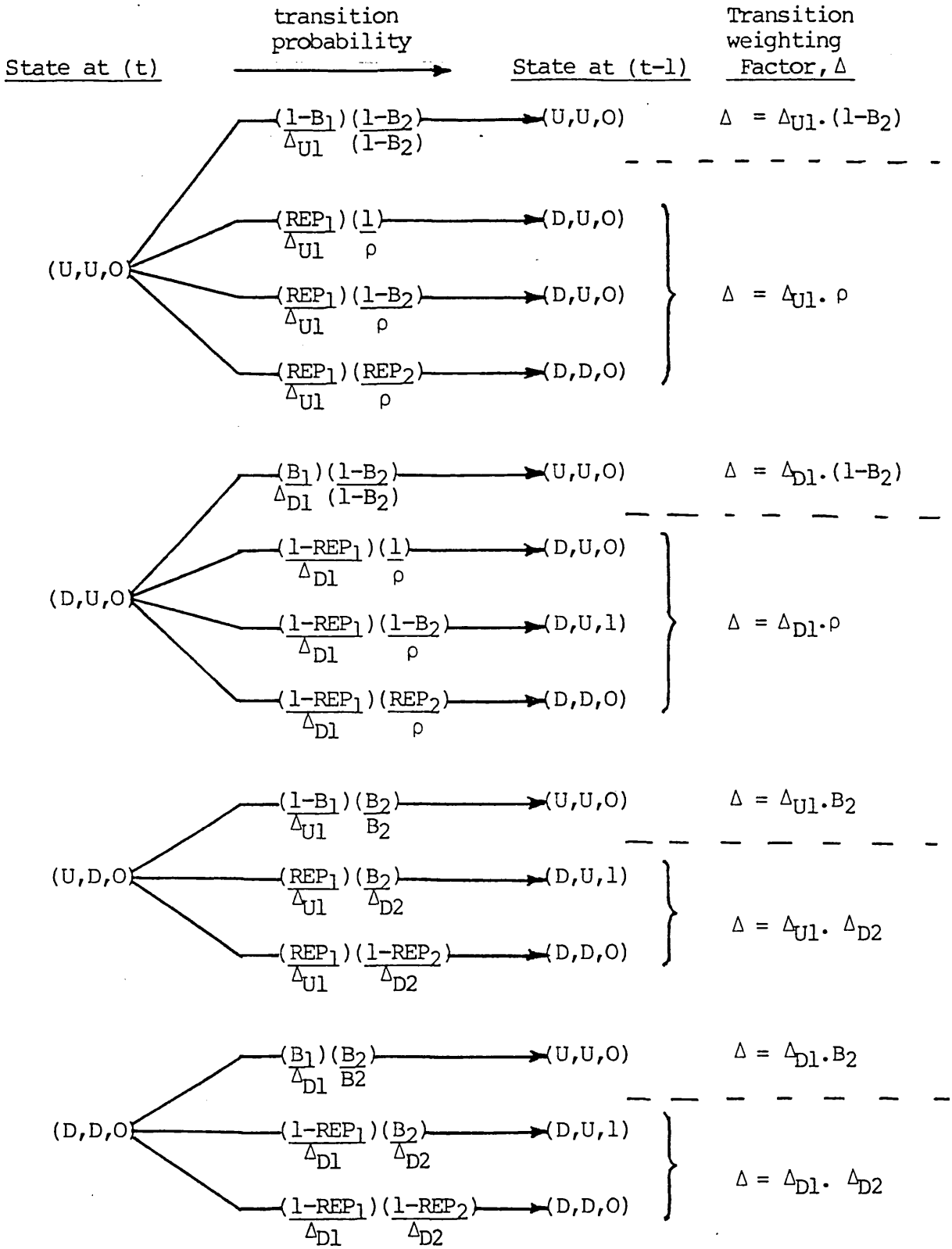
Table 5.3: Calculating buffer store contents as function of machine states at t.

State of machine i at t	Transition weighting factor	Conditions to be satisfied for state of machine i at (t-1) to be:	
		D	U
D	Δ_{Di}	$0 \leq v_i < \bar{B}_i$	$\bar{B}_i \leq v_i \leq 1$
U	Δ_{Ui}	$0 \leq v_i < \bar{R}_i$	$\bar{R}_i \leq v_i \leq 1$

where $\left(\frac{REP_i}{\Delta_{Ui}}\right) \triangleq \bar{R}_i$, and $\left(1 - \frac{B_i}{\Delta_{Di}}\right) \triangleq \bar{B}_i$

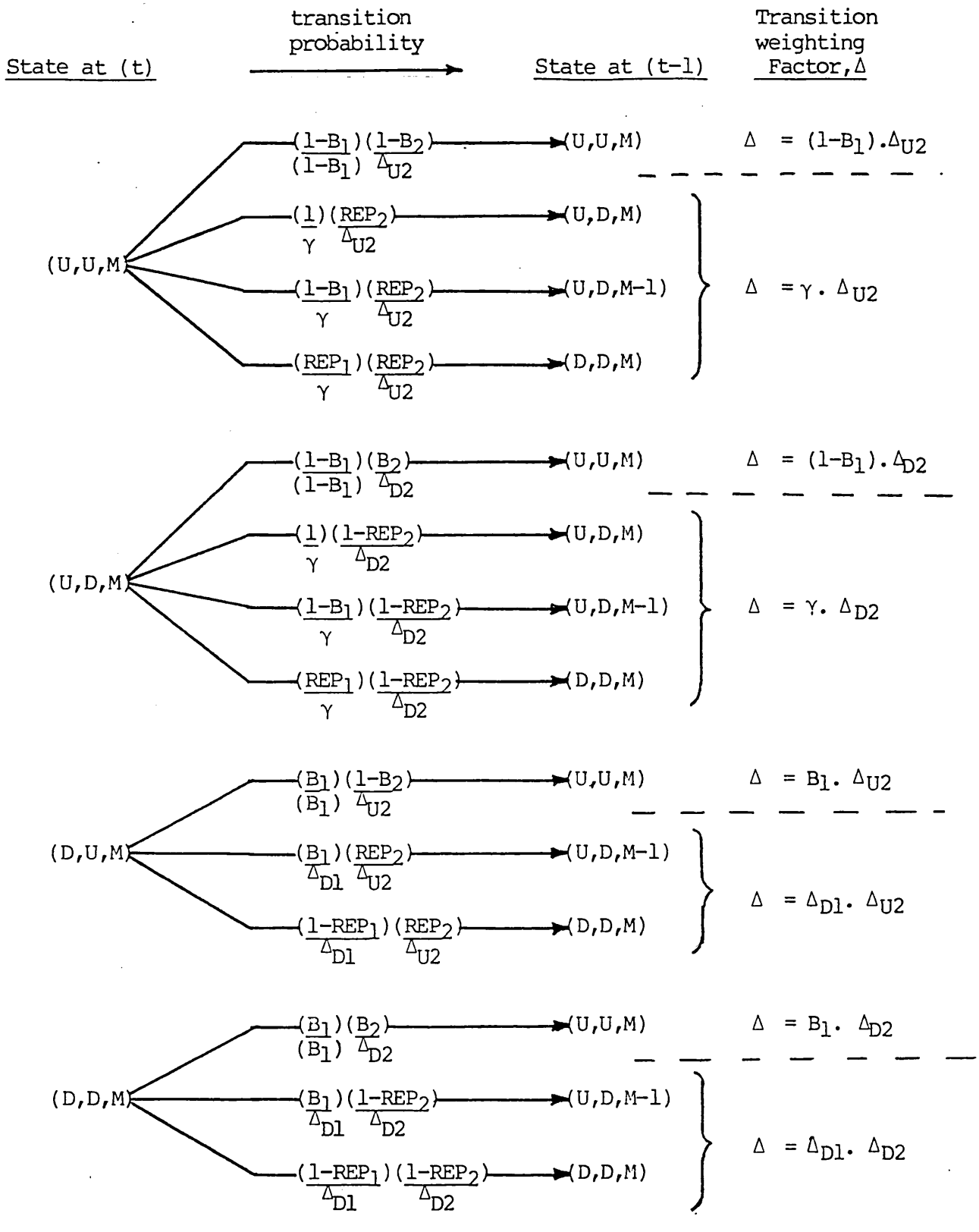
Rules for simulating transitions from states where the buffer store is full or empty are given in Figs. 5.6 and 5.7 and are seen to be more complicated for reverse-time than for forward-time simulations.

Table 5.4: Simulating independent state transitions for machine i in reverse-time



Note: notation as in Fig. 5.5
with $\rho \triangleq 1 + REP_2 + (1-B_2)$

Fig. 5.6: Reverse-time simulation rules for a two-stage transfer-line - transitions from states with $X_3 = 0$.



Note: Notation as in Fig. 5.5
with $\gamma = 1 + REP_1 + (1-B_1)$

Fig. 5.7: Reverse-time simulation rules for a two-stage transfer-line - transitions from states with $X_3 = M$.

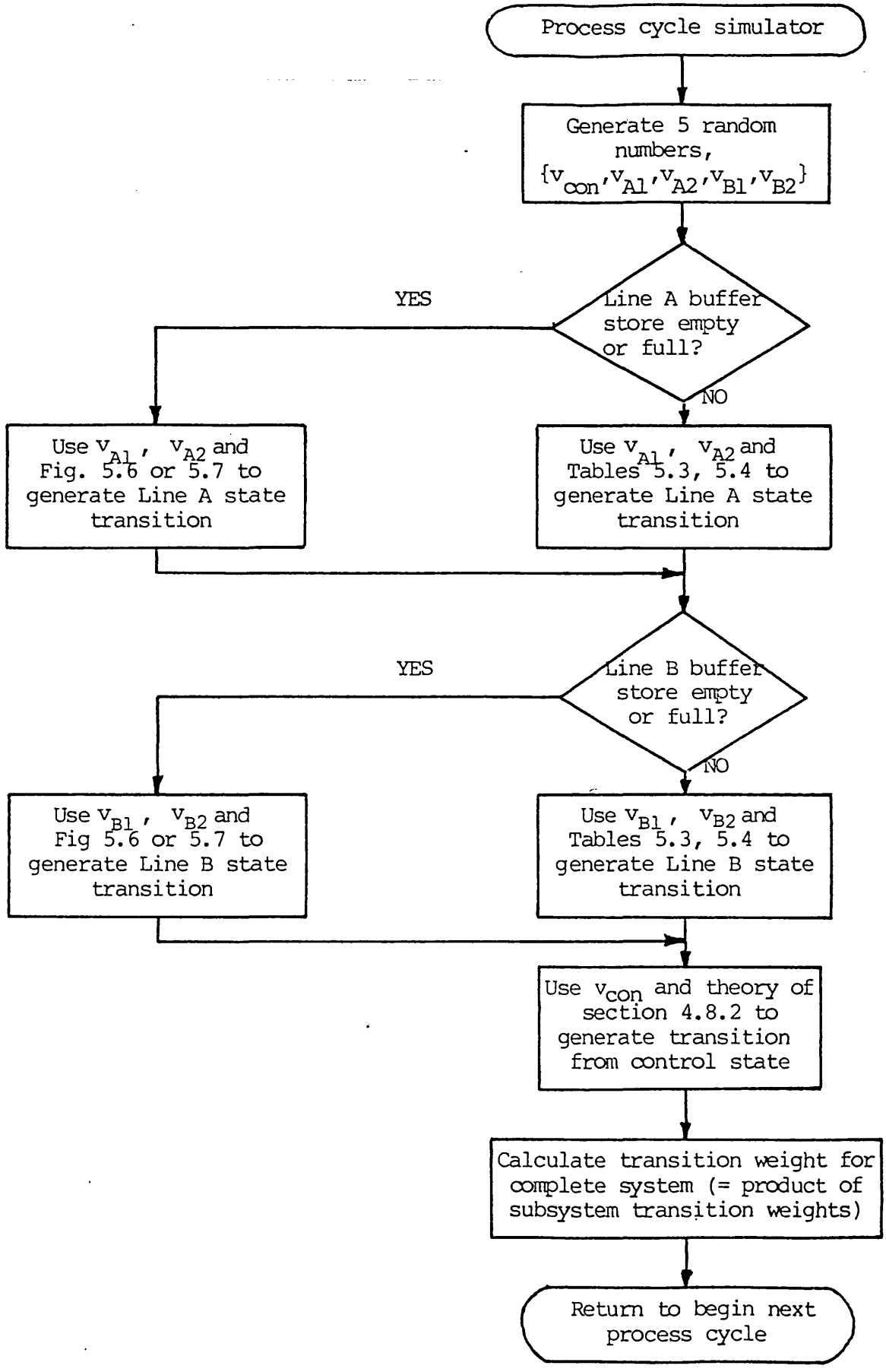


Fig. 5.8: Flow diagram showing fundamental rules for simulating a process cycle in reverse-time.

Under these conditions, subsystem transitions are generated according to conditional probabilities as in equation 5.5, adjusting the weighting factors accordingly.

From Fig. 5.2, we note that if the buffer store is empty, then the state of machine 1 at (t-1) influences the operation of machine 2 during the next transition (i.e. machine 1 in a down state can force down machine 2). The dual nature of reverse-time simulation implies that it is the state of machine 1 at the end of the transition which influences the operation of machine 2. It follows that one random number, v_1, is used to generate a transition of machine 1 according to the rules of Table 5.4, using an appropriate weighting factor. Conditional on the outcome of this simulated transition, a transition of machine 2 is generated using another random number, v_2, the transition probabilities shown in Fig. 5.6, and the theory of Section 4.8.2. It can be seen that, depending on the simulated behaviour of machine 1, machine 2 might make a certain (probability = 1) transition, a transition according to Table 5.4, or a transition to one of a set of states including the N.I. force-down state.

In contrast, Fig. 5.2 indicates that when the buffer store is full, the state of machine 2 at (t-1) influences the operation of machine 1 during the next transition. A dual version of Fig. 5.6 therefore exists for reverse-time transitions from states with $X_3 = M$. Fig. 5.7 summarizes the simulation rules for this case, and it is clearly necessary that the state transition for machine 2 should be generated before the state transition for machine 1. Fig. 5.8 is the equivalent version of Fig. 5.4 for simulation of a process cycle in reverse-time.

5.2.4 Reverse-time simulation of cross-linked transfer-lines by Method 2

A detailed analysis of this method should not be necessary as the simulation rules are a combination of those given in Sections 5.2.2, 5.2.3 and 4.8.2. Again we split the process cycle into two phases, sequentially generating transitions of the two independent lines, and the transition from control state to process state. Independent line transitions can be illustrated by considering possible reverse-time transitions from state (D,U,X_3). Fig. 5.9 shows states which can be reached if both of the indicated conditions are satisfied for random numbers w_1 and w_2.

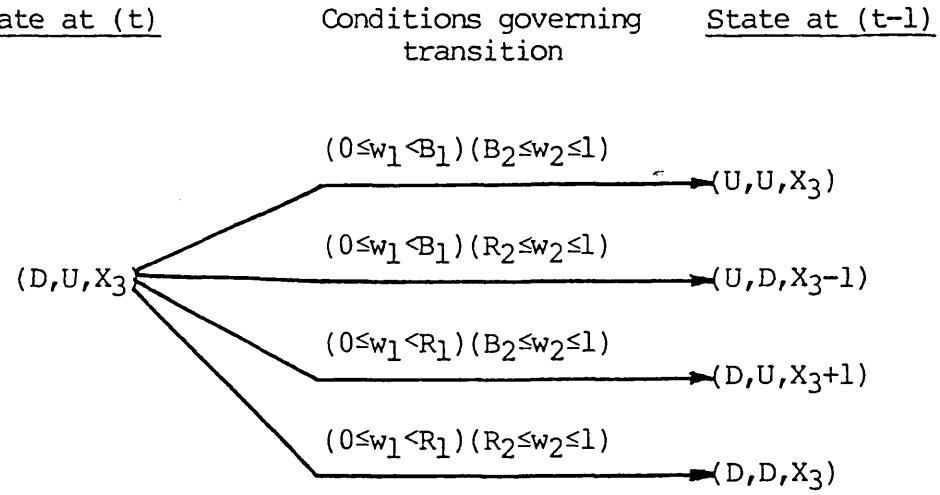


Fig. 5.9: Conditions governing transitions from (D,U,X_3), for $0 < X_3 < M$.

It is possible that none of the conditions are satisfied by w_1 and w_2, indicating that the transition weight is zero (the state trajectory has effectively been terminated). Alternatively, conditions may be

satisfied for transitions to more than one state, forcing the use of another random number and the theory of Section 4.8.2.

5.3 Gradient estimates for the standard transfer-line examples

In order to compare the efficiency of each of the gradient estimators when applied to an O.R. problem, we take as an example the standard 64-state cross-linked transfer-line problem of Appendix I and estimate the gradient for a nominal control parameter, $\bar{\alpha} = 0$, using four gradient estimators. The accurate value for this gradient has been calculated using equation 3.62, and is -0.01772. The four gradient estimators each use the same method for estimating the π-sequence, but obtain adjoint estimates using Methods 1, 2, 3b and 3c, hence the labelling of gradient estimation programs as G1, G2, G3b and G3c. Program G3b is included to show the loss of accuracy which results from using weighting methods to reduce the work-load presented to the computer. In this case, the computer generates the main state trajectory and no more than one additional state transition per process cycle, using weighting methods to avoid biased estimates.

Results from gradient estimation routines using reverse-time simulations are given in Tables 5.5 and 5.6. Comparing the variances (with an adjustment for differing computing times) of gradient estimates obtained from 50 runs of 1000 cycles for each program, we find that program G2 reduces variance by 98% (by a factor of 75) with respect to program G1.

Two forward-time estimators, implemented as programs G3b and G3c can be compared using Tables 5.7 and 5.8. It can be seen that it is worthwhile avoiding the use of weighting methods in forward-time adjoint estimation if at all possible. For example, the variance of results

No. of runs	Length of simulation (cycles)	Gradient estimate	Estimated variance for a single run	Estimated S.D.	Computing time for experiment (secs)
500	100	+1.697	2748.060	52.420	62.46
10	1000	-0.273	73.815	8.592	46.33
20	1000	-1.525			94.98
50	1000	-0.547			224.15

Table 5.5: Experimental results from program G1 - time averaged gradient estimates using adjoint estimation method 1.

No. of runs	Length of simulation (cycles)	Gradient estimate	Estimated variance for a single run	Estimated S.D.	Computing time for experiment (secs)
500	100	-6.608	15212.58	123.34	66.49
10	1000	-1.142	3.339	1.827	15.38
20	1000	-0.766			28.20
50	1000	-0.444			65.99

Table 5.6: Experimental results from program G2 - time averaged gradient estimates using adjoint estimation method 2.

No. of runs	Length of simulation (cycles)	Gradient estimate	Estimated variance for a single run	Estimated S.D.	Computing time for experiment (secs)
500	100	+0.0674	2.2276	1.4925	40.41
10	1000	+1.0248	1.3832	1.1761	9.82
50	1000	+0.1958			43.54
5	10000	-0.0181	0.0003	0.0180	44.48

Table 5.7: Experimental results using program G3b - time averaged gradient estimates using adjoint estimation method 3b.

No. of runs	Length of simulation (cycles)	Gradient estimate	Estimated variance for a single run	Estimated S.D.	Computing time for experiment (secs)
500	100	-0.0208	0.0136	0.1168	86.56
10	1000	+0.0065	0.0019	0.0437	19.39
50	1000	+0.0082			89.80
5	10000	-0.0101	0.0001	0.0113	88.77

Table 5.8: Experimental results using program G3c - time averaged gradient estimates using adjoint estimation method 3c.

from program G3b can be reduced by 98.7% (for 100 cycle run) or 99.7% (for 1000 cycle run) by avoiding weighting methods. These figures have been adjusted to allow for the extra computing time required for program G3c. Though these figures indicate significant improvements made possible by avoiding weighting methods in this case, it should be appreciated that we have assumed extreme restrictions on computing resources for program G3b in order to illustrate the estimators.

Finally, we note that programs G3b and G3c are more efficient gradient estimators than programs G1 and G2. Comparing results from programs G2 and G3c for 50 runs of 1000 cycles, the variance reduction is found to be about 99.92% (reduction by a factor of 1284).

5.4 Solution of Markov decision problems using cost gradient estimates

5.4.1 Discussion

Consider an unconstrained function minimization problem where a system has an adjustable parameter, x, and a scalar cost function taking the value f(x) at x. Assume that the law taking x into f(x) is known in principle, but is so complicated that it is virtually impossible to compute. The problem is to find some recursive numerical procedure which sequentially yields or approximates a value of x which minimizes f(.).

One possible approach might be to generate a "sufficiently large" ensemble of system simulations using a nominal value of x, and to average the system responses to provide a good estimate of f(x). This estimate could then be used in a non-linear programming algorithm which would generate a sequence $\{x_n\}$, where x_n = n'th estimate of the optimal

value of x. Farah (53) has developed earlier work by Mayne (6) on algorithms utilizing estimated system responses. A feature of these algorithms is that the standard deviation of δx_n (= $x_n - x_{n-1}$, for each n) is arranged to be small compared with δx_n, by adjusting the size of the ensemble used to estimate $f(x_n)$. This method is computationally expensive because the number of simulations in the ensemble increases near the optimal point, where δx_n becomes small. It is also inefficient because f(.) must be estimated at many parameter values which are not near the optimal point. It makes sense to use the information from earlier experiments to select better values of x in later experiments, which is what stochastic approximation algorithms do. Though this characteristic of stochastic approximation algorithms makes them more efficient in one sense, they suffer from two significant drawbacks. Firstly, the step-length (and thus the magnitude of δx_n) at each iteration is decided to a great extent by an algorithm parameter which is chosen at the experiment design stage. Choosing this parameter unwisely can lead to slow convergence, which is demonstrated later in the chapter. A further drawback is that definitions of stochastic approximation algorithms given in the literature do not include a stop rule. The algorithms are shown to converge assymptotically (i.e. after an infinite number of iterations) but the literature fails to define a suitable test which can be used to indicate that a small neighbourhood of the solution point has been reached after a finite number of iterations. One possibility is to perform some measure of variance analysis during each simulation which can be used to indicate when a stationary point has been reached. In Sections 5.4.2 and 5.4.3 we are able to define a stop rule using knowledge of estimate variance obtained from previous experiments.

Convergence properties of stochastic approximation algorithms are discused in detail by Kushner and Clark (54). They describe (p.189) a

projection algorithm which is a form of the Kiefer-Wolfowitz function minimization procedure employing a finite difference approximation to the gradient of the cost function with respect the parameters. The algorithm solves problems with inequality constraints, and is therefore suitable for solving the Markov decision problems formulated in this thesis. It produces a sequence $\{x_n\}$, where $x_n \in G = \{ x : q_i(x) \leq 0, i = 1, \ldots, k \}$ for all n. It is proved that the sequence $\{x_n\}$ converges (w.p.1) to a point satisfying the Kuhn-Tucker necessary condition for a constrained minimum.

The projection algorithm is defined by equation 5.7.

Algorithm

$$\left.\begin{aligned} \tilde{x}_{n+1} &= x_n + a_n[-f_x(x_n) + \xi_n + \beta_n] \\ x_{n+1} &= \pi_G(\tilde{x}_{n+1}) \end{aligned}\right\} \qquad 5.7$$

where (i) $\pi_G(x)$ is defined as the nearest point on G to x. Thus $\pi_G(x) = x$, if $x \in G$.

(ii) β_n is the error due to approximating the derivative $f_x(.)$ by a finite difference approximation.

(iii) ξ_n is the observation noise.

and (iv) $\{a_n\}$ is a sequence of positive numbers, tending to zero, such that $\sum_n a_n = \infty$.

As we intend to estimate cost gradients using the theory of Chapter 4, we see that $\beta_n = 0$. Also, sequences satisfying the given conditions for $\{a_n\}$ are $\{a/n^p\}$ where $0<p\leq 1$ and a is a constant (see Rudin (60,p.62)), so we set $a_n = a/n$. If a single simulation provides an estimate of the cost gradient $h(x_n)$, then the algorithm can be re-written,

$$\left.\begin{aligned} \tilde{x}_{n+1} &= x_n + \frac{a}{n}[-h(x_n)] \\ x_{n+1} &= \pi_G(\tilde{x}_{n+1}) \end{aligned}\right\} \qquad 5.8$$

The properties of $\{a_n\}$ ensure that as x_n approaches the optimal point, δx_n tends to 0, even though $h(x_n)$, being random, is nonzero.

The following sections give results obtained by applying the algorithm (equations 5.8) to the standard 64-state transfer-line problem of Appendix I (having one control parameter), and to the two-state problem of Section 3.5.3 (having two control parameters). In both cases gradient estimates were obtained using program G3c.

5.4.2 Solution of the 64-state transfer-line problem

This problem is formulated as a function maximization problem where we wish to find the control parameter value which maximizes the mean system output (number of pieces from the two lines) per cycle of operation. Because the size of the Markov model is small enough to allow direct calculation of the performance function, we begin by considering the graph of performance function vs. α, shown in Fig. 5.10. The graph indicates that output is maximized at $\alpha = 0.0$, i.e. when the lines are operating independently. This contrasts with the results of an earlier study (see Appendix I) where it was shown that increased output could be obtained through the use of cross-linking if independent

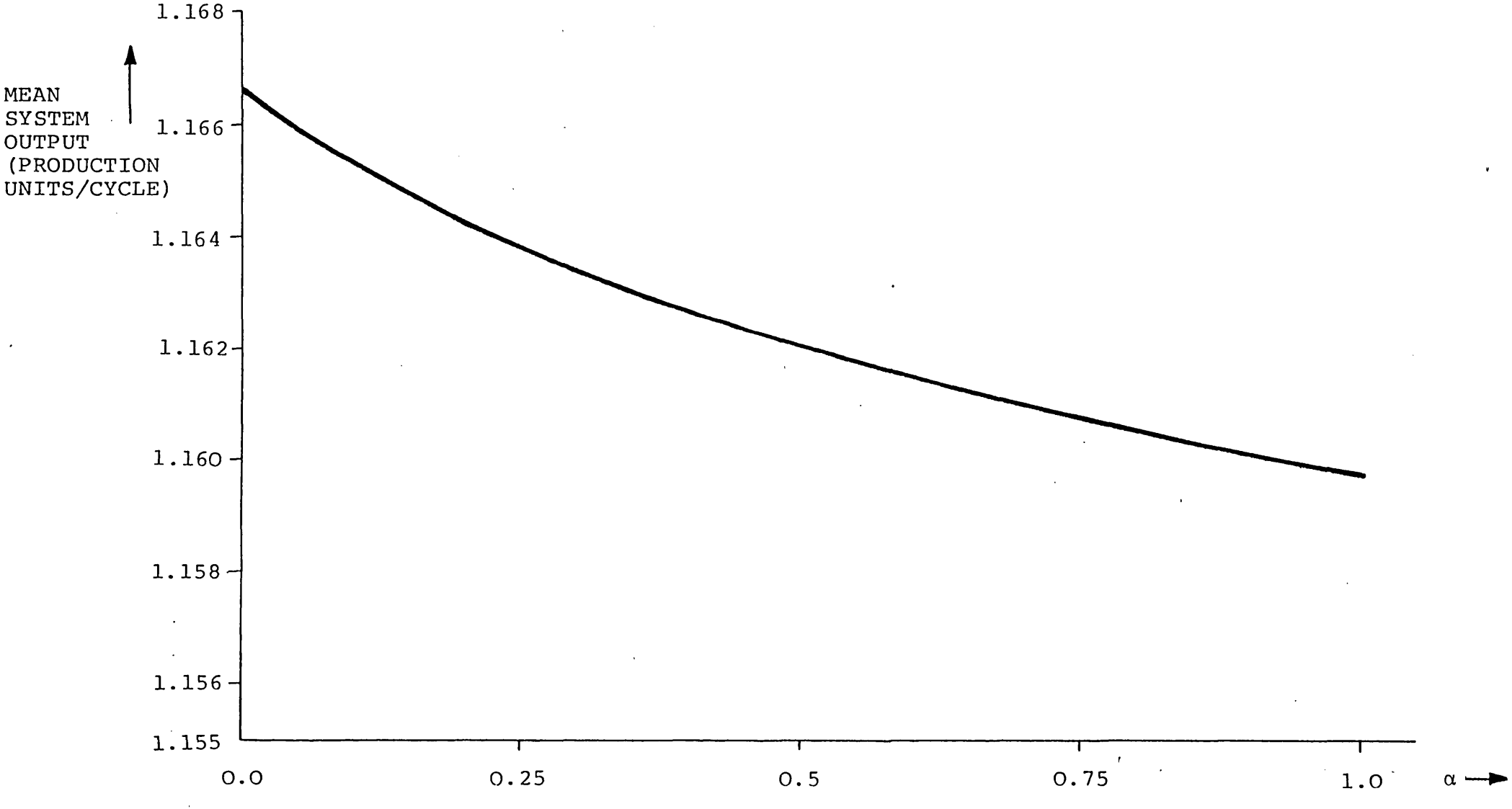


FIG. 5.10: Performance characteristic for standard 64-state transfer-line example

control actions are allowed for each of the 64 states. We nevertheless continue to use the control problem of Appendix I as an illustration throughout the rest of the thesis, noting meanwhile that care should be taken when specifying the structure of grouped-state controllers.

In order to solve the problem using the projection algorithm, we must replace $-h(x_n)$ by $h(x_n)$ in equations 5.8, and define $\pi_G(x)$ by

$$\left.\begin{aligned} \pi_G(x) &= 0.0, && \text{if } x < 0.0 \\ &= x && \text{if } x \in [0.0, 1.0] \\ &= 1.0, && \text{if } x > 1.0 \end{aligned}\right\} \qquad 5.9$$

We are now left with a choice of algorithm parameters:

(i) the length of each simulation used to estimate $h(x_n)$ - only one simulation run is used per iteration, but the accuracy of the estimates is controlled by adjusting the run-length,

(ii) the value of 'a',

and (iii) the initial parameter value, x_1.

Some comments will be made in Section 5.4.4 on the selection of these parameters, so for now we consider the results obtained using a range of parameter settings, with full details in Tables 5.9, 5.10, 5.11 and 5.12. From previous experiments using program G3c, we know that the cost gradient at the optimal point is -0.01772, with standard deviations of 0.04 and 0.10 for 1000-cycle and 100-cycle estimates, respectively. Thus with 95% certainty we would expect an estimate $\in$ [-.098, +.06] from a 1000-cycle simulation, and an estimate $\in$ [-.218, +.18] from a 100-cycle simulation. Thus for an algorithm using 1000-cycle estimates, where $a_n \simeq 1$ and $x_n \simeq 0.0$ for some value of n, we see from equation 5.8 that δx_{n+1} $(=x_{n+1}-x_n)$ will be small and we can take x_n as the optimal

Iteration number n	System parameter x_n
1	0.5
20	0.0
40	0.548
60	0.388
80	0.281
100	0.263
120	0.0
140	0.457
160	0.389
180	0.406
200	0.390
220	0.238
240	0.297
260	0.237
280	0.072
300	0.180
320	0.168
340	0.048
360	0.080
380	0.067
400	0.012
420	0.021
440	0.0
460	0.095
480	0.061
500	0.0

Algorithm Parameters

(i) Simulation length: 100 cycles
(ii) a = 100
(iii) $x_1 = 0.5$

Optimal point, $x_\infty = 0.0$

Table 5.9: 500 iterations of projection algorithm solving 64-state problem.

Iteration number n	System parameter x_n
1	1.0
2	0.957
4	0.798
6	0.686
8	0.540
10	0.445
12	0.404
14	0.344
16	0.327
18	0.321
20	0.269
22	0.378
24	0.388
26	0.348
28	0.373
30	0.358
32	0.368
34	0.373
36	0.350
38	0.345
40	0.332
42	0.306
44	0.290
46	0.250
48	0.199
50	0.197

Algorithm Parameters

(i) Simulation length: 1000 cycles
(ii) a = 50
(iii) $x_1 = 1.0$

Optimal point, $x_\infty = 0.0$

Table 5.10: 50 iterations of projection algorithm solving 64-state problem.

Iteration number n	System parameter x_n
1	1.0
2	0.936
4	0.661
6	0.571
8	0.384
10	0.336
12	0.265
14	0.0
16	0.0
18	0.219
20	0.122
22	0.105
24	0.063
26	0.0
28	0.037
30	0.0
32	0.053
34	0.010
36	0.0
38	0.0
40	0.0
42	0.014
44	0.0
46	0.097
48	0.134
50	0.123

Algorithm Parameters

(i) Simulation length: 1000 cycles
(ii) a = 75
(iii) $x_1 = 1.0$

Optimal point, $x_\infty = 0.0$

Table 5.11: 50 iterations of projection algorithm solving 64-state problem.

Iteration number n	System parameter x_n
1	1.0
4	0.631
8	0.349
12	0.410
16	0.357
20	0.064
24	0.160
28	0.0
32	0.373
36	0.359
40	0.313
44	0.298
48	0.124
52	0.017
56	0.029
60	0.0
64	0.0
68	0.059
72	0.0
76	0.119
80	0.141
84	0.065
88	0.0
92	0.0
96	0.035
100	0.019

Algorithm Parameters

(i) Simulation length: 1000 cycles
(ii) a = 100
(iii) x_1 = 1.0

Optimal point, x_∞ = 0.0

Table 5.12: 100 iterations of projection algorithm solving 64-state problem.

parameter value. For an algorithm using 100-cycle estimates, n should be large enough to ensure than $a_n \simeq 0.3$ in order to apply the same criterion; hence the impressive results of Table 5.9, where for n = 500, $a_n = 0.2$.

5.4.3 Solution of a problem having two control parameters

Section 3.5.3 gives details of (and solves, by an analytic method) a two-state Markov decision problem. Its solution by stochastic approximation algorithm is of interest because two control parameters are involved, and the complications of projecting onto the feasible control set become apparent. Furthermore, the optimal controls (not a single point, but a set of points in this case) can fall within the interior of the feasible region, unlike the problem of the last section.

We are required to choose controls in order to minimize a cost function, allowing direct application of equations 5.8. It is useful to consider how the definition of $\pi_G(x)$ as the "nearest point to x in the feasible set" is calculated in practice. We use Fig. 5.11 to show how two points, $\tilde{x}_{n+1}$ and $\tilde{y}_{n+1}$ are projected onto the feasible region formed by the interior of a triangle.

Having detected that $\tilde{x}_{n+1}$ is not a feasible point, we know that at least one constraint will be active at $\pi_G(\tilde{x}_{n+1})$, so we calculate the distances a, b and c, and because a is the shortest distance, set $\pi_G(\tilde{x}_{n+1}) = x_{n+1}$ as shown. The projection of $\tilde{y}_{n+1}$ is more difficult. In this case we calculate and compare the distances d, e and f_2. f_1 cannot be used because the point $\bar{y}$ is not in the feasible region, and so we select the nearest feasible point to $\bar{y}$. In the event, d turns out to be the shortest distance, and y_{n+1} is selected as $\pi_G(\tilde{y}_{n+1})$.

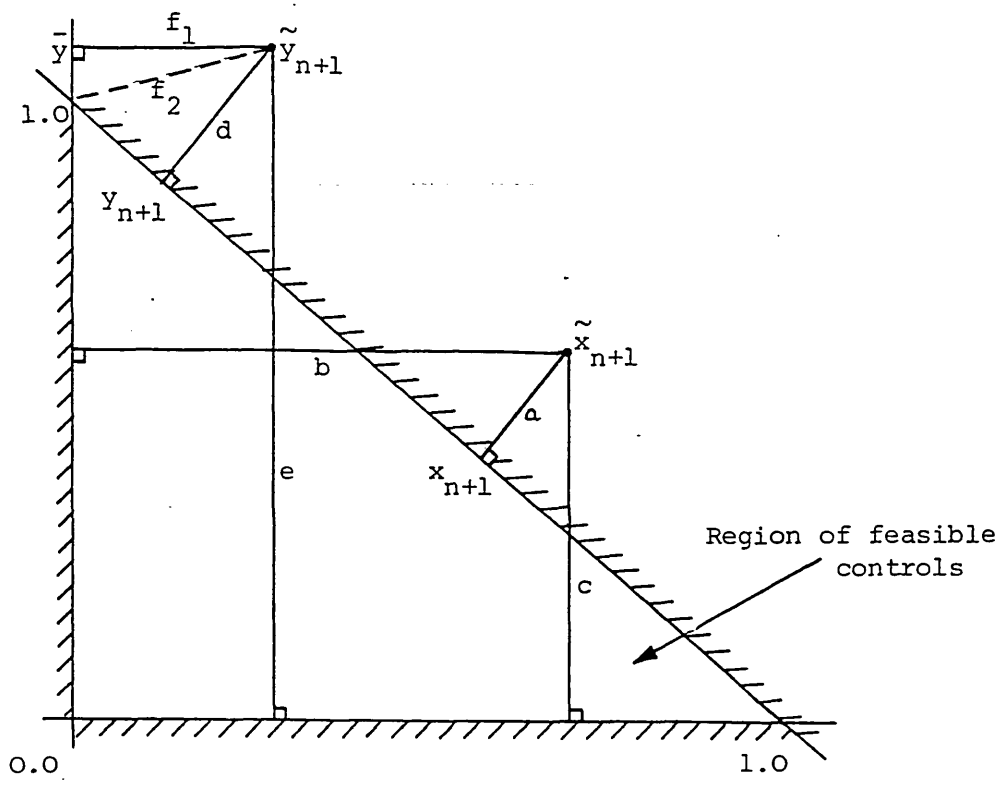


Fig. 5.11: Calculating $\pi_G(\tilde{x}_{n+1})$ and $\pi_G(\tilde{y}_{n+1})$ for two-parameter problem.

Again we consider a range of algorithm parameters, presenting the results in graphical form. Fig 5.12 shows three sequences produced by the projection algorithm using 1000-cycle estimates of cost gradients and starting from three different points. The sequences were continued up to 20 iterations in each case, but convergence was found to be rapid and points not indicated in Fig. 5.12 can be assumed to be within a small neighbourhood of the last indicated point of the sequence. The same comments apply to Fig. 5.13, where less accurate cost gradient estimates are used. 400 iterations of the algorithm were recorded for each set of parameters shown in Fig. 5.13 and convergence was established after about 40-50 iterations.

5.4.4 Choosing algorithm parameters

Kushner and Clark (54, Chapter VII) discuss rates of convergence and selection of parameters, and their comments are confirmed by the experimental results of the two preceeding sections.

Algorithm Parameters

(i) Simulation Length : 1000 cycles

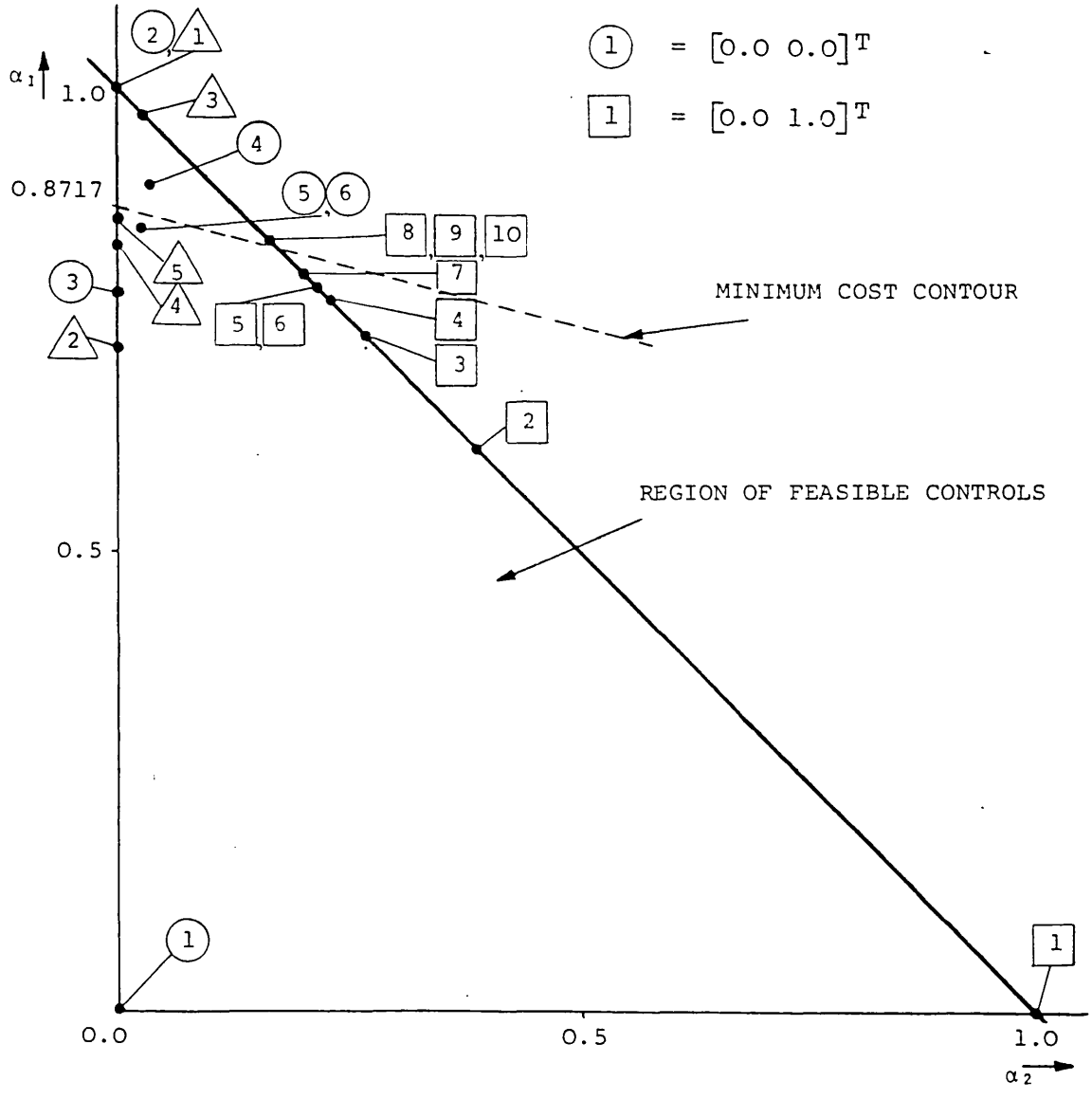
(ii) a = 0.5

(iii) Three $\{X_n\}$ sequences starting from

$\triangle 1 = [1.0\ 0.0]^T$

$(1) = [0.0\ 0.0]^T$

$[1] = [0.0\ 1.0]^T$



NOTES:

(i) $\triangle i$, (i), $[i]$ indicate the i'th estimate of the optimal control parameter $[\alpha_1\ \alpha_2]^T$ for sequences starting from three different points.

FIG. 5.12 : Solution of problem having two parameters using projection algorithm

Algorithm Parameters

(i) Simulation Length : 100 cycles

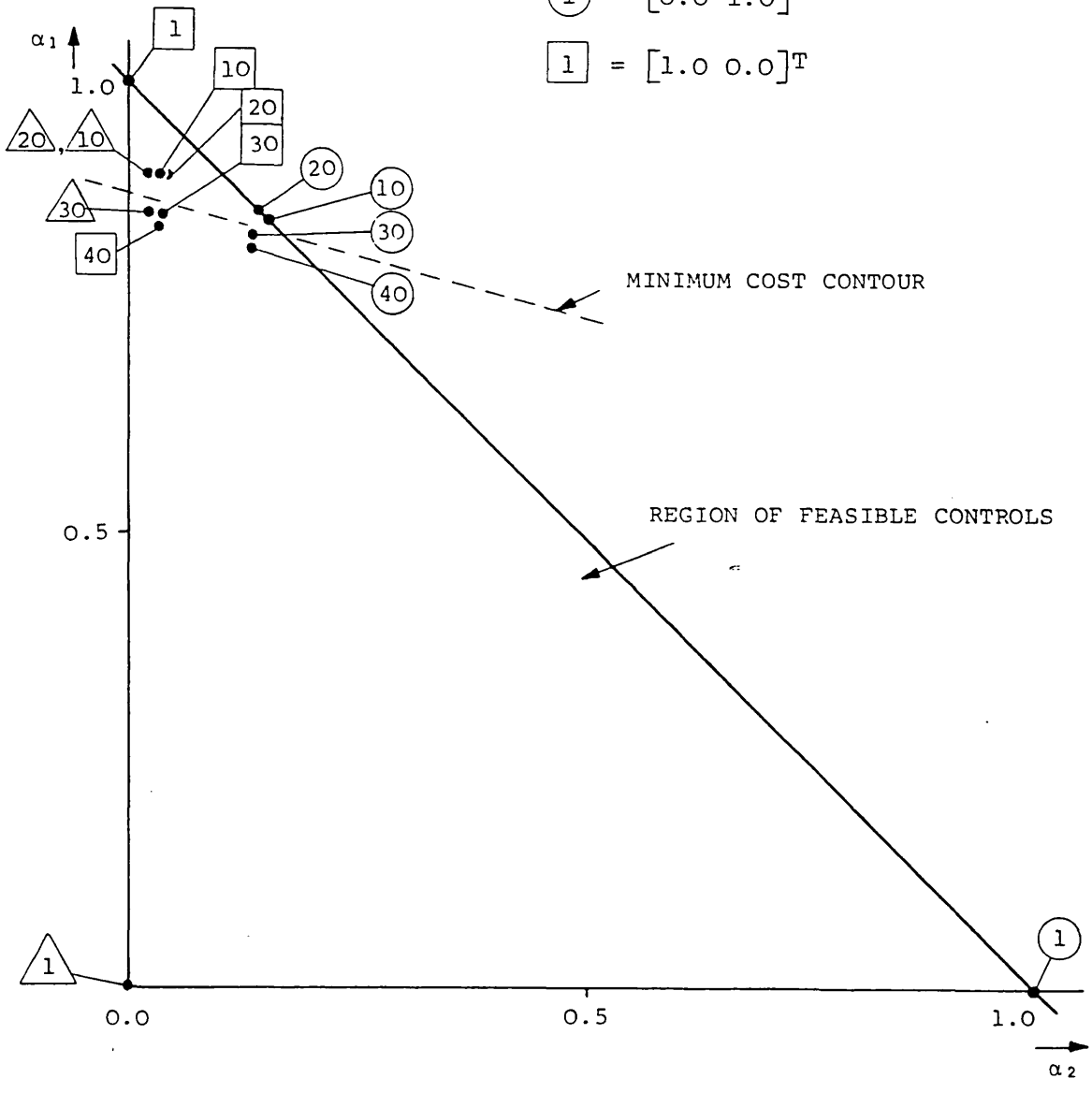
(ii) a = 0.5

(iii) Three $\{X_n\}$ sequences starting from

△1 = $[0.0\ 0.0]^T$

(1) = $[0.0\ 1.0]^T$

□1 = $[1.0\ 0.0]^T$



Notation as in FIG. 5.12

FIG. 5.13 : Use of 100-cycle estimates in projection algorithm

Firstly, we conclude that averaging over increased run lengths (or alternatively averaging over an ensemble of runs) at each iteration does not substantially affect the computing time required to achieve a solution to the optimal control problem. Increasing the run length by a factor K causes the estimate variances to decrease by K, but the same total number of simulated process cycles are required to achieve a solution, because the stochastic approximation algorithm uses (1/K)th as many iterations.

Secondly, we find that the parameter 'a' should be adjusted to a value large enough to ensure convergence and small enough to prevent "oscillation". Essentially it acts in the same way as the loop gain factor in a closed-loop control system, where too little gain results in a sluggish system response and too great a gain drives the system into oscillation. It was found that for the transfer-line problem, where gradients are between -.003 and -.017, a = 100 was a suitable value, thereby causing $(a/n).h(x_n)$ to fall between -.3 and -1.7 at the first iteration. A larger value than this would be unsuitable bearing in mind the system constraints. Likewise the gradients of the two-state problem reach a maximum value of about 8.8 and applying the same rule of thumb gives a = 0.5 as a reasonable value.

Finally, the experimental results indicate that the choice of x_1 is far less important in a stochastic approximation algorithm than in an analytic algorithm. The rate of a convergence depends more on the other parameters than on x_1 due to the random components of each gradient estimate.

5.5 Summary

Simulating large Markovian systems like transfer-line problems is best done by splitting up the complete system description into a number of subsystems, simulating each subsystem using different random numbers. It turns out that reverse-time, as well as forward-time simulations can be treated in this way and a dual relationship exists between the simulation rules for each. This observation is offered as an original contribution in the thesis, and is important because straightforward weighting methods like those described in Section 4.7.1 are of little practical use when applied to large transition matrices. For these, reverse-time simulation only becomes a practical proposition when subsystems can be identified. The rules applying to the cross-linked transfer-line problem are discussed in the first two sections of the chapter. Cost gradient estimates for the 64-state transfer-line example of Appendix I can be obtained by a number of methods combining forward- and reverse-time simulation. Results are obtained using four of these methods and the relative efficiencies of the methods are assessed.

Optimal control problems for which cost gradient estimates are available can be solved using stochastic approximation algorithms. A projection algorithm of this type is described and used to solve two grouped-state optimal control problems. The results are discussed and comments are included on the choice of algorithm parameters.

CHAPTER 6

VARIANCE REDUCTION METHODS

6.1 Introduction

From the previous two chapters, we see that gradients of total process cost over N cycles can be estimated from a single (or possibly two, if a reverse-time adjoint estimator is used) N-cycle simulation(s) in a number of ways. Each estimator, given the size of the problem, and the magnitude of N, will have associated with it a certain variance. Even the most accurate gradient estimates, obtained using adjoint estimators 3a, 3b and 3c, have a high and unacceptable level of variance. The simplest ways of making an estimate more accurate have already been used in Chapter 5. Naïve, or crude Monte Carlo sampling involves taking the mean of several independent estimates. If T seconds of computing time are required to produce a single-run estimate having variance v, then the crude Monte Carlo estimate obtained by taking the mean of k independent estimates will have variance v/k and will require kT seconds of computing time. Alternatively, for systems described by a single ergodic set of states, it is possible to estimate the gradient of mean process cost per system transition. Thus we can reduce variance by averaging over one long simulation run rather than by averaging over a large ensemble of runs. This is an alternative form of the naïve Monte Carlo estimator, i.e. if an N-cycle estimate of the mean cost gradient has variance v, and is computed in T seconds, then a kN-cycle estimate has variance v/k and is computed in kT seconds.

Thus we seek variance reducing methods which reduce variance (or computing time) to a greater extent than they increase computing time (or variance). Some authors (mentioned later) quote figures for

variance reduction without any adjustment for extra computing time, and it often turns out that such methods are no better, or are sometimes worse than the crude Monte Carlo estimator. Thus in the following sections we assess methods with both variance and computing time taken into account. We use the definition of 'normalized variance reduction' given by Moy (57, Table 3), i.e. normalized variance reduction = {1-[(average variance of alternative sampling procedure).(computer running time of alternative sampling procedure)]/[(average variance of crude estimate).(computer running time of crude estimator)]} x 100%.

Whereas a straightforward approach to estimation might be to generate records of a system's performance, and then to analyse input and output data, the use of Monte Carlo techniques implies that the simulation process can be radically changed in order to improve the accuracy of estimates without introducing bias. The discussion of variance reducing methods included here reviews the standard procedures and concludes that only two of these seem worthwhile when applied to simulations of a finite-state discrete-time process. It turns out, however, that a refinement technique often used in numerical analysis can be employed in a successful variance reduction method. A good approximation to the relative cost vector $\bar{\lambda}$ is an initial requirement, and is obtained through use of one of a range of reduced system models which are also discussed in some detail. The standard transfer-line problems described in Appendix I are used to indicate levels of variance reduction to be expected using two of these reduced models.

6.2 Variance reduction: a review of standard techniques taken from the literature

6.2.1 Use of Expected Values (see Kahn (29) and Handscomb (56))

This simple but effective technique combines an element of

analytical calculation with the normal simulation process. It follows the general rule of Monte Carlo sampling, namely that whenever possible, parts of the problem should be solved analytically, substituting these partial solutions into the estimation process where appropriate.

Thus at some point in a simulation, rather than counting actual occurrences of some future event, we might be able to calculate the expected number of occurrences. To quote Handscomb's example, if the probability of some future event occurring is known to be p, then instead of later scoring 1 when the event occurs or 0 otherwise, we score p in either case, leaving the expected score unchanged. Variance is thereby reduced by p(1-p) for every p scored. It will be shown later that this simple modification to the estimation process can bring about a reduction in variance when used in the transfer-line example.

6.2.2 Antithetic Variates (see Handscomb (56), Hammersley and Handscomb (31), Moy (57) and Kleijnen (20))

The term antithetic variates is generally used to describe any set of estimators which mutually compensate each other's variations. Negative correlation is created between (usually) two unbiased estimates by generating one using a random number, r, selected from a uniform distribution on the interval [0,1], and the other using another random number (1-r), known as the antithetic partner. If the two estimates are x_1 and x_2 then the antithetic estimate is the mean of x_1 and x_2, i.e. $\bar{x} = \frac{1}{2}(x_1 + x_2)$, whose variance is

$$\text{var}(\bar{x}) = \tfrac{1}{4}\text{var}(x_1) + \tfrac{1}{4}\text{var}(x_2) + \tfrac{1}{2}\text{cov}(x_1, x_2) \qquad 6.1$$

and if $\text{cov}(x_1,x_2)$ is negative (equivalently if x_1 and x_2 are negatively correlated) then the variance of the antithetic estimate, $\bar{x}$, is less

than half of the variance of either individual score x_1 or x_2, resulting in a more efficient estimation procedure.

This technique can bring about dramatic improvements when used in solutions of uncomplicated mathematical problems like the evaluation of the integral $\theta = \int_0^1 f(x)dx$. This type of example is used by Hammersley and Handscomb (31) to demonstrate variance reduction by 97% (i.e. by a factor of 31), these figures having been adjusted to allow for the extra work involved in calculating the antithetic estimates. Even greater improvements are recorded for solution of this type of problem using more sophisticated antithetic variate methods which employ variations of the "stratified sampling" variance reduction technique. Stratified sampling has proved a difficult technique to apply to Markov chain problems, and is therefore not discussed here.

As will be seen later, the effectiveness of antithetic variate methods when applied to multi-stage experiments (i.e. simulations) is much less predictable, and invariably of a lower level. Kleijnen (20) reports on antithetic variate applications in the simulation of single server queuing systems by four authors, and records average variance reductions of 71%, 41%, 46% and 42%. He also reports on more complicated queuing system simulations where variance was reduced by 40% or less. It appears that he has not made any allowance for the additional work required.

6.2.3 Use of Common Random Numbers (see Kahn (29) Kleijnen (20), and Hammersley and Handscomb (31))

This is the technique most widely used in simulation studies and has been applied in many cases where several systems are to be simulated with the object of choosing the 'best' one. Here we are more interested

in the relative values of two or more system responses, rather than their absolute values. Hence the systems are compared "under the same circumstances", i.e. the same random numbers are used to generate the same event at the same stage of simulation of two or more systems. In doing this, we are attempting to introduce a positive correlation between the system responses. We can only achieve this if each system responds to the random number inputs in a similar way, e.g. a random number of low (high) value giving rise to an estimate of low (high) value for each system simulated.

When comparing two systems by simulation, with unbiased estimated responses x and y, an unbiased estimate of the difference between the responses is (x-y), having variance

$$\text{var}(x-y) = \text{var}(x) + \text{var}(y) - 2 \cdot \text{cov}(x,y) \qquad 6.2$$

If x and y are positively correlated, i.e. if cov(x,y) is positive, then this estimator is more efficient than the crude Monte Carlo estimator, where use of independent sets of random numbers would imply that

$$\text{var}(x-y) = \text{var}(x) + \text{var}(y) \qquad 6.3$$

Kleijnen discusses the technique in detail with reference to "synchronization" (identical random numbers being used for each system variant at the same juncture in the realization of the two processes - achieved by the use of multiple random number streams, or by the use of a single stream of random numbers, where some are not used). Kleijnen also goes to considerable lengths to warn the reader about the drawbacks of indiscriminately applying common random numbers (requiring positive correlation between system responses) together with antithetic variates

(requiring negative correlation).

6.2.4 Control Variates, or Regression Sampling (see Moy (57), Hammersley and Handscomb (31), Handscomb (56) and Kleijnen (20))

Many variations (Kleijnen lists eight) of this idea have appeared in the literature. We discuss here the one best known and most relevant to the Markov decision problem.

A related, simpler system with easily calculated average response μ_z is simulated together with the system of interest using common random numbers, thus providing a measure of positive correlation between the two simulations. If the estimated responses of the original and simplified systems are x and z respectively, then the control variate estimator is defined by:

$$x_c = x - a(z - \mu_z) \qquad 6.4$$

where a is a scalar chosen to minimize variance.

The estimator x_c is unbiased:

$$\begin{aligned} E\{x_c\} &= E\{x\} - a\, E\{z - \mu_z\} \\ &= E\{x\} - a(E\{z\} - \mu_z) \\ &= E\{x\} \end{aligned} \qquad 6.5$$

and the variance is given by:

$$\mathrm{var}(x_c) = \mathrm{var}(x) + a^2 . \mathrm{var}(z) - 2a . \mathrm{cov}(x,z) \qquad 6.6$$

Again it can be seen that positive correlation between the two

system responses allows the possibility of variance reduction to an extent determined by the accuracy with which the large system is modelled by the reduced one, and the choice of the scalar, a. Kleijnen discusses at length ways of calculating (or estimating by prior Monte Carlo experiment) the value for 'a' giving maximum variance reduction, but many authors accept that if the reduced system is a good approximation to the original then significant variance reduction can be achieved by setting a = 1. It should be noted that when crude Monte Carlo sampling is applied to J independent pairs of estimates $\{ x_j, z_j ; j = 1, \ldots, J \}$, then from equation 6.4 the control variate estimator is

$$\bar{x}_c = \frac{1}{J}\left(\sum_{j=1}^{J} [x_j - a \cdot z_j] \right) + a\mu_z \qquad 6.7$$

The effectiveness of the control variate technique can be judged from the following figures taken from the literature. For estimation of a simple integral, Hammersley and Handscomb show that 97% variance reduction has been achieved, allowing for the extra work involved. Moy reports reductions between 20% and 50% for simulations of simple queuing systems, while Kleijnen quotes reductions of 50% or less for a range of simulations, falling to single figure variance reductions for complex (queuing system) simulations.

6.3 Application of standard techniques to the transfer-line problem

6.3.1 A necessary feature of the simulation

In order to apply antithetic variates, common random numbers, and control variates, we need to program the problem in such a way that regardless of the state assumed by the system at the start of the cycle,

the system always reacts in the same way to a given random number. Thus we program the simulation so that random numbers with high values tend to keep machines working, rather than broken down, implying that the system responses (number of pieces at the output of the system) have high values. Obviously the opposite effect is achieved for random variables with low values. We note that for a broken-down machine, the probability of repair in the next cycle is a constant, REP, and we set R = (1-REP). A random number ϵ [0,R) implies no change of state, whereas a random number ϵ [R,1] gives rise to a simulated repair. Similarly, for a working machine, the probability of breakdown in the next cycle is a constant, B. Random numbers ϵ [0,B) cause simulated breakdowns while those ϵ [B,1] imply no change of state.

6.3.2 Use of expected values

It has already been established that the accuracy of a gradient estimate depends on the accuracy of each of its two components, the state distribution and adjoint vector sequences. While it is theoretically possible to improve estimates of each sequence by the use of expected values, in practice it makes sense only to apply the technique to the estimation of adjoint vectors. Furthermore, benefits only become apparent when using forward-time adjoint estimators, because the reverse-time methods involve sampling from large vectors. Sampling from the vector c is straightforward because of its structure (in fact it is only necessary to generate random integers, as shown in Section 5.2.3) but sampling from a more complex vector like q (=Pc) would be tedious.

Recalling that a process cycle is composed of a 'control transition' followed by a 'process transition' (see Section 2.2.5), we propose the following improvement to the forward-time adjoint estimators. Rather than scoring $< e_j, c >$ for a system transition ending

in state e_j, we score $\langle e_i, Pc \rangle$ for a system transition beginning in state e_i. This is a simple and convenient procedure to program in the case of the cross-linked transfer-line model. During the process transition phase, the two lines are acting independently, and it is therefore possible to generate distinct transitions using each single line model. We can easily store the vectors $(P^A c^A)$ and $(P^B c^B)$ corresponding to the individual lines. If the overall system state, e_i at the beginning of the system transition is identified by the smaller vectors e_m and e_n for lines A and B respectively, then the score recorded at the beginning of the transition is

$$\langle e_i, Pc \rangle = \langle e_m, P^A c^A \rangle + \langle e_n, P^B c^B \rangle \qquad 6.8$$

That individual scores can be added in this way for independent systems is proved by Norman (43, Appendix I).

To quantify the benefits of applying this method to the transfer-line example, program G3c was run both with and without expected value variance reduction. The average variance reduction (calculated from estimated variances over a range of simulation conditions) was 59%. Table 6.1 gives details of the simulations with results. The estimated improvements are likely to be optimistic and we might do well to pay more attention to the first two figures (which are reasonable estimates of the variance over 100 and 1000 cycles). Taking the mean of these figures, we see that 25% is a worthwhile improvement, bearing in mind that virtually no extra computing time or storage is required for the method.

6.3.3 Antithetic Variates

Several authors including Tocher (18) and Moy (57) agree that the

No. of runs	Length of simulation (cycles)	Estimated variance for a single run		Variance reduction
		Using expectations	Not using expectations	
10	100	.0173976	.061009	71%
10	1000	.0015054	.001239	-21.5%
1	10000	.0000211	.000196	89%
1	20000	.0000004	.0000184	97.8%

Simulation Conditions: (i) Standard 64-state model (2 transfer-lines each having buffers of capacity 1 production unit)

(ii) Nominal control parameter, $\bar{\alpha} = 0$

(iii) Variance figures relate to mean cost gradient estimates.

Table 6.1 : Estimated variance reduction through use of expected values

No. of runs	Length of simulation (cycles)	Gradient estimate (expected value = -.017)	Estimated variance for single run	Computing time for experiment (seconds)	Variance reduction	Variance reduction, adjusted for extra work
500	100	-.0149 (-.0208)	.0061 (.0136)	163.3 (86.6)	55%	15%
10	1000	-.007 (+.0065)	.0011 (.0019)	36.0 (19.4)	47%	1%
50	1000	-.0128 (-.0082)		170.7 (89.8)	42%	-10%

Figures in parentheses are the results from corresponding experiments using program G3c.

Simulation Conditions: (i) Standard 64-state model (2 transfer-lines each having buffers with capacity 1 production unit)

(ii) Nominal control parameter, $\bar{\alpha} = 0$.

Table 6.2: Estimated gradients and variance reduction using the antithetic variates estimator.

use of random numbers $r_1, r_2, \ldots\ldots$ in one simulation run and $(1-r_1)$, $(1-r_2), \ldots$ in an antithetic simulation run lead to negatively correlated system responses if the simulation is carefully programmed (see Section 6.3.1). Kleijnen (20), however, points out that in the simulation of complex systems, negative correlations cannot be assumed, and can only be shown to exist by trial. Nevertheless, it seems reasonable to assume that an antithetic variate estimator using the pair of simulations described above should offer some variance reduction. The result of applying this antithetic variate method to program G3c is shown in Table 6.2. It can be seen that variance reductions of between 40% and 50% compare favourably with the range of results reported in Kleijnen for simulations of simple queuing systems. Note however, how the figures are much less attractive when adjusted for the extra work involved in implementing the method (typically doubling the run-time of the standard program). In fact the adjusted figures show that we derive almost no benefit from using this form of antithetic variate estimator for this type of problem, and we question whether Kleijnen's reported variance reductions for complex queuing systems (5 to 10% and 18 to 27% were quoted), when corrected for extra computing time, would still show the method to be worthwhile. Handscomb (56, p.260) warns that when applying antithetics in simulation studies, "its effectiveness depends rather heavily on the care with which we decide when to use it, since every application multiplies the minimum number of samples we have to take by a factor of at least two," and this study shows that the extra computing time is significant.

It is worth mentioning that the gradient estimation method forming the basis of this antithetic variate estimator incorporates forward-time adjoint vector estimation with implicit use of common random numbers (see Section 6.3.4). The resulting positive correlation employed in this estimation process could lead to undesirable cross-

correlation terms in the antithetic gradient estimate (as shown by Kleijnen (20, p.209)), thus reducing the accuracy of the results in this case. Had we been observing some simple feature of the system like average output, we might achieve much better results using this antithetic variate technique. Though Kleijnen agrees with other authors that variance reduction methods are "by no means mutually exclusive" (and we shall see by the end of this study that at least three methods can be simultaneously applied to the transfer-line problem with significant benefits) his study of the joint application of antithetic variates and common random numbers shows that in many cases, variance increases can result.

6.3.4 Common Random Numbers

Common random numbers are used in adjoint estimation methods 3a, 3b and 3c. In Method 3c they are used for other reasons than just to reduce variance (which they undoubtedly do, though the reduction has not been quantified by experiment). To begin with, it is expedient to do so: a vast amount of computing time is saved by using the same random numbers for the additional "adjoint runs", and the resulting element estimates are obviously unbiased. Another reason is that as we are estimating, for each partial derivative, the difference between two non-terminating sequences of costs, the use of independent random numbers in the generation of these sequences would force us to terminate the adjoint estimation process when each sequence had been judged to have achieved its steady state, i.e. each sequence increasing by the same mean cost every transition. Terminating such sequences without introducing bias is a well-known problem (see Kahn (29), where "Russian Roulette" is suggested as a possible method). However, by using common random numbers for the simulation of the same system starting in two different states, we identify the end of the transient easily. At

some stage, the states assumed by each simulation will be identical (signalling the end of the transient), and because common random numbers are used, will continue to coincide ad infinitum. Thus we have no problem in identifying the end of the transient, and can estimate elements of the 'relative cost vector', $\bar{\lambda}$, without introducing bias.

When comparing the performance of two different systems by simulation, the benefit due to using common random numbers is less obvious than when (as in this procedure) we are comparing responses of the same system, starting in two different states, and this fact is especially true when a simulation of that system initiates additional "adjoint runs" at random times.

6.3.5 Control Variate Methods

From the previous general discussion of control variates, it should be apparent that the efficiency of the method depends largely upon the extent to which the simpler model mimics the original. Therefore it is essential that we spend some time considering ways of reducing the complexity of our large Markov model. A requirement of the control variate formulation is that the expected cost gradient for the simpler system can be calculated. Thus we seek a rule for mapping the original system states into a smaller space, so that matrices and vectors for the reduced model can be accommodated by the computer, allowing computation of the required gradient. This type of reduced model satisfies another requirement of the control variates method, namely that the similarities between the two models allow similar events to be generated by the same random numbers.

It turns out that by identifying problem characteristics, it is possible to reduce the number of states but at the same time retain

features of the original model which ensure close correspondence between simulations of each system. Norman (43) has made a study of such reduction methods in connection with his heuristic procedures in dynamic programming. Appendix IV reviews those reduction methods most appropriate in the treatment of transfer-line problems and adds some novel variations. The reduction method referred to in Appendix IV as "State Grouping" was used to derive a 64-state Markov model which acted as a control variate during the simulation of our standard 144-state transfer-line problem. Most features of the two models are identical, but whereas the buffer of each transfer-line of the large system contains 0, 1 or 2 units, the corresponding variable of the reduced system represents 0 or 1 unit. Thus "buffer states" 1 and 2 of the original system are mapped into the single state 1 of the reduced system. We assume for this study that state 1 of the reduced system represents states 1 and 2 of the large system with equal probability (0.5), but prior calculation or estimation of the steady-state probabilities for states 1 and 2 of the large system leads to a very accurate control variate model which has the same essential properties as the original, at least in the steady-state.

The initial work on this method was directed towards quantifying the improvements (if any) which resulted from estimating the first four adjoint vectors using adjoint estimation Method 1, together with the 64-state control variate above, assuming a zero-valued nominal control parameter, $\bar{\alpha}$, throughout the experiment. By calculating the first four adjoint (64-) vectors of the reduced system, we can obtain approximate versions of the required 144-vectors by the rule given in Appendix IV. Each of these vectors is then used as μ_z in (four) vector versions of equation 6.7, representing control variate estimators of the four adjoint vectors of the original system. Thus, having μ_z for each estimate, we can obtain vectors x_j by estimation Method 1,

simultaneously generating adjoint simulations of the 64-state model to provide the vectors z_j.

From the early results, it became obvious that this control variate method is well suited to improving the "shape" of adjoint vector estimates. Whereas one run using a reverse-time method provides non-zero estimates of only a few elements of each vector, these sparse vectors are "filled in" by the sequence of approximate adjoints calculated using the control variate model. Unfortunately, it was found that in simulating the control variate using the same random numbers as the original process, the associated trajectories in (144-) state space were rarely seen to coincide no matter what refinements to the control variate were attempted, with the result that although the adjoint vectors were no longer sparse, there were significant variance contributions due to simulations of both the original, and the related process. Nevertheless, because the gradient estimation process requires only a small number of elements from each vector, it seemed likely that we should achieve some reduction in the variance of gradient estimates using the above method.

Table 6.3 gives results obtained by calculating (rather than estimating) the first four state distribution vectors, and using four adjoint vectors estimated in reverse-time, with and without the use of the control variate.

The more accurate variance estimates from the 500-run ensembles suggest that sample variance is changed very little and in view of the fact that computing time is increased, we are not encouraged to use this type of control variate.

The same control variate has been used in conjunction with gradient

	No. of runs	Gradient estimate	Estimated variance for a single run	Estimated S.D
Without control variate	10	+0.5282	2.4744	1.5730
	500	+0.0059	4.2773	2.0682
With control variate	10	+0.2647	1.0650	1.0320
	500	-0.0501	4.5799	2.1401

Simulation conditions: (i) True cost gradient over 4 cycles = -0.0136
(ii) Nominal control parameter, $\bar{\alpha} = 0$
(iii) 144-state standard model with 64-state control variate

Table 6.3 : Use of control variates to improve adjoint estimates

estimation routines which use adjoint estimators 3a and 3b over longer simulation runs to estimate the mean cost gradient per cycle. Here the control variate was employed as a means of directly improving gradient estimate accuracy. Equation 6.7 reverts to its scalar form, where μ_z is the gradient calculated directly (in 64-space) from the reduced model, and x_j and z_j are the gradient estimates from the 144-state and 64-state models, simulated using common random numbers. All variations of the control variate model produced slight increases in variance compared with the crude Monte Carlo estimates. Because gradient estimators using forward-time adjoint estimates are the most accurate of the methods studied, we can say with more confidence that this type of control variate is not effective when used in the simulation of discrete-state Markov processes.

As a postscript to this discussion of the use of control variates we note that although the method proved to be ineffective in this case, we have evidence that the approximate adjoint vectors derived from the related process were quite accurate, and we therefore conclude that this valuable information was being used in the wrong way. It is gratifying to be able to report that the model reduction techniques can be put to

good use in a successful variance reduction scheme described in the following section.

6.4 Variance reduction by refinement of an approximation

6.4.1 Origins of the method

We propose the use of a refinement technique which is well known in the field of numerical analysis. The method is normally used to refine a solution to the linear vector equation

$$Ax> = b> \qquad 6.9$$

which has been obtained either by Gaussian elimination (which is invariably subject to roundoff errors) or by an iterative method like Gauss-Seidel, where a large number of iterations might be required for a satisfactory solution, in which case use of this technique would accelerate convergence. For a detailed discussion of the method see any basic text on numerical analysis, like Dorn and McCracken (58, p.175). The technique was later put forward by Halton (30) (also see Hammersley and Handscomb (31)) as a sequential Monte Carlo method for estimating solutions to an alternative form of equation 6.9, i.e.

$$x> = a> + Hx> \qquad 6.10$$

where a is a constant vector.

6.4.2 Use of the technique to reduce variance of gradient estimates

The refinement technique is described using Halton's notation. We assume that a rough estimate, $\hat{x}$, of the solution to equation 6.10 has

* The references point out that H must have certain properties for the technique to be effective. It can be shown that the transition matrix for a regular Markov chain always has these properties.

been obtained by prior calculation or experiment.

Putting

$$y> = (x - \hat{x})> \qquad 6.11$$

and

$$d> = a> + H\hat{x}> - \hat{x}> \qquad 6.12$$

then equation 6.10 can be re-formulated as

$$y> = d> + Hy> \qquad 6.13$$

where components of d are smaller than those of a if $\hat{x}$ is a close approximation to x (if $\hat{x} = x$, $d = 0$).

It turns out that Monte Carlo estimates of the vectors x and y of equations 6.10 and 6.13 are obtained by similar methods to those used in this study of adjoint estimation. In fact, adjoint estimation method 3c, used to estimate the relative cost vector $\bar{\lambda}$, implicitly solves an equation of the same form as equations 6.10 and 6.13.

Proposition 6.1

The relative cost vector $\bar{\lambda}>(= (\lambda - b1)>$, where b is scalar valued) is a solution of

$$\bar{\lambda}> = v> + P(\alpha)\bar{\lambda}> \qquad 6.14$$

where $v> = (P(\alpha)c - g1)>$ and g is a scalar representing the mean process

cost per cycle.

Thus v is a constant vector for a given Markov process operating with nominal control parameter, α.

Proof

Repeating equation 4.42.

for large n ($\leq$ N)

$$\lambda_{N-n}\rangle = ng\ 1\rangle + \lambda\rangle \qquad 4.42$$

Also, from equation 4.4a,

$$\lambda_{N-n}\rangle = P(\alpha)c\rangle \quad + \quad P(\alpha)\lambda_{N-n+1}\rangle$$

therefore using equation 4.42,

$$ng\ 1\rangle + \lambda\rangle = P(\alpha)c\rangle + P(\alpha)[(n-1)g\ 1\rangle + \lambda\rangle]$$

$$= P(\alpha)c\rangle + (n-1)g\ 1\rangle + P(\alpha)\lambda\rangle, \quad \text{as } P(\alpha)\ 1\rangle = 1\rangle$$

$$\therefore \quad \lambda\rangle = (P(\alpha)c\rangle - g\ 1\rangle) + P(\alpha)\lambda\rangle \qquad 6.15$$

which can be re-written as equation 6.14 because:

substituting $\lambda\rangle = \bar{\lambda}\rangle + b\ 1\rangle$ into equation 6.15,

$$\bar{\lambda} + b\ 1\rangle = (P(\alpha)c\rangle - g\ 1\rangle) + P(\alpha)\ (\bar{\lambda}\rangle + b\ 1\rangle)$$

$$\therefore \bar{\lambda}\rangle = (P(\alpha)c\rangle - g\ 1\rangle) + P(\alpha)\bar{\lambda}\rangle \qquad 6.16$$

which is equivalent to equation 6.14.

Using 'expected value' variance reduction with adjoint estimator 3c, it can be seen that when generating two 'adjoint simulations' in order to estimate the relative values of two components $\langle e_m, \bar{\lambda}\rangle$ and $\langle e_n, \bar{\lambda}\rangle$, the expected score for the first transition will be $\langle [e_m - e_n][P(\alpha)c - g1]\rangle = \langle [e_m - e_n][P(\alpha)c]\rangle$, this being added to similar 'scores' recorded at the beginning of subsequent transitions.* It is not difficult to see that these scores can be reduced in magnitude by exploiting a rough estimate of $\bar{\lambda}$ using equations 6.11-6.13. It follows that we obtain a more accurate estimate of $\bar{\lambda}$ and therefore a gradient estimate with reduced variance. We begin by specifying a reduced model (one of those described in Appendix IV) and using it to calculate an approximate form, $\hat{\lambda}$ of $\bar{\lambda}$. When dealing with systems having a very large number of states, it is essential that both $\hat{\lambda}$ and $\hat{d}$ $(= P(\alpha)c\rangle + P(\alpha)\hat{\lambda}\rangle - \hat{\lambda}\rangle)$ can be easily stored in some form. For this reason Halton's sequential Monte Carlo method cannot be applied directly, whereas prior calculation or estimation using a reduced model implies that the reduced form of $\hat{\lambda}$ can be stored, expanding into the full state space as required by the gradient estimator. The type of reduced model used limits the ease with which the vector $\hat{d}$ is stored and retrieved, and further comments will be made later when results are discussed.

6.4.3 A simple example

To illustrate the reductions in variance made possible by this method, we consider the estimation of $\hat{\lambda}$ for the uncontrolled two-state system defined by:

* The additions are made according to the rules of Method 3c described in Section 4.8.8.

$$P = \begin{bmatrix} .75 & .25 \\ .1 & .9 \end{bmatrix}, \qquad c\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\therefore q\rangle = Pc\rangle = \begin{bmatrix} .25 \\ .9 \end{bmatrix}$$

and

$$\bar{\lambda}\rangle = \begin{bmatrix} -1.857 \\ 0 \end{bmatrix} = \lambda\rangle - (\langle e_2, \lambda\rangle)\ 1\rangle$$

Thus we intend to estimate $[-1 \quad 1]\,\bar{\lambda}\rangle$ using adjoint estimation method 3c, comparing crude estimates from a range of ensemble sizes to those obtained using the refinement technique with several prior estimates, $\hat{\lambda}$. Comparison of Tables 6.4 to 6.7 indicates the considerable benefits to be realized through application of this method. Our estimate of single run variance from Table 6.4 indicates that an ensemble of no less than 5100 runs would be required to obtain a estimate accurate to one decimal place (with 95% certainty) using crude Monte Carlo sampling. Table 6.5 shows quite clearly that even for a bad initial estimate like $[+1.0 \quad 0]^T$ we still get a significant variance reduction of the order of 80%. Furthermore, the method requires relatively little extra computation, even for the larger transfer-line problem (depending on the choice of reduced model), as unlike the standard control variates method, no parallel simulation of the related process is required. Table 6.7 indicates the value of improving the initial approximation. With $\hat{\lambda} = [+1.8 \quad 0]^T$, variance is reduced by a factor of 1067 compared with the crude estimator. Put another way, for accuracy to one place of decimals, we require an ensemble average over only 10 runs using this prior estimate, compared with over 5000 runs using crude Monte Carlo estimation.

No. of runs	Estimate of $[-1 \quad 1]\bar{\lambda}>$	Estimated variance for a single run	Estimated S.D.	Computing time for experiment (secs)
10	2.21	2.2457	1.4986	0.906
1000	1.8226			1.292
20000	1.8624			8.937
200000	1.85236			81.677

Table 6.4 : 2-state example - crude Monte Carlo estimates of $[-1 \quad 1]\bar{\lambda}>$

No. of runs	Estimate of $[-1 \quad 1]\bar{\lambda}>$	Estimated variance for a single run	Estimated S.D.	Computing time for experiment (secs)
10	2.02	0.4836	0.6954	0.884
1000	1.8412			1.276
20000	1.859575			8.862

Table 6.5 : 2-state example - use of refinement technique with prior estimate = $[+1.0 \quad 0]^T$

No. of runs	Estimate of $[-1 \quad 1]\bar{\lambda}>$	Estimated variance for a single run	Estimated S.D.	Computing time for experiment (secs)
10	1.83	0.0134	0.1159	0.893
1000	1.8598			1.303
20000	1.856737			8.952

Table 6.6 : 2-state example - use of refinement technique with prior estimate = $[+2.0 \quad 0]^T$

No. of runs	Estimate of $[-1 \quad 1]\bar{\lambda}>$	Estimated variance for a single run	Estimated S.D.	Computing time for experiment (secs)
10	1.868	0.0021	0.0464	0.884
1000	1.85608			1.275
20000	1.857305			8.959

Table 6.7 : 2 state example - use of refinement technique with prior estimate = $[+1.8 \quad 0\ ]^T$

6.5 Application of the refinement technique to the transfer-line problem

6.5.1 Use of Norman's reduced models

Appendix IV gives two ways of simplifying the state description of the complete transfer-line model according to its characteristics. Sections 6.5.2 and 6.5.3 illustrate the use of these reduced models in conjunction with the refinement technique, resulting in considerable reductions in the variance of gradient estimates. The technique, using an appropriate reduced model, is seen to have the potential for reducing variance to very low levels (reduction by as much as 98% for our example) encouraging the use of Monte Carlo gradient estimators for very large Markov decision problems.

6.5.2 Model reduction by 'State Grouping'

In the discussion of the standard control variates method applied to the transfer-line model (Section 6.3.5), we established that the reduced model resulting from grouping together certain states could be manipulated to provide very close approximations to the adjoint vectors. However, we concluded that this information was not being fully exploited by the standard control variates approach, and it is now apparent that the refinement technique is likely to make best use of this information. Furthermore, it is likely to be more efficient due to the fact that no control variate simulation is required. After specification of the reduced model, and calculation or estimation of the relative cost vector for this model, the only further calculation required is the mapping of the reduced relative cost vector into the approximate complete model vector, $\hat{\lambda}$, and the calculation of vector $\hat{d}$. Certain elements of these vectors would normally be calculated during a

simulation run for large problems but such calculations would take relatively little time.

As an illustration, we again estimate cost gradients for the standard 144-state transfer-line problem using a 64-state reduced model derived by the state grouping technique of Appendix IV. Our aim is to compare results obtained using estimate refinement with the crude Monte Carlo estimates obtained for the same problem using program G3c, for a nominal control parameter $\bar{\alpha} = 0$. The results are given in Tables 6.8 and 6.9. After correcting for the extra computing time required for calculation of approximate vectors, the variance reductions appear to be of the order 83-89%.

6.5.3 Use of independent sub-systems

We see from Appendix IV that one way of obtaining approximate results for the complete model is to combine vectors calculated from two or more reduced models which represent independent sub-systems.

This technique is particularly effective when applied to the transfer-line problem for two reasons:

(i) The type of control action assumed by the problem formulation implies that two normally independent sub-systems (transfer-lines A and B) are made to interact in a probabilistic manner. Thus simulations using nominal control parameters, $\bar{\alpha} \simeq 0$ are effectively simulations of two independent transfer-lines. Moreover, if we also assume lines A and B to be identical, then only one reduced model (representing line A or line B) is required for calculation of $\hat{\lambda}$. If the complete system has s states then $\hat{\lambda}$ is obtained by calculation using a

No. of runs	Length of simulation (cycles)	Gradient estimate	Estimated variance for a single run	Estimated S.D.	Computing time for experiment (secs)
10	1000	+.000455	.0003312	.0182	25.281
50	1000	-.007184			107.242
1	50000	-.008009	.0000302	.0055	107.008

Simulation Conditions: (i) 144-state example with 64-state reduced model for approximation

(ii) Nominal control parameter, $\bar{\alpha} = 0$

(iii) Accurate time-averaged cost gradient = -0.014

Table 6.8 : Variance reduction using the refinement method with 'state grouping' model reduction

No. of runs	Length of simulation (cycles)	Gradient estimate	Estimated variance for a single run	Estimated S.D.	Computing time for experiment (secs)
10	1000	-.002537	.0021252	.0461	20.162
50	1000	+.000892			95.973
1	50000	+.003294	.0002992	.0173	98.351

Simulation Conditions: As for Table 6.8 except solutions obtained by basic forward-time gradient estimator.

Table 6.9 : Estimated gradients for 144-state problem without use of variance reduction methods.

single reduced model having $\sqrt{s}$ states. Taking a realistic value for s, we see that the reduction from a 10,000-state model to a 100-state model makes possible the use of a direct solution algorithm like Gaussian elimination.

(ii) Independent sub-systems analysis frequently provides a great deal of information about systems having non-zero nominal control parameters. A case in question is when cross-over control is exercised in relatively few process states (e.g. our standard 64-state example, where only 16 states allow the use of cross-over control). If a system makes extensive use of cross-over control, and control actions applied to individual lines are similar for all states, it may be possible to consider a pair of independent sub-systems where each is "controlled" regardless of the other's state. For example, the complete model tells us that transfer-line A accepts a piece from transfer-line B when A and B are in certain states. Using independent sub-systems, controlling actions for each sub-system are determined with reference to the state of just that sub-system, i.e. A 'accepts' a piece if it can, regardless of B's state, and likewise B 'gives up' a piece if it can, regardless of A's state. It is recognised that this type of model could only be useful in certain cases, but is included to show the range of reduced models made possible by this approach.

For our first set of results, we estimate cost gradients for the standard 64-state transfer-line example, initially by program G3c, and then using the refinement method with two independent (and uncontrolled) sub-systems as the means of deriving $\hat{\lambda}$. These two estimators were compared using long (1000 cycle) simulations having a range of nominal

control parameters, $\bar{\alpha} \in [0,1]$. The results are summarized in Fig. 6.1. As we would expect, variance reduction reaches a maximum when $\bar{\alpha} = 0$, i.e. when we are simulating independent transfer-lines. Under these conditions we see that $\hat{\lambda} = \bar{\lambda}$, removing the requirement for adjoint estimation. It is interesting to note that for this example at least, 98% of the variance of gradient estimates obtained from program G3c is attributed to the adjoint estimates. Thus we feel that the approach adopted in this chapter, i.e. reducing the variance of gradient estimates by improving adjoint vector estimates, is justified.

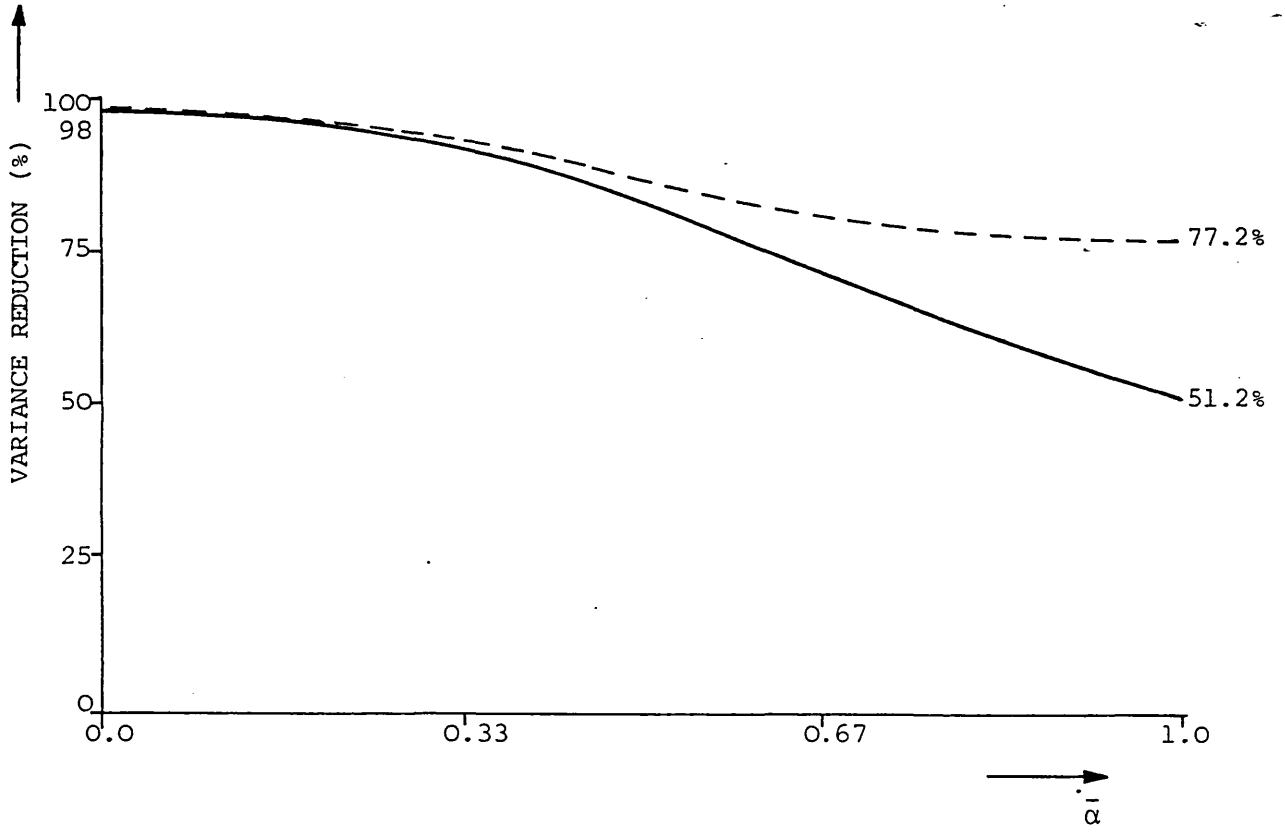
The broken curve of Fig. 6.1 shows how independently controlled sub-systems (see (ii) above) can be used to improve the estimates still further. Some interesting points were raised during experiments using this estimator which warrant a more detailed discussion of the sub-system models.

As a simple illustration, suppose we have two identical sub-systems, A and B, each modelled by a two-state Markov chain with process transition matrix P_{AB}. The process transition matrix for the complete system is thus a 4 x 4 matrix, P.

$$\text{If } P_{AB} = \begin{bmatrix} .75 & .25 \\ .1 & .9 \end{bmatrix}, \text{ then}$$

$$P = \begin{bmatrix} .75 \begin{bmatrix} .75 & .25 \\ .1 & .9 \end{bmatrix} & .25 \begin{bmatrix} .75 & .25 \\ .1 & .9 \end{bmatrix} \\ .1 \begin{bmatrix} .75 & .25 \\ .1 & .9 \end{bmatrix} & .9 \begin{bmatrix} .75 & .25 \\ .1 & .9 \end{bmatrix} \end{bmatrix} \qquad 6.17$$

Suppose also that (with probability α) we can exert a control



——— Reduction due to use of refinement technique with independent (uncontrolled) subsystem approximations.

– – – – Reduction due to use of refinement techique with independent (controlled) subsystem approximations. [also see Fig. 6.2]

Note: All reductions are calculated relative to gradient estimates from program G3c.

Fig. 6.1: Variance reduction through independent sub-system analysis.

action only when A is in state 1 and B is in state 2 which sends A to state 2 and B to state 1.

$$\text{i.e. } D(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & (1-\alpha) & \alpha & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad 6.18$$

A cycle of the complete system is characterised by the system transition matrix $P(\alpha) = D(\alpha)P$. Following the outline of (ii) above, we define the following independent (controlled) sub-systems by 2 x 2 transition matrices:

$$D_A(\beta) \cdot P_{AB} = \begin{bmatrix} (1-\beta) & \beta \\ 0 & 1 \end{bmatrix} \begin{bmatrix} .75 & .25 \\ .1 & .9 \end{bmatrix} \text{ for subsystem A}$$

and

$$D_B(\beta) \cdot P_{AB} = \begin{bmatrix} 1 & 0 \\ \beta & (1-\beta) \end{bmatrix} \begin{bmatrix} .75 & .25 \\ .1 & .9 \end{bmatrix} \text{ for subsystem B} \qquad 6.19$$

The control parameter β can be adjusted so that for a given α, the model produced by combining the independent sub-systems is in some sense a good approximation to the original.

Having produced similar independent sub-system models for the 64-state transfer-line problem, a sequence of experiments was conducted to determine the relationship between β and α providing the best independent sub-system model in the sense of achieving maximum variance reduction for a given α. The results of these experiments are summarised in Fig. 6.2. It seems unlikely that this relationship could be determined analytically and further research into this seems

unjustified in view of the marginal improvements indicated by Fig. 6.1.

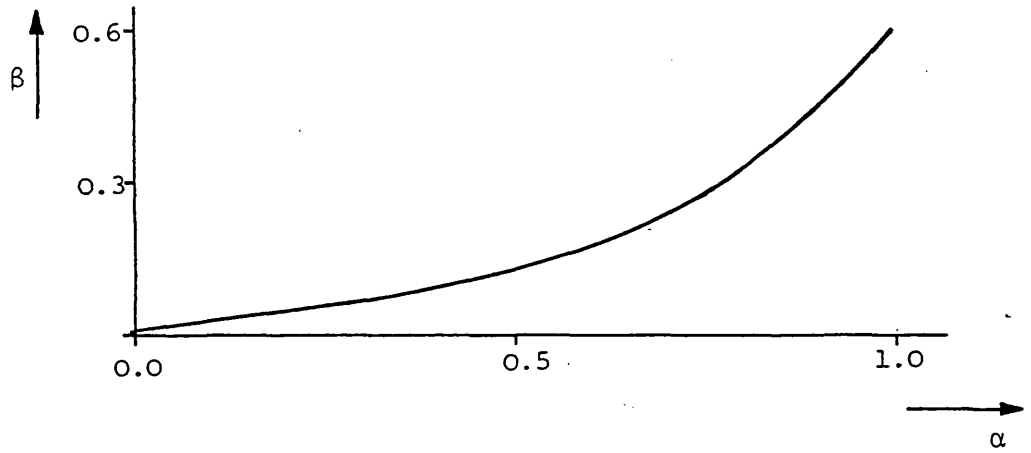


Fig.6.2: Relationship between β and α .

6.5.4 Some comments on the refinement method

The previous sections have established the potential of the refinement method as a powerful technique for reducing estimate variance in complex simulations. Some of the main features are:

(i) Very little extra computing time is required.

(ii) Six types of reduced model have been established by Norman (43) and are readily applied to complex O.R. problems modelled by Markov chains.

(iii) Very large variance reductions are possible because the technique improves the most inaccurate part of the gradient estimation process. For the standard transfer-line examples considered in this thesis, adjoint vector estimation accounts for up to 98% of the variance associated with gradient estimates. Results have shown that approximate relative cost vectors calculated from 'reduced' Markov models are often

sufficiently accurate to reduce the overall variance level by more than 80%.

(iv) We see that our numerical results compare favourably with variance reductions reported by Kleijnen and others in the simulation of complex systems.

6.6 Summary

This chapter reviews four well-known variance reduction techniques, and concludes that the most popular ones, antithetic variates and control variates, are not effective when applied to simulations of discrete-time finite-state Markov decision problems. The other two methods, common random numbers and expected values, are conveniently applied to the best of the 'forward-time' gradient estimators, i.e. program G3c.

Solutions of linear vector equations computed using analytic methods are invariably inaccurate to some extent, due to computer round-off errors, etc. These deterministic errors can be reduced by application of a refinement technique which has been known to numerical analysts for some time. In Sections 6.4 and 6.5 we show that the technique can be used to reduce statistical errors (i.e. reduce variance) in gradient estimates obtained from program G3c. An approximate relative cost vector, $\hat{\lambda}$, is calculated using one of a range of 'reduced' Markov models, and the difference between this vector and the true relative cost vector, $\bar{\lambda}$, is estimated using adjoint estimation method 3c. The only substantial computational overhead is the prior calculation of $\hat{\lambda}$ using a reduced model, and the variance reductions can be dramatic. We conclude that the use of the refinement technique with gradient estimator G3c represents an extremely efficient procedure for

estimating cost gradients which can be used in non-linear programming or stochastic approximation algorithms to solve Markov decision problems.

CHAPTER 7

CONCLUDING REMARKS

7.1 Introduction

Working from the basic assumption that a finite-state, discrete-time Markov decision process can be used to describe a given problem in the real world, it is shown in Chapter 1 that determining the optimal decision policy using the conventional approach, i.e. formulating a "classical Markov decision problem" and solving it using existing algorithms, is acceptable in theory, but can in practice lead to computational (and other) difficulties, especially when the model has many states and alternative actions. Having identified some of the drawbacks of the conventional approach in Chapter 1, we proceed by simplifying the decision structure of the Markov model, leading to the definition of a "grouped-state Markov decision problem" and use the rest of the thesis to develop analytical and experimental solution methods for this problem.

In this chapter, we try to give some idea of the size of problem which can be solved using the proposed methods, by assessing the computational efficiency of the recommended analytical and experimental methods. Also, we review the contributions of the thesis bearing in mind the objectives which were defined in Chapter 1, thereby determining what measure of success can be claimed for the complete research programme.

Finally, we suggest some opportunities for future research, either by extending results given in this thesis, or by considering adaptive control schemes.

7.2 Assessing the usefulness of the recommended methods for solving problems of a realistic size

7.2.1 Discussion

In this section we try to give some idea of the size of problem which can be solved by the methods recommended earlier in the thesis. We calculate some approximate figures for computing time and storage required by methods solving large problems using 'grouped-state' and 'classical' control structures. This allows a direct comparison of the new and existing solution methods on grounds of efficiency and therefore applicability to large problems. The most obvious way of making the comparison would be to observe the performance of each method when used to solve a given problem. Unfortunately this procedure is unacceptable because of the vast amount of computing time and storage which would be required to solve a problem of suitable size (having (say) 10,000 states) even by the most efficient method. All results appearing in this thesis were obtained using the CDC Cyber 174/6500 computer installation at Imperial College which at the time of writing allowed user jobs a maximum of 50,000 words of (central) memory and a maximum of 600 seconds of computing time. It is helpful to bear in mind these practical limitations when discussing the solution of large problems in the following sections. However, we also note that some companies, when faced with problems involving inventory or production line control, have been willing to run system simulations on a mainframe computer for a day or more, offsetting the cost of computation against increased profits resulting from the use of an improved control policy.

The approach we shall take is to calculate approximate values for storage and computing time, making the following assumptions:-

(i) Compile/access time and storage for the program can be ignored.

(ii) Sets of simultaneous linear equations are solved directly by Gaussian elimination. (However, we comment on the use of iterative methods).

(iii) Computer operation times are:-

add	-	2.5 μsec
compare	-	5.0 μsec
multiply	-	5.0 μsec
divide	-	15.0 μsec

The assumptions coincide with these made by Norman (43, Appendix IV) who also used a CDC computer in his study of analytic methods, solving a 2,400 state problem. He has shown that a good estimate of the time taken to solve a set of N simultaneous equations by Gaussian elimination (using the figures from (iii) above) is $7.5N^3$ μsec. He also shows that L iterations of Howard's policy iteration algorithm require a total computing time of $7.5LN^2(M+N)$ μsecs where N is the number of states, and M is the average number of alternative actions for each state. Norman assumes that all variants of his 2,400 state problem can be solved in 10 iterations.

The comparisons are bound to be rough and ready, not only because the computing time is dependent upon the abilities of the programmer, and on the actual computer used, but also because the grouped-state decision structure allows the problem-solver to redefine the problem in many different ways, within constraints set by the original system description. It is therefore very difficult to assess gains in efficiency which might result from the use of grouped-state controls, without also knowing something about the specific problem.

7.2.2 Analytic Methods

From the discussion above, we see that the time taken for one iteration of Howard's policy iteration algorithm is $7.5N^2(M+N)$ μsecs. When N, the number of states, is significantly greater than M, the average number of alternatives in each state, we see that each iteration takes approximately $7.5N^3$ μsecs, i.e. most of the computing time is required by the value determination routine, which solves N simultaneous equations.

Without quoting any figures, we can immediately see that the linear approximation algorithm solving any form of grouped-state problem having N states, is bound to require more computing time than the policy iteration algorithm solving a classical problem having N states. This must be so because the linear approximation algorithm requires at each iteration, the solution of two sets of N simultaneous equations in order to determine a search direction, and the solution of a further two sets of N simultaneous equations (on average) to determine the step length. In the following section we suggest a method for improving the line search subroutine of this algorithm, but it is clear that unless the number of iterations is relatively small (say 2-3 for the grouped-state solution, compared with 10 for the classical solution) the linear approximation algorithm will be less efficient. However, this algorithm is a valid way of solving grouped-state Markov decision problems, and the advantages offered by a simplified optimal control policy (e.g. allowing less complicated real-time control software) might override any disadvantages.

It is difficult to assess how much computing time would be required by the dynamic programming algorithm to determine a stationary grouped-state control policy, unless something is known about the process

transition matrix. This comment also applies to the use of iterative algorithms for solving simultaneous equations, where if g is the mean process cost per cycle, the number of iterations required to calculate stationary forms of π_k and $(v_k - kg1)$ using

$$<\pi_k = <\pi_{k-1}P \qquad 7.1$$

and

$$v_{k-1}> = q> + Pv_k> \qquad 7.2$$

would be determined by the eigenvalues of the process transition matrix. However, we see that the time taken for one iteration of both equations 7.1 and 7.2 (assuming the matrix to be dense) is 15 N^2 μsecs so we can give an indication of computing time for large problems. We note that the cross-linked transfer-line problems have a sparse transition matrix which reduces the computing time to 270N μsec per iteration of both equations 7.1 and 7.2. We summarize some relevant values for storage and computing times in Table 7.1.

	Solution of N simultaneous equations by Gaussian elimination		Time taken for one iteration of equations 7.1 and 7.2 using dense matrix		Time taken for one iteration of equations 7.1 & 7.2 using sparse matrix	
N	Storage (words)	Time	Storage (words)	Time	Storage (words)	Time
10,000	10^8	87 days	10^8	25 mins	2.10^5	2.7secs
2,000	4.10^6	16.7 hours	4.10^6	1 min	4.10^4	0.54sec
1,000	10^6	2.08 hours	10^6	15 secs	2.10^4	0.27sec

Table 7.1: Estimated storage and computing time for three relevant subroutines

We see that for a Markov model having 10,000 states, the storage requirements would almost certainly be greater than the available central memory of most computers, and we would be forced to use disc storage, or similar, which would increase computing time considerably. Table 7.1 indicates that only problems having 1000 or less states could realistically be solved by analytic methods.

7.2.3 Experimental Methods

We have established in the previous section that because of restricted storage and computing time we could not seriously consider solving grouped-state, or classical Markov decision problems having more than 1000 states using analytic methods. We therefore turn to experimental methods for larger problems, and in this section we estimate the time required to solve a grouped-state problem using the results given for the cross-linked transfer-line problems in Chapters 5 and 6. We have already stated in Chapter 2 that the transfer-line problem was specifically chosen to illustrate the new solution methods because, having solved a small (64 or 144-state) problem, it is then possible to increase the model size without changing the structure, therefore allowing the use of the same simulation program and gradient estimator. This means that without solving a large problem, we have an indication of the time which would be required to solve it.

We take as our objective the solution of a problem having 10,000 states (which corresponds to the cross-linked transfer-line configuration shown in Fig. 2.5, with buffer capacities of 24 units). We assume that an estimate of only one partial derivative is required.

The most efficient of the new gradient estimators uses adjoint estimator 3c, and has previously been used to solve standard transfer-

line problems in Chapters 5 and 6. From results given in Chapter 5, we see that gradient estimation program G3c takes about 1.8 m.sec (on average) to generate one simulated cycle of operation. For the 10,000 state problem, we aim to estimate the gradient to about the same accuracy as that obtained for the 64-state problem using 1000 cycle simulations. Previous studies (e.g. Curtiss (59)) have shown that the run length of a simulated Markov process should be proportional to the dimension of its state space, for an estimate to a given accuracy. We therefore choose to run the 10,000 state system for 100,000 cycles, and assume that the stochastic approximation algorithm would produce a solution in no more than 100 iterations. Thus the time required to compute a gradient estimate at each iteration is (100,000 x 1.8 10^{-3})secs = 3 minutes, and the problem would be solved in (100 x 3) minutes = 5 hours. The figures chosen above are unlikely to be very accurate (e.g. each simulation seems unnecessarily long for the accuracy we require, judging by Ho's results, which we mention later) but are nevertheless used to indicate that solutions of large problems can be computed in hours rather than in days or weeks. Also, it is important to note that variance reducing methods have not been mentioned in the previous analysis. We have shown in Chapter 6 that the refinement technique can be used to reduce variance by 75-98%. If we assume that two independent (100-state) subsystem models are used to provide the approximate form of $\bar{\lambda}$, and that this calculation, solved by Gaussian elimination, takes 7.5 secs, then the total time required to produce a gradient estimate to the required accuracy would be 35 seconds (assuming 85% variance reduction) and the overall solution would take 58 minutes.

A more realistic estimate of the computing time required to solve a large problem can be obtained by considering the problems solved by Ho, et al. (39) as illustrations of their buffer sizing algorithm for transfer-lines. When simulating transfer-lines we note that the

computing time per simulated cycle is proportional to the number of machines in the line. This applies whether all machines are linked together in series, or whether two line sections are cross-linked. It follows that a program designed to simulate a transfer-line comprising four machines linked together in series using three buffer stores, requires approximately the same amount of time as (and is also very similar to) one simulating the cross-linked transfer-line configuration shown in Fig. 2.5.

Ho's paper gives details of the simulation run lengths required to provide results which are statistically significant (in the sense that accurate probability distributions are generated, and gradient estimates have low variance) for four problems involving serially linked machines. Moreover, Ho's gradient estimation method, though approximating the gradient using a finite difference formula, bears a close resemblance to program G3c. As Ho does not consider a four machine problem, we take his three machine problem (represented by an 81,608 state Markov model) and, assuming that a computing time of 1.8 m.sec/cycle would be required by program G3c to generate one partial derivative estimate to at least the same accuracy offered by Ho's method, calculate the computing time per estimate using Ho's quoted run length. Thus the total computing time required by program G3c to estimate one partial derivative is $(2.5.10^{6} \times 1.8.10^{-3})$ secs = 1.25 hours. Ho uses these estimates in a non-linear programming algorithm which solves a buffer sizing problem in relatively few iterations. If we assume that 16 iterations are required (this figure was quoted for a larger problem having similar gradients and machine breakdown distributions) then the optimal buffer design can be found in 20 hours. When the refinement technique is used for variance reduction, the total computing time is reduced to 3 hours (assuming 85% variance reduction) which is not unreasonable for such a large problem.

The computing time required to solve a large Markov decision problem using program G3c with variance reduction, can only be estimated by actually solving the problem, or by taking an "educated guess". It is hoped that the rough estimates given in this section can provide some idea of the feasibility of the method.

7.3 Assessment of contributions in relation to the objectives set in Chapter 1

Section 1.4 gives a list of specific contributions, which we need not repeat here. The purpose of this section is to judge the value of the new results bearing in mind the objectives set at the beginning of the thesis.

In Chapter 1, we identified three difficulties associated with the use of existing methods for solving classical Markov decision problems. New solution methods were developed in an attempt to overcome these difficulties, which we repeat below:

(a) Existing analytic methods can only be used to solve classical problems having relatively few (e.g. < 1,000 states). The limiting factor in this case is usually the amount of central computer memory available.

The alternative procedure for general problems involves generating separate, long simulations of a number of system variants and using the responses to indicate the best variant.

(b) The classical control structure allows actions to be chosen independently in each state, resulting in a large number of feasible policies. It might be possible to devise more

efficient solution methods for large problems if the number of policies is deliberately restricted.

(c) When the system description includes many states for which alternative actions exist, the use of the classical control structure results in detailed control policies which record the chosen action for each state. It might therefore prove difficult to implement the optimal control policy using a computer with limited storage facilities.

By defining actions as functions of pre-specified groups of states, we obtain a simpler form of control policy, and effectively overcome difficulty (c), above. Depending on constraints set by the system itself, it is possible to specify beforehand a policy of virtually any complexity ranging from open-loop controls to full state feed-back controls. All solution methods (two analytic and a range of experimental methods) given in this thesis for the grouped-state problem can therefore be viewed as significant contributions.

It is difficult to devise a more efficient algorithm than Howard's policy iteration algorithm for the Markovian decision problem. Its major disadvantage is the amount of storage and time required to solve the "value determination" subproblem. Also we know that the linear approximation algorithm requires more computation than Howard's algorithm at each iteration. Therefore, unless the linear approximation algorithm can be shown to require fewer iterations, (which it might, when applied to large problems having many control parameters) we are unable to claim that difficulty (b) has been overcome.

The thesis appears to make a significant contribution in problem area (a) above. We initially assume that it is not possible to develop

efficient analytic methods for solving problems having many thousands of states, and procede to compare existing experimental methods with those suggested in this thesis. It is helpful to describe the existing methods before making the comparison.

One method exploits prior knowledge of the system to define a set of policies which is likely to include the optimal policy, and this is then found by generating separate simulations of the system, using each policy in the set.

A more systematic approach is possible if the system has parameters which can be varied. For any particular parameter vector it is possible to estimate the gradient (i.e. the vector consisting of partial derivates of the system response with respect to each parameter) and to use the estimated gradient in a steepest ascent algorithm (or similar) climbing the response surface to find the optimal parameter vector. It is normal (see Kushner and Clark (54, Chapter 1) and Ho et al. (39)) to estimate partial derivates by making a small perturbation to each parameter in turn, generating a new simulation using each perturbed set of parameters, and then calculating a noisy finite-difference approximation to each partial derivative. Thus for a model having n parameters, at least (n+1) separate simulations are required to obtain an estimate of the gradient at any given point in parameter space. Though Ho's method is very efficient for the buffer sizing problem, it still uses finite difference approximations, which suffer from the disadvantage that long simulation runs are required to show any appreciable change in response for each (small) parameter perturbation. The use of larger perturbations can result in unacceptable deterministic errors due to inherent non-linearities in the system.

It follows that neither of the existing simulation approaches is

very efficient. The first advantage offered by each of the gradient estimators proposed here is that partial derivatives are estimated directly, i.e. finite-difference approximations are not used. This means that we avoid any deterministic errors, and should be able to use shorter simulation runs than would be required by perturbation methods.

An obvious disadvantage of the recommended forward-time gradient estimator (using adjoint estimator 3c) is that the number of simulations being generated at any instant is a random variable (depending partly on the number of 'additional' runs initiated by the main simulation, and partly on the minimum length of each additional simulation, determined by the transient response of the system). This could mean that more computation is required by this method than by the perturbation methods to provide a gradient estimate to a given accuracy. However, it has been observed that for the examples studied, no more than two simulations (on average) need be generated simultaneously to provide each gradient estimate, and this extra computation is more than offset by the elimination of deterministic errors and by the possibility of shorter simulation runs, as mentioned earlier.

It is important to note that variance reduction methods have not been successfully applied to complex system simulations previously and the use of the refinement technique with the forward-time gradient estimator (G3c) represents a far more efficient experimental procedure than those currently available. It is difficult to determine from the results in this thesis whether the gradient estimates are better employed in a stochastic approximation algorithm or in a non-linear programming algorithm (there are drawbacks to each). Whichever algorithm is used, it is clear that by solving a grouped-state Markov decision problem using the recommended gradient estimator, we have to a great extent overcome difficulties (a) and (c), above.

7.4 Suggestions for further work

7.4.1 Analytic methods using quadratic approximations to the cost function

It was pointed out in Chapter 3 that the linear approximation algorithm is not an ideal solution method for Markov decision problems for two reasons. Firstly, it converges slowly to the solution because the routine determining search directions inevitably causes the sequence of points to zig-zag back and forth. Secondly, it is necessary to evaluate the cost function at an ever increasing number of points near the solution, primarily due to the type of line search subalgorithm used. We would prefer to evaluate the cost function for a Markov decision problem at as few points as possible, because each evaluation requires the solution of s simultaneous equations for an s-state model. It is likely that the use of quadratic approximations could improve both situations.

In Proposition 3.2 we have proved the existence of a convenient expression for the required quadratic approximation using standard Markov chain notation. This information can be used to improve the convergence properties of the existing algorithm in two ways.

(i) Using a standard linear approximation algorithm to locate the region in which a solution can be found, the cost function could be approximated locally by the quadratic function given in equation 3.63. The next point in the sequence would then be determined by minimizing the quadratic function directly (i.e. by calculating $\alpha_{k+1}> = \alpha_k> - [G(\alpha_k)]^{-1} \nabla_\alpha g(\alpha_k)>$, where G(.) is the cost function's Hessian matrix). The rate of convergence near the solution point would be improved by generating successive points in this way.

(ii) Line searches could be carried out using a one-dimensional form of equation 3.63. Thus, given a search direction $(\hat{\alpha} - \alpha_k)$, we could determine the step length τ_k by directly calculating the value of τ which minimizes the function

$$f_\tau(\hat{\alpha}, \alpha_k) \triangleq g(\alpha_k) + \tau.<\pi(\alpha_k),[q(\hat{\alpha}) - \Psi(\hat{\alpha}).z(\alpha_k)]>$$

$$-\tau^2.<\pi(\alpha_k).S(\hat{\alpha} - \alpha_k).\Psi^{-1}(\alpha_k),[q(\hat{\alpha}) - \Psi(\hat{\alpha}).z(\alpha_k)]> \qquad 7.3$$

The first suggestion is likely to offer the greatest benefits but both have the disadvantage that the full matrix inverse $\Psi^{-1}(.)$ is required. However, the work involved in inverting the matrix might be offset by the improved convergence properties of the algorithm.

7.4.2 Adaptive controllers for large Markov chains

In this thesis our approach is to identify the system model and solve the optimal control problem off-line, and then to implement the optimal control policy using some form of real-time controller. An equally valid approach would be to use a computer to carry out all tasks on-line, using some form of adaptive control scheme. Some studies (e.g. Riordon (16), Borkar and Varaiya (17)) have proposed adaptive control schemes which are based on Markov chain models of the underlying process. However, when dealing with systems having many states, we would probably be forced to specify a 'reduced model' by lumping states together, to allow manipulation of the model by one of the relatively small computers which are normally used for real-time control. System identification using this type of reduced model would not be straightforward, as indicated by the following discussion.

The probability of a transition from one state group to another is related to the stationary probability distribution for states within the

group (because transitions to another group would be made from some states but not others). By specifying a different control policy, we affect the stationary distribution, and therefore change the transition probabilities for the reduced model.

Nevertheless it seems reasonable to suppose that some form of adaptive control scheme should be possible, and this would represent an interesting topic for further work.

APPENDIX I

STANDARD 64-STATE AND 144-STATE TRANSFER-LINE PROBLEMS

The real-time control of two transfer-lines linked by cross-conveyors is discussed in Chapter 2. In this appendix, we give details of two specific problems which are used to illustrate various aspects of the Monte Carlo methods described in Chapters 4,5 and 6. These problems have been chosen because they are small enough for solution by direct methods, but are also suitable for simulation studies because they retain the same structure as models of a more realistic size. Most of the information required for these problems has already been given in Chapter 2, but we repeat the relevant data here and give a formal statement of each problem.

Each problem is described with reference to the schematic diagram of two cross-linked lines, i.e. Fig. 2.5. When buffer A and buffer B each have a capacity of 1 production unit, then each individual line (A or B) is described by (2 x 2 x 2 =) 8 states, and the cross-linked system model therefore requires (8 x 8 =) 64 states. Similarly, if buffers A and B each have a capacity of 2 production units, then each line is described by 12 states and the complete system is described by 144 states.

AI.1 The 64-state problem

Assuming the configuration shown in Fig. 2.5, each buffer having a capacity of 1 production unit, we can describe the cross-linked system using 64 states. We have already discussed in some detail (Section 2.3.2) the procedure for identifying operating and repair time distributions for unreliable machines. The distributions for each of

the four can making machines in the 64-state problem are taken to be geometric, and to have the following parameters:

$$B \triangleq \left\{ \begin{array}{l} \text{probability of breakdown (if machine is working)} \\ \text{during next cycle.} \end{array} \right\} = 0.1$$

$$REP \triangleq \left\{ \begin{array}{l} \text{probability of repair (if machine is down)} \\ \text{during next cycle.} \end{array} \right\} = 0.25$$

Section 2.3.4 provides the necessary information for setting up the transition matrix for line A (or B). Using Fig. 2.4 and equation 2.35 with Z = 1 we obtain, substituting $B = B_1 = B_2$ and $REP = REP_1 = REP_2$,

$$P_1 = \begin{bmatrix} 0.75 & 0.25 & & & & & & \\ 0.09 & 0.81 & 0.01 & 0.09 & & & & \\ 0.1875 & 0.0625 & 0.5625 & 0.1875 & & & & \\ & & & & 0.025 & 0.225 & 0.075 & 0.675 \\ 0.675 & 0.225 & 0.075 & 0.025 & & & & \\ & & & & 0.09 & 0.81 & 0.01 & 0.09 \\ & & & & 0.1875 & 0.0625 & 0.5625 & 0.1875 \\ & & & & & 0.25 & & 0.75 \end{bmatrix} \qquad \text{AI.1}$$

and using equation 2.36, with Z = 1, we obtain the 'cost' vector for a two-stage transfer-line:

$$c_1 = [0\ 1\ 0\ 0\ 1\ 1\ 0\ 0]^T \qquad \text{AI.2}$$

Using the state numbering system defined by equations 2.33 and 2.38, we

see that if the 64 x 64 system transition matrix $P(\alpha)$ is given by

$$P(\alpha) = D(\alpha)P \qquad \text{AI.3}$$

then P is the 64 x 64 process transition matrix, which is a nested version of P_1, defined by

$$P = \begin{bmatrix} 0.75[P_1] & 0.25[P_1] & \cdots & \cdots \\ 0.09[P_1] & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \vdots & & & 0.1875[P_1] \\ \cdots & \cdots & [0] & 0.75[P_1] \end{bmatrix} \qquad \text{AI.4}$$

The 'cost' vector for the complete system is the 64-vector c, which can also be written in a partitioned form:-

$$<c = [<c_1 \,|\, <1+<c_1 \,|\, <c_1 \,|\, <c_1 \,|\, <1+<c_1 \,|\, <1+<c_1 \,|\, <c_1 \,|\, <c_1] \qquad \text{AI.5}$$

Control transitions (and therefore the structure of the matrix $D(\alpha)$) are defined by Table AI.1.

Buffer contents at start of cycle		Probability of transferring one unit from buffer B to buffer A
Buffer A	Buffer B	
0	0	0
0	1	$\alpha \in [0,1]$
1	0	0
1	1	0

Table AI.1: Control structure for 64-state problem.

It can be seen that we have defined a grouped-state (probabilistic) control structure because the same control action is specified for each of 16 states (all combinations of states [Up or Down] for the four machines in the system description). Of course, it is possible to translate Table AI.1 into an explicit expression for the matrix $D(\alpha)$, which is essentially the 64 x 64 identity matrix with 16 of its rows replaced by rows of the form $[0....0\ (1-\alpha)\ 0....0\ (\alpha)\ 0...0]$.

Having defined 'costs' (or more precisely 'system output'), transition probabilities and control structure for the system, it remains only to state the optimal control problem which is:

$$\underset{\alpha\in[0,1]}{\text{maximize}} \{J(\alpha)\} \qquad \text{AI.6}$$

where $J(\alpha)$ is the long run expected output of the system per cycle, defined by

$$J(\alpha) = \lim_{T\to\infty} \frac{1}{T+1} E\left\{\sum_{t=0}^{T} \langle x_t, q(\alpha)\rangle\right\} \qquad \text{AI.7}$$

where $q(\alpha) = D(\alpha)Pc$ and $\{x_t\}$ is a sequence of process states.

AI.2 The 144-state problem

In order to demonstrate the use of a model reduction principle known as "state-grouping" in Chapter 6, it is necessary to identify states which are "close" to each other in a cost and transition sense (a full discussion is found in Appendix IV). A cross-linked transfer-line model of realistic size would provide ample opportunity to apply this

principle and it is found that the smallest model likely to benefit from the technique has 144 states. The problem is identical in virtually all respects to the one previously described. The increase from 64 to 144 states is due to an increase in buffer capacity in each line from 1 to 2 production units. The 12 x 12 transition matrix for each two-stage line is defined (using the previously quoted breakdown and repair parameters) by Fig. 2.4. The 12-vector defining 'costs' for a two-stage line is given by equation 2.36, with Z = 2, and the state numbering system defined by equations 2.33 and 2.38 is used to obtain a 144 x 144 'nested' matrix P and the combined 'cost' vector c.

The control structure for this problem is defined by Table AI.2 below.

Buffer contents at start of cycle		Probability of transferring one unit from buffer B to buffer A
Buffer A	Buffer B	
0	0	0
0	1	$\alpha \in [0,1]$
0	2	$\alpha \in [0,1]$
1	0	0
1	1	0
1	2	0
2	0	0
2	1	0
2	2	0

Table AI.2: Control structure for 144-state problem.

The grouped-state Markov decision problem is formally stated in equations AI.6 and AI.7.

AI.3 Solution of the classical 64-state Markov decision problem using Howard's policy iteration algorithm.

We consider the problem described in Section AI.1 but, rather than grouping together 16 states and specifying the same probabilistic control action for each, we allow the action of transferring or not transferring one production unit from buffer B to buffer A to be chosen independently for each of the 16 states in question. The model is identical in all other respects to that described in Section AI.1. Thus the classical 64-state Markov decision problem is to choose a stationary control policy such that the long run expected output of the system per cycle is maximized.

This problem was posed (and solved, using the policy iteration algorithm) in an attempt to indicate benefits which might be realized through the use of cross-conveying systems. The optimal policy gives rise to an improvement in mean system output per cycle of 2.6% compared with that obtained from two lines operated independently. Thus we know that cross-conveyors can improve output even if some grouped-state control schemes indicate little or no improvement.

APPENDIX II

PROOF OF EXISTENCE AND CONTINUITY OF THE DERIVATIVES $g_\alpha(.)$ AND $g_{\alpha\alpha}(.)$

The vector valued function $z(.)$ has been defined in Section 3.4 and is evaluated at any feasible point $\hat{\alpha}$ in continuous k-dimensional parameter space by solving the s-vector equation:

$$\psi(\hat{\alpha})z(\hat{\alpha}) > = q(\hat{\alpha}) > \qquad \text{AII.1}$$

where $\psi(\hat{\alpha}) \triangleq (I - P(\hat{\alpha}))(I - e_s><e_s) + 1><e_s$ AII.2

also by definition,

$$g(\hat{\alpha}) = <e_s, z(\hat{\alpha})> \qquad \text{AII.3}$$

In Chapter 3 we derive first and second order approximations to the mean cost function $g(.)$ near the point $\hat{\alpha}$. These functions have meaning only if $g_\alpha(.)$ and $g_{\alpha\alpha}(.)$ exist, which we now proceed to prove. The following propositions are proved by bringing together a number of well-known results.

Proposition AII.1

The function $z_\alpha(.)$ exists and is a continuous mapping represented by an s x k matrix.

Proof

From equations 3.42 and 3.43 we see that the elements of $P(.)$ and $q(.)$ are affine functions of α. It follows that $\psi(.)$ has elements which

are either constants, or affine functions of α. Furthermore, we see that for any feasible parameter vector $\hat{\alpha}$, $P(\hat{\alpha})$ is a probability matrix i.e. it satisfies

$$\left.\begin{aligned} &P(\hat{\alpha}) \,.\, 1\!\!1> = 1\!\!1> \\ &0 \leq (<e_i, P(\hat{\alpha})e_j>) \leq 1 \\ &\qquad\qquad \text{for } i, j \in \{1, \ldots, s\} \end{aligned}\right\} \qquad \text{AII.4}$$

It has been shown by Paige, et al (51,p.177) that if the matrix $\psi(\hat{\alpha})$ is defined by equation AII.2 using a probability matrix $P(\hat{\alpha})$, then its determinant is non-zero. Thus for any feasible parameter vector $\hat{\alpha}$, $\det(\psi(\hat{\alpha}))$ is non-zero and $\psi^{-1}(\hat{\alpha})$ exists.

If $z(\hat{\alpha})$ has elements $\{z_i(\hat{\alpha}),\ i = 1,\ldots.s\}$, then applying Cramer's rule to equation AII.1 gives

$$z_i(\hat{\alpha}) = \frac{\det(\psi^i(\hat{\alpha}))}{\det(\psi(\hat{\alpha}))}, \text{ for } i = 1, \ldots, s \qquad \text{AII.5}$$

where $\psi^i(\hat{\alpha})$ is the matrix $\psi(\hat{\alpha})$ with its i'th column replaced by $q(\hat{\alpha})$.

For any s x s matrix A, having elements $\{a(i,j)\}$ we have, by definition (Rudin (60, p.232))

$$\det(A) = \sum r(j_1, .., j_s).\ a(1,j_1).\ a(2,j_2).\ldots.\ a(s,j_s)$$

where r is a function of an ordered n-tuple of integers and takes values 1,-1, or 0. The summation extends over all ordered n-tuples of integers $(j_1, \ldots, j_s)$, each integer taken from the set $\{1, \ldots., s\}$.

Thus the determinants of both matrix valued functions, $\psi^i(.)$ and $\psi(.)$ are polynomial functions of the parameters. The basic rules of differentiation can thus be applied to equation AII.5 to confirm the existence and continuity of the partial derivatives $\{\frac{\partial}{\partial\alpha_j} z(.),\ j = 1, \ldots,k\}$. A standard result given by Bartle (55, p.355) states that if all partial derivatives are continuous, then the derivative mapping $z_\alpha(.)$ exists and is represented by an s x k matrix. That $z_\alpha(.)$ is continuous follows from a result (Bartle (55, p.143)) stating that combinations (sums, products, quotients, etc.) of continuous functions are also continuous.

Proposition AII.2

The functions $g_\alpha(.)$ and $g_{\alpha\alpha}(.)$ exist and are continuous.

Proof

Existence and continuity of $g_\alpha(.)$ follows directly from equation AII.3 and Proposition AII.1.

The functions $\psi(.)$ and $q(.)$ can be defined in the following way for any feasible parameter vector, $\hat{\alpha}$, having elements $\{\hat{\alpha}_i,\ i = 1, \ldots, k\}$:

$$\left.\begin{aligned} \psi(\hat{\alpha}) &\triangleq S_1\hat{\alpha}_1 + S_2\hat{\alpha}_2 + \ldots + S_k\hat{\alpha}_k + T \\ q(\hat{\alpha}) &\triangleq u_1\hat{\alpha}_1 + u_2\hat{\alpha}_2 + \ldots + u_k\hat{\alpha}_k + w \\ \text{where}\quad & T, S_i,\ i = 1\ldots,k \quad \text{are real s x s matrices} \\ \text{and}\quad & w, u_i,\ i = 1,\ldots.k \quad \text{are real s-vectors} \end{aligned}\right\} \qquad \text{AII.6}$$

Differentiating both sides of equation AII.1 and rearranging yields

$$\psi(\hat{\alpha})z_\alpha(\hat{\alpha}) = [u_1 \vdots - - - \vdots u_k] - [S_1 z(\hat{\alpha}) \vdots - - - \vdots S_k z(\hat{\alpha})] \qquad \text{AII.7}$$

thus

$$\psi(\hat{\alpha}) z_{\alpha_i}(\hat{\alpha}) = [u_1 - S_i z(\hat{\alpha})], \quad i = 1, \ldots, k \qquad \text{AII.8}$$

It is apparent that the RHS of equation AII.8 is continuous, and Proposition AII.1 implies that its derivative exists and is continuous. Equation AII.8 is of a similar form to equation AII.1 and the proof of Proposition AII.1 can be used to show that the functions $\{z_{\alpha_i}(\cdot),\ i = 1, \ldots, k\}$ and hence $\{g_{\alpha_i}(\cdot),\ i = 1, \ldots, k\}$ are continuously differentiable. Thus the existence and continuity of the function $g_{\alpha\alpha}(\cdot)$ having elements $\{\frac{\partial^2}{\partial\alpha_i \cdot \partial\alpha_j} \cdot g(\cdot)\}$ has been established.

APPENDIX III

EXPERIMENTAL RESULTS FOR THE 3-STATE EXAMPLE DESCRIBED IN CHAPTER 4

Full details of the example and a general discussion of these results are given in Chapter 4. The experiments were designed to illustrate features of the state distribution and adjoint vector estimators. Accordingly, sequences (each component being the average of an ensemble of values) are estimated by each method, using a range of ensemble sizes. It can be seen that the state distribution estimator and the two reverse-time adjoint estimators give rise to sparse vectors when only one simulation is generated. Also, of the two "reverse-time" adjoint estimators, Method 1 is significantly more efficient. Two forward-time methods are used to estimate selected components of the adjoint and relative cost vectors. The first two tables list the calculated sequences of state distributions and adjoint vectors which can be compared with estimated versions which appear later.

t	$<\pi_t$		
0	0	0	1.0000
1	.1000	.6000	.3000
2	.1600	.5000	.3400
3	.1960	.4860	.3180
4	.2176	.4730	.3094
5	.2306	.4657	.3038
6	.2383	.4612	.3005
7	.2430	.4585	.2985
8	.2458	.4569	.2973
9	.2475	.4560	.2965
10	.2485	.4554	.2961

Table AIII.1 : Standard (calculated) sequence of state distribution vectors

t	$<\lambda_t$		
0	21.3277	24.1745	23.9927
1	18.9952	21.8308	21.6490
2	16.6684	19.4852	19.3034
3	14.3508	17.1365	16.9547
4	12.0489	14.7826	14.6008
5	9.7730	12.4200	12.2382
6	7.5404	10.0430	9.8612
7	5.3800	7.6420	7.4600
8	3.3400	5.2000	5.0200
9	1.5000	2.7000	2.5000
10	0	0	0

Table AIII.2 : Standard (calculated) sequence of adjoint vectors

t	Estimate of $<\pi_t$		
0	0	0	1.0000
1	0	1.0000	0
2	0	0	1.0000
3	0	0	1.0000
4	0	0	1.0000
5	0	1.0000	0
6	0	0	1.0000
7	0	0	1.0000
8	1.0000	0	0
9	0	1.0000	0
10	1.0000	0	0

Table AIII.3 : State distribution estimates from 1 run.

t	Estimate of $<\pi_t$		
0	0	0	1.0000
1	.1000	.7000	.2000
2	.2500	.3000	.4500
3	.2500	.4000	.3500
4	.1500	.5500	.3000
5	.1500	.4500	.4000
6	.2000	.5000	.3000
7	.2000	.5000	.3000
8	.2000	.5000	.3000
9	.2000	.3500	.4500
10	.3000	.5000	.2000

Table AIII.4 : State distribution estimates from an ensemble of 20 runs.

t	Estimate of $<\pi_t$		
0	0	0	1.0000
1	.1150	.5990	.2860
2	.1940	.4530	.3530
3	.2220	.4720	.3060
4	.2360	.4530	.3110
5	.2690	.4400	.2910
6	.2550	.4390	.3060
7	.2570	.4480	.2950
8	.2330	.4490	.3180
9	.2410	.4670	.2920
10	.2680	.4450	.2870

Table AIII.5 : State distribution estimates from an ensemble of 1000 runs.

t	Estimate of $<\pi_t$		
0	0	0	1.0000
1	.0956	.5989	.3055
2	.1603	.4963	.3434
3	.1941	.4877	.3182
4	.2110	.4759	.3131
5	.2292	.4671	.3037
6	.2323	.4617	.3060
7	.2389	.4657	.2954
8	.2410	.4573	.3017
9	.2387	.4555	.3058
10	.2454	.4485	.3061

Table AIII.6 : State distribution estimates from an ensemble of 10,000 runs.

t	Estimate of $<\lambda_t$		
0	41.0333	0	0
1	38.5925	0	0
2	35.8806	0	0
3	32.8673	0	0
4	29.5192	0	0
5	25.7991	0	0
6	21.6657	0	0
7	17.0730	0	0
8	11.9700	0	0
9	6.3000	0	0
10	0	0	0

Table AIII.7 : Method 1 - adjoint vector estimates from 1 run.

t	Estimate of $<\lambda_t$		
0	28.9497	17.6586	18.5344
1	25.3161	16.8827	17.2448
2	21.3752	16.7086	14.8371
3	18.5783	15.6536	11.5447
4	16.0421	12.2586	10.9845
5	13.6161	10.0082	9.2563
6	10.6756	7.7755	8.1869
7	7.9968	6.0487	6.2031
8	4.9840	4.6095	3.9445
9	1.9600	2.8700	1.7850
10	0	0	0

Table AIII.8 : Method 1 - adjoint vector estimates from an ensemble of 20 runs.

t	Estimate of $\langle\lambda_t$		
0	21.3969	24.7586	24.2182
1	19.0637	22.4364	21.6928
2	16.5268	20.1566	19.2755
3	14.1032	17.7179	16.9225
4	11.9068	15.1876	14.4859
5	9.8277	12.6745	11.9547
6	7.6995	10.1405	9.5623
7	5.6926	7.5037	7.2868
8	3.6499	5.1472	4.7649
9	1.6184	2.6992	2.3772
10	0	0	0

Table AIII.9 : Method 1 - adjoint vector estimates from an ensemble of 1000 runs.

t	Estimate of $\langle\lambda_t$		
0	21.0112	24.3560	24.1224
1	18.7104	21.9822	21.7429
2	16.3967	19.5909	19.4140
3	14.1559	17.2005	17.0271
4	11.9544	14.8237	14.5933
5	9.7527	12.4704	12.1293
6	7.5551	10.0568	9.7500
7	5.4570	7.5898	7.3809
8	3.4376	5.1548	4.9419
9	1.5308	2.7180	2.4303
10	0	0	0

Table AIII.10: Method 1 - adjoint vector estimates from an ensemble of 10,000 runs.

t	Estimate of $<\lambda_t$		
0	0	0	49.0000
1	0	42.0000	0
2	0	0	14.0000
3	0	0	0
4	0	0	14.0000
5	0	0	0
6	7.0000	0	0
7	0	0	0
8	0	0	56.0000
9	0	0	21.0000
10	0	0	0

Table AIII.11: Method 2 - adjoint vector estimates from 1 run.

t	Estimate of $<\lambda_t$		
0	3.8500	4.5500	8.4000
1	2.4500	5.6000	3.8500
2	4.2000	1.0500	6.6500
3	14.3500	9.8000	0
4	9.4500	4.9000	10.5000
5	10.5000	4.9000	9.1000
6	3.8500	9.4500	5.6000
7	3.1500	1.7500	5.9500
8	2.1000	2.8000	9.1000
9	1.7500	1.7500	3.5000
10	0	0	0

Table AIII.12: Method 2 - adjoint vector estimates from an ensemble of 20 runs.

t	Estimate of $<\lambda_t$		
0	16.7930	16.0090	16.0160
1	16.7650	13.5870	10.7660
2	14.1260	11.8160	22.5260
3	13.4960	14.3080	18.1510
4	11.0670	14.7000	13.9790
5	8.9600	12.5370	13.7130
6	7.6440	8.2460	10.9130
7	5.8730	7.2030	7.7700
8	2.9400	5.4460	5.7400
9	1.5470	2.7370	2.6320
10	0	0	0

Table AIII.13: Method 2 - adjoint vector estimates from an ensemble of 1000 runs.

t	Estimate of $<\lambda_t$		
0	20.2461	27.8173	22.3552
1	17.4097	25.0432	18.6354
2	16.6852	18.9574	19.5251
3	14.3465	16.6649	19.4985
4	11.9560	15.0983	14.6020
5	9.7517	13.3385	12.3865
6	7.1750	9.8651	10.8122
7	5.3893	7.6510	7.6405
8	3.3054	5.3977	4.9665
9	1.5106	2.6754	2.5361
10	0	0	0

Table AIII.14: Method 2 - adjoint vector estimates from an ensemble of 10,000 runs.

t	Component estimate variances (estimated from an ensemble of 1000 runs)		
0	363.7850	524.5836	276.4397
1	299.5836	420.8462	218.0319
2	246.8388	342.8763	171.0577
3	197.8735	267.0383	132.4154
4	151.2789	191.8614	98.1510
5	115.5464	134.3359	71.8336
6	83.2965	88.6167	55.6463
7	52.7642	48.8401	38.7319
8	27.4687	28.3959	23.5268
9	8.6158	11.2579	11.7512
10	0	0	0

Table AIII.15: Method 1 adjoint estimator - analysis of variance of component estimates from a single run.

t	Component estimate variances (estimated from an ensemble of 10,000 runs)		
0	50000.0	100000.0	60000.0
1	10000.0	90000.0	10000.0
2	10000.0	20000.0	20000.0
3	6000.0	8000.0	10000.0
4	3000.0	5000.0	5000.0
5	1000.0	2000.0	2000.0
6	556.7	1000.0	1000.0
7	238.0	472.4	512.4
8	72.3	166.0	156.4
9	15.6	30.9	31.4
10	0	0	0

Table AIII.16: Method 2 adjoint estimator - analysis of variance of component estimates from a single run.

t	Estimate of $\langle e_1, \lambda_t \rangle$	Estimate of $\langle e_3, \lambda_t \rangle$
0	25.00000	28.00000
1	24.00000	26.00000
2	20.00000	22.00000
3	16.00000	18.00000
4	13.00000	14.00000
5	12.00000	12.00000
6	6.00000	8.00000
7	4.00000	4.00000
8	3.00000	5.00000
9	1.00000	1.00000
10	0	0

Table AIII.17: Method 3a - estimates of $\langle e_1, \lambda_t \rangle$, $\langle e_3, \lambda_t \rangle$; $t = 0, 1, \ldots, 10$ from 1 run.

t	Estimate of $\langle e_1, \lambda_t \rangle$	Estimate of $\langle e_3, \lambda_t \rangle$
0	21.50000	24.60000
1	19.65000	22.70000
2	17.55000	20.30000
3	15.20000	17.30000
4	12.35000	15.30000
5	9.95000	12.90000
6	8.10000	10.30000
7	5.65000	7.50000
8	3.55000	5.40000
9	1.60000	2.55000
10	0	0

Table AIII.18: Method 3a - estimates of $\langle e_1, \lambda_t \rangle$, $\langle e_3, \lambda_t \rangle$; $t = 0, 1, \ldots, 10$ from an ensemble of 20 runs.

t	Estimate of $\langle e_1, \lambda_t \rangle$	Estimate of $\langle e_3, \lambda_t \rangle$
0	21.27600	23.90300
1	18.98200	21.61800
2	16.66100	19.20000
3	14.39900	16.89600
4	12.01900	14.60800
5	9.80300	12.30200
6	7.60300	9.90300
7	5.33800	7.47400
8	3.35400	5.02400
9	1.52700	2.50600
10	0	0

Table AIII.19: Method 3a - estimates of $\langle e_1, \lambda_t \rangle$, $\langle e_3, \lambda_t \rangle$; $t = 0, 1, \ldots, 10$ from an ensemble of 1000 runs.

t	Estimate of $\langle e_1, \lambda_t \rangle$	(Estimated) single run variance	Estimate of $\langle e_3, \lambda_t \rangle$	(Estimated) single run variance
0	21.40190	27.2872	24.03240	23.2785
1	19.09110	23.8859	21.72020	20.2351
2	16.68860	20.8161	19.35850	17.1034
3	14.38330	17.7010	17.00740	14.1397
4	12.10070	14.9411	14.63200	11.5367
5	9.79070	11.7637	12.26250	9.0190
6	7.56480	8.4660	9.86440	6.6102
7	5.38890	5.3997	7.45030	4.3876
8	3.36540	2.7877	5.02740	2.5118
9	1.51400	.8932	2.51450	1.0652
10	0	0	0	0

Variances are estimated from an ensemble of 5000 runs.

Table AIII.20 : Method 3a - estimates of $\langle e_1, \lambda_t \rangle$, $\langle e_3, \lambda_t \rangle$; t = 0, 1, ..., 10 from an ensemble of 10000 runs. Single run variance analysis.

Size of ensemble (simulation runs)	Estimate of $\langle e_2, \bar{\lambda} \rangle$	(Estimated) single run variance	(Estimated) single run S.D.	Estimate of $\langle e_3, \bar{\lambda} \rangle$	(Estimated) single run variance	(Estimated) single run S.D.	Computing time required for both estimates (secs)
1	2.0000	5.0986	2.2580	2.0000	5.1344	2.2659	.961
20	2.3300			2.1400			.996
1000	2.8166			2.6336			1.799
10,000	2.8493			2.6667			9.370
100,000	2.8510			2.6696			84.078
True values	$\langle e_2, \bar{\lambda} \rangle = 2.8636$			$\langle e_3, \bar{\lambda} \rangle = 2.6818$			

Variance figures apply to an estimate obtained from one simulation run.

Table AIII.21 : Method 3c - estimating second and third elements of $\bar{\lambda}$ without application of 'Russian Roulette'.

APPENDIX IV

REDUCED MARKOV MODELS

In his book on heuristic procedures in dynamic programming (43, Chapter 6), Norman describes a technique for reducing the number of states in a Markov model by identifying problem characteristics. His aim in formulating the technique is to reduce model complexity proportionately more than he loses information about the system. He procedes to calculate (using Gaussian elimination) a solution to equation AIV.1, below:

$$g^*|1\rangle + |v^*\rangle = |q^*\rangle + P^*|v^*\rangle \qquad \text{AIV.1}$$

where g^* is the mean process cost (per cycle) incurred by the reduced system. P^* and q^* are the transition matrix and the one-stage expected cost vector for the reduced system.

In fact for a reduced chain having m states, he actually calculates the m-vector $|\bar{v}^*\rangle = [|v^*\rangle - \langle e_m, v^*\rangle .|1\rangle]$ as described in Section 2.2.4. Using $\bar{v}^*$ and a suitably defined function ϕ, he obtains an approximate form of $\bar{v}$ which we shall denote by the s-vector $\hat{v}$. Thus $\hat{v} = \phi(\bar{v}^*)$ and $\bar{v}$ is given by

$$|\bar{v}\rangle = [|v\rangle - \langle e_s, v\rangle .|1\rangle] \qquad \text{AIV.2}$$

where v is the solution of

$$g|1\rangle + |v\rangle = |q\rangle + P|v\rangle \qquad \text{AIV.3}$$

and g, q and P relate to the complete system description having s states. Norman goes on to use $\hat{v}$ in an analytic procedure which produces suboptimal solutions to classical Markov decision problems.

It turns out that Norman's results are of immediate use in connection with the variance reduction methods of Chapter 6. The refinement technique described in Section 6.4 requires an approximate form $\hat{v}$, of the relative cost vector $\bar{v}$, (though in Chapter 6 they are denoted by $\hat{\lambda}$ and $\bar{\lambda}$) which is defined by equations AIV.2 and AIV.3.

Norman identifies six characteristics of a state description, and shows how those characteristics, where they exist, can be used to form a reduced model. In the following sections, we describe two state reduction techniques which are most appropriate to transfer-line studies.

AIV.1 Independent sub-systems

The flow diagram for the independent sub-systems technique is shown in Fig. AIV.1. By ignoring some of the interaction which can take place in the complete model we reduce the model to two (or more) smaller systems whose relative cost vectors can be computed separately. For a cross-linked transfer-line model, lines A and B can immediately be identified as two suitable subsystems. If the complete model has 10,000 states then application of this method allows the calculation of an approximate 10,000-element vector $\hat{v}$, by solving two sets of 100 simultaneous equations using Gaussian elimination. In fact, as the two subsystems A and B are identical in this case, we only have to solve a single set of 100 equations. For the cross-linked line model, we have defined the one-to-one function f, (appearing in Fig. AIV.1) by equation 2.38. The validity of additive values assuming no interaction between

subsystems, i.e. the last operation shown in Fig. AIV.1, is proved by Norman (43, Appendix I).

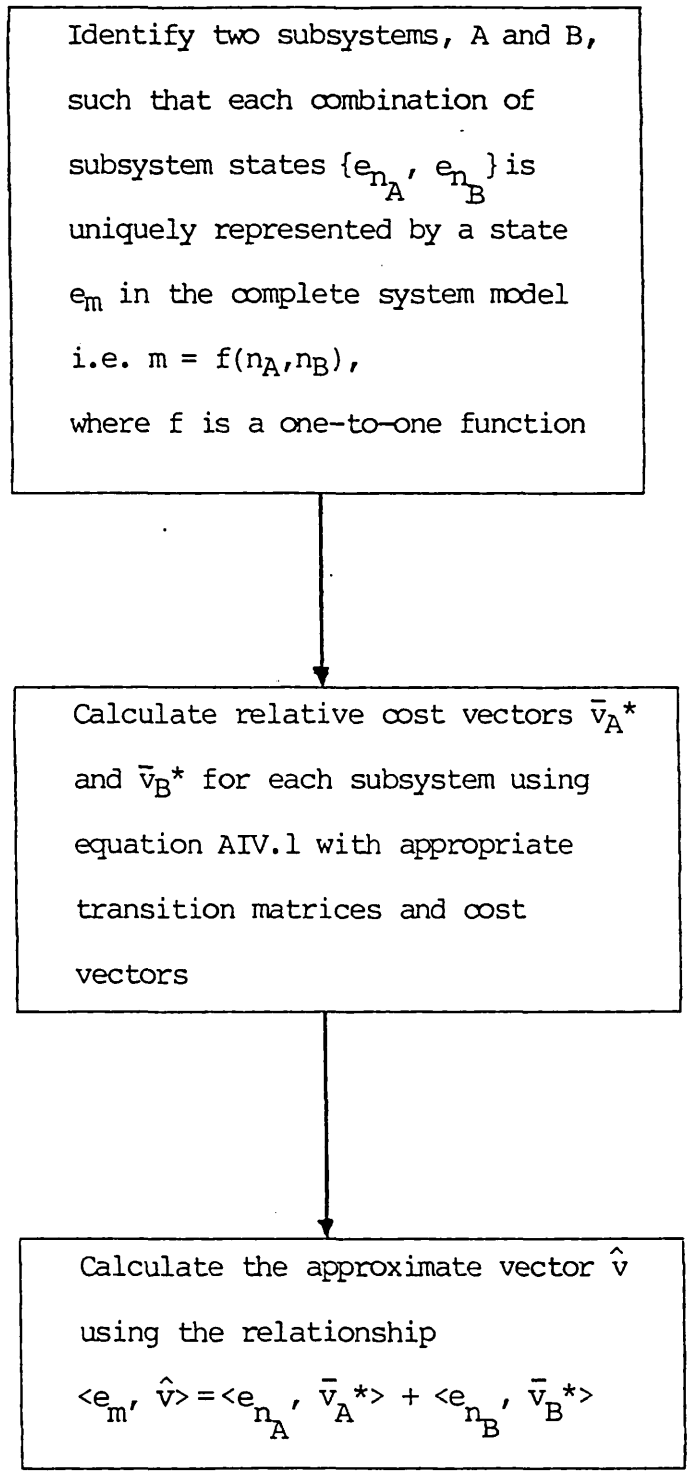


Fig. AIV.1 Independent sub-systems analysis

AIV.2 State Grouping

We define the technique using the flow diagram shown in Fig. AIV.2

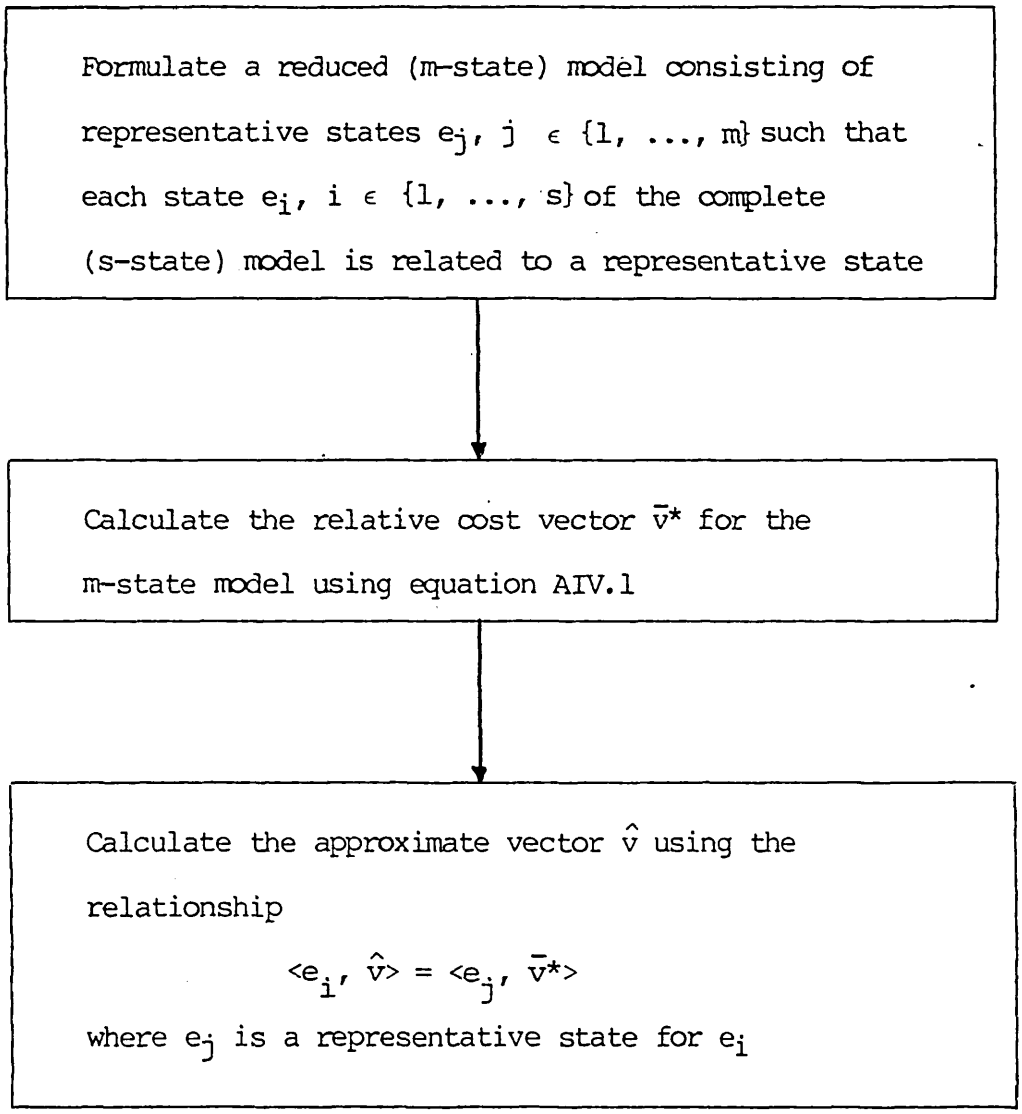


Fig. AIV.2. State-grouping model reduction method.

A reduced model is obtained by considering certain states as representative states for their neighbours. We understand "neighbours" to mean states which are similar in that transitions are made to the same, or similar states, with similar probabilities, and at similar costs. Thus for a given parameter vector $\hat{\alpha}$, if the system transition matrix is $P(\hat{\alpha})$, having elements $\{p_{ij}(\hat{\alpha})\}$, and the one-stage expected

cost vector is $q(\hat{\alpha})$ having elements $\{ q_i(\hat{\alpha}) \}$, then state e_i is a neighbour of state e_j if $p_{ik}(\hat{\alpha}) \simeq p_{jk}(\hat{\alpha})$ for most k, and $q_i(\hat{\alpha}) \simeq q_j(\hat{\alpha})$.

We illustrate this procedure by giving details of the reduced model used in Section 6.5.2. The state grouping principle is used to reduce the standard 144-state transfer-line model to a 64-state model. The example is to be simulated using a nominal control parameter $\hat{\alpha} = 0$, which implies that no simulated transfers are possible between the two lines. Thus the 12-state description of each independent line can be reduced to 8 states implying a reduced model for the total system having 64 states.

The following comments are made with reference to the 12-state single line model described by Fig. 2.4 and equation 2.36. Comparing transition costs and probabilities, it can be seen that states e_9, e_{10}, e_{11} and e_{12} can represent neighbouring states e_5, e_6, e_7, and e_8 respectively. A suitable reduced form of the standard 144-state model thus turns out to be the standard 64-state model.

It is convenient that in this instance we can apply the state grouping principle to single-line descriptions rather than to the complete 144-state model. Having reduced two independent 12-state models to two independent (identical) 8-state models, we can calculate the 144-vector $\hat{v}$ using the following procedure.

(i) Solving equation AIV.1 for an 8-state model we obtain the vector $\bar{v}_8^*$. This is the relative cost vector for each identical reduced model.

(ii) Using the previous definition of neighbouring states we can "expand" the 8-element vector $\bar{v}_8^*$ into a 12-vector $\bar{v}^*$, i.e. $\bar{v}^*$ is

calculated using:

$$\left.\begin{aligned} \langle e_i, \bar{v}^* \rangle &= \langle e_i, \bar{v}_8^* \rangle, \quad i = 1, \ldots, 8 \\ \langle e_i, \bar{v}^* \rangle &= \langle e_{(i-4)}, \bar{v}_8^* \rangle, \quad i = 9, \ldots, 12 \end{aligned}\right\} \qquad \text{AIV.4}$$

(ii) $\hat{v}$ is obtained from $\bar{v}^*$ using the following equations

$$\langle e_m, \hat{v} \rangle = \langle e_{n_A}, \bar{v}^* \rangle + \langle e_{n_B}, \bar{v}^* \rangle \qquad \text{AIV.5}$$

where $n_A = 1, \ldots, 12; \quad n_B = 1, \ldots, 12$

and m is found by making suitable substitutions in equation 2.38 i.e.

$$m = [(n_A - 1) \times 12] + n_B \qquad \text{AIV.6}$$

It can be seen that the state grouping technique has been applied without difficulty to this example, primarily because of the lack of subsystem interaction. When a non-zero nominal control parameter is specified it becomes necessary to identify representative states from the full 144-state Markov model.

REFERENCES

1. N.A.J. Hastings and D. Sadjadi, "Markov Programming with Policy Constraints", European Journal of Operational Research, 3, 253-255; 1979.
2. R. Bellman, "Dynamic Programming", Princeton Univ. Press; 1957.
3. R. Bellman and S.E. Dreyfus, "Applied Dynamic Programming", Princeton Univ. Press; 1962.
4. D.C. Collins, "Reduction of Dimensionality in Dynamic Programming via the Method of Diagonal Decomposition", J. Math. Anal. Appl., 30, 223-234; 1970.
5. D.C. Collins and A. Lew, "Dimensional Approximation in Dynamic Programming by Structural Decomposition", J. Math. Anal. Appl., 30, 375-384; 1970.
6. D.H. Jacobson and D.Q. Mayne, "Differential Dynamic Programming", Elsevier Publishing Co., Inc.; 1970.
7. R.A. Howard, "Dynamic Programming and Markov Processes", Wiley; 1962.
8. A.S. Manne, "Linear Programming and Sequential Decisions", Management Sci., 6, 259-267; 1960.
9. P. Wolfe and G.B. Dantzig, "Linear Programming in a Markov Chain", Operations Res., 10, 702-710; 1962.
10. H. Mine and S. Osaki, "Markovian Decision Processes", Elsevier Publishing Co., Inc.; 1970.
11. D.J. White, "Dynamic Programming, Markov Chains, and the Method of Successive Approximations", J. Math. Anal. Appl., 22, 373-376; 1963.
12. P. Varaiya, "Optimal and Suboptimal Stationary Controls for Markov Chains", I.E.E.E. Trans. Auto. Control, AC-23, 3, 388-394; 1978.

13. K.J. Åström, "Optimal Control of Markov Processes with Incomplete State Information", J. Math. Anal. Appl., 10, 174-205; 1965.

14. R.D. Smallwood and E.J. Sondik, "The Optimal Control of Partially Observable Markov Processes over a Finite Horizon", Operations Res., 1071-1088; 1972.

15. E.J. Sondik, "The Optimal Control of Partially Observable Markov Processes over the Infinite Horizon : Discounted Costs", Operations Res., 26, 282-303; 1978.

16. J.S. Riordon, "Adaptive Control of Discrete State Markov Processes", Ph.D. Thesis, Univ. of London, England; 1967.

17. V. Borkar and P. Varaiya, "Adaptive Control of Markov Chains I : Finite Parameter Set", I.E.E.E. Trans. Auto. Control, AC-24, 6; 1979.

18. K.D. Tocher, "The Art of Simulation", English Universities Press, London; 1963.

19. J.R. Emshoff and R.L. Sisson, "Design and Use of Computer Simulation Models", Macmillan, New York; 1971.

20. J.P.C. Kleijnen, "Statistical Techniques in Simulation", Marcel Dekker, Inc., New York; 1974.

21. D.L. Iglehart, "Simulating Stable Stochastic Systems, VII : Selecting the Best System", TIMS Studies in the Management Sciences, 7, 37-49; 1977.

22. E.J. Dudewicz and N.A. Zaino, Jr., "Allowance for Correlation in Setting Simulation Run-Length via Ranking-and-Selection Procedures", TIMS Studies in the Management Sciences, 7, 51-61; 1977.

23. R.E. Bechhofer, "A Single-Sample Multiple-Decision Procedure for Ranking Means of Normal Populations with Known Variances", Annals of Mathematical Statistics, 25, 16-39; 1954.

24. H. Robbins, M. Sobel, and N. Starr, "A Sequential Procedure for Selecting the Largest of k Means", Annals of Mathematical

Statistics, 39, 88-92; 1968.

25. E.J. Dudewicz and N.A. Zaino, Jr., "Ranking and Selection Procedures with Correlated Observations and Simulation", Proceedings of Computer Science and Statistics, Seventh Annual Symposium on the Interface, Iowa State Univ., Ames, Iowa, 150-155; 1973.

26. E.J. Dudewicz and S.R. Dalal, "Allocation of Observations in Ranking and Selection with Unequal Variances", Sankhya, Series B, 37, 27-78; 1975.

27. Y.C. Ho, "Parametric Sensitivity of a Statistical Experiment", I.E.E.E. Trans. Auto. Control, AC-24, 6, 982-983; 1979.

28. D.S. Burdick and T.H. Naylor, "Response Surface Designs", published in "The Design of Computer Simulation Experiments", edited by T.H. Naylor, Duke University Press, Durham, N.C.; 1969.

29. H. Kahn, "Use of Different Monte Carlo Sampling Techniques", published in "Symposium on Monte Carlo Methods", edited by H.A. Meyer, Wiley; 1954.

30. J.H. Halton, "A Retrospective and Prospective Survey of the Monte Carlo Method", SIAM Rev., 12, 1-63; 1970.

31. J.M. Hammersley and D.C. Handscomb, "Monte Carlo Methods", Methuen, London; 1964.

32. R. Wild, "Mass-Production Management", Wiley; 1972.

33. G.C. Hunt, "Sequential Arrays of Waiting Lines", Operations Research, 4, 674-683; 1956.

34. F.S. Hillier and R.W. Boling, "Finite Queues in Series with Exponential or Erlang Service Times - A Numerical Approach", Operations Research, 15, 286-301; 1967.

35. M.C. Freeman, "The Effects of Breakdowns and Interstage Storage on Production Line Capacity", Journal of Industrial Engineering, 15, 194-200; 1964.

36. J.A. Buzacott, "Automatic Transfer Lines with Buffer Stocks", Int. Jnl. Prod. Res., 5, 3, 183-200; 1967.

37. J.A. Buzacott, Ph.D Thesis, Dept. of Eng. Production, University of Birmingham, England; 1967.

38. S.B. Gershwin and I.C. Schick, "Analytic Methods for Calculating Performance Measures of Production Lines with Buffer Storages", Proceedings of the I.E.E.E. Conference on Decision and Control, 618-624; 1978.

39. Y.C. Ho, M.A. Eyler and T.T. Chien, "A Gradient Technique for General Buffer Storage Design in a Production Line", report for Harvard University and The Charles Stark Draper Laboratory, Inc., published as "A Gradient Technique for General Buffer Storage Design in a Serial Production Line", Int. J. Prod. Control, 17, 6, 557-580; 1979.

40. Y.C. Ho and C. Cassandras, "Computing Co-State Variables for Discrete Event Systems", Proc. 19th I.E.E.E. Conf. on Decision and Control, Albuquerque, N.M.; 1980.

41. G.W. Brilman, "Mass Conveying of Tinplate Cans", International Technical Committee, Annual Technical Conference, Chicago, paper TD.329; 1973.

42. W.I. Zangwill, "Nonlinear Programming : A Unified Approach", Prentice-Hall, Englewood Cliffs, N.J.; 1969.

43. J.M. Norman, "Heuristic Procedures in Dynamic Programming", Manchester University Press; 1972.

44. C.R. Heathcote, "Probability : Elements of the Mathematical Theory", George Allen and Unwin Ltd.; 1971.

45. J.G. Kemeny and J.L. Snell, "Finite Markov Chains", D. Van Nostrand Co., Inc., Princeton, N.J.; 1960.

46. R.H. Arnold, R.S. Goldsmith and H. Fanshawe, "Can Line Performance Analysis", Management Services Dept. Report No. 215, Metal Box p.l.c.

47. G.B. Dantzig, "Linear Programming and Extensions", Princeton University Press, Princeton, N.J.; 1963.

48. D.G. Luenberger, "Introduction to Linear and Nonlinear Programming", Addison - Wesley Publishing Co., Inc.; 1973.

49. R.L. Zahradnik, "Theory and Techniques of Optimization for Practicing Engineers", Barnes and Noble, Inc.; 1971.

50. E. Polak, "Computational Methods in Optimization", Academic Press; 1971.

51. C.C. Paige, G.P.H. Styan and P.G. Wachter, "Computation of the Stationary Distribution of a Markov Chain", J. Statist, Comput. Simul., 4, 173-186; 1975.

52. H.J. Kushner, "Introduction to Stochastic Control", Holt, Rinehart and Winston, New York; 1971.

53. J.L. Farah, "Optimization of Stochastic Systems", Ph.D Thesis, University of London, England; 1977.

54. H.J. Kushner and D.S. Clark, "Stochastic Approximation Methods for Constrained and Unconstrained Systems", Applied Mathematical Sciences, Vol. 26, Springer-Verlag, New York; 1978.

55. R.G. Bartle, "The Elements of Real Analysis", Wiley; 1976.

56. D.C. Handscomb, "Monte Carlo Techniques : Theoretical", published in "The Design of Computer Simulation Experiments", edited by T.H. Naylor, Duke University Press, Durham, N.C.; 1969.

57. W.A. Moy, "Monte Carlo Techniques : Practical", published in "The Design of Computer Simulation Experiments", edited by T.H. Naylor, Duke University Press, Durham, N.C.; 1969.

58. W.S. Dorn and D.D. McCracken, "Numerical Methods with Fortran IV Case Studies", Wiley; 1972.

59. J.H. Curtiss, "A Theoretical Comparison of the Efficiencies of Two Classical Methods and a Monte Carlo Method for Computing One Component of the Solution of a Set of Linear Algebraic Equations",

published in "Symposium on Monte Carlo Methods", edited by H.A. Meyer, Wiley; 1954.

60. W. Rudin, "Principles of Mathematical Analysis", McGraw-Hill; 1976.