

COMPUTING PROCEDURES FOR A
LEARNING MACHINE

A thesis presented for the degree of
Doctor of Philosophy in Electrical Engineering
in the University of Canterbury,
Christchurch, New Zealand

by

P.M. CASHIN, B.E.(Hons), M.E.(Dist.)

1970

0
325.5
C338
1970

ACKNOWLEDGEMENTS

I am deeply indebted to Professor J.H. Andreae who has been my supervisor and has provided continual guidance, support and enthusiasm. I have also had valuable assistance from his methodically indexed library of papers and books.

Unfortunately, it is impossible to individually acknowledge all the people who have helped me through discussion, argument and cooperative effort during the course of this work. In particular the staff and post-graduate students in the Electrical Engineering Department have contributed; many actively, all by good will.

I am grateful to the University Grants Committee for their scholarship which has supported me through this work.

Finally, I am grateful to my wife, Trish, for her patience and interest in my work.

TABLE OF CONTENTS

	<u>Page</u>
<u>CHAPTER 1:</u> INTRODUCTION	1-1
1-1 Introduction	1-1
1-2 Path Finding	1-5
1-3 Stochastic Learning Automata	1-8
1-4 Rote learning and Markov Process Theory	1-9
 <u>CHAPTER 2:</u> THE BANDIT ALGORITHM FOR MINIMUM COST PATH FINDING WITH INCOMPLETE COST INFORMATION	
2-1 Introduction	2-1
2-2 Problem Statement	2-1
2-3 Illustrative Example	2-2
2-4 The Two Armed Bandit Problem	2-5
2-5 The BANDIT Algorithm	2-7
2-6 Some Comparative Results	2-10
2-7 Extensions to Path Finding	2-13
2-8 An Admissible Algorithm	2-13
2-9 Arc Cost Estimates	2-15
2-10 On-line Algorithm	2-15
2-11 Convergence Theorem	2-16
2-12 Applications of the BANDIT Algorithm	2-16
2-13 Results	2-18
2-14 Conclusions	2-25
References	2-27

<u>CHAPTER 3: STOCHASTIC LEARNING AUTOMATA</u>	<u>Page</u>
3-1 Introduction	3-1
3-2 Notation	3-5
3-3 Modified Linear Reinforcement Procedure	3-8
3-4 The BANDIT Algorithm	3-9
3-5 Environments with Perception and Performance Measures	3-16
3-6 Results	3-25
3-8 Conclusions	3-35
References	3-37

CHAPTER 4: ROTE LEARNING AND MARKOV PROCESSES

4-1 Table Building	4-1
4-2 Operator Selection Strategy	4-6
4-3 Planning from Rote Learning	4-10
4-4 Interaction as a Markov Process	4-11
4-5 An Example of Optimal Policy Failure	4-20
4-6 Stochastic Simulation	4-26
4-7 Operator Decision Procedure	4-33
4-8 Expectance Function	4-36
4-9 Operator Decision based on Expectance	4-40
4-10 Expectance Entry in the Rote Learning Table	4-49
4-11 BANDIT-EXPECTANCE Machine	4-52
4-13 Fox and Dogs Game	4-55
4-14 Extensions to the Rote Learning Table	4-62
References	4-67

<u>CHAPTER 5: CONCLUSIONS</u>	<u>Page</u>
5-1 Reinforcement Learning	5-1
5-2 Against Reinforcement Learning	5-2
5-3 Learning by Being Told	5-3
5-4 Why the Gap?	5-7
5-5 Grafting Learning Ability onto a Program	5-8
5-6 Summary of Main Points	5-10

APPENDICES

APPENDIX A: LINKNET: A Structure for Computer

Representation and Solution of Network Problems	
Abstract	A-1
A-1 Introduction	A-2
A-2 The Basic Structure	A-3
A-2.1 Graphs	A-3
2.2 LINKNET Elements	A-4
2.3 Lists	A-7
2.4 Access to arc attributes	A-10
2.5 Access to node attributes	A-10
A-3 Creation of LINKNET	A-10
3.1 Construction of a LINKNET Structure	A-10
A-4 Applications of LINKNET	A-13
4.1 Minimum length path finding	A-13
4.2 Finding Meshes and Spanning Trees	A-17

	<u>Page</u>
A-5 Conclusions	A-22
References	A-24

APPENDIX B: GRAPHIC DISPLAY SYSTEM

B-1 Introduction	B-1
B-2 Basic System Requirements	B-4
B-3 The Edit Phase	B-5
- Direct Draw Facility	B-5
- Display Language Facility	B-7
B-4 The Run Time Phase	B-10
- Display Order Inter	B-11
- Display File Editing	B-15
B-5 Editor-Interpreter	B-19
References	B-26

APPENDIX C: Analytic Calculation of BANDIT

Selection Probability for Normal Probability
Densities.

APPENDIX D: The BANDIT Algorithm in Heuristic Search Algorithms.

CHAPTER ONE

INTRODUCTION

CHAPTER ONE1 - 1 INTRODUCTION

This thesis presents the highlights of work done in the field of artificial intelligence - more particularly machine learning. Artificial intelligence research is accepted [3] as a wide ranging discipline, and no attempt will be made to define or delimit it. The areas of most importance to this thesis are, heuristic programming, problem solving and associated learning models.

Such important areas as pattern recognition are scarcely mentioned in this thesis; this is not to imply that such areas do not contribute to or supplement the main theme of machine learning. It is simply that the work reported has contributed no new concepts in these areas, or linked them any closer to machine learning.

The view has been taken that every learning machine must face the problem of continually having to decide on an action on the basis of some current set of collected data and deductions. Each action can be thought of as producing a 'value'. The problem is that the estimated 'value' of each action is based on the current data, while each action may produce a 'side effect' of contributing more data.

This problem is exemplified by the 'Dual Control' problem [4] and in its most basic form by the 'Two Armed Bandit Problem' [2]. The Two Armed Bandit Problem is first considered in Chapter 2, where the BANDIT algorithm is first introduced. The BANDIT algorithm is not only contributed as an algorithm for solving the Two Armed Bandit Problem, but it is designed to be a basic mechanism in the learning machine faced with the more complex and general problem outlined above.

The heuristic that the BANDIT algorithm is based on can be stated like this:

If one of a number of alternatives has a probability 'p' of being the best alternative, then choose this alternative 100.p% of the time.

To assess the probability 'p' of one alternative being better than any other, it is necessary to know not only the estimated mean 'value' of each alternative but also the probability density of these mean 'value' estimates. The BANDIT algorithm provides a concise computational procedure to perform this decision process.

The applications, implementation and results from using the BANDIT algorithm form a central core to this thesis.

The second contribution that plays a major part in this thesis is the 'expectance' function. This function is based on the 'expectation' used by Andreade [1] and Gaines and Andreade [6] in the STeLLA learning machine. It is similar to the expectimaxing scheme proposed by Michie and Chambers [9].

The purpose of the expectance function can be thought of (for now) as a way of assessing an action's long term 'value'. That is, not only is the immediate 'value' resulting from the use of the action considered, but also account is taken of the future actions that will become available, and of their expectance functions or expected 'values'.

The development contributed by the expectance function is its generality, and equally important is its recursive formulation and on-line evaluation. The expectance function is introduced near the end of Chapter 3, and is fully discussed in Chapter 4.

Just as important as the BANDIT algorithm and the expectance function themselves, is their use in linking together and extending several distinct areas of current research interest. The three main areas concerned are:

- Path finding (graph searching) - considered from a particular point of view where incomplete information is involved,

- . Stochastic automata - with the introduction of an extended problem class for these machines, and
- . Markov process theory and its use in the development of a rote-learning table-based learning machine.

These topics are dealt with in Chapter 2, 3 and 4 respectively. A brief 'over-view' of these topics is given in this introductory chapter under sections 1-2, 1-3 and 1-4.

The function of the appendices is two fold. First they contain some support material that is not appropriate 'in-line', but more important they contain some original material of their own. This material is not included in the main body since it is concerned with computational tools that have been used (transparently) to develop the algorithms and examples contained in the main body. The two main topics in this class are:

1. A technique based on linked list structures that enables problems involving networks or graphs to be implemented in a rather uniform manner. It is not so much that the techniques involved are in any way new, but rather that the particular way of applying the techniques to the network itself - rather than to the various information structures that may arise in the course of a particular problem - leads to structural and procedural convenience. The ideas here have been developed in conjunction with M.R. Mayson and R. Podmore who have

used the technique on several power system problems.

2. A discussion of work done in implementation of a graphical display system for the Electrical Engineering Department's EAI 640 computer. Although this display system was developed from scratch in cooperation with M.R. Mayson, the details of this work are not considered relevant to this thesis. The philosophy developed is considered relevant however and is based on a large effort devoted to establishing a framework for the software.

1 - 2 PATH FINDING

Chapter 2 is concerned with a particular class of path finding problems. Briefly these problems involve repeated traversal of the minimum cost path that can be found on the basis of current (incomplete) arc or path-segment cost information. This is combined with the updating of the arc cost information for those arcs that are traversed, on any one path traversal. This problem has been given the name 'on-line' path finding.

Chapter 2 is written in the form of a paper describing an operations research technique for this class of problem.

The presentation in Chapter 2 is thus rather closer to the basic problem than if the sophisticated heuristic graph searching techniques used by artificial

intelligence workers had been explicitly employed. It should be made clear however that by the very nature of graph searching in problem solving or game playing they often fall into the 'on-line' path finding class.

Consider for example a small board game where it is possible for a machine to search enough of the game graph (tree) to establish that it can not possibly win if the opponent plays optimally. An example of such a problem is considered near the end of Chapter 4, with the French Military Game or Fox and Dogs.

In such situations we would like the machine to make a move that maximized its chances of a win - that is, try to put the opponent in a position where he is most likely to blunder. Such performance is just not possible by many successful tree searching programs, since the basis of back-tracking up the game tree is a mini-max strategy. On the other hand the efficiency of the search (the work that it involves) is very dependent on the search strategy and this aspect has received considerable attention [7][10][11].

A similar problem can occur in situations where a complete search is not possible by any strategy. In this (normal) case the usual technique is to back-track up the game tree from an estimation of the value or merit of the various terminal nodes that have been established. Probably the best known program of this

form is Samuel's checker player [12]. An evaluation of various search techniques is given by Slagel and Dixon [14].

The problem arises not from the search and back up procedures themselves but occurs as soon as the estimates used for the value of each node are allowed to be learned by the machine from its own experience. In fact as soon as the learning is directly derived from the machine's own play we have an 'on-line' path finding problem. Chapter 2 shows that without an algorithm such as the BANDIT algorithm the learning process in such cases is liable to get 'stuck' below the optimal performance level.

The above comments apply equally well to problem solving and theorem proving - except that in these cases it may only be required to search once for a solution. In other words the information update from one solution to the next is not present. In such cases the 'on-line' path finding problem does not exist and the best available estimate gives the best that can be achieved. There is no benefit from the 'side-effect' of the actions giving more data.

1 - 3 STOCHASTIC LEARNING AUTOMATA

Stochastic automata have recently been receiving attention as models for learning behaviour and a survey of this work is given by Fu (1970) [5]. Chapter 3 is concerned with this approach to learning machines, starting with a brief introduction to the current established work.

One recent scheme in particular [13] is then developed and shown to be similar to a BANDIT algorithm stochastic learning automaton, which is introduced at this point. A benefit of this is that a proof of convergence is given for the first automata scheme [13] and a modified BANDIT automaton can be derived which falls within the scope of this proof.

Stochastic learning automata schemes are viewed in Chapter 3 as procedures. For this point of view a notation used in computer algorithm formulation is shown as an attractive method for presenting the stochastic learning automata procedures.

Finally the environment-automaton interaction is generalized to enable this approach to tackle the class of problems considered by several 'heuristic programming' schemes (for want of a better definitive). STeLLA [1,6] in particular tackles such a class of problems.

1 - 4 ROTE LEARNING AND MARKOV PROCESS THEORY

The basic memory structures and strategy used by STeLLA were seen as similar to work on Markov process theory developed by Howard [8]. With this starting point an attempt was made to bridge this gap by building from the Markov theory towards the STeLLA strategy. Unfortunately the complexity of the STeLLA heuristics and special purpose parameters proved too great to allow the theory to meet up with the STeLLA implementation.

By working in reverse a very basic STeLLA structure was extracted in order to move closer to the Markov theory. This basic structure was (eventually) formed into the BANDIT-EXPECTANCE algorithm as presented in Chapter 4. At this point the algorithm proved of enough interest in its own right - and the road back to STeLLA rather torturous - that the linking of the Markov theory through to the BANDIT-EXPECTANCE algorithm was considered as replacing the original objective.

Chapter 4 briefly presents the relevant Markov theory and develops from this to end up with the BANDIT-EXPECTANCE algorithm. Throughout this development the idea of a rote learning table is used to tie the presentation together.

The idea of a rote learning table is a simplification of the 'control policy' which STeLLA employed. In the course of Chapter 4 the rote learning table

progresses from a simple record of events, through to a more functional 'control policy' form. The structure of the rote learning table is not too important in Chapter 4, except that it contains the learning machine's long term memory. However, the format is developed in such a way as to be extensible, and an indication of such future directions completes Chapter 4.

REFERENCES

- 1 Andreae, J.H., Learning Machines: A Unified View.
Encyclopaedia of Linguistics Information and
Control. . Ed. Meethan,A.R. and Hudson,R.A.
Pergamon Press, 1969.
- 2 Bellman,R. A Problem in the Sequential Design of
Experiments. SANKHYA Vol.16, parts 3 and 4, 1956.
pp 221-229.
- 3 Feigenbaum,E.A. Artificial Intelligence: Themes
in the Second Decade. Stanford Artificial
Intelligence Project Memo AI-67. Invited paper
IFIP68 Congress, Edinburgh, Aug. 1968.
- 4 Fel'dbaum,A.A. Dual Control Theory I-IV in Optimal
and Self-Optimizing Control, Ed. Oldenburger,R.
M.I.T. Press, 1966.
- 5 Fu,K.S. Stochastic Automata as Models of Learning
Systems. pp 393-431, Adaptive, Learning and
Pattern Recognition Systems, Ed. Mendel,J.M. and
Fu,K.S. Academic Press, 1970.
- 6 Gaines,B.R. and Andreae,J.H. A Learning Machine
in the Context of the General Control Problem,
3rd IFAC Congress, London, June 1966. (Inst. Mech. Eng.)

- 7 Hart, P., Nilsson, N., Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Trans. on Sys. Sci. Cybernetics, July 1968, pp 100-107.
- 8 Howard, R.A. Dynamic Programming and Markov Processes. MIT Press 1960.
- 9 Michie, D. and Chambers, R.A. Boxes: An experiment in adaptive control. Machine Intelligence 2, Edinburgh University Press 1968.
- 10 Pohl, I. First Results on the Effect of Error in Heuristic Search. Machine Intelligence 5, Ed. Meltzer, B. and Michie, D. Edinburgh University Press, 1969, pp219-236.
- 11 Samuel, A.L. Some Studies in Machine Learning Using the Game of Checkers. IBM J. Res. Develop. 3 (July 1959), pp 211-229.
- 12 Samuel, A.L. Some Studies in Machine Learning Using the Game of Checkers. II - Recent Progress. IBM J. Res. Develop. 11, 6 (Nov. 1967), pp 601-617.
- 13 Shapiro, I.J. and Narendra, K.S. Use of Stochastic Automata for Parameter Self-Optimization with Multimodal Performance Criteria. Trans. IEEE Systems Science and Cybernetics, Vol. SSC-5 No.4, October 1969, pp352-360.
- 14 Slagle, J.R. and Dixon, J.K. Experiments with some Programs that Search Game Trees. JACM Vol.16, No.2, April 1969, pp 189-207.

CHAPTER TWO

THE BANDIT ALGORITHM FOR MINIMUM COST PATH
FINDING WITH INCOMPLETE COST INFORMATION

CHAPTER TWO2 - 1 INTRODUCTION

The problem of finding the minimum cost path through a graph (network) of interconnected nodes where the arcs, or node interconnections have associated costs has been solved in many ways. The applications of this problem range from transportation routing problems [1] through automatic control, [2] to artificial intelligence research [3].

2 - 2 PROBLEM STATEMENT

Existing algorithms cannot satisfactorily tackle certain problems having incomplete cost information. Here we attempt to solve a class of problem that we have termed 'on-line'. These problems have the following characteristics:

- a) The graph is to be traversed from a start node to one of a set of goal nodes N times.
- b) The costs associated with the arcs are not fully known and may be stochastic.
- c) Information gained from each traversal of the graph is to be used to update the corresponding arc cost estimates.
- d) We wish to minimize the total cost incurred by the N traversals of the graph.

In descriptive terms the problem is that of deciding whether to travel a known path or to spend money in exploring for a short cut. It is clear that a 'search for the best path' policy may well precede a 'use the best path that has been found' policy.

THE ON-LINE PROBLEM

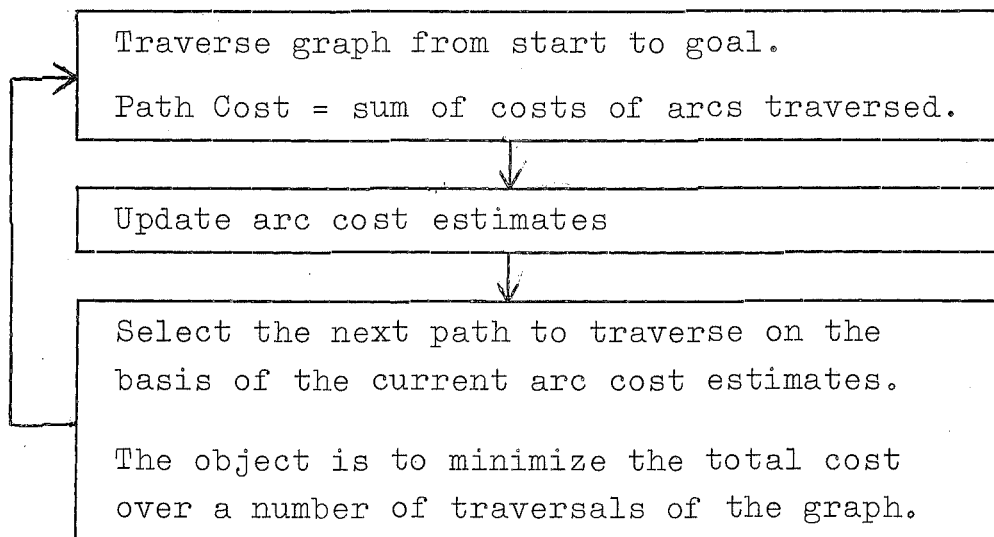


Figure 1

2 - 3 ILLUSTRATIVE EXAMPLE

To illustrate the on-line path finding problem consider the case of a transport operator who has a contract to transport goods between city A and city B. The contract is such that time is the important factor, so that the cost of any particular route (path) from

city A to city B is the travel time rather than the mileage or fuel cost or anything else.

In this example the arcs are the separate lengths of road that may be travelled as part of some path from A to B. The arc cost is the travel time for an arc. Notice that the arc cost is a random variable since hills, bends, traffic density and so on will all affect the travel time. Notice also that a reasonable apriori estimate of the mean travel time is available using road maps and so on.

Every time the transport operator runs an assignment from A to B he is able to update his cost estimates for the roads (arcs) that he chose to travel.

Consider the simple case where there are only two possible paths from A to B. Assume that the paths have costs uniformly distributed in the range 0.7 to 0.8 and 0.8 to 0.9 respectively. In our ignorance we may well assign apriori estimates of 1.0 for the cost of each route. Making an arbitrary choice we proceed by the second path on the first occasion and find it better than the apriori assumption. From this moment onwards the simple strategy of travelling the minimum (estimated) cost path, would never get around to trying the other route, although quite obviously it is better.

A heuristically derived algorithm - the BANDIT algorithm - is proposed to tackle this and similar

problems. The BANDIT algorithm accepts apriori estimates not simply as a mean cost but as a probability distribution for the mean cost.

Although the BANDIT algorithm is not guaranteed to be optimal in the sense that it will minimize the total cost over a number of traversals, it is shown to be very near optimal for simple problems and is computationally feasible for large problems. All known methods for optimal solutions are impractical (or even currently impossible) for large problems, but they can be used on small 'artificial' problems.

The optimal solution for simple problems will be given by use of Bellman's method [4], which was proposed for the 'two armed bandit problem'. This problem (to be described in the next section) can be taken as equivalent to the two path problem described above by considering the two slot machines to be the two paths, and the payoff probability as relating to the path cost. For example, a slot machine payoff probability of 0.75 can be interpreted as 0.25 mean path cost (normalized).

Note that the two armed bandit problem (next section) is considered because it is only for this simple case that optimal solutions can be computed. The BANDIT algorithm is then described and shown to match up very well to these optimal solutions.

2 - 4 THE TWO ARMED BANDIT PROBLEM

The simplest form of the path finding problem is equivalent to the two-armed bandit problem. This is a classical mathematics problem that is still not completely solved. The basic two-armed bandit problem is outlined below:

Suppose that we have two slot machines in front of us, one with known properties and one with unknown properties. When the handle on the first machine is pulled, there is a known probability, s , of receiving a dollar; when the second machine is played, there is a fixed, but unknown, probability of success, r .

The process assumes the following form. We try the second (unknown) machine a number of times to be determined by the outcomes, and then* decide to use the first machine from then on.

The object is to maximize the expected value of the criterion function:

$$R = \sum_{n=0}^{\infty} a^n z_n$$

where $0 < a < 1$ is a discount factor and z_n represents the return obtained on the n th trial.

This criterion function enables the problem to be treated as an unbounded process with discount factor, a , rather than a finite sequence of choices where we have

*could be never

N trials.

Bellman's dynamic programming approach [4] to this problem gives a computational method that is feasible for the case of one unknown and one known payoff probability to choose between (as above). For multiple choice problems the computation would quickly become impractical. No analytic solution has been derived for an optimal policy.

Bellman defines:

$f_{mn}(s)$ = the expected return obtained using an optimal policy for an unbounded process after the second (unknown) machine has had m successes and n failures.

It is assumed [4] that the probability distribution $F_{mn}(r)$, for r in $[0,1]$ after m successes and n failures, is updated from the apriori estimate. Let the expected value of $F_{mn}(r)$ be p_{mn} .

The basic functional equation can now be written,

$$f_{mn}(s) = \text{Max} \left[\begin{array}{l} (1+a \cdot f_{m+1,n}(s))p_{mn} + a(1-p_{mn})f_{m,n+1}(s) \\ s/(1-a) \end{array} \right]$$

Bellman gives an existence and uniqueness theorem that enables the above equation to be solved by a method of successive approximations.

Bellman's method has been outlined above since it shows the difficulty involved in this problem; also it provides a computational method for the optimal policy. These optimal policies will be compared with the results from a heuristic algorithm which is described below.

2 - 5 THE BANDIT ALGORITHM

We will leave the two-armed bandit problem for a moment in order to set out a heuristic decision procedure that will be central to the rest of this chapter.

The heuristic that the BANDIT algorithm is based on can be stated like this:

If one of a number of alternatives has a probability 'p' of being the best alternative, then choose this alternative p.100% of the time.

To assess this probability of one alternative being better than any other, it is necessary to know not only the estimated mean costs for each alternative but also the distribution probability of these mean costs.

The apriori distribution to be used is not an objective probability corresponding to some random experiment, but rather degrees of belief based on prior analysis of conditions relevant to the particular problem. Thus apriori distributions including a zero probability for some range of values imply a complete belief or certainly that values in this range never occur. A

common apriori distribution would be a normal (Gaussian) form, the variance reflecting the confidence in the mean estimate.

Consider the case of only two alternatives as in the case of two possible paths from A to B. If the cost of one path is estimated to be 0.85 with variance of 0.15; and the other path is known to have a mean cost of 0.75; roughly, the BANDIT algorithm will give the 0.75 cost path preference about 84% of the time. The other 16% of the traversals are used to establish a better estimate of the 0.85 estimated mean cost path - making sure that it really is a greater cost than 0.75. As we become surer the variance drops and the 16% falls lower.

We will now frame the BANDIT algorithm in a more formal and precise manner:

Let $S_1, S_2, S_3, \dots, S_N$ be N alternatives. We must select one of these and the estimated mean cost of selecting S_i is x_i . Let $f_i(x)$ be the probability distribution for x_i , it is to be understood that this is the current distribution and updating occurs to $f_i(x)$ after S_i has been selected and its cost measured. The measured cost is treated as a sample from an unknown distribution $g_i(x)$ and the expected value of $f_i(x)$ is a mean estimator for $g_i(x)$. The object is to minimize the total cost incurred.

From the $f_i(x)$ we can calculate the current probability that the cost of S_i will be less than any of the other alternatives, $p(x_i = \min \{x_j; j=1,2,\dots,N\})$.

BANDIT Algorithm

Select S_i with a probability $p(S_i)$ such that $p(S_i) = p(x_i = \min \{x_j; j=1,2,\dots,N\})$.

The direct computation of $p(S_i)$ would require evaluation of the integral:

$$p(S_i) = \int_0^{\infty} \prod_{j \neq i} (1 - F_j(x)) \cdot f_i(x) \cdot dx$$

Where $F_j(x) = \int_0^x f_j(x) \cdot dx$; the cumulative distribution (see Appendix C for details).

Fortunately since we only need to select one of the possible strategies, $S_j \in \{S_i\}$, we can employ a Monte Carlo type procedure to avoid calculating $p(S_i)$ for each i , i.e. $\{p(S_i)\}$. The procedure is to take a set of random samples, $\{y_i; i=1,2,\dots,N\}$ where each y_i is a random sample from the probability distribution $f_i(x)$. If the minimum of this set is y_j , where $y_j = \min \{y_i; i=1,2,\dots,N\}$, then select strategy S_j . Notice that unlike normal Monte Carlo procedures only one set of random samples is taken in this procedure. If the

procedure were repeated a large number of times then the set of probabilities of selection for each strategy, $\{p(S_i)\}$, could be evaluated.

An additional simplification can be made in the procedure by assuming that the $f_i(x)$ are all normal distributions. This will usually be an acceptable assumption since the central limit theory proves that the distribution of a mean estimator will tend to be normal as the number of samples becomes large, regardless of the form of the parent population. Further, if the parent population is normal in its distribution then the distribution of the mean estimator will always be normal.

2 - 6 SOME COMPARATIVE RESULTS

In order to illustrate the operation of the BANDIT algorithm we have compared its performance on some two-armed bandit problems with the optimal solution (computed by Bellman's method).

The optimal solution involves a switch from the unknown machine to the known one after a particular sequence of successes and failures. For any given number of trials, the probability of different sequences of success and failure can be computed and hence the probability of a switch to the known machine.

The BANDIT algorithm has a probability of selecting either machine that depends on the sequence of successes and failures for each machine. In the same manner as

above we compute for each given number of trials the probability of each possible sequence of success and failure, and hence the overall probability that the known machine will be selected.

The above procedure has given the results for both the optimal solution and the BANDIT algorithm in the form of a probability that the known machine will be selected after any given number of trials. These results can be directly compared and this is done in figure 2. Discount factors, a , of 0.5 and 0.8 are shown for the optimal policy. Both methods used the same apriori estimate for the unknown machine payoff and also the same updating procedure. The distribution of the payoff probability for the unknown machine was assumed normal with an apriori variance of 0.25.

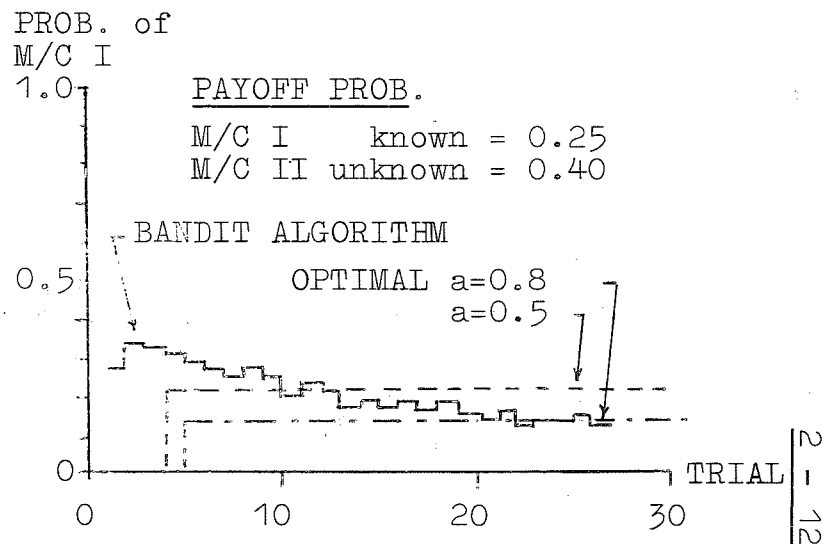
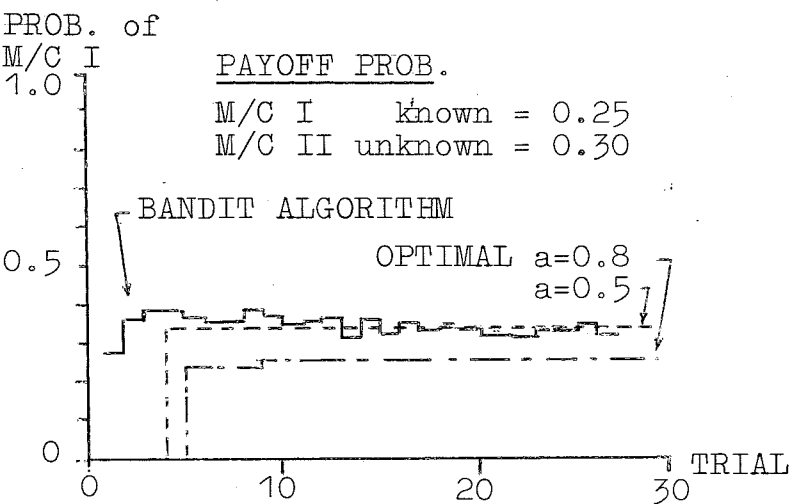
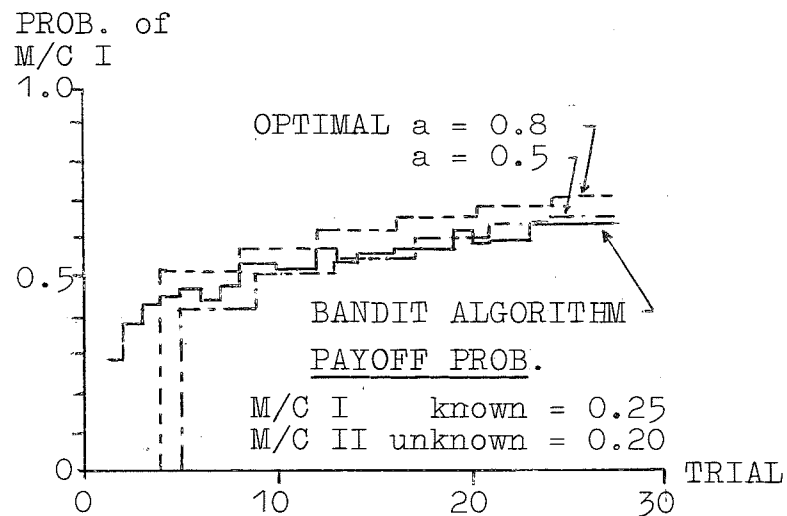
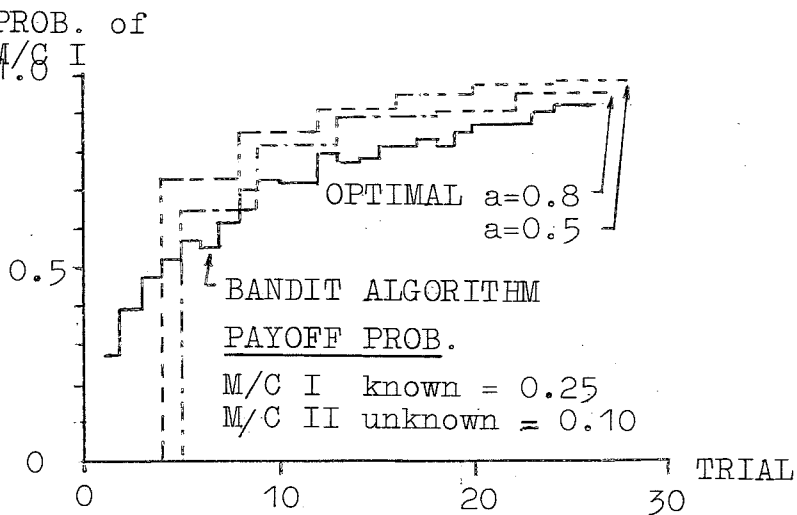


Figure 2

TWO ARMED BANDIT PROBLEM

2 - 7 EXTENSION TO PATH FINDING

It is not practical to calculate an optimal solution for even a small on-line path finding problem. We can appreciate this by considering the extensions that need to be made to the two-armed bandit problem:

1. Extension to an m-armed bandit problem. (m alternative paths).
2. The cost of all m paths may be unknown apriori.
3. The traversal of one of the paths allows updating of a set of arcs. These sets of arcs for each of the m paths are not necessarily mutually exclusive.
4. It requires considerable calculation to evaluate the set of arcs that form the minimum cost path (on the basis of the cost estimates). In general it is impracticable to list all of the m alternative paths and their costs.

Point 4 leads on to the procedure that will be used to find a path through the network.

2 - 8 AN ADMISSIBLE ALGORITHM

We shall use the definition in [5] for an admissible algorithm, which (briefly) is any algorithm that is guaranteed to find an optimal path from the start node to a goal node. Dynamic Programming [1] and the A* algorithm [5] are examples of admissible algorithms.

An optimal path from node i to node j has the lowest cost over all possible paths from node i to node j . An admissible algorithm considers only the set of arc costs it has to work with so that the optimal path that it finds is more correctly the current-optimal path. It is only optimal on the basis of the current arc cost estimates.

In the on-line path finding problem the current-optimal path on the basis of the current arc cost estimates may not be the true minimum cost path (the optimal based on the, unknown, true mean arc costs). This fact can cause the admissible algorithm to 'sit' on one particular path each time it is used during the course of an on-line path finding problem, even though this path may not be the true-optimal. The feedback that comes from the measured arc costs between each use of the admissible path finding algorithm (used to update the arc cost estimates) may not help simply because only those arc costs on the current-optimal are being measured. This failure of an admissible algorithm in the on-line case is considered further in the next sections.

2 - 9 ARC COST ESTIMATES

We will denote the cost of the arc $i-j$ from node i to node j as c_{ij} , and its estimate \hat{c}_{ij} . The \hat{c}_{ij} are updated every time arc $i-j$ is traversed. A form of stochastic approximation can be used;

$$\hat{c}_{ij} \leftarrow a \cdot \hat{c}_{ij} + (1-a) \cdot c_{ij}'$$

where c_{ij}' is the arc cost as measured, and $0 \leq a \leq 1$.* If c_{ij} is known to be deterministic, use $a = 0$. More sophisticated forms of this process can be found in [6]. Figure 3 illustrates this process, and shows how the variance of the estimate can be maintained in a similar manner.

We will use C_i to denote the cost of path i , one of the set of possible paths from the start node to a goal node. \hat{C}_i will be the estimate of C_i ; i.e. the sum of \hat{c}_{ij} for all arcs $i-j$ on path i .

2 - 10 ON-LINE ALGORITHM

The on-line algorithm applies an admissible algorithm at each traversal under the on-line conditions.

If $\hat{C}_j < \hat{C}_i$ for all $i \neq j$, and for simplicity $\hat{C}_j = C_j$, then the on line algorithm will select path j to be traversed. However, if $\hat{C}_k > C_j$ and $C_j > C_k$ for

*The \leftarrow implies that at each updating, the variable on the LHS is given the value of the expression on the RHS.

some k then the on-line algorithm will have converged onto a non-optimum path since $C_k < C_j$.

2 - 11 CONVERGENCE THEOREM

The on-line algorithm is guaranteed to converge onto the optimum path under the following conditions:

$$\begin{array}{ll} \hat{c}_{ij} \leq c_{ij} & \text{for all } i-j \\ \hat{c}_{ij} \rightarrow c_{ij} & \text{as arc } i-j \text{ is repeatedly} \\ & \text{traversed.} \end{array}$$

The proof of the convergence theorem follows from contradiction of the hypothesis of convergence onto a non-optimum or non-convergence.

If we cannot satisfy the conditions of the convergence theorem we have shown that the use of the on-line algorithm will not always be successful. The BANDIT algorithm is proposed to overcome this difficulty.

2 - 12 APPLICATION OF THE BANDIT ALGORITHM

We have shown that the BANDIT algorithm performs well in comparison to the optimal solution for some simple two armed bandit problems, which is the basis of the on-line path finding problem. There is no known method that is practical for computation of the optimal solution for the more complex on-line problem. The BANDIT algorithm (which requires only modest computation) cannot therefore be compared with an optimal

solution. Instead we must rely on the 'reasonable' nature of the algorithm - as with most heuristic procedures.

We have found proof of convergence for a modified form of the BANDIT algorithm in terms of the theory of variable structure automata, [8]. This is discussed in the next chapter (Chapter 3). This chapter however will rely on an example to support the worth of the algorithm.

The application of the BANDIT algorithm to the on-line path finding problem requires:

1. A procedure to maintain and update a distribution for the mean estimator for each arc cost. If advantage is taken of the fact that by the central limit theorem this distribution will tend to be normal, then only two parameters (mean and variance) will be required for each arc cost estimation. One possible procedure for updating these parameters has been outlined in figure 3.
2. A procedure for obtaining a random sample from each of the mean arc cost estimator distributions. There are many random number generators available - again the computation is simplified if a normal distribution is assumed.
3. An admissible algorithm that can be applied to the set of arc costs obtained by 2. There may well be

a standard minimum-cost path finding program to hand and this could be used directly.

The operation of these three procedures is shown in figure 4.

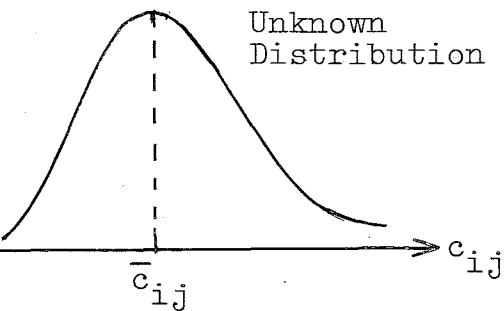
The application of the BANDIT algorithm to heuristic search algorithms used in artificial intelligence research is discussed in Appendix D. However, the notation for algorithm presentation developed in Chapter 3 is used, so that a detailed study of Appendix D is best left for the present.

2 - 13 RESULTS

The small network shown in figure 5 was used to demonstrate the BANDIT algorithm as applied to the on-line path finding problem. The true mean arc costs for this example are shown in figure 5 and the true optimal path is shown with the dotted line. The apriori arc cost estimates were all taken as equal so that the apriori current optimal path will be either around the top or around the bottom in figure 5.

Two other methods apart from the BANDIT algorithm were applied to the same example. One was the on-line algorithm which in this case will not have the conditions specified in the convergence theorem satisfied. The results from this show the trap of applying a standard minimum cost finding program to the on-line

Probability



Mean estimator distribution:

Probability

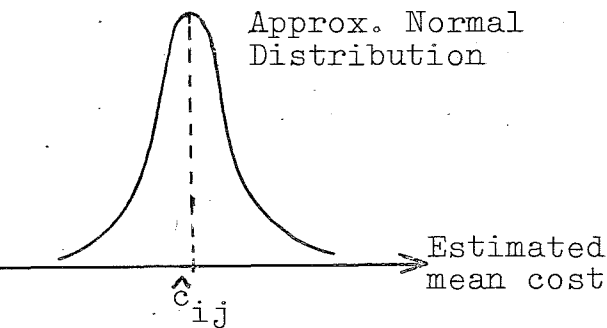


Figure 3

Each traversal of arc ij incurs a cost c_{ij} .

Cost at time $t = c_{ij}(t)$.

Estimate of $\bar{c}_{ij}(t) = \hat{c}_{ij}(t)$.

Variance of $\hat{c}_{ij}(t) = v_{ij}(t)$.

S.T.D. of $\hat{c}_{ij}(t) = d_{ij}(t)$.

Update of mean and variance for the cost of arc ij .

$$\hat{c}_{ij}(t) = a \cdot \hat{c}_{ij}(t-1) + (1-a) \cdot c_{ij}(t); \quad 0 < a < 1$$

$$v_{ij}(t) = a \cdot v_{ij}(t-1) + (1-a) \cdot (\hat{c}_{ij}(t) - c_{ij}(t))^2$$

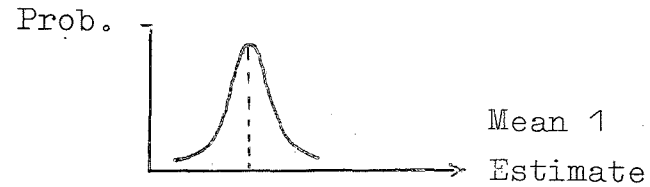
$$d_{ij}(t) = \frac{(v_{ij}(t))^{1/2}}{m}; \quad m = \text{the number of samples,}$$

m is limited to $(1+a)/(1-a)$, the moving average sample number that is equivalent to the exponential smoothing equation, Brown [7]. Notice that if $a=(m-1)/m$, and m is the total number of samples available, then the above equations reduce to the standard (classical) mean and variance estimators.

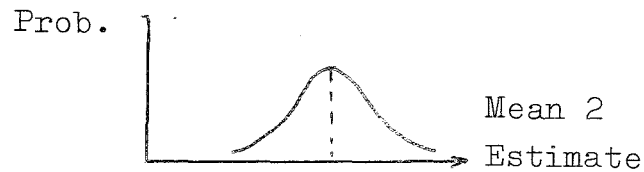
THE BANDIT ALGORITHM
APPLIED TO ON-LINE
PATH FINDING.

Mean Arc Cost Estimates

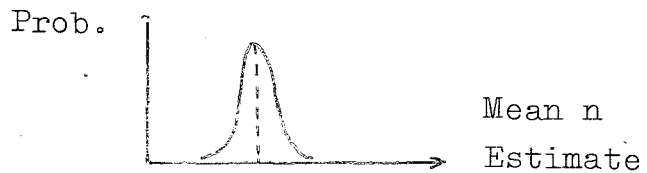
Random Samples



\hat{c}_1



\hat{c}_2



\hat{c}_n

Apply Dynamic Prog.
 or A* algorithm
 Travel Minimum Cost Path
 (Any minimum cost path
 finding algorithm may
 be used).

Update Estimates

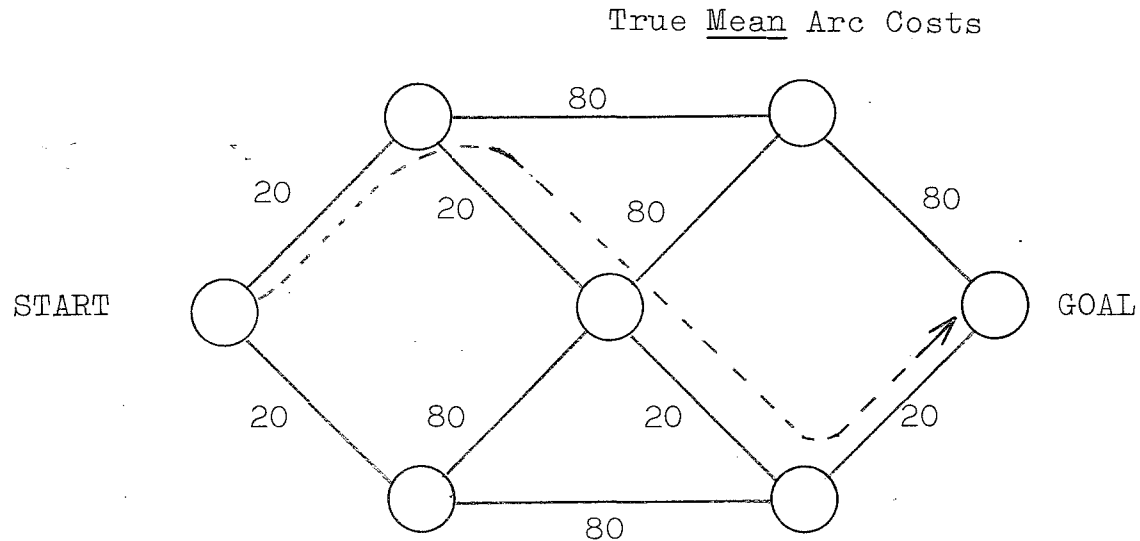
Figure 4

problem. The other method was an algorithm that selected a path with probability inversely proportional to the relative cost of the path.

Figure 6 has a graph of the results expressed as incremental costs (total cost incurred divided by the number of trials). These graphs can be viewed as a form of 'learning' curve. The BANDIT algorithm is the only one to converge onto the true minimum cost path.

The same results obtained from the BANDIT algorithm are shown in a different form in figure 7. Here the probability that each of the possible paths will be chosen is graphed against the number of trials (plotted for three of the paths).

SMALL NETWORK EXAMPLE



Apriori estimates: All arc costs = 50
Std deviation = 30

True arc costs: As shown above with
Std deviation = 10.

Figure 5

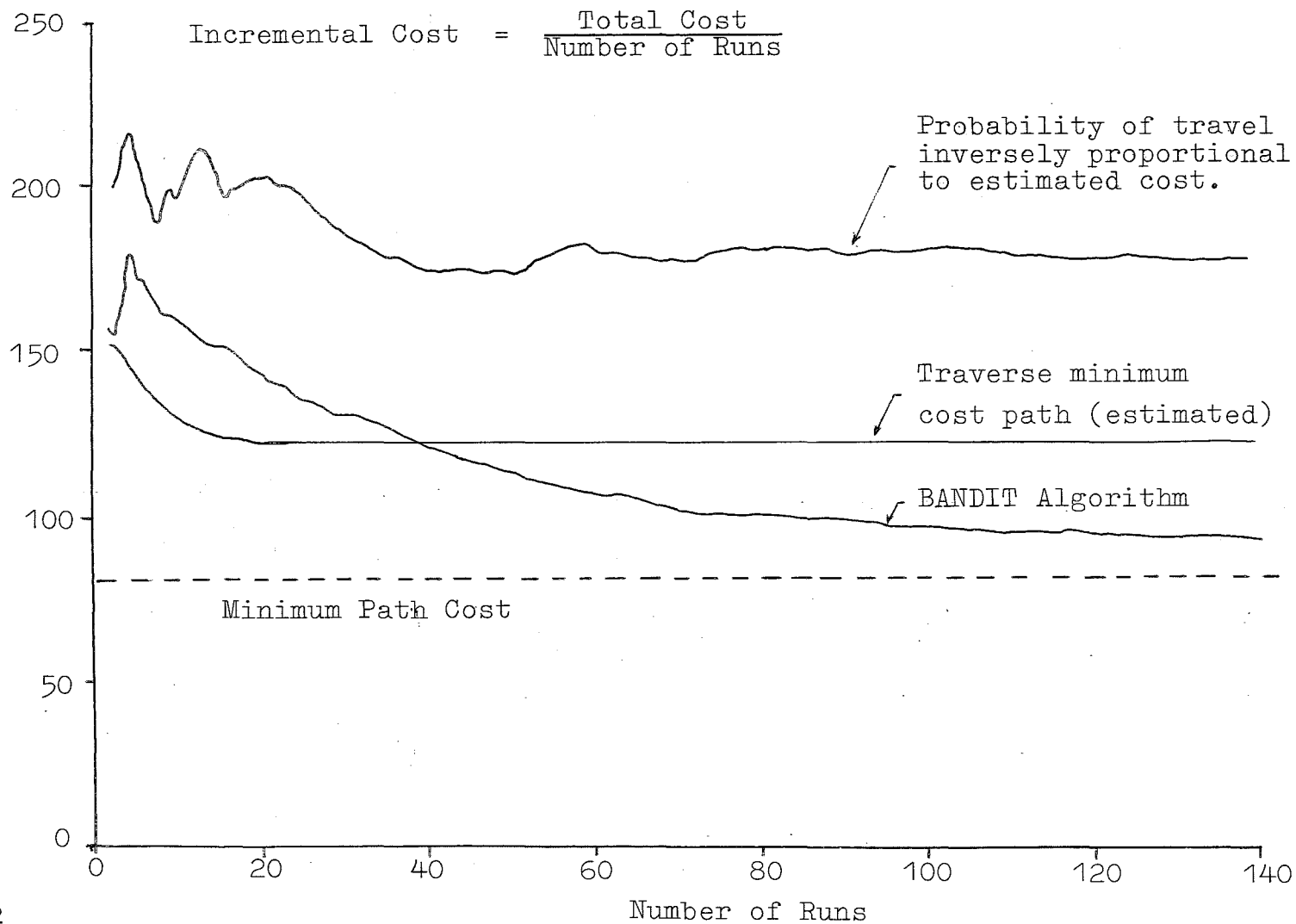
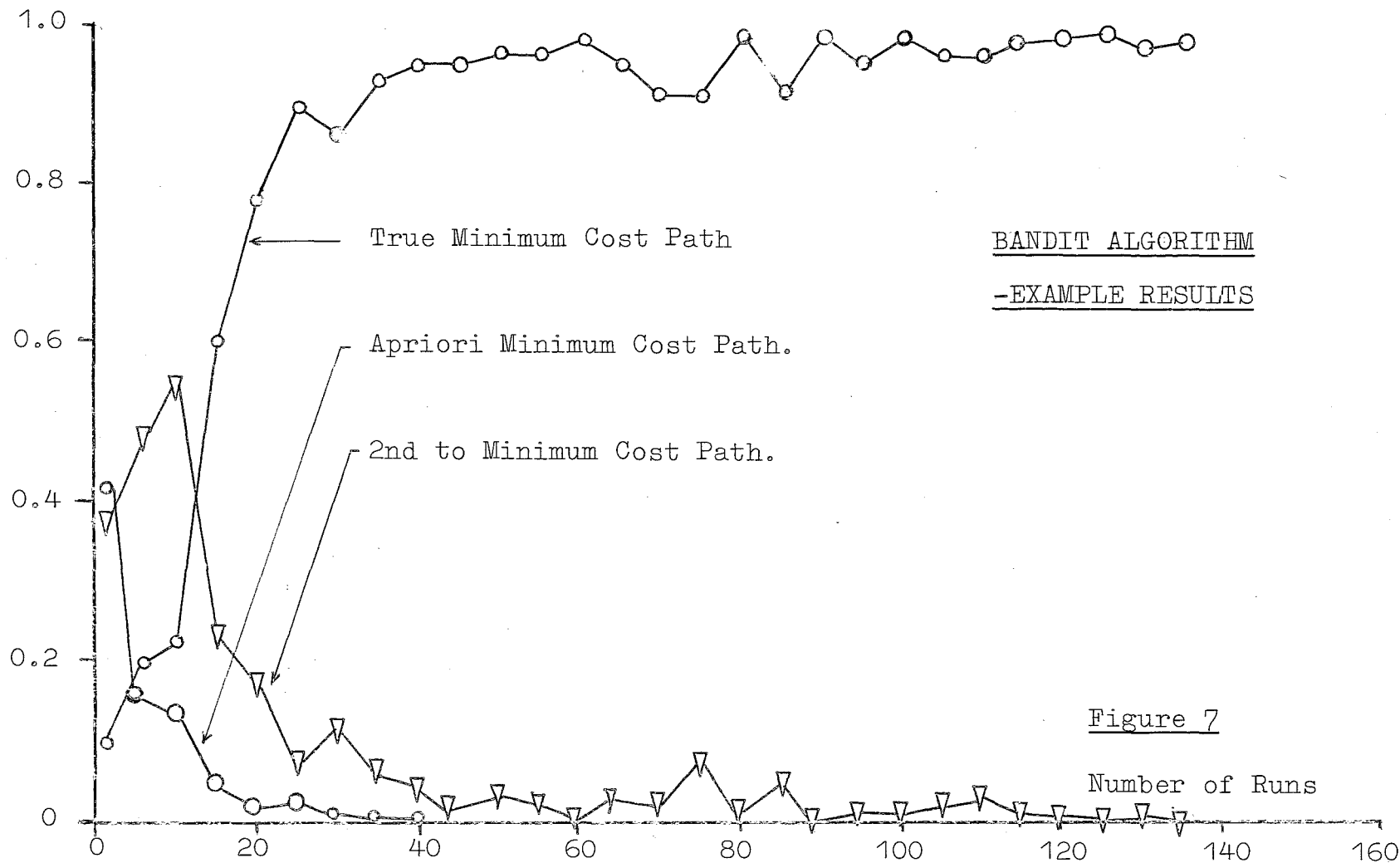


Figure 6

Prob. of Traversal



BANDIT ALGORITHM

-EXAMPLE RESULTS

Figure 7

Number of Runs

2 - 14 CONCLUSIONS

A class of path finding problems that involve a simultaneous maximization of information gain and minimization of incurred cost has been described as the 'on-line' path finding problem. A heuristically derived algorithm - the BANDIT algorithm - is proposed to tackle these problems.

Simplest form of the on-line path finding problem is the two armed bandit problem. An optimal solution for this problem is compared with results from the BANDIT algorithm.

The application of the BANDIT algorithm to the full on-line path finding problem is then outlined and some results given for a small example problem.

The computation of the optimal solution for the two armed bandit problem is an iterative procedure requiring considerable computation. The extension of this method to the on-line path finding problem is not practical. The BANDIT algorithm on the other hand requires very little computation if the method outlined is followed.

A convergence theorem with the particular conditions under which a standard path finding method (such as dynamic programming) can be successfully applied to the on-line path finding problem is given. The BANDIT algorithm can however be applied to any standard procedure (admissible algorithm) in order to extend the range of

problems that can be successfully tackled.

No discussion has been given for the case where the random arc costs have time variable statistics (such as drift in the mean cost of some arcs). Preliminary results show that the BANDIT algorithm is capable of tackling such problems and the methods suggested leave this possibility open. There is no known optimal solution to this class of problem even for the simple two-armed bandit situation.

A convergence proof for a modified form of the BANDIT algorithm has been found in terms of the theory of variable - structure automata [8]. This will be discussed in Chapter 3.

REFERENCES

- 1 Kaufman, A., Graphs Dynamic Programing and Finite Games, Academic Press, 1967.
- 2 Bellman, R., & Kalaba, R., Dynamic Programing and Modern Control Theory, Academic Press, 1965.
- 3 Slagle, J.R. & Dixon, J.K., Experiments with some Programs that Search Game Trees, JACM Vol.116, No.2 April 1969; pp 189-207.
- 4 Bellman, R., A Problem in the Sequential Design of Experiments, SANKHYA Vol.116 pt. 3 & 4, 1956; pp 221-229.
- 5 Hart, P.E., Nilsson, N.J. & Raphall, B., A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEE Trans. on System Science & Cybernetics, S.S.C. 4 No. 2 July 1968; pp 100-107.
- 6 Fu, K.S., Stochastic Automata as Models of Learning Systems, Computer & Information Sciences II, Ed. J.T. Tou, Academic Press, 1967; pp 177-192.
- 7 Brown, R.G., Smoothing forecasting and prediction of discrete time series, Prentice Hall, 1962.
- 8 Chandrasekaran, B. & Shen, D.W.C., On Expediency and Convergence in Variable - Structure Automata, IEEE Trans. on Systems Science & Cybernetics, March 1968, SSC - 4 No. 1; pp 52-59.

CHAPTER THREE

STOCHASTIC LEARNING AUTOMATA

3 - 1 INTRODUCTION

This introduction gives a brief survey of the development of stochastic learning automata and develops an algorithmic notation for the concise description of these schemes. Following this, a new development called the BANDIT scheme is described and some of its advantages considered.

The class of problems considered, or the environment for the stochastic learning automaton, is then extended to a much wider problem class that cannot be tackled by the automata schemes described, but has been considered in the field of 'heuristic programming' for learning machines as robots and game players. The BANDIT automaton is extended by the development of an 'expectance' function in order to tackle this extended problem class. Illustrative results are then given.

A more detailed historical development of most of the material outlined here can be found in Fu (1970) [11]

The behaviour of a finite state (deterministic) automaton operating in a random environment was first studied by Tsetlin (1961) [15]. A simple block diagram of the system is shown in figure 1.

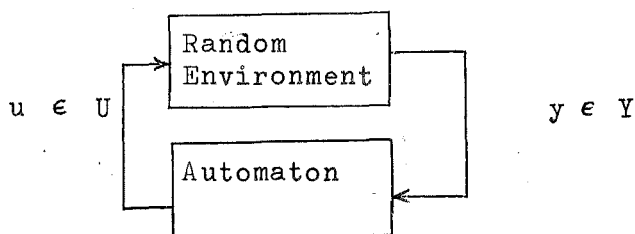


Figure 1.

The response of the environment can be $y = 1$, called a penalty response, or $y = 0$ called a nonpenalty or reward response. Tsetlin characterised the environment by a vector $C = (C_1, \dots, C_r)$ where:

$$C_j = \Pr (y(t)=1 \mid u(t)=u_j) \quad j=1, \dots, r \quad (1)$$

$u(t)$ = automaton's output or action at time t ,

$y(t)$ = the environments response to this action.

C_j = the probability of a penalty response for action u_j ,
and $(1-C_j)$ = the probability of a reward response.

A stationary random environment is considered which means that the C_i are all constant values over time.

Following Tsetlin's approach Varshavskii and Vorontsova (1963) [16] have used a variable structure automaton as a model of a learning system operating in a random environment. Their approach was to modify the state transition probabilities in such a way as to minimize the probability of a penalty response. If $p_{ij}^1(t)$ is the transition probability from Automaton state q_i to state q_j for the environment's response $y=y_1$, at time t , then:

$$\sum_{j=1}^r p_{ij}^1(t) = 1 \text{ for all } i \text{ and } t. \quad (2)$$

The idea is to minimize the probability of a penalty response by a decrease of p_{ij}^1 every time a transition from state q_i to q_j due to response y_1 , is followed by a penalty response. If however a nonpenalty or reward response follows, then p_{ij}^1 is increased. This modification of the

state transition probabilities constitutes a transition probability update procedure. An admissible transition probability update procedure must be such that (2) applies after every update.

Fu et al (1965,1966) [7] [8] have extended Varshavskii and Vorontsova's work [16] to the more general case where:

1. Instead of updating the transition probabilities, any convenient probability function is treated. (The total state probabilities for example).
2. The performance of the automaton is measured not by a 1 (penalty) or 0 (reward), but by any value between 0 and 1.

The system now looks like figure 2.

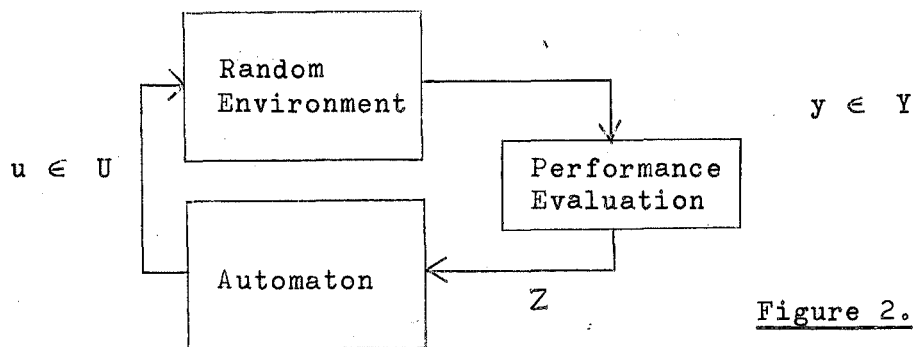


Figure 2.

Let the probability of the automaton taking the i th decision, choosing the i th action, or being in the i th state be p_i . The index $i=1, \dots, r$, where r is the total number of possible (quantized) parameter settings, or the number of states of the automaton.

A linear reinforcement algorithm to update p_i can be written as:

$$p_i^1(t+1) = p_i^1(t) + (1-\alpha)z_i^1(t) \quad (3)$$

Where $0 < \alpha < 1$, $0 \leq z_i^1(t) \leq 1$, and $\sum_{i=1}^r z_i^1(t) = 1$.

$p_i^1(t)$ = the probability of action u_i due to input (environment response) y_1 , at time t .

$z_i^1(t)$ = a performance measure evaluated from the environments response to action u_i (u_i due to y_1) at time t .

A slightly different viewpoint is to consider the use of a learning automaton as a hill climber to maximize or minimize some performance measure. The environment can now be considered as providing an output that is a performance measure $I(u_i)$ for action or parameter setting u_i . The output from the environment that is available to the automaton is denoted $g(u_i)$ and this will be a noisy observation or measurement of $I(u_i)$. The expectation of $g(u_i)$ will be the actual performance measure for action u_i , $I(u_i)$. The aim of the automaton is to give an output or parameter setting u_{opt} such that $I(u_{opt})$ is an extremum. From now on only a maximum will be considered, in contrast to the minimization of a penalty probability considered previously.

Shapiro and Narendra (1969) [14] have considered this use of a learning automaton as a hill climber in situations where only noisy measurements of performance are available and the performance hill may be multi-modal. Their scheme is outlined below because it comes closest to the BANDIT algorithm that is to be described later, and results from the scheme are compared with results from the BANDIT algorithm.

An important point about the scheme proposed by Shapiro and Narendra is that it can achieve optimal performance, whereas the schemes previously outlined can, in general, only achieve expedient performance. Convergence and expediency have been considered for several learning procedures by Chandrasekaran and Shen (1968). [5] Expediency in the context of hill climbing, is said to occur when for;

$$I(u_j) > I(u_k) \quad k=1, \dots, r, \quad k \neq j \quad (4)$$

in the limit as $t \rightarrow \infty$, the probability $p_j(t)$ of selecting action j is greater than $p_k(t)$, in either a deterministic or probabilistic sense as the particular case may apply.

Optimality is a stronger result than expediency and it requires that;

$$\lim_{t \rightarrow \infty} p_j(t) = 1 \quad \text{if } I(u_j) > I(u_k) \text{ for all } k \neq j. \quad (5)$$

and

$$\lim_{t \rightarrow \infty} p_k(t) = 0 \quad \text{for } k \neq j$$

Before giving more details of particular procedures some notation will be introduced to clarify the presentation.

2. NOTATION.

An algorithmic notation is used because it eliminates an explicit reference to time which is a particular advantage for variables that are updated but not necessarily updated every time interval. More than this it is desirable to give a clear indication of the 'form' of a procedure, without restricting the procedure to a particular set of rules. The format for presenting the algorithms is based on the excellent

work of Knuth (1968)[10] and is introduced below.

Consider for example the statement:

A1. [Update] $\hat{x} \leftarrow \alpha \hat{x} + (1 - \alpha)x$. ($0 < \alpha < 1$).

Here \hat{x} is an estimate (indicated by the hat) of a variable x .

Whenever the step A1 is carried out \hat{x} is updated by replacing its current value by the evaluation of the expression to the right hand side of the arrow (\leftarrow). The phrase in the square brackets at the start of the step, in this case [Update] is a brief summary of the principal content of the step. There will sometimes be parenthesized comments at the end of the steps; in A1 the range for alpha is a comment.

Now consider a more detailed procedure:

B1 [Initialize.] $\alpha \leftarrow 0$, $k \leftarrow 1$.

B2 [Observation.] $x \leftarrow$ observation.

B3. [Update mean estimator.] $\hat{x} \leftarrow \alpha \hat{x} + (1 - \alpha)x$.

B4 [Count.] $k \leftarrow k+1$, $\alpha \leftarrow (k-1)/k$, go to step B2.

This B-procedure is simply a method for evaluating the classical mean of a set of observations (samples) x_j . Note that there is no final answer, the procedure is 'on line' and simply maintains an estimator \hat{x} .

$$\hat{x} = \frac{1}{k} \sum_{j=1}^k x_j \quad (6)$$

If however α is a fixed constant:

C1 [Initialize.] $\alpha \leftarrow$ value in the range 0 to 1.

C2 [Input.] $x \leftarrow$ observation.

C3 [Update mean estimate.] $\hat{x} \leftarrow \alpha \hat{x} + (1 - \alpha)x$, go to C2.

The C-procedure is a moving average or exponential smoothing procedure, Brown (1962) [2]. The procedure may also be interpreted as a linear reinforcement learning procedure, Bush and Mosteller (1958) [3].

In a similar manner we can maintain some estimator of the variance of observations x , written:

D1 [Variance update.] $\hat{v} \leftarrow \alpha \hat{v} + (1 - \alpha) (\hat{x} - x)^2$. ($0 < \alpha \leq 1$).

And for the variance of the mean estimator:

E1 [Mean variance update.] $\hat{w} \leftarrow \beta \hat{v}$ ($\beta \leq 1/\text{number of samples}$)

Where for a classical estimator of the variance of the mean, β would be $1/(k-1)$, k being the number of samples or observations of x .

Now the step A1 by itself can be interpreted as part of a B-procedure or part of a C-procedure or any similar procedure where α may be a more complex function Fu (1967) [9]

However instead of leaving a step like A1 in an algorithm where α is not specified the notation:

$\hat{x} \leftarrow \text{UPDATE} (\hat{x}, x)$, will be used.

It is to be understood that any of the procedures described or referred to above are appropriate for the UPDATE function. For that matter it may be any procedure to do a similar job under the particular circumstances that may apply.

3. MODIFIED LINEAR REINFORCEMENT PROCEDURE.

The modified linear reinforcement scheme developed by Shapiro and Narendra [14] will now be described in terms of the notation discussed above. Remember that

g = the environments response to the automaton's action u ,

$u \in \{u_i; i=1, \dots, r\}$, u is a noisy performance measure.

\hat{p}_i = the probability that the automaton will select action u_i ,
the hat indicating that these probabilities are estimates
of the probabilities that will give optimum performance.

F1. [Observation.] $g \leftarrow$ observed environment response.

F2. [Reward set up.] Let $g_j = \max \{g_i; i=1, \dots, r\}$.

If $g < \hat{g}_j$, then $z_i \leftarrow 0$, $i=1, \dots, r$. (No reward.)

Else if $g \geq \hat{g}_j$, and $u = u_i$, (The last action was u_i)

then $z_i \leftarrow 1$, $z_k \leftarrow 0$, $k=1, \dots, r$, $k \neq i$. (Reward ith action).

F3. [Record performance.] if $u = u_i$ (Last action was u_i).

then $\hat{g}_i \leftarrow \sigma \hat{g}_i + (1 - \sigma)g$

$k_i \leftarrow k_i + 1$, $\sigma \leftarrow (k_i - 1)/k_i$.

F4. [Update.] if $z_i = 0$, $i=1, \dots, r$, then go to F5.

(No update unless reward).

Else $\hat{p}_i \leftarrow \alpha \hat{p}_i + (1 - \alpha)z_i$, for $i=1, \dots, r$.

($\alpha = \text{constant}$, $0 < \alpha < 1$).

F5. [Select action.] Select action u_i with probability p_i .

$u \leftarrow u_i$ (execute action u_i)

go to step F1.

...

In the above procedure \hat{g}_i is a mean estimator for the performance measure resulting from the use of action u_i .

Reward is given for the current action if the observed performance is better than any of the current mean performance estimators. Notice that the update procedure for the performance estimators is a classical mean estimation, while the update procedure for the action probabilities is a linear reinforcement procedure. The function UPDATE is not used because a particular procedure was being described.

The update procedure for the action probabilities only modifies the \hat{p}_i 's for a reward condition, not for a non-reward or penalty condition. This is a feature introduced by Shapiro and Narendra [14] to obtain optimal performance rather than expedient performance.

4. THE BANDIT ALGORITHM

The BANDIT Algorithm description

Following the general approach outlined above let \hat{g}_i be a mean estimator of the noisy observations of performance for action u_i , as before. In addition let \hat{w}_i be an estimator of the variance of the mean estimator \hat{g}_i (as in E1 and D1).

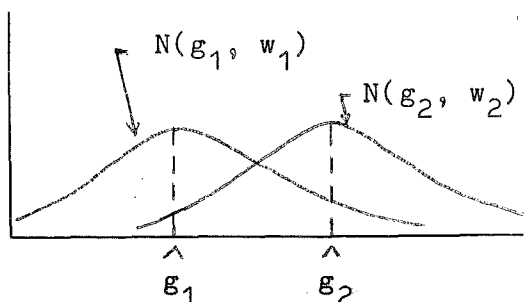
A normal distribution based on these two estimators as mean and variance, $N(\hat{g}_i, \hat{w}_i)$, will be taken as a summary of the current information about the performance evaluation of each action u_i . The validity of assuming a normal distribution will be dealt with later. The BANDIT algorithm for the functioning of an automaton is now given:

BANDIT Algorithm.

- G1. [Observation.] $g \leftarrow$ Environment response. (noisy performance)
- G2. [Update estimators.] For $u = u_i$, (current action is u_i)
- $$\hat{g}_i \leftarrow \text{UPDATE}(\hat{g}_i, g) \quad (\text{mean estimator})$$
- $$\hat{w}_i \leftarrow \text{UPDATE}(\hat{w}_i, (\hat{g}_i - g)^2) \quad (\text{variance of mean estimator})$$
- G3. [Sample.] $x_i \leftarrow$ sample from $N(\hat{g}_i, \hat{w}_i)$.
- G4. [Select action.] if $x_j = \max \{ x_i ; i=1, \dots, r \}$
 then $u \leftarrow u_j$, (output action u_j)
 go to step G1. ...

To illustrate the process involved in the BANDIT Algorithm the case with only two possible actions u_1 , and u_2 , will be considered. The distributions $N(\hat{g}_1, \hat{w}_1)$ and $N(\hat{g}_2, \hat{w}_2)$ will both start off as 'broad' distributions indicating the initial lack of exact knowledge about the true expectation of g following action u_1 and u_2 respectively. As the mean and variance of these distributions are updated the distributions will become 'sharp' with the means \hat{g}_1 and \hat{g}_2 being more and more accurate estimates of the expectation of g after action u_1 and u_2 . Figure 3 indicates the general process.

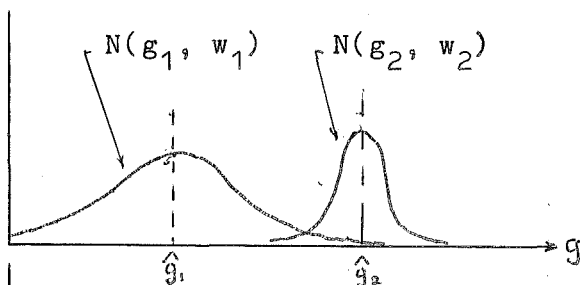
Prob.
Density



Before many samples have been used to update the distributions.

Figure 3 (a).

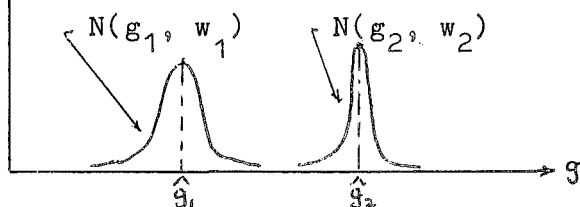
Prob.
Density.



After some
updating.

Figure 3(b).

Prob.
Density.



After consider-
able updating.

Figure 3(c).

The probability of the automaton selecting action u_i is defined by the BANDIT equation as:

$$\hat{p}_i = \Pr \left[x_i = \max \{ x_j ; j=1, \dots, r \} \right] \quad (7)$$

Where $x_i =$ a random sample from $N(\hat{g}_i, \hat{w}_i)$.

The BANDIT algorithm is a procedure that will obey the BANDIT equation, without explicit evaluation of the \hat{p}_i 's. The corresponding reduction of calculation is very significant since methods to explicitly evaluate the BANDIT equation involve either a convolution integral over the distributions $N(\hat{g}_i, \hat{w}_i)$, or a Monte Carlo evaluation. Notice that the BANDIT algorithm is almost a Monte Carlo method except that only one sample is needed. If steps G3 and G4 are repeated a large number of times the \hat{p}_i can be calculated by the standard Monte Carlo technique of counting the number of times each possible action u_i is selected.

4.1. COMMENTS ON THE BANDIT ALGORITHM.

In the presentation of the BANDIT algorithm a normal distribution $N(\hat{g}_i, \hat{w}_i)$ was assumed for the following

reasons:

1. The procedure and its description are simplified.
2. The distribution for any mean estimator will tend to be normal in the limit, by the central limit theorem. This is true even if the parent distribution (that the g 's are drawn from) is not a normal distribution. If the parent distribution is a normal distribution then the distribution for \hat{g}_i 's will always be a normal distribution.

A general BANDIT algorithm does not exclude a more general distribution function, although the class of distributions that could be used may be limited by the particular update scheme used.

As mentioned above the BANDIT algorithm has no explicit evaluation of the action probabilities \hat{p}_i , unlike all the previous work outlined in the introduction. Since the sum of the action probabilities must sum to unity (analogous to equation (2)) explicit update of one \hat{p}_i always requires the adjustment of all the other \hat{p}_i 's to meet this constraint. The BANDIT algorithm avoids this while introducing only one other set of estimators to be updated; a reduction of computation but an increase in storage (if explicit p_i 's kept).

Notice also that there is no need to normalize the performance evaluation into the range 0 to 1 before it is used in the update procedure. This normalization is done by Fu (1970) [11] and is also done in effect by Shapiro and Narendra with the creation of a reward function.

4.2 CONVERGENCE

It would be very desirable to give a convergence proof for the BANDIT algorithm but this has only been achieved for a modified form of the BANDIT algorithm. Results are presented later that show the convergence of the BANDIT algorithm for the particular problems used by Shapiro and Narendra [14] and we have no reason to suppose that this is a special case.

MODIFIED BANDIT ALGORITHM

- H1. [Observation.] $g \leftarrow$ noisy performance measure observation
- H2. [Update.] For $u = u_i$, (last action was u_i)
 $\hat{g}_i \leftarrow \text{UPDATE}(\hat{g}_i, g)$
 $\hat{w}_i \leftarrow \text{UPDATE}(\hat{w}_i, (\hat{g}_i - g)^2)$ (mean variance)
- H3. [Sample.] $x_i \leftarrow$ sample from $N(\hat{g}_i, \hat{w}_i)$.
- H4. [Reward.] If $X_i = \max \{x_j ; j=1, \dots, r\}$
 and $u = u_i$ (last action was u_i)
 then $z_i = 1 ; z_k = 0, k=1, \dots, r, k \neq i$.
 otherwise go to H6.
- H5. [Action probabilities.]
 $\hat{p}_i \leftarrow \alpha \hat{p}_i + (1 - \alpha) z_i, i=1, \dots, r$ (update prob.'s)
- H6. [Select action.] Select action u_j with probability p_j ;
 $u \leftarrow u_j ;$ (execute action u_j)
 go to step H1. ...

Notice that the modified BANDIT algorithm has the essential features of the BANDIT algorithm (G-procedure) but

it has been put into the same F-procedure form used by Shapiro and Narendra [14]. The update of \hat{p}_i only for a reward response and the use of z_i as a reward derived from the current information (estimation) of performance measures \hat{g}_i , are both established in the F-procedure.

The basis of a convergence proof relies on the stationary state of the action probabilities \hat{p}_i being defined by the expectation of \hat{p}_i at time $t+1$ being equal to the value of \hat{p}_i at time t .

$$E(\hat{p}_i(t+1)) = \hat{p}_i(t) \quad (8).$$

This is one of the criteria originally given by Varshavskii and Vorontsona (1963) [16].

Details of the convergence proof will not be given for the H-procedure since it closely copies the proof given by Shapiro and Narendra (for their method which has been presented here as the F-procedure).

4.3 COMMENTS ON CONVERGENCE.

An intuitive understanding of why the BANDIT algorithm should converge onto an optimal rather than an expedient strategy can be gained by consideration of the distributions $N(\hat{g}_i, \hat{w}_i)$. Referring back to figure 3 showing the trend of these distributions as they are updated, notice that if there is any probability of using action u_i then $N(\hat{g}_i, \hat{w}_i)$ becomes 'sharper'. It will tend towards a delta function at $\hat{g}_i = \bar{g}_i$, where \bar{g}_i is the true mean value of g for action u_i , that is $I(u_i)$. For a stationary environment $I(U_i)$ will be a constant value. Since all the distribution

functions tend towards a delta function in the limit so all the random samples x_i from distribution $N(\hat{g}_i, \hat{w}_i)$ tend to the value \hat{g}_i (which itself tends to the value $I(u_i)$). Thus in the limit, and assuming no equal maxima, the probability of selecting action u_j corresponding to the maximum performance index $I(u_j)$ becomes unity, and the probability of selecting any other action tends to zero. Before this limiting case is reached the probability of choosing action u_j (the optimal action) will become very large even though the distribution $N(\hat{g}_i, \hat{w}_i)$ for the non optimal actions may be far from a delta function. This is illustrated by figure 3(c) where the probability of action u_2 would be very large, and $N(\hat{g}_2, \hat{w}_2)$ often updated whereas $N(\hat{g}_1, \hat{w}_1)$ is infrequently updated since the probability of action u_1 is very small.

The situation described above indicates that the information collected about $N(\hat{g}_j, \hat{w}_j)$ where u_j is the optimal action, is more than that for $N(\hat{g}_i, \hat{w}_i)$ $i \neq j$. If this were not the case convergence would be slow and in the worst case the adaptive automaton scheme would have no advantage as a hill climber over an exhaustive search strategy. This conflict between gaining information to update each distribution $N(\hat{g}_i, \hat{w}_i)$ and so gain more accurate information about the performance of action u_i , as against the use of the optimal action as it is currently known, is the heart of the learning problem. This basic problem in a simple form is embodied in the well known Two Armed Bandit (T.A.B.) problem; Yakowitz (1969) [17] gives a detailed account and historical references.

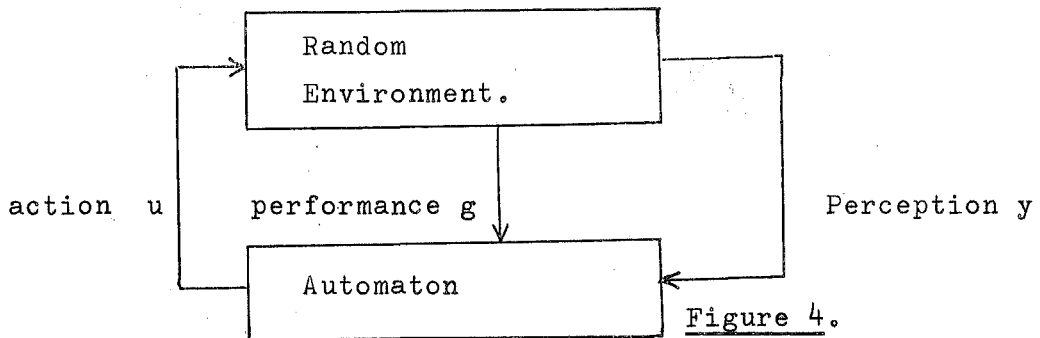
The use of the BANDIT algorithm for minimum cost path finding Cashin (1970) [4] gave rise to its application to the T.A.B. problem and hence the name BANDIT algorithm. (Chapter 2)

5. ENVIRONMENTS WITH PERCEPTION AND PERFORMANCE MEASURES

5.1 An Extended Problem Class

All the work mentioned in the introduction, in fact all known work on stochastic adaptive automata, has considered a performance measure as the only output or response of the environment. It is true that the response has been generalized a long way from the first steps with a 1 or 0 penalty or nonpenalty response but never the less the output of the environment or input to the automaton has been a performance indication. Consideration will now be given to the case depicted in figure 4, where there are two separate channels from the environment to the automaton. The important point is that the environment output y is available to the automaton as a 'perception' of the environment; it gives the automaton a 'look' at the state of the environment but does not directly indicate performance. The response from the environment labelled g is a performance measure in the same sense as the (only available) environment response considered before this section. The automaton input g will direct the performance of the automata as before, the object being to maximize the expectation of g . There is no reason why g may not be a function of y in which case the original arrangement of figure 2 may be considered adequate. But this is not necessarily so; the problems that will now be considered have

the property that the perception y is absolutely necessary for the automaton to be successful in maximizing the expectance of the performance g .



This class of problems has been considered by Andrae (1968) [1] Michie and Chambers (1968) [12] and Doran (1969) [6]. None of this work arose directly in the field of stochastic automaton learning theory as considered in the introduction, but rather in the field of 'heuristic learning machine work'. The problems will now be put as a logical extension to the learning automaton work considered previously. Thus a link will be formed between two distinct research areas which will hopefully lead to advantages for both.

5.2 AN EXAMPLE PROBLEM.

The idea of a perception signal y being necessary for the maximization of the expectance of the performance g is best seen from an example. The example used here will be the game of NIM. In this game two players take turns to remove 1, 2 or 3 stones from a pile initially containing 16 stones. The player to remove the last stone of the pile is, in this version, the winner. However it will always be allowable to remove 1, 2 or 3 stones, to be subtracted modulo

16 from the pile. The game thus cycles continuously with a win for the player leaving his opponent with exactly 16 stones.

In terms of figure 4 the possible automaton actions u_i , $i=1,2,3$, are the moves corresponding to the removal of 1,2 or 3 stones. The perception of the environment y , will be considered as the number of stones on the pile just prior to the automaton choosing an action (making a move). Notice that the automaton can not 'see' the result of its action, only the result of its action followed by the environments action.

The environment will be considered to be a pseudo random player that knows the optimal moves but occasionally makes random mistakes. If the environment were to play only optimal moves the automaton would never win a game, and could not hope to learn anything useful at all.

The performance indication g given from the environment will be $g=1$ for a move by the automaton that wins a game, otherwise $g=0$. This performance indication g need not come from the environment but could be calculated from y and u by the automaton itself. This does not alter the concept of having both g and y as necessary inputs to the learning procedure, so that the arrangement in figure 4 will still be considered as an adequate description.

Table 1 gives a complete description of the game, together with the optimal moves that the automaton should learn to play. These optimal moves are closely related to the

perception y and a little consideration will show that only the last three optimal moves could be learned by an automaton denied the perception y .

TABLE 1. Description of the game of NIM.

Perception $y =$	Action 1 $u = u_1$ $g =$	Action 2 $u = u_2$ $g =$	Action 3 $u = u_3$ $g =$	Optimal Moves $u =$
16	0	0	0	u_1^*
15	0	0	0	u_3
14	0	0	0	u_2
13	0	0	0	u_1
12	0	0	0	u_1^*
11	0	0	0	u_3
10	0	0	0	u_2
9	0	0	0	u_1
8	0	0	0	u_1^*
7	0	0	0	u_3
6	0	0	0	u_2
5	0	0	0	u_1
4	0	0	0	u_1^*
3	0	0	1	u_3
2	0	1	0	u_2
1	1	0	0	u_1

*There is no true optimal move in that a win cannot be forced from these positions, but a move of u_1 maximizes the probability of a win.

It is interesting to note that given only a set of lights for the y and g signals and asked to select a series of actions u_i (3 buttons) to maximize the probability of light g being on a human player often takes several hours to learn a near optimal set of moves.

One of the difficulties of the situation is, that a non optimal move may, because of a 'mistake' by the environment, lead to a win. The probability of a win for this move is determined by the 'mistake' probability of the environment. If such moves are discovered by the learning player (human or automaton) care must be taken to ensure that these moves do not become established as the best moves simply because of inadequate 'research' on the other moves. This is the old Two Armed Bandit Problem again, that of information gain against cost of a move as currently known. Michie and Chambers (1968) [12] discuss this problem with reference to NIM and present their solution to this problem in terms of a control problem. The following section is a similar approach that is suitable to stochastic learning automata. It is essentially a development of the 'expectation function' used in the STELLA learning machine of Andreae (1968) [1].

6. EXPECTANCE.

A function is now developed called the expectance \emptyset that gives a performance measure for each perception y , associated response u , and environment response g . The expectance value for perception y_i followed by action u_j will

be denoted ϕ_{ij} .

The purpose of the expectance function is to give a value to each input-action association ($y-u$) that reflects the probability of getting a reward response, $g=1$, but also takes into account the probability of getting a reward response in the future. Thus once the expectance function has been derived the extended problem can be viewed as a series of problems suitable for the automata first considered in this paper.

For example consider the NIM game of Table 1, and take the situation $y = 6$ or y_6 . The optimal move in this situation is $u = u_2$ (or move 2) but the reward for this action $g = 1$, will not come until after the next move, and only then if the next move is optimal. The expectance ϕ_{62} should however have a value higher than ϕ_{61} or ϕ_{63} (non optimal moves).

This gives rise to what can be viewed as a new set of subproblems: Given situation i ($y = y_i$) choose an action u_j based on a performance measure ϕ_{ij} . The situation is shown in figure 5, where each of the 16 parallel automata is similar to those originally discussed (eg. a G-procedure).

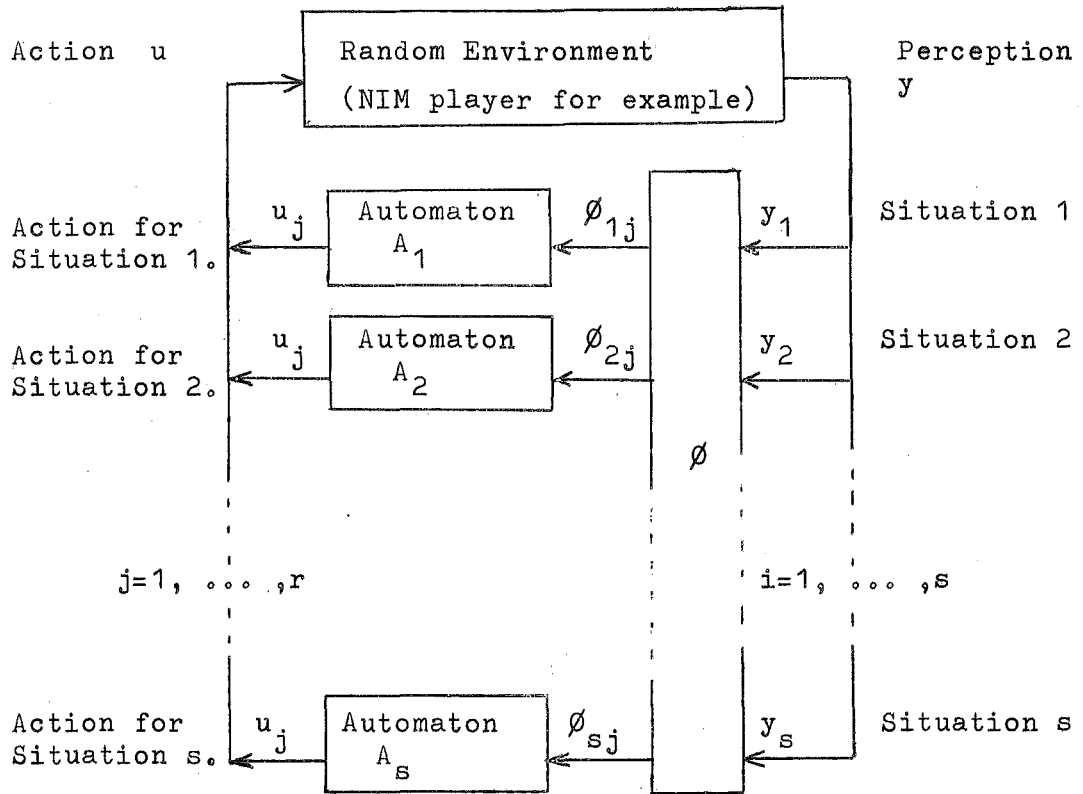


Figure 5.

It can be seen from figure 5 that there is an automaton for each situation y_i . In practice a completely separate set of automata is not needed but this representation serves to give an understanding of the situation. The automaton A_2 would in the NIM problem handle situation 2 (There being 16 possible situations). Situation 2 corresponds to 2 stones in the pile, and the action u_j (1,2, or 3) is based on the performance measure ϕ_{2j} which are provided to it for each action by the expectance function calculation.

Now that the expectance function's purpose has been established, the function can be defined recursively as .

$$\phi_{ij} = \frac{1}{(1 + \gamma)} \left[g_{ij} + \gamma \sum_{k=1}^s \sum_{l=1}^r p_{ij,kl} \phi_{kl} \right] \quad (9)$$

Where γ = a discount factor, $0 \leq \gamma \leq 1$.

$p_{ij,k1}$ = the probability of situation y_k with action u_1 ,
given situation y_i and action u_j .

g_{ij} = the performance measure on action u_j given situation y_i .

s = the number of possible situations.

r = the number of possible actions.

The discount factor γ can be thought of as setting the importance of potential future reward against immediate reward ($\gamma=1$) response for this action. If $\gamma = 1$ equal account is taken of the probability of future reward and immediate reward, if $\gamma = 0$ then no account is taken of future reward probability and the situation reverts to the original problem with a performance signal g , but no perception signal y .

The probability $p_{ij,k1}$ can be thought of as the situation - action transition probability. An on-line calculation similar to the B or C-procedure could be used to estimate this probability but even given this the expectance function would be difficult to calculate directly from the recursive definition (9).

In order to avoid the iterative calculation inherent in equation (9) the following on-line procedure can be used to give a 'running' estimate of the expectance values ϕ_{ij} without any explicit calculation of the transition probabilities $p_{ij,k1}$.

Expectance Algorithm.

- I1. [Perform action.] If $y=y_i$, let $u=u_j$ be the action selected and used by the automaton.
- I2. [Observe response.] $y \leftarrow$ new situation observed.
 $g \leftarrow$ new performance measure.
- I3. [Select action & Update $\hat{\phi}$.] If $y=y_k$ and the automaton selects action $u = u_1$, then:

$$\hat{\phi}_{ij} \leftarrow \text{UPDATE}(\hat{\phi}_{ij}, \frac{1}{1+\gamma} (g + \gamma \hat{\phi}_{k1})) ,$$

go to I1. (γ = discount factor, $0 \leq \gamma \leq 1$)

....

Now that the procedure for calculation of the expectance has been described the complete procedure for an automaton that acts as if it were s parallel automata (as shown in figure 5) will be defined. The selection of actions based on a performance measure, that is now an expectance function, will be done by the same BANDIT algorithm described by the G-procedure.

- J1. [Observe.] $y \leftarrow$ environment situation response.
 $g \leftarrow$ environment performance response.
- J2. [Sample.] For $y = y_i$, (situation i , i th automaton)
 $x_i \leftarrow$ sample from $N(\hat{\phi}_{ij}, \hat{\theta}_{ij})$, $i=1, \dots, r$
 ($\hat{\phi}_{ij}$ = expectance estimator, action j)
 ($\hat{\theta}_{ij}$ = variance of $\hat{\phi}_{ij}$)
- J3. [Select action.] If $x_j = \max \{x_k ; k=1, \dots, r\}$,
 $u \leftarrow u_j$. (output action u_j to environment)

J4. [Update expectance.] For $y = y_k$, $u = u_1$,
 and $oldy = y_i$, $oldu = u_j$; (last situation & action)

$$z \leftarrow \frac{1}{1 + \gamma} (oldg + \gamma \hat{\phi}_{k1}) , \quad (0 \leq \gamma \leq 1, \text{ discount factor})$$

$$\hat{\phi}_{ij} \leftarrow \text{UPDATE}(\hat{\phi}_{ij} , z) ,$$

$$\hat{\theta}_{ij} \leftarrow \text{UPDATE}(\hat{\theta}_{ij} , (\hat{\theta}_{ij} - z)^2)$$

J5. [Step on one.]

$$oldy \leftarrow y ,$$

$$oldu \leftarrow u ,$$

$$oldg \leftarrow g ,$$

go to step J1.

7. RESULTS

7.1 BANDIT Automaton Results

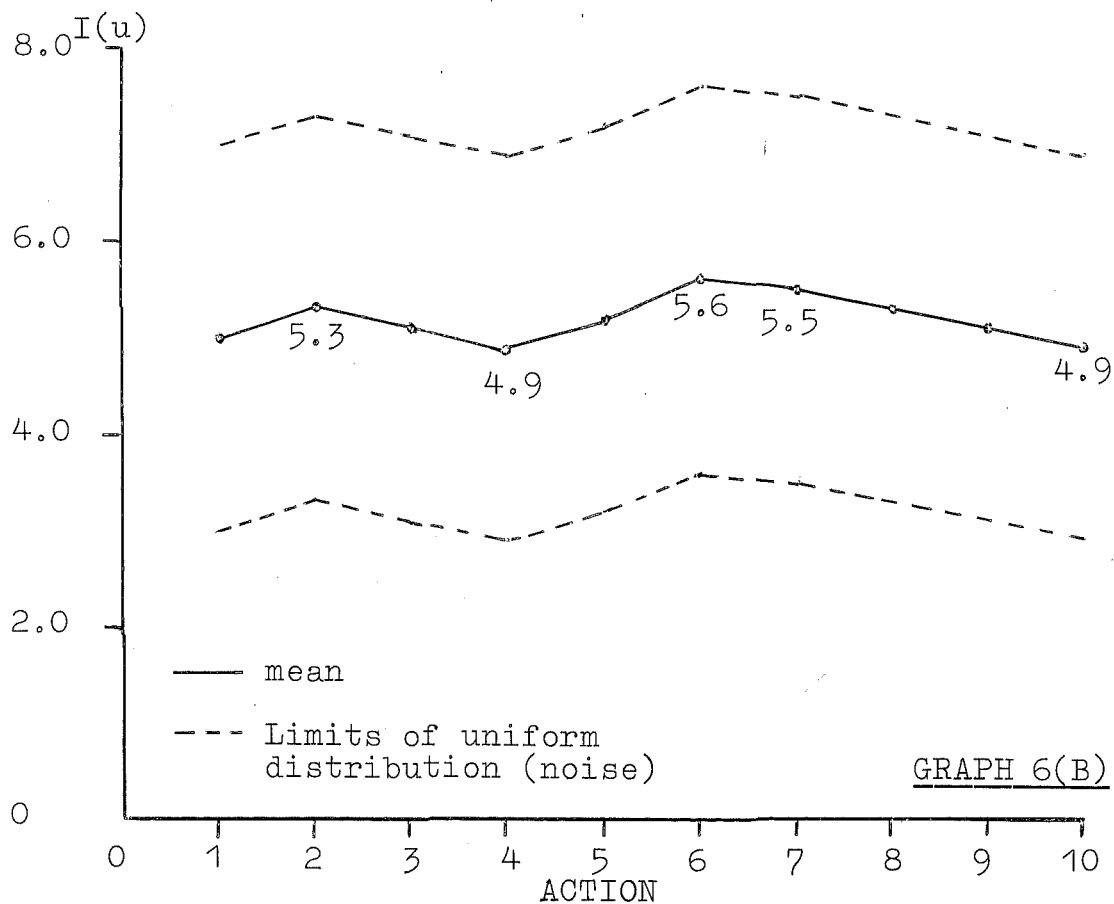
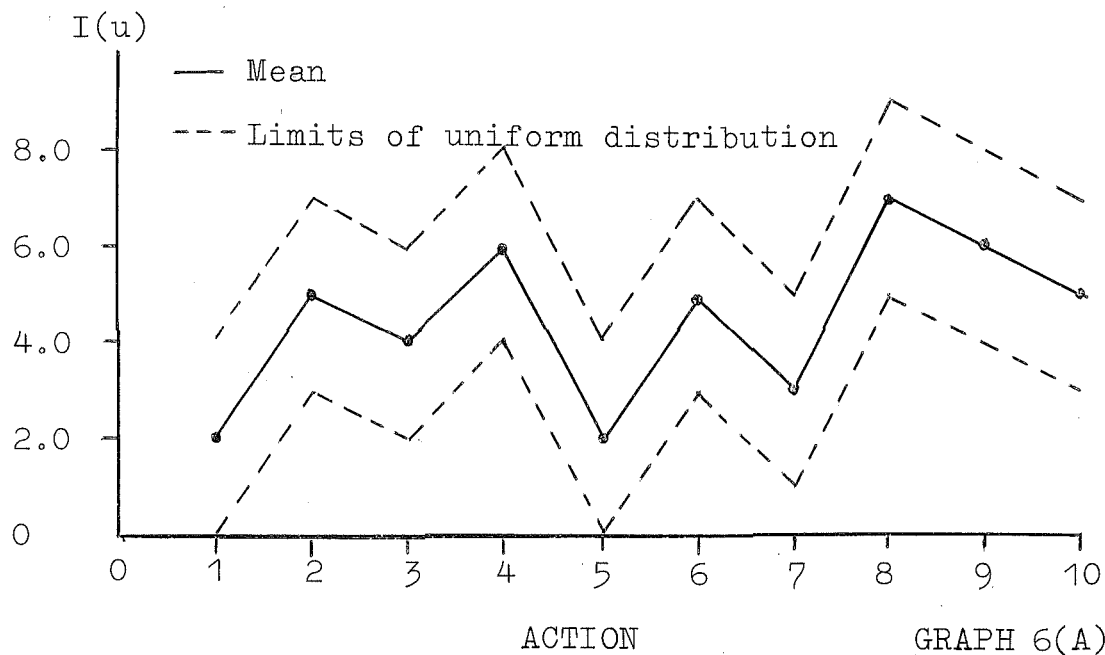
Results from the G-procedure (BANDIT algorithm) and the H-procedure (modified BANDIT algorithm) are compared with results from the F-procedure. The problems used are from Shapiro and Narendra (1969) [14] and their results agree with the results from the F-procedure as they should.

The problem can be described by the two simulated multi-modal performance curves shown in figure 6. The environment performance response g is the height of the performance curve $I(u)$ at the particular value of u output by the automaton, with a superimposed noise uniformly distributed, in the range ± 2 units.

Figure 6 (a) is a relatively easy case, while 6 (b) is a more difficult case since the differences of performance for the different actions are masked by the large noise component.

The particular UPDATE procedures used were a classical mean and variance estimation (B-procedure) for the estimation of g_i and w_i , and a linear reinforcement procedure (C-procedure) for the estimation of p_i 's. This applies to all the results except that the BANDIT algorithm which does not use an explicit UPDATE procedure for the p_i 's. The coefficient, α , used in the linear reinforcement equation (B3) are given on the results in graph 7.

As can be seen from the results in graph 7 the modified BANDIT algorithm (H-procedure) was quicker to converge than the S & N scheme (F-procedure), while the BANDIT algorithm was faster than both. These results are accounted for by the fact that the estimation of w , the variance of the mean estimator g , gives more information per step. Also avoiding a double stage estimation, that is an estimator that is derived from an estimator, gives a faster over-all time constant. Table II summarises these factors as they apply to each of the three procedures (F,G,&H).



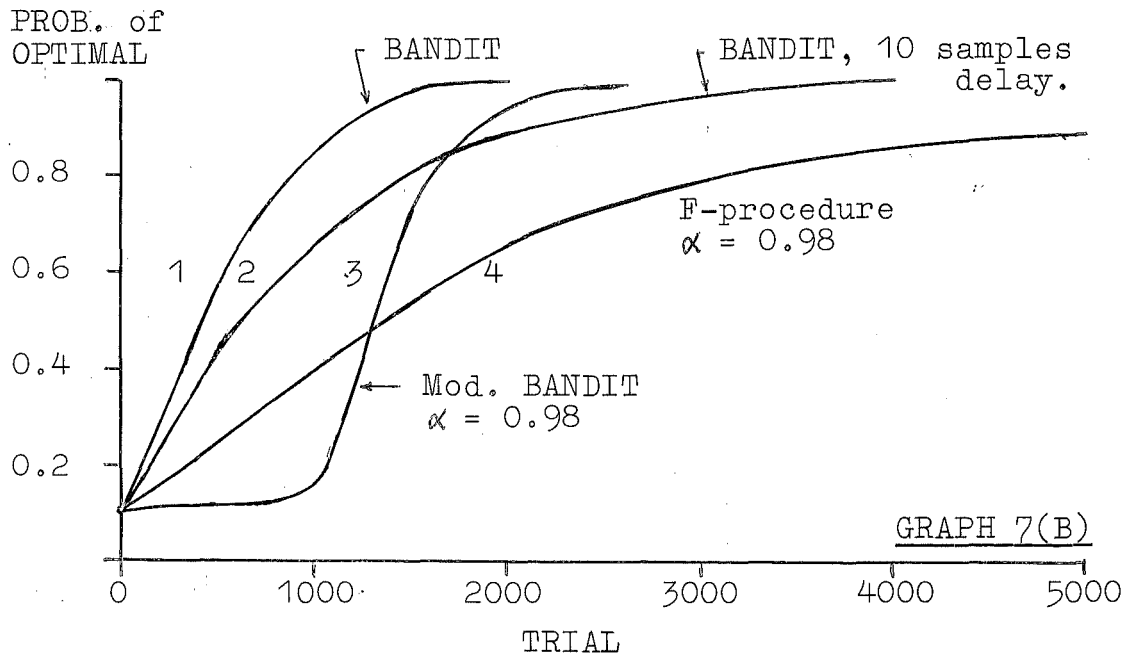
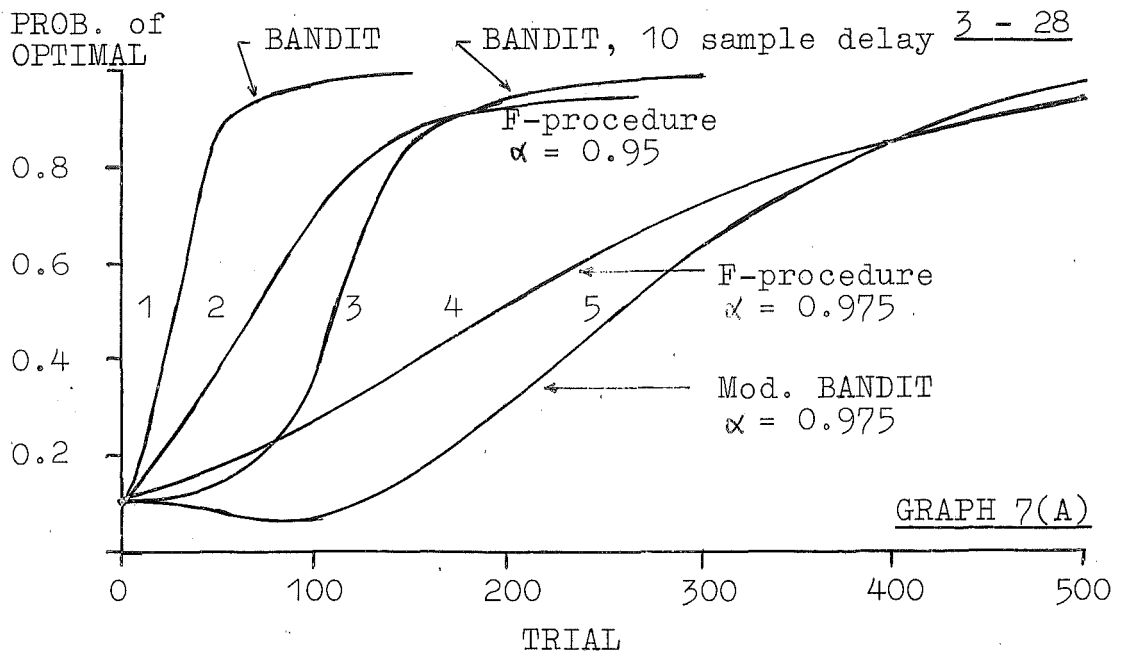


TABLE II

Procedure	Primary estimators mean g.	var of, w.	Secondary estimator action prob. p.
S&N, F-procedure	yes	no	yes
Mod. BANDIT (H)	yes	yes	yes
BANDIT (G)	yes	yes	no

Care is needed in interpreting the results since they are averages over a number of runs (at least 20). It is not true that the algorithms with faster convergence rates are by this fact alone preferable to ones with slower convergence.

The convergence rates of the different algorithms need careful interpretation and for this two definitions will be helpful. Let 'primary convergence' be defined as the condition when the probability of one particular action exceeds some arbitrary value, say 0.99. If primary convergence occurs for a suboptimal action it will be called a 'premature primary convergence'.

Premature primary convergence occurs because there is always a probability that over any finite number of samples for each action, the optimal action does not show itself as the best action available. The smaller the number of samples the greater the probability of the true optimal action having a mean performance estimator that is less than its true mean performance measure and also less than some other suboptimal actions true mean performance measure.

With premature primary convergence there can only be a small probability of the optimal action being used. It is thus very probable that a considerable number of actions will have to be taken before the performance estimator(s) for the optimal action can be updated sufficiently to cause a 'switch' to convergence on the optimal action.

It can be seen that for any given situation the faster the rate of convergence the greater the probability of

a premature primary convergence. Notice that the rate of convergence for any given small probability of premature primary convergence is dependent on the random environment. In the special case where the noise is very small compared to the difference in true mean performance measures for each action, one sample from each action would be sufficient evidence on which to allow primary convergence. For ten actions in this environment primary convergence after 10 or 20 actions could be quite satisfactory. In the environment with performance measure as shown in figure 6(b) the performance measure difference for different actions is 'buried' under all the noise. In this case several thousand actions need to be taken before primary convergence occurs, as shown in the results on graph 7(b).

In the case of the S&N scheme (F-procedure) and also the modified BANDIT algorithm (H-procedure) the rate of convergence can be set by the parameter α in the linear reinforcement equation for the p_i 's. This is not the case for the BANDIT algorithm (G-procedure). However even in the difficult case for the performance measure in graph 6(b) there was about 0.8 probability of convergence in the practical sense on to the optimal action. The result shown in graph 7(b) 2 is better than this (100% optimal convergence over the 20 trial runs) because it incorporates a feature to be described below.

The reason that the BANDIT algorithm can sometimes converge (in a practical sense) onto a suboptimal action is that the optimal action occasionally gives its first few performance responses lower than the long term mean and close to each other. This results in the mean estimator being low and also the variance of the mean estimator being low. Because of the assumption (that is made for computational convenience) that the mean estimator has a normal distribution the optimal action can in this case be 'discarded' by the algorithm. That is, it gets only a small probability of being tried again. There are at least two methods of overcoming this problem without introducing any significant extra computation.

The first method is to use an UPDATE procedure that has the property that it converges smoothly from apriori selected values onto the estimated value. The linear reinforcement equation (C-procedure) or a more complex form of stochastic approximation (Fu & Nickolic 1966 [8]) have this property, while the classical estimators (B-procedure) do not. The reason for using such UPDATE procedures is that the apriori variance of the mean estimator can be set to some suitably high value so that the BANDIT algorithm will need to take a significant number of samples before the probability of any action can become very small. The BANDIT algorithm can be thought of as being forced to try each alternative a number of times before it can be discarded. Alternatively it can be thought of as a relaxation of the assumption of a normal distribution for the mean estimators. A variation of this

approach is to keep using an apriori variance of the mean estimator until some arbitrary number of samples have been acquired. This simple scheme was used for graph 7(b)2 .

A second method for avoiding a long time convergence (in the practical sense) onto a suboptimal action is to decay the variance estimator for each step that the estimator does not get updated. This method is suggested mainly for time varying situations since it essentially says that if a mean estimator has not been updated then the confidence in it should be decreased, or the variance should be increased. Notice that unlike 'forgetting' schemes that have in the past been suggested (Samuel 1963 [13]) this decay of the variance does not alter the value of the mean estimator. Information is not lost, but the confidence in the information is decreased.

Both the methods suggested above improve the assumption that the distribution for the mean estimator is normal. All the above suggestions have been successfully demonstrated, but not fully investigated.

The comments made above apply in the main to the extended problem class where the automaton is learning to maximize the expectance of each move, rather than the performance measure itself.

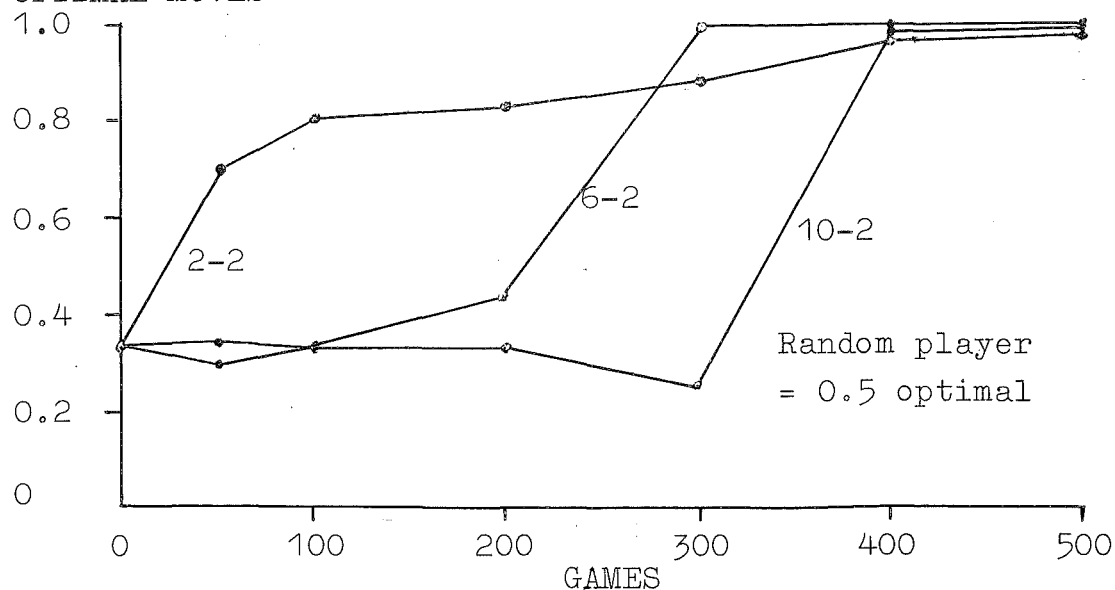
7.2 NIM Results

For the game of NIM discussed earlier (refer to TABLE I) some of the results from use of the BANDIT-EXPECTANCE algorithm, or J-procedure, are shown in figure 8. Only three of the optimal moves are shown simply to keep the figure

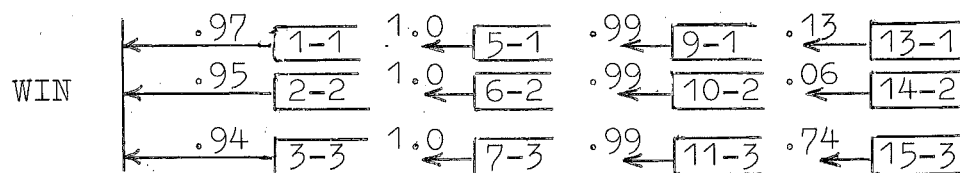
uncluttered. Notice that the first optimal actions to be established are those leading directly to a win; in figure 8 the move shown is for $y=2$, $u=2$, resulting in $g=1$ (TABLE I). After these 'direct win' moves have been established and their expectance values have increased, the moves 'one away from reward' are established. In figure 8 the move for $y=6$, $u=2$ always results in $g=0$, but the next situation must be $y=1, 2$ or 3 for which the optimal moves are now established. Hence the expectance of the moves 'one away from reward' build up and so the process continues.

A point of interest is the fact that the probability of the optimal actions leading to immediate reward (a win) are not as great as those for actions one or more step from immediate reward (see figure 8). This arises from the particular UPDATE procedure used in the BANDIT-EXPECTANCE algorithm (J-procedure). There was no update of an expectance value if the update information (z in step J4) was zero. That is, if the action resulted in situations still having their apriori expectance and no reward ($g=0$), then it was considered that no information had been gained to update the expectance of the action just used. The non optimal moves from $y=1, 2$ or 3 result in $g=0$, and $y=15$ or 16 . Until a complete set of optimal moves has been established (so that optimal moves from $y=15$ and $y=16$ are established) the non optimal moves from $y=1, 2$ or 3 do not have their expectance values updated, for the reason given above. Hence the optimal actions from $y=1, 2$ and 3 are competing with the apriori (non updated) expectance for the non optimal

PROB. of
OPTIMAL MOVES



500 game optimal move probabilities:



1000 game optimal move probabilities

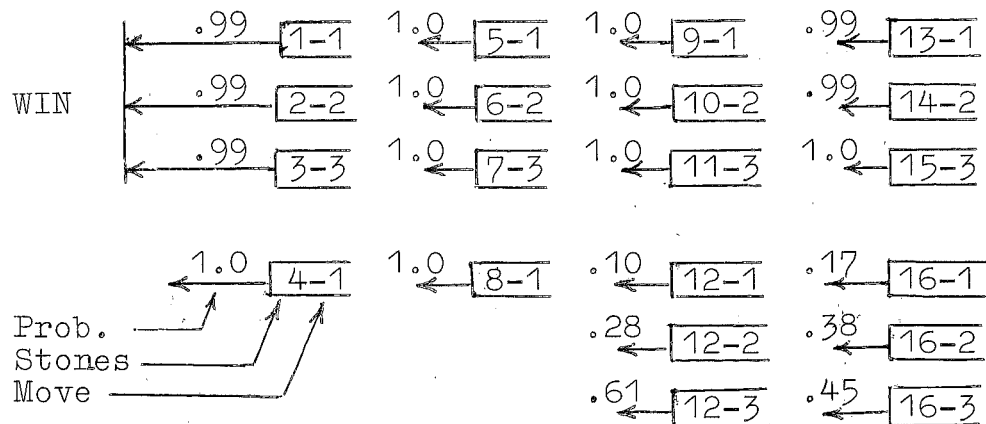


Figure 8

actions, resulting in the lower probability seen in figure 8. The situation can be thought of as 'known good actions verses actions of unknown worth'.

8. CONCLUSIONS

Automata schemes have been described in a new way for the following reasons:

- By their nature schemes for learning stochastic automata are procedures and an algorithmic notation can describe them concisely.
- Algorithmic presentation has been developed in the computer science literature that can be used to advantage.
- An attempt has been made to give this notation some generality so that one procedure can be set down to describe a whole class of particular implementations.
- When presented in this uniform manner it is much easier to compare different schemes, see their differences, get an idea as to the differing computational requirements, and so on.

After introducing the main points in the development of stochastic learning automata, one of the latest published schemes is presented as a procedure in the notation mentioned above. Using this as a basis of comparison a new scheme - the BANDIT scheme - is described. The key points about the BANDIT algorithm are:

- Mean and variance information is gained from each action that the automaton performs, so that the convergence rate can be increased.

- No explicit estimation of the action probabilities is made so that the storage of an extra estimator, variance, need not take additional storage space.
- Estimators that are estimated from other estimators are eliminated.
- The performance measure does not have to be normalized to any particular range.
- The algorithm can be implemented with very modest computation.
- The algorithm is suited for 'on-line' use coping with time varying stochastic environments.

In addition to this new algorithm an extended class of problems for stochastic learning automata are introduced, together with a scheme, the BANDIT-EXPECTANCE algorithm, to enable these problems to be tackled. The main points about this extended problem class are:

- The extended class of problems cannot be satisfactorily tackled by the learning automata discussed in the early parts of this paper.
- This extended class of problems has been considered in 'heuristic learning machine' research, so that a link is formed between this and the stochastic learning automata considered here.
- The extended problem class includes the interesting cases of board games and robot problems.

REFERENCES

- 1 Andreae, J.H., Learning Machines: a unified view.
In Encyclopaedia on Linguistics, Information and Control
Pergamon Press, 1968.
- 2 Brown, R.G., Smoothing Forecasting and Prediction of
Discrete Time Series, Prentice Hall, 1962.
- 3 Bush, P.R. and Mosteller, F., Stochastic Models for
learning, Wiley, 1958.
- 4 Cashin, P.M. The Bandit Algorithm for Minimum Cost Path
Finding with Incomplete Cost Information,
Refer to chapter 2, also Proc. 3rd International Conf.
on system Science. Hawaii 1970.
- 5 Chandrasekaran, B. and Shen, D.W.C., On expediency and
convergence in variable-structure automata, IEEE Trans.
Systems Sci. Cybernetics, SSC-4, No.1, pp52-60, March 1968.
- 6 Doran, J.E. Planning Generalization in an automaton/
environment system, Machine Intelligence 4, pp433-454,
Ed. Meltzer, B. Michie, D., Edinburgh University Press
1969.
- 7 Fu, K.S. and McMurtry, G.J., A variable structure
automaton used as a multi-modal searching technique.
Proc. Nath. Conf., 21, pp494-499. 1965.

REFERENCES

- 8 Fu, K.S. and Nickolic, A.J., On some reinforcement techniques and their relation with stochastic approximation. IEEE Trans, Auto. Control AC-11, pp756-758, 1966.
- 9 Fu K.S., Stochastic Automata as Models of Learning Systems, Computer & Information Science - II, Ed. Tou, J.T. Academic Press, pp177-191, 1967.
- 10 Knuth D.E. The Art of Computer Programming, Vo.1, Fundamental Algorithms, Addison-Wesley, 1968.
- 11 Mendel, J.M., & Fu, K.S. Adaptive, Learning & Pattern Recognition Systems Theory and application. Academic Press 1970.
- 12 Michie, D., and Chambers, R.A., Boxes: An experiment in Adaptive Control, Machine Intelligence 2, pp137-152 Ed. Dale, E. and Michie, D., Oliver & Boyd, 1968.
- 13 Samuel, A.L. Some Studies in Machine Learning using the game of Checkers. Computers & Thought, Ed. Feigenbaum, E.A. and Feldman, J. McGraw - Hill, 1963, pp71-105.
- 14 Shapiro, I.J. and Narendra, K.S., Use of Stochastic Automata for Parameter Self-optimization with Multimodal Performance criteria. IEEE Systems Sci. Cybernetics, SSC-5, No.4, pp352-360, October 1969.

REFERENCES

- 15 Tsetlin, M.L., On the behaviour of finite automata in random media. Automation & Remote Control 22 No.10 pp1210-1219, 1961.
- 16 Varshavskii, V.I. and Verontsova, I.P., On the behaviour of stochastic automata with variable structure. Automation & Remote Control 24, No.3, pp327-333, 1963.
- 17 Yakowitz, S.J. Mathematics of Adaptive Control Processes, Elsevier, 1969.

CHAPTER FOUR

ROTE LEARNING AND MARKOV PROCESSES

CHAPTER FOUR4 - 1 TABLE BUILDING

A table can be constructed to record the history of all machine/environment interactions. The basic events consist of 4-tuples:

$$\langle y, u, z, t \rangle \quad (1)$$

where y = the observed state, $\in Y = \{y_a, y_b, \dots\}$,

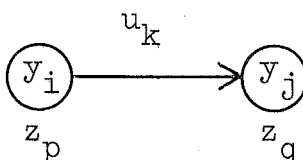
u = the operator applied, $\in u = \{u_a, u_b, \dots\}$,

z = the valuation resulting from this,

and t = the time.

Notice that the z in these 4-tuples is the valuation that is observed after the application of operator u to state y . This is a basic assumption of cause and effect, an example of which is given in figure 1.

Initial
observed state
and valuation



Resulting
observed state
and valuation.

Recorded as

4-tuple: $\langle y_i, u_k, z_q, t \rangle$

Figure 1

We now make the important assumption that the z appearing in $\langle y, u, z, t \rangle$ has an expected value \bar{z} up to this particular event.

That is the expectation of z , $E(z)$, in the event $\langle y, u, z, t_n \rangle$ given by the history $\langle y, u, z, t_{n-1} \rangle, \dots$
 $\dots, \langle y, u, z, t_0 \rangle$, is equal to $E(z)$ of z in the event $\langle y, u, z, t_n \rangle$ with unknown history.

A stronger assumption that will be made for the present but will be relaxed later, is that of time stationary:

$$E(z) \text{ of } \langle y, u, z, t_n \rangle = E(z) \text{ of } \langle y, u, z, t_{n+m} \rangle \quad (2)$$

for all n and m .

Because of these assumptions we can 'condense' the historical record of 4-tuples (1) into a smaller set of triples:

$$\langle y, u, \hat{z} \rangle \quad (3)$$

where \hat{z} = the mean value of z in $\langle y, u, z, t \rangle$ for each unique pair $y = y_i$ and $u = u_j$ over all t .

The \hat{z} will be treated as a running estimate of the true mean value of $z, (\bar{z})$, by updating it after each event. For example, given event $\langle y_i, u_k, z, t \rangle$:

$$\hat{z} \text{ of } \langle y_i, u_k, \hat{z} \rangle \leftarrow \text{UPDATE } (\hat{z} \text{ of } \langle y_i, u_k, \hat{z} \rangle, z')$$

$$z' = z \text{ of } \langle y_i, u_k, z, t \rangle \quad (4)$$

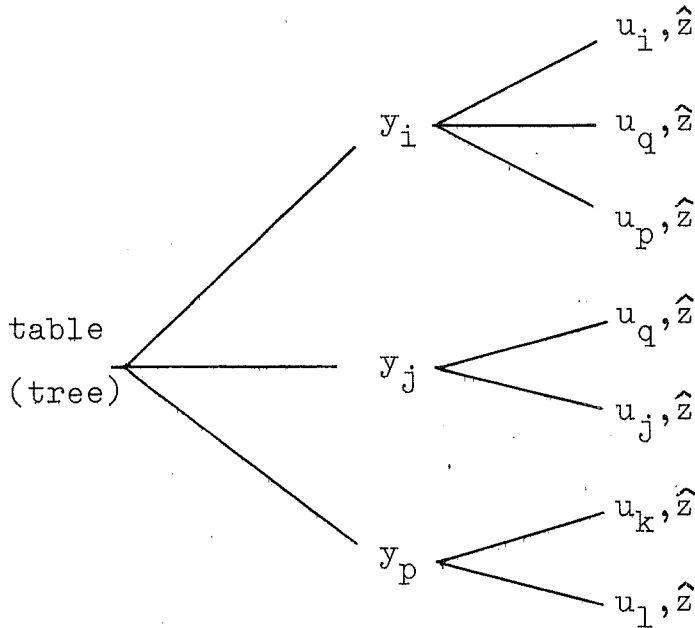
1.1. Information Structure

The set of triples $\langle y, u, \hat{z} \rangle$ can be thought of as recorded in an information table in the form of a tree; represented by the list structure:

$$\text{list } (y, \text{list}(u, \hat{z})) \quad (5)$$

$$\text{or simply } (y, (u, \hat{z})) \quad (6)$$

This list structure has the form of a tree since it has a number of particular y 's, the states that have been observed and recorded in the table, and each y branches out to a number of particular operators u that have been applied to each state. For example:



$$\text{table} = (y, (u, \hat{z}))$$

Figure 2

1.2. Transition Probabilities

Although this table has all the events that have occurred stored in it, it does not give any record of the sequence of events. The table will enable us to answer the question: "what is the expected value $E(z)$ after the use of operator u_k if the current state is y_p ?" But the question: "what is the expected state y if

operator u_k is used from the current state y_p ?", cannot be answered from the table.

This information will be required, and it can be included in the table (tree) by inserting an estimate of each particular state following particular state operator pairs, as they are observed. For example, if state y_j has been observed as a result of the use of operator u_k in state y_i , then the table will have an estimator for the probability of y_j given y_i, u_k , i.e. $\Pr(y_j | y_i, u_k)$. The table will contain an entry (or a branch in the information tree):

$$(y_i, (u_k, \hat{z}, (y_j, \hat{p}))) \quad (7)$$

where \hat{p} = estimate for $\Pr(y_j | y_i, u_k)$. The form of the table is now:

$$\text{list}(y, \text{list}(u, \hat{z}, \text{list}(y, \hat{p}))) \quad (8)$$

or $(y, (u, \hat{z}, (y, \hat{p}))) \quad (9)$

To make the table construction quite clear consider the situation illustrated in figure 3.

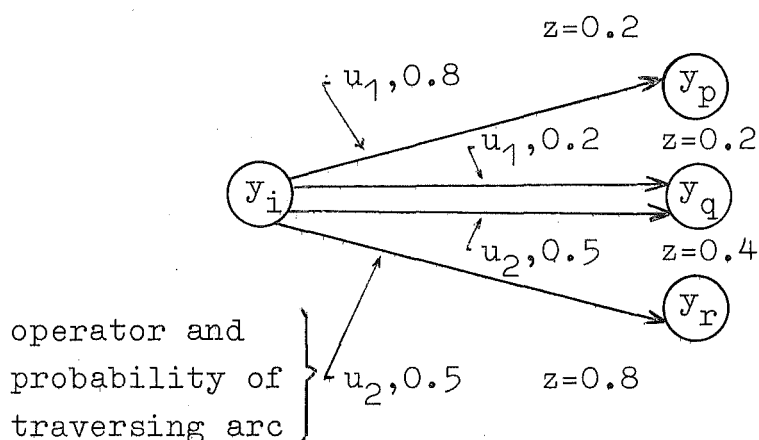


Table entries:

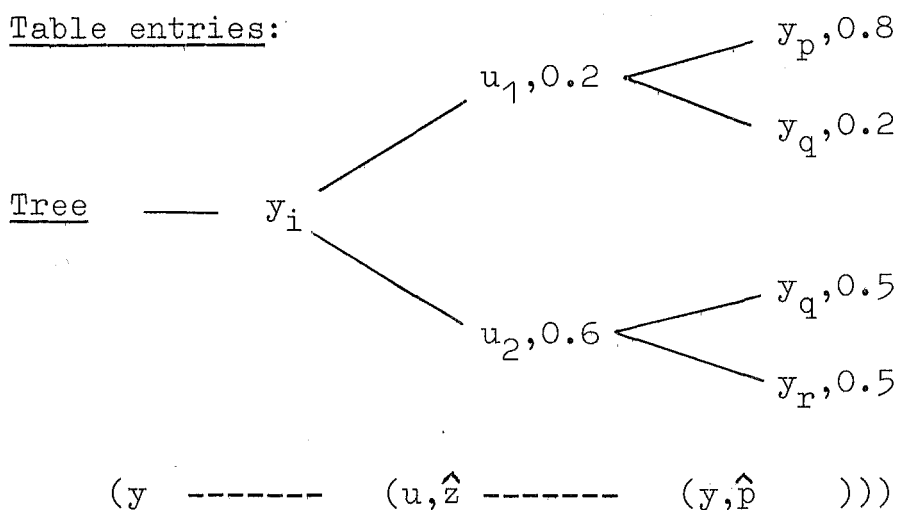


Figure 3

1.3. Information Extraction

Consideration will now be given to the extraction of information from the information tree. The form $(y, (u, \hat{z}, (y, \hat{p})))$ can take a slightly more general form:

$$(y, \dots, x', \dots, (u, \dots, x'', \dots, (y', \dots, x''', \dots)))$$

where x' = information item pertaining to y ,

x'' = information pertaining to operator u applied to state y ,

x''' = information pertaining to state y' that is observed after the use of operator u in state y .

Now to extract the items x' , x'' and x''' from the table the following notation will be used:

$$x' < y > \qquad x'' < y, u > \qquad x''' < y, u, y' >$$

Examples of this notation used on the table given in the example of figure 3

$$\hat{z} < y_i, u_2 > = 0.6$$

$$\hat{p} < y_i, u_1, y_q > = 0.2$$

and $\hat{z} < y_i, \forall u > = \{0.2, 0.6\}$, \forall implies all instances of u in the table, given y_i .

4 - 2 OPERATOR SELECTION STRATEGY

Consideration is now given to the utilization of the table $(y, (u, \hat{z}, (y, \hat{p})))$ in order to select a 'good' operator u to apply to a given state y . To decide if an operator is 'good' or not it is necessary to define a criterion or objective function.

Objective_1: Given a state y , choose u such that the expected value of the resulting valuation z is maximized:

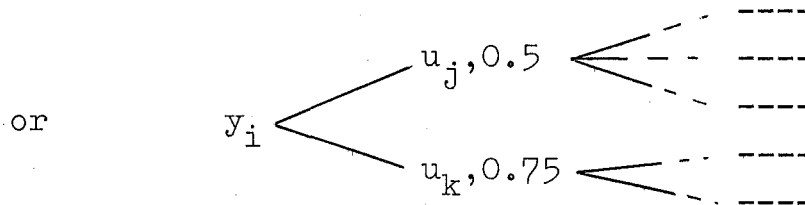
In order to meet objective_1 the first strategy that comes to mind is simply to choose the operator u such that $\hat{z} < y, u >$ is maximized:

U_MAX_Z(y): Find u_k such that
 $\hat{z}\langle y, u_k \rangle = \text{maximum over } \hat{z}\langle y, \forall u \rangle,$
 $U_MAX_Z(y) \leftarrow u_k.$ (Random choice for
several equal max.)

This seemingly reasonable strategy suffers a severe defect as will be illustrated by the following example:

Table: $(y, (u, \hat{z}, (y, \hat{p})))$

Entry: $y_i((u_j, 0.5((---)), (u_k, 0.75, (---)))$



for which $U_MAX_Z(y_i) = u_k.$

Now $\hat{z}\langle y_i, u_j \rangle = 0.5$ and $\hat{z}\langle y_i, u_k \rangle = 0.75$, but it could well be that the true values are 0.8 and 0.75, since \hat{z} is only an estimate, and it may be based on only a very few samples. In this case the operator that satisfies objective_1 is $u = u_j$, not $u = u_k$ as given by $U_MAX_Z(y_i)$. The serious defect with $U_MAX_Z(y)$ is that not only can it be misled, but as in this example it will never discover its error, i.e. $\hat{z}\langle y_i, u_k \rangle$ will be updated (and retain its value of 0.75) but $\hat{z}\langle y_i, u_j \rangle = 0.5$ will never be updated to reveal the true value of $\bar{z} = 0.8$ for operator u_j .

This example illustrates a general defect in $U_MAX_Z(y)$ that arises from the basic problem of utilization of information, versus the gathering of more information. This is a key problem in machine learning and the same problem posed by the Two Armed Bandit problem [2], [3]. To overcome this problem some form of probabilistic selection must be used.

2.1. Probabilistic Selection

Consider first the probability of different operators u being selected by $U_MAX_Z(y)$, using the example from the last section —

$$\begin{aligned} \Pr(U_MAX_Z(y_i) = u_j) &= 0 \\ \Pr(U_MAX_Z(y_i) = u_k) &= 1 \end{aligned} \quad (10)$$

This is typical of $U_MAX_Z(y)$, unless there are several equal maximum values in the set $\hat{Z}\langle y_i, \forall u \rangle$ the probability of selecting any given u will be 0 or 1. It is this deterministic property that can cause $U_MAX_Z(y)$ to get 'stuck' and keep returning a u that does not satisfy objective_1.

A probabilistic operator selection strategy will now be defined. This strategy, $U_RAN_Z(y)$ will select an operator with a probability determined by the relative value of \hat{Z} for each alternative. This strategy may be called a 'linear probability weighting rule'.

$U_RAN_Z(y)$: Label $\hat{z}\langle y, \forall u \rangle$ as $\{z(1), z(2), \dots\}$,
 $ZTOT \leftarrow \sum_{\forall u} \hat{z}\langle y, u \rangle$,
 $R \leftarrow$ a random number in the range 0 to 1,
 $R \leftarrow R \cdot ZTOT$ (scale the range)
 $Z \leftarrow 0$, $I \leftarrow 1$,
loop: $Z \leftarrow Z + z(I)$
If $Z \geq R$ go to set,
else $I \leftarrow I + 1$, go to loop.
set: If $z(I) = \hat{z}\langle y, u_k \rangle$ then
 $U_RAN_Z(y) \leftarrow u_k$. (11)

Now we will write

$Pr(u_i | y_j)$ to mean $Pr(U_RAN_Z(y_j) = u_i)$. (12)

For any operator selection strategy we desire one and only one operator to be selected so that

$$\sum_{\forall i} Pr(u_i | y_j) = 1, \forall j. \quad (13)$$

This will be true for $Pr(u_i | y_j)$ defined by equation (12), and if (as we shall later) the procedure $U_RAN_Z(y)$ is replaced by some other procedure it must be ensured that (13) still holds true.

4 - 3 PLANNING FROM ROTE LEARNING

In the last section we considered the selection of an operator u given some observed state y in order to maximize the expected value of z , as dictated by objective_1. This is planning, but only in a limited sense since it is only the immediate outcome of an operator that is considered. As sufficient rote learning is entered into the information table it is possible to plan not just for the immediate outcome of an operator but the planning horizon can be extended to maximize the expected value of z over 2,3 or many more steps into the future.

In this section it will be assumed that there is a procedure ($U_MAX_Z(y)$, $U_RAN_Z(y)$ or some other) that has been defined to determine $Pr(u|y)$. Given this it is possible to calculate state transition probabilities and the expected value of each state, which are needed to be able to plan over many steps into the future.

3.1. State transition probabilities

The estimated probability of observing state y_j after use of action u_k while in state y_i can be extracted directly from the rote learning table as $\hat{p}\langle y_i, u_k, y_j \rangle$. The state transition probability $Pr(y_j | y_i)$ is the probability of observing state y_j given state y_i . These transition probabilities can be estimated in the following manner:

$$\Pr(y_j | y_i) = \sum_{\forall k} \Pr(y_j | y_i, u_k) \cdot \Pr(u_k | y_i) \quad (14)$$

for which an estimate can be made using:

$$\hat{\Pr}(y_j | y_i) = \sum_{\forall k} \hat{p}_{\langle y_i, u_k, y_j \rangle} \cdot \Pr(u_k | y_i) \quad (15)$$

3.2. State valuation

The valuation z exhibited by the environment has been attributed to the preceeding state action pair (Section 4 - 1). To evaluate the expected value of a state $E(z|y)$ the following equations can be used:

$$E(z|y_i) = \sum_{\forall k} E(z | y_i, u_k) \cdot \Pr(u_k | y_i), \quad (16)$$

for which an estimate can be made using:

$$\hat{E}(z|y_i) = \sum_{\forall k} \hat{z}_{\langle y_i, u_k \rangle} \cdot \Pr(u_k | y_i) \quad (17)$$

4 - 4 INTERACTION AS A MARKOV PROCESS

4.1. Relationship to Markov Processes

A Markov process is a mathematical model that is useful in the study of complex systems. The purpose of relating the machine-environment interaction to a Markov process is to enable use to be made of results established for this mathematical model [7]. In particular, reference will be made to the book by R. Howard, 'Dynamic Programming and Markov Processes', [5].

The basic concepts of a Markov process are those of 'state' of a system and state 'transition'. A system is said to occupy a state when it is completely described by the values of the variables that define the state. For a simple Markov process the state transition probabilities are dependent on the current state and not on the previous states.

The system we are now considering is the complete machine-environment with the state of the system taken to be the observation of the state of the environment y , with the machine and environment 'frozen' in time.

For the frozen system:

1. The environment is assumed to retain its current performance so that $\Pr(y | u)$ remains constant.
2. The machine is assumed to retain its current performance so that $\Pr(u | y)$ remains constant, also the table $(y, (u, \hat{z}, (y, \hat{p})))$ will be constant and not updated after each machine-environment interaction step.
3. The machine is assumed to have some memory that is not part of the rote learning table - a scratch pad memory. This memory is used for planning operations. It may for example hold a temporary estimate of the probability of a transition from y_i to y_j by any 3 step path. This memory can be actively used in the frozen state.

The frozen system can be thought of as continuing to interact with time frozen, by means of a simulation of reality, or in more colourful terms as 'thinking' or 'dreaming'. This is the mode that the machine will use to undertake planning into the future on the basis of its current (frozen) knowledge.

It is important to remember that we are considering the machine-environment system as a Markov system at some given time. At some time later it may again be considered as a Markov process, but not necessarily the same Markov process. The transition probabilities, for example, may have changed considerably with better estimates available; more than this the states in the Markov process may have been expanded as more entries are made into the table $(y, (u, \hat{z}, (y, \hat{p})))$.

4.2. Total Expected Earnings

Let us now consider the observed value of z at each step as the 'earnings' for that step. With this interpretation a question we may ask is: What will be the expected total earnings in the next n steps? This question has been answered for a Markov process by Howard [5], and the following sections outline the methods that can be used.

Let $v(n | y_i)^*$ be defined as the expected total earnings over the next n steps given the current state is y_i . From this definition we can formulate the recursive relationship,

$$v(n | y_i) = E(z | y_i) + \sum_{y_j} \text{Pr}(y_j | y_i) \cdot v(n-1 | y_j). \quad (18)$$

In a step from y_i to y_j , the expected earnings are the expected value of z , $E(z | y_i)$. The total earnings are the expected value for this one step plus the total expected earnings with one fewer step remaining from the state y_j , weighted by the probability of a transition from state y_i to y_j .

4.3. Value Iteration

We now look at the problem of determining the operator selection rule to achieve objective_2:

Objective_2: Maximize the expected total earnings over the next n step period.

The operator selection rule can be thought of as giving a decision $d(y)$ that determines the operator for state y ; if this decision is dependent on the stage n then the decision will be denoted as $d(y, n)$. When $d(y, n)$ has been specified for all y and for all n then

*Note that $v(n | y_i)$ is not in the rote-learning table; it is in another memory space (scratch pad), and can thus not be denoted $u(n) \langle y_i \rangle$.

a 'policy' has been determined. The optimal policy is the policy which meets objective_2 and maximizes the expected total earnings.

We now redefine $v(n | y_i)$ as the total expected return in n steps starting from state y_i , if an optimal policy is followed. For any n :

$$v(n | y_i) = \underset{\substack{\text{maximum} \\ \text{over all} \\ \text{possible} \\ \text{policies.}}}{\left[E(z | y_i) + \sum_{\forall j} \text{Pr}(y_j | y_i) \cdot v(n-1 | y_j) \right]} \quad (19)$$

This is the application of the 'Principle of Optimality' of dynamic programming to the Markovian decision process, as given by Howard [5] and, along with other applications, Bellman [2].

The solution of the recursive relation (19) gives the set of decisions, $d(y, n)$, that determine the operator u to be selected for each y at each stage n in order to follow an optimal policy.

This method of finding the optimal policy is called the value-iteration method since the $v(n|y)$ or 'values' are determined iteratively. The following sections will indicate the basis of an alternative method directed at long term optimal policies, that is at decisions when n is very large.

4.4. Discounting

Discounting has the effect of giving less and less weight to steps further and further ahead as we are planning. Planning is done on the Markov system corresponding to the frozen machine-environment as a 'thinking' process, as discussed previously. In economic terms the present value of earnings has a greater value than that of future earnings; for the learning machine the future is uncertain and future earnings are not as certain as immediate expected earnings. Discounting can be thought of as describing a process with uncertain duration, the discount factor being the probability that the process will continue to earn after each step.

The expected value of earnings over n steps with a discount factor of γ can be written as

$$v(n | y_i) = E(z | y_i) - \gamma \sum_{y_j} \text{Pr}(y_j | y_i) \cdot v(n-1 | y_j) \quad (20)$$

where $0 \leq \gamma < 1$.

4.5. Policy Improvement

The policy-improvement method (Howard [5]) of obtaining an optimal policy is aimed directly at long term policies where the decisions are for large n , and there is a discount factor γ . Without the discount factor it is still possible to use a method that is very similar to the policy-improvement routine but it requires consideration of asymptotic behaviour and will not be

included in the present discussion.

The basis of the policy-improvement method is the replacement of the expected total value for n steps, $v(n|y)$, with the limit as n tends to infinity $v(y)$, called the present value. With this substitution we obtain

$$v(y_i) = E(z | y_i) + \gamma \sum_{y_j} \text{Pr}(y_j | y_i) \cdot v(y_j) \quad (21)$$

For a given set of transition probabilities and a given set of expected immediate earnings equation (21) can be used to find the present value $v(y_i)$ for each state y_i , $i=1,2,\dots$. The particular values for the expected immediate earnings $E(z|y)$ and the transition probabilities $\text{Pr}(y_j | y_i)$ are dependent on the particular policy that is being used.

The optimal policy is the one that has the highest present value, $v(y)$, in all states. Suppose that the present values for an arbitrary policy have been determined. Then a better policy, one with higher present values in every state, can be found by the following procedure, which is called the policy-improvement routine.

For each state y_i , find the decision $d(y_i)$ which gives the operator that maximizes

$$E(z | y_i) + \gamma \sum_{y_j} \text{Pr}(y_j | y_i) \cdot v(y_j) \quad (22)$$

using the present values $v(y)$ determined for the original policy. When a new decision $d(y_i)$ has been found for every state, then a new policy has been determined. At this point we can go back to the present value equations (21) to determine the new present values for the new policy. This iterative loop has been shown to converge onto the optimal policy, each successive iteration produces a better policy (with higher present values $v(y)$) so that the optimal policy is found when two iterations produce the same policy.

The policy-improvement iteration loop as it could be carried out using estimates from the rote learning table $(y, (u, \hat{z}, (y, \hat{p})))$, is shown in figure 1. The present values $\hat{v}(y)$ are given a hat to indicate that they are estimates because they are based on the current values in the rote learning table. Although this procedure produces an optimal policy it is an optimal policy on the basis of the current estimates in the rote learning table and this is of key importance in the learning machine system since the improvement of the table is dependent on the policy that is used. The deficiency of an optimal policy in this regard is considered in the following section.

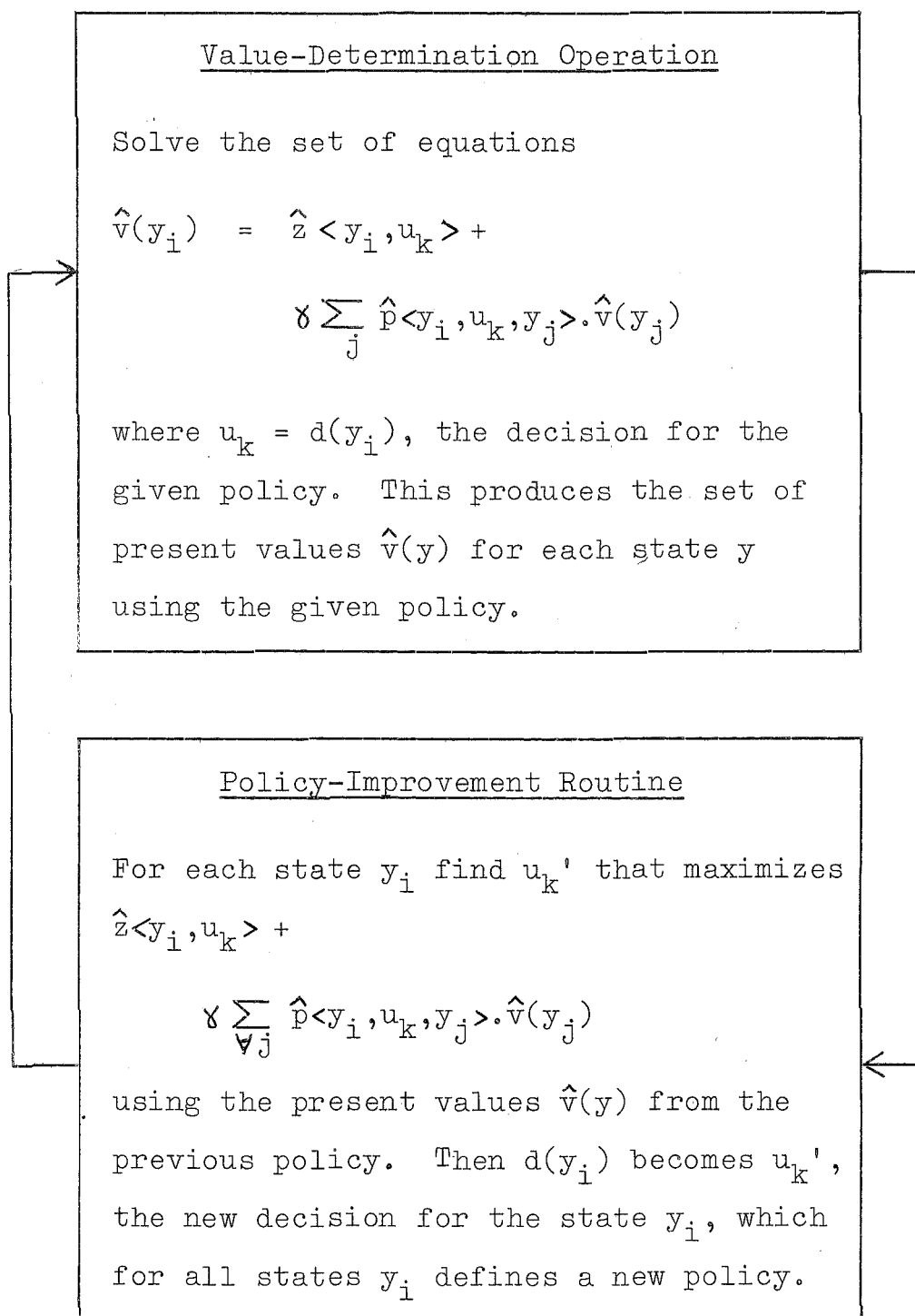


Figure 4.1: Iteration cycle for discrete decision process with discounting. Following Howard [5].

4 - 5 AN EXAMPLE OF OPTIMAL POLICY FAILURE

5.1. Failure of the Optimal Policy

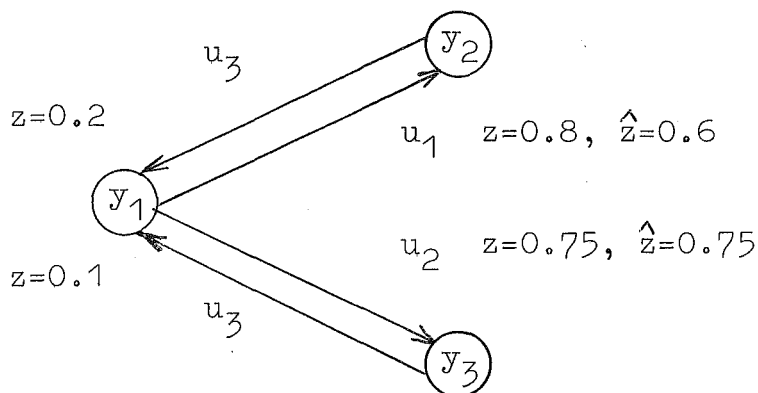
The last section considered the machine-environment as frozen and then showed how to obtain the optimal policy for such situations. Optimality was defined as the maximization of total expected earnings over an n step period with a discount factor γ . For a limited range of n the value-iteration method was given and for large n the policy-improvement method.

The optimal policy consists of a set of decisions, $d(y,n)$, giving the operator u to be used for state y at step n . This policy has the property that $\Pr(u | y,n)$ will be 1 or 0. As discussed in Section 4-2 this form of deterministic operator selection policy takes no account of the uncertainty in the properties of the Markov process it is based on. The optimal policy can get 'stuck' by not allowing for the update of transition probability estimates and immediate expected earnings estimates in the rote learning table.

The two examples in the following section will illustrate the use of both the value-iteration and the policy-improvement methods, and will show for this very simple system how the optimal policy can be undesirable.

5.2. Policy Planning Example

The Markov system will be taken to be:



$$\left. \begin{aligned}
 \Pr(y_2|y_1, u_1) &= 1.0, \quad E(z|y_1, u_1) = 0.8 \\
 \Pr(y_3|y_1, u_2) &= 1.0, \quad E(z|y_1, u_2) = 0.75 \\
 \Pr(y_1|y_2, u_3) &= 1.0, \quad E(z|y_2, u_3) = 0.2 \\
 \Pr(y_1|y_3, u_3) &= 1.0, \quad E(z|y_3, u_3) = 0.1
 \end{aligned} \right\} \text{ True values.}$$

For this example the operator selection decisions for states y_2 , and y_3 , are uniquely determined since there is only one possible operator, u_3 .

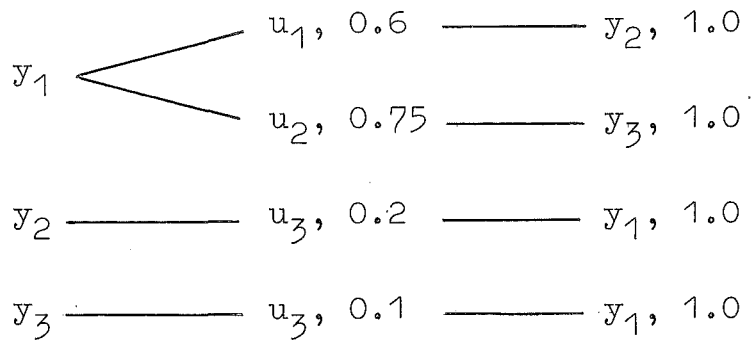
$$d(y_2, n) = u_3, \text{ and } d(y_3, n) = u_3.$$

The optimal policy will determine whether

$$d(y_1, n) = u_1 \text{ or } u_2.$$

Assume that the state of the rote learning table is:

$(y, \text{----} (u, \hat{z}, \text{----} (y, \hat{p} \text{ })))$



Note that $\hat{z}_{\langle y_1, u_1 \rangle}$ is the only estimate not = the true value.

Value-iteration method

The basic equation used by the value-iteration method is

$$v(n|y_i) = \underset{\text{over all policies}}{\text{maximum}} \left[E(z|y_i) + \gamma \sum_j \Pr(y_j|y_i) \cdot v(n-1|y_j) \right]$$

and when computed by use of the rote learning table

$$\hat{v}(n|y_i) = \underset{\substack{\text{maximum} \\ \text{over all} \\ d(y_i, n)}}{\left[\hat{z}_{\langle y_i, d(y_i, n) \rangle} + \gamma \sum_j \hat{p}_{\langle y_i, d(y_i, n), y_j \rangle} \cdot \hat{v}(n-1|y_j) \right]}$$

With $\gamma = 0.5$, and using $\hat{v}(0|y) = 0$, the value-iteration method produces the following -

n=	0	1	2	3	4	5	6 ...
$\hat{v}(n y_1)$	0	0.75	0.800	0.988	1.000	1.047
$\hat{v}(n y_2)$	0	0.20	0.575	0.600	0.694	0.700	...
$\hat{v}(n y_3)$	0	0.10	0.475	0.500	0.594	0.600	...
$d(y_1, n)$	-	u_2	u_2	u_2	u_2	u_2

An example step in this process

If $d(y_1, 2) = u_2$

$$\hat{v}(2|y_1) = 0.75 + 0.5 * 1.0 * 0.10 = 0.80$$

If $d(y_1, 2) = u_1$

$$\hat{v}(2|y_1) = 0.6 + 0.5 * 1.0 * 0.20 = 0.70$$

and so for the optimal policy $d(y_1, 2) = u_2$.

For the true values (not known in the rote-learning table) -

If $d(y_1, 2) = u_2$

$$\hat{v}(2|y_1) = 0.75 + 0.5 * 1.0 * 0.10 = 0.80$$

If $d(y_1, 2) = u_1$

$$\hat{v}(2|y_1) = 0.80 + 0.5 * 1.0 * 0.20 = 0.90$$

and so the true optimal policy $d(y_1, 2) = u_1$.

Policy-improvement method

We shall start the policy-improvement iteration with the assumption of the decision $d(y_1) = u_1$. Remember that n is assumed to be large and v (the expected

total earnings for n steps $v(n|y)$ have been replaced by the present values $v(y)$.

Value-determination:

$$\hat{v}(y_i) = \hat{z}_{\langle y_i, d(y_i) \rangle} + \gamma \sum_{y_j} \hat{p}_{\langle y_i, d(y_i), y_j \rangle} \cdot \hat{v}(y_j)$$

and for $d(y_1) = u_1$

$$\hat{v}(y_1) = 0.6 + 0.5(1.0 \ v(y_2))$$

$$\hat{v}(y_2) = 0.2 + 0.5(1.0 \ v(y_1))$$

$$\hat{v}(y_3) = 0.1 + 0.5(1.0 \ v(y_1))$$

giving

$$\hat{v}(y_1) = 0.934$$

$$\hat{v}(y_2) = 0.667$$

$$\hat{v}(y_3) = 0.567$$

and now the policy-improvement routine can be used, for maximizing

$$\hat{z}_{\langle y_i, u_k \rangle} + \gamma \sum_{y_j} \hat{p}_{\langle y_i, u_k, y_j \rangle} \cdot \hat{v}(y_j)$$

$d(y_1) = u_1$ gives

$$0.6 + 0.5(1.0 \ 0.667) = 0.933$$

$d(y_1) = u_2$ gives

$$0.75 + 0.5(1.0 \ 0.567) = 1.03 \quad \dots \text{maximum}$$

now back to the value-determination with $d(y_1) = u_2$ gives

$$\hat{v}(y_1) = 0.75 + 0.5(1.0 \ \hat{v}(y_3))$$

$$\hat{v}(y_2) = 0.2 + 0.5(1.0 \ \hat{v}(y_1))$$

$$\hat{v}(y_3) = 0.1 + 0.5(1.0 \ \hat{v}(y_1))$$

with solutions

$$\hat{v}(y_1) = 1.07$$

$$\hat{v}(y_2) = 0.745$$

$$\hat{v}(y_3) = 0.645$$

Returning once more to the policy-improvement routine with this set of present values yields

for $d(y_1) = u_1$

$$0.6 + 0.5(1.0 + 0.745) = 0.973$$

and for $d(y_1) = u_2$

$$0.75 + 0.5(1.0 + 0.645) = 1.073 \dots \text{maximum.}$$

Because the policy-improvement routine has produced the same policy with $d(y_1) = u_2$, this must be the optimum policy and the iteration terminates. (The policy is only optimum with respect to the values in the table, it is not necessarily the true optimal policy).

Comments

The optimal policy based on the table with $\hat{z}_{\langle y_1, u_1 \rangle} = 0.6$ gives $d(y_1) = u_2$. If the table were updated so that $\hat{z}_{\langle y_1, u_1 \rangle}$ converged onto the true value of 0.8, then the optimal policy would become $d(y_1) = u_1$. If the optimal policy as found above is used exclusively then this updating will never occur, and the true optimal policy will never be discovered.

The optimal policy on the basis of a frozen system is thus not suitable to determine the policy for a learning machine because the learning (by estimator

updating) can be impeded, as in the example just considered. The following sections are based on the idea of an optimal policy but take into account the uncertainty in the estimators contained in the rote learning table.

4 - 6 STOCHASTIC SIMULATION

6.1. Operator Selection Probability

The present value equations that have been used are,

$$v(y_i) = E(z|y_i) + \gamma \sum_j \text{Pr}(y_j|y_i) \cdot v(y_j) \quad (23)$$

and for a particular policy with decision $d(y_i)$,

$$\hat{v}(y_i) = \hat{z}_{\langle y_i, d(y_i) \rangle} + \gamma \sum_j \hat{p}_{\langle y_i, d(y_i), y_j \rangle} \cdot \hat{v}(y_j) \quad (24)$$

Now we will return to the idea used in 4-2(1) that $\text{Pr}(u|y)$ is not restricted to the value 0 or 1, and the present value equations become,

$$\begin{aligned} \hat{v}(y_i) = & \sum_k \text{Pr}(u_k|y_i) \cdot \hat{z}_{\langle y_i, u_k \rangle} + \\ & \gamma \sum_j \sum_k \text{Pr}(u_k|y_i) \cdot \hat{p}_{\langle y_i, u_k, y_j \rangle} \cdot \hat{v}(y_j) \end{aligned} \quad (25)$$

When there is a policy with this probabilistic nature, $d(y)$ may be a range of possible operators u , with associated probabilities $\text{Pr}(u|y)$. To keep this in

mind decisions for such policies will be denoted $D(y)$ rather than $d(y)$ to indicate that the decision does not always result in the selection of the same operator u .

$$D(y_i) = u_k \text{ with probability } \Pr(u_k | y_i) \quad (26)$$

6.2. Simulation Method

The value-determination routine in the policy-improvement method if extended to the present value equations (25) would treat them as a set of simultaneous equations and solve them by standard techniques. An alternative method for solving these equations is to use an iterative method that simulates the machine-environment interaction.

To use this simulation method we will need a routine $\text{SIMULATE}(y, u)$ that, for a given state y and operator u , uses the rote learning table $(y, (u, \hat{z}, (y, \hat{p})))$ to select a subsequent state.

$$\Pr [\text{SIMULATE}(y_i, u_k) = y_j] = \Pr(y_j | y_i) \quad (27)$$

and this is estimated by

$$\Pr [\text{SIMULATE}(y_i, u_k) = y_j] = \hat{p}_{\langle y_i, u_k, u_j \rangle} \quad (28)$$

We can now simulate a trip from any state, say y_i , and as we travel the present value $\hat{v}(y_i)$ can be updated by use of the simulated z and $\hat{v}(y_j)$ that result from a

step to y_j from y_i . If a number of trips starting from y_i are simulated, then the present value $\hat{v}(y_i)$ will converge onto the value given by the particular policy decision being used, $D(y_i)$. There will be a number of different trips involved, the probability of any particular route depending on the transition probabilities $\Pr(y_j|y_i)$ and the operator selection probability $\Pr(u|y_i)$.

An algorithm for the simulation approach will now be given. The main points should now be clear and the details will be discussed in what follows.

TRIP(y_i):

1. [Initialize] $p' \leftarrow 1$, $y \leftarrow y_i$. (p' will keep track of the probability along the path.)
 2. [Select operator] $u \leftarrow D(y)$. ($u = u_k$ with probability $\Pr(u_k|y_i)$.)
 3. [Expected value] $z \leftarrow \hat{z}\langle y, u \rangle$
 4. [Simulate a step] $y' \leftarrow \text{SIMULATE}(y, u)$ ($y' = y_j$ with probability $\Pr(y_j|y_i)$)
 5. [Update present value estimator]
 $\hat{v}(y) \leftarrow \text{UPDATE}(v(y), z + \gamma \cdot \hat{v}(y'))$
 6. [Check probability of travelling this far]
 $p' \leftarrow p' \cdot \hat{p}\langle y, u, y' \rangle \cdot \gamma$
 If $p' < p_{\min}$, exit from TRIP.
 7. [Simulate move onto next step]
 $y \leftarrow y'$, go to 2.
-

Notice that the direct solution to the present value equations (25) will give the present value for all states whereas the simulation method can generate the present value for state y_i , $\hat{v}(y_i)$, without necessarily evaluating the present value of all the other states. To see this imagine the states as a network connected together by possible paths, the present value of a state is determined by its location in the network; that is it depends on the present values of all its near neighbours. This comes about because the transition probability for a long trip away from the immediate neighbourhood of the starting state is the product of the probability for each step. Just as important the discount factor weights the contribution of distant states to make them less important. For example, if the discount factor is 0.5 then the importance of direct neighbours (one step from the start) is at least 1024 times the importance of the present value of states 10 steps away.

In step 6 of the TRIP algorithm the probability p' is updated to give the probability of travelling to this point from the start, weighted by the discount factor γ at each step. This variable p' is used to determine if it is worth continuing on this trip, in a similar manner to a convergence error stopping an iterative calculation.

To evaluate $\hat{v}(y_i)$ by the simulation method the procedure is to execute $\text{TRIP}(y_i)$ repeatedly until $\hat{v}(y_i)$ is changed to an acceptably small increment each time.

6.3. Estimator distributions

Up to this stage the estimators \hat{z} and \hat{p} in the rote learning table have been simply mean estimators. Clearly the information that has been collected for particular estimators may be vastly different, some estimators may have been well established giving reliable mean estimates, others may have had only one or two occurrences to base their estimates on. To embody this information as to the reliability of the mean estimator into the rote learning table we will talk of an estimated distribution being maintained; rather than the storage of the number of samples (observations) their mean, their variance and so on. It may well be that the latter method is used in practice but it will still be assumed that an estimator distribution is available.

Let z_1, z_2, \dots, z_n be n observations of the z that has occurred after some particular state y_i has been observed and operator u_k used. The probability density distribution that these observations can be thought of as being drawn from, will be called the parent distribution and denoted $g(z|y_i, u_k)$. From this set of observations we can obtain a mean estimator \hat{z} which estimates $E(z|y_i, u_k)$,

the mean of $g(z|y_i, u_k)$. The mean estimator $\hat{z}_{\langle y_i, u_k \rangle}$ will itself a probability density distribution, which we will denote $f(\hat{z}|y_i, u_k)$. Similarly the parent distribution for the transition probability y_i to y_j given operator u_k will be $g(p|y_i, u_k, y_j)$. (This will be a binomial distribution.) The probability density distribution of the mean estimator, $\hat{p}_{\langle y_i, u_k, y_j \rangle}$, of this distribution will be $f(\hat{p}|u_i, u_k, y_j)$. These mean estimator distributions contain the best mean estimate (their mean) together with the probability of the mean of the parent distribution being other values.

The rote learning table now becomes

$$(y, (u, f(\hat{z}), (y, f(\hat{p})))) \quad (29)$$

and $f(\hat{z})_{\langle y_i, u_k \rangle}$ = the probability density distribution
for the mean estimator of $g(z|y_i, u_k)$.
 $f(\hat{p})_{\langle y_i, u_k, y_j \rangle}$ = the probability density distribution
for the mean estimator of
 $g(p|y_i, u_k, y_j)$.

The simulation method is ideal for accommodating these distributions. The only alterations to the TRIP algorithm being to replace

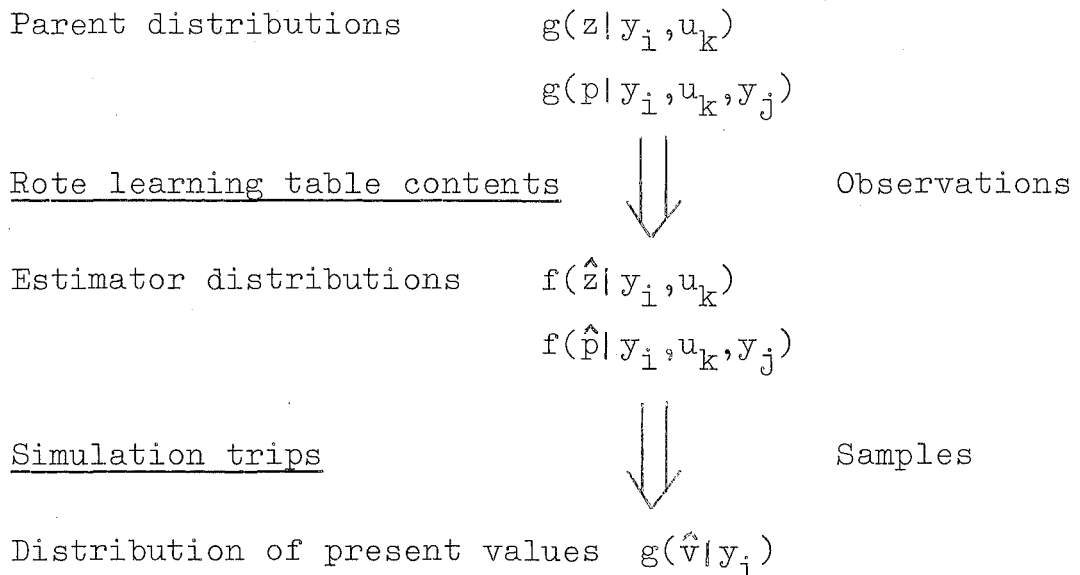
$$z \leftarrow \hat{z}_{\langle y_i, u_k \rangle} \text{ by } z \leftarrow \text{a sample from } f(\hat{z})_{\langle y_i, u_k \rangle} \quad (30)$$

and to extend SIMULATE(y, u) to produce a state y' based on the probability density distribution $f(\hat{p})_{\langle y, u, y \rangle}$.

6.4. Present Value Distribution

The uncertainty in the mean estimators for $E(z|y,u)$ and $\Pr(y_j|y_i, u_k)$ will produce uncertainty in the present value estimates $\hat{v}(y)$. This uncertainty in the value of $\hat{v}(y)$ can be represented by a probability density distribution for $\hat{v}(y)$, written $g(\hat{v}|y)$. Notice that the simulation method uses a series of simulations to supply samples to update the mean estimate of $v(y)$. It is not a big step to extend this into an estimation $g(\hat{v}|y)$, of the distribution of $\hat{v}(y)$.

Unlike the estimates of z and p we will not be concerned with the distribution of the mean estimator of $v(y)$ but with the distribution of the variable itself (the parent distribution). This is because the values or samples for $\hat{v}(y)$ are derived from a simulation that uses the distribution of mean estimators as parent distributions for generating samples. The sequence is shown below.

Machine-environment interaction4 - 7 OPERATOR DECISION PROCEDURE

We are now in the position that a distribution for the present value can be calculated using the distributions of the mean estimators that are currently available in the rote learning table. With this distribution the operator decision strategy is not restricted to choice of an operator leading to the best present value, but account can be taken of both the value of the present value estimate and its uncertainty - as given by the distribution $g(\hat{v}|y)$.

Since we are going to use the simulation method only the $D(y_i)$ that occur in TRIP need be decided - a complete policy as considered for the policy improvement method is

not necessarily needed. The decision process can be summed up as:

Problem: Given state y_i , choose $D(y_i) = u_k$ such that operator u_k will maximize the estimated present value $\hat{v}(y_i|u_k)$ but also minimize the chance that the true present value $v(y_i|u_l)$ for some other operator u_l may be larger than $\hat{v}(y_i|u_k)$. The estimate $\hat{v}(y_i|u_k)$ is based on the current contents of the rote learning table.

This is an n-armed bandit problem of the same form that has been discussed in Chapters 2 and 3. Applying the BANDIT algorithm to this situation,

BANDIT Algorithm:

$$\text{Set } \Pr[D(y_i)=u_k] = \Pr[\hat{v}(y_i|u_k) \geq \hat{v}(y_i|u_l), \forall u_l \neq u_k] \quad (31)$$

The probability of one estimated present value being greater than another

$$\Pr[\hat{v}(y_i|u_k) \geq \hat{v}(y_i|u_l), u_l \neq u_k] \quad (32)$$

can be estimated by sampling as

$$\Pr[S(g(\hat{v}|y_i, u_k)) \geq S(g(\hat{v}|y_i, u_l)), u_l \neq u_k] \quad (33)$$

where $S(y(x)) =$ a sample of x given probability density function y of x .

Since the simulation method has been described as producing a probability density distribution for $\hat{v}(y)$ rather than $\hat{v}(y|u)$, samples $S(g(\hat{v}|u, y))$ are not directly available, but they can be created as

$$S(g(\hat{v}|y_i, u_k)) = S(f(\hat{z}|y_i, u_k)) + \gamma \cdot S(f(\hat{v}|y_j)) \quad (34)$$

where the probability of the use of a particular y_j in this equation for a particular sample is

$$\Pr(y_j) = \Pr \left[S(f(\hat{p}|y_i, u_k, y_j)) \geq S(f(\hat{p}|y_i, u_k, y_1)), \right. \\ \left. y_1 \neq y_j \right] \quad (35)$$

We are now in a position to calculate the $\Pr[D(y_i) = u_k]$ by use of

$f(\hat{p}|y_i, u_k, y_j)$ = $f(\hat{p})_{\langle y_i, u_k, y_j \rangle}$ from the rote learning table, used in equation (35), and

$f(\hat{v}|y_j)$ = value computed by the simulation method as explained in the previous sections, and

$f(\hat{z}|y_i, u_k)$ = $f(\hat{z})_{\langle y_i, u_k \rangle}$ from the rote learning table, used in equation (34).

4 - 8 EXPECTANCE FUNCTION

As seen in the last section the BANDIT algorithm is not directly estimated from $g(\hat{v}|y_i)$ but rather from $g(\hat{v}|y_i, u_k)$. This leads us to define a new function $h(y_i, u_k)$ called the expectance of the state-action pair y_i, u_k . The expectance can be used directly by the BANDIT algorithm, and it has the advantage of being readily computed on-line, as will be discussed later. The definition of the expectance $h(y_i, u_k)$ is given by the (recursive) equation:

$$h(y_i, u_k) = E(z|y_i, u_k) + \gamma \sum_j \sum_m \Pr(y_j|y_i) \cdot \Pr(u_m|y_j) \cdot h(y_j, u_m) \quad (36)$$

This can be seen to be similar to the definition of present value v , except that the expectance h is defined on state-action pairs rather than on states. From equation 36, it may be thought that the expectance would be more difficult to calculate than the present value v , however, this is not the case despite the greater complexity of the defining equation.

The current estimate of $h(y_i, u_k)$ denoted $\hat{h}(y_i, u_k)$ can be obtained from the rote learning table $(y, (u, \hat{z}, (y, \hat{p})))$ by the (recursive) equation:

$$\hat{h}(y_i, u_k) = \hat{z}(y_i, u_k) + \gamma \sum_{j} \sum_{m} \hat{p}(y_i, u_k, y_j) \cdot \Pr(u_m | y_j) \cdot \hat{h}(y_j, u_1) \quad (37)$$

Just as for the present value $\hat{v}(y_i)$ we will extend from $\hat{h}(y_i, u_k)$ to the probability density distribution of $\hat{h}(y_i, u_k)$, $g(\hat{h} | y_i, u_k)$.

To obtain the probability density $g(\hat{h} | y_i, u_k)$, an extension of the simulation method described in section 4 - 6(2) can be used. A rote learning table of the form - $(y, (y, f(\hat{z}), (y, f(\hat{p}))))$ will be assumed, where $f(\hat{z})(y_i, u_k)$ is the probability density for the mean estimator of z resulting from state y_i with operator u_k , and $f(\hat{p})(y_i, u_k, y_j)$ is the probability density for the estimator of the probability of state y_j following state y_i with operator u_k used.

The procedure to simulate a step can be redefined as -

SIMULATE(y_i, u_k):

Let $x_q = S(f(\hat{p})\langle y_i, u_k, y_q \rangle), \forall y_q$ (see footnote*)

label $\{x_q, \forall q\}$ as $\{x(1), x(2), \dots\}$

$R \leftarrow$ a random number in the range 0 to 1.

$xsum \leftarrow \sum_{\forall q} x_q$

$R \leftarrow R \cdot xsum$ (Scale the random number)

$x \leftarrow 0, \text{index} \leftarrow 1.$

Loop: $x \leftarrow x + x(\text{index})$

If $x \geq R$ go to set.

else $\text{index} \leftarrow \text{index} + 1$, go to loop.

Set: If $x(\text{index}) = S(f(\hat{p})\langle y_i, u_k, y_j \rangle)$

then $\text{SIMULATE}(y_i, u_k) \leftarrow y_j$

...

The extended UPDATE procedure to handle a probability density rather than a simple mean estimator will not be detailed here since a variety of algorithms may be used, dependent mainly on the assumptions that are made about the form of the density function. In general;

$g(\hat{h}|y, u) \leftarrow \text{UPDATE}(g(\hat{h}|y, u), \text{sample})$

will be taken to mean the value of 'sample' is to be

*Where $S(y(x)) =$ a sample of x from the probability density y of x .

incorporated into the probability density $g(\hat{h}|y,u)$. For example if $g(\hat{h}|y,u)$ is a histogram then the probability of \hat{h} falling in the range containing the value of 'sample' will be incremented and the other ranges of the histogram will be decremented.

The basic algorithm for the simulation method TRIP(y) will now be given in outline; for simplicity a fixed number of steps, 'limit', will be used rather than a probability limit of TRIP(y) as described in section 4-6(2).

TRIP(y_i):

1. [Initialize.] $y \leftarrow y_i, u \leftarrow D(y), \text{step} \leftarrow 0$
 (u will be u_k with probability $\text{Pr}(u_k|y_i)$.)
2. [Simulate a step.] $y' \leftarrow \text{SIMULATE}(y,u)$
 $u' \leftarrow D(y')$
3. [Simulate earnings.]
 $\text{sample} \leftarrow S(f(\hat{z}) < y, u) + \gamma \cdot S(g(\hat{h}|y', u'))$
 ($S(y(x))$ = sample of x, prob. density y of x.)
4. [Update density.]
 $g(\hat{h}|y,u) \leftarrow \text{UPDATE}(g(\hat{h}|y,u), \text{sample})$
5. [Next step if necessary.]
 If $\text{step} = \text{limit}$, exit from TRIP(y_i).
 else $\text{step} \leftarrow \text{step} + 1$,
 $y \leftarrow y', u \leftarrow u'$, go to 2.

To summarize; we have defined an expectance function which is very similar to the present value discussed previous to this section, except that the expectance is defined over a state-action pair rather than on a state alone, as was the case with present values. The calculation of expectance by the simulation method has been considered and the main procedures $\text{TRIP}(y)$ and $\text{SIMULATE}(y,u)$ have been outlined. These result in values for the probability density functions $g(\hat{h}|y,u)$ for each state-action pair given a rote learning table of the form $(y,(u,f(\hat{z}),(y,f(\hat{p}))))$. It only remains now to consider the operator decisions $D(y)$ in terms of expectances rather than present values.

4 - 9 OPERATOR DECISION BASED ON EXPECTANCE

In section 4-7 we considered the application of the BANDIT algorithm to the present values $\hat{v}(y)$ of states y , in order to select an operator u . Restating the object of the decision procedure:

Given state y_i , choose $D(y_i) = u_k$, such that the expectance $\hat{h}(y_i, u_k)$ is maximized, but also minimize the chance that the true expectance $h(y_i, u_1)$ for some other operator u_1 may be larger than the current estimate $\hat{h}(y_i, u_k)$.

Applying the BANDIT algorithm:

$$\text{Set } \Pr[D(y_i) = u_k] = \Pr[\hat{h}(y_i, u_k) \geq \hat{h}(y_i, u_l), \forall u_l \neq u_k]$$

Unlike the case for present values, the procedure to implement the BANDIT algorithm with expectances is given directly by:

$D(y_i)$:

Let $x_q = S(g(\hat{h}|y_i, u_q)), \forall u_q$. (samples)

if $x_k \geq x_q, \forall q$

then $D(y_i) \leftarrow y_k$.

...

It can be seen from this that the decision procedure is quite straightforward given the probability density function for the expectance of each possible action from state y_i . It is because of this direct use of the expectance by the BANDIT algorithm, that the expectance $h(y, u)$ is preferred to the present value $v(y)$, and expectance will thus be used from now on. The advantage of present values is that these are used for Markov processes and details of their properties and uses are available (Howard [5]).

The calculation of the probability density for expectance, and its use in the operator selection decision are summarized in figure 9.1.

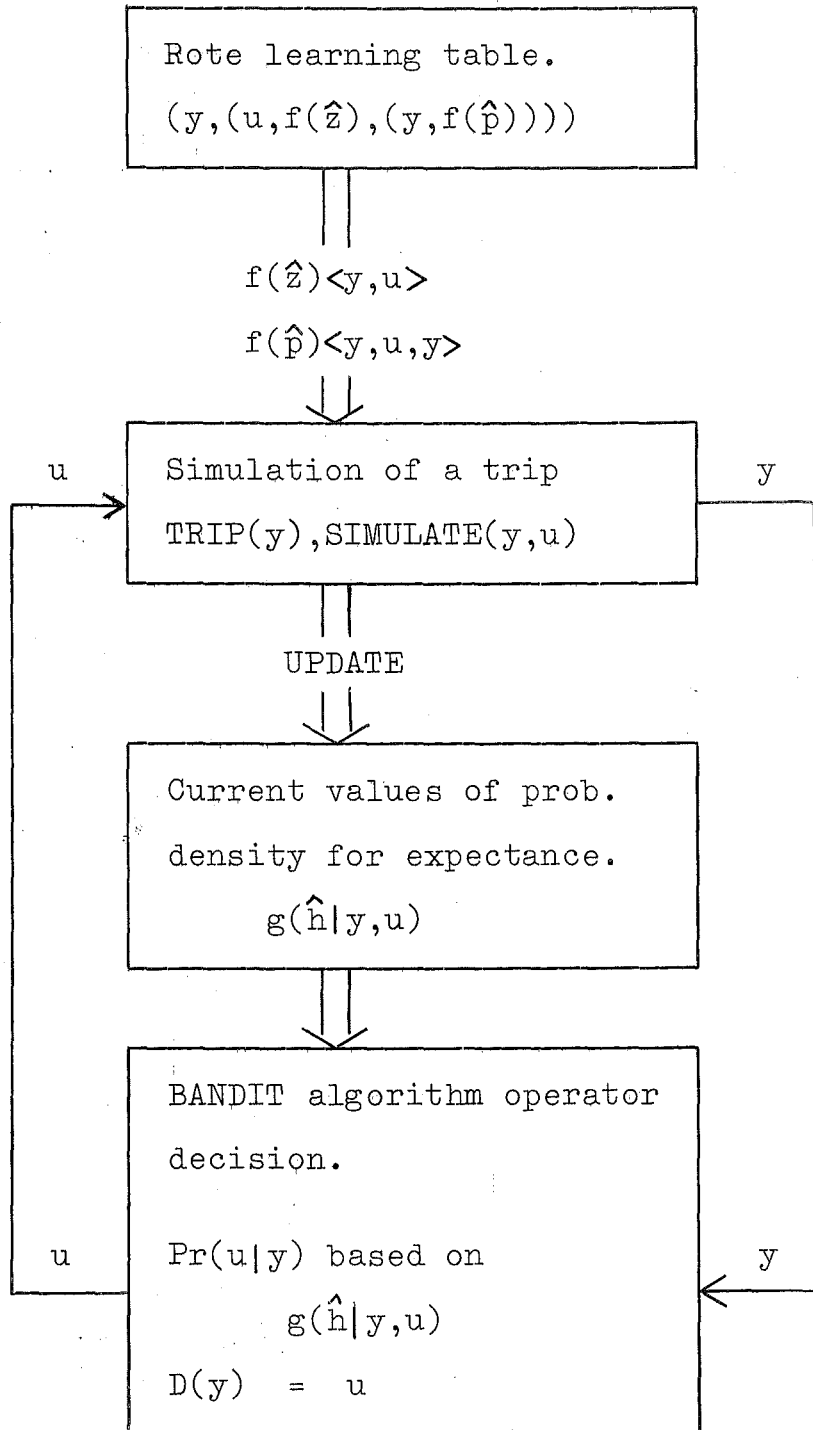
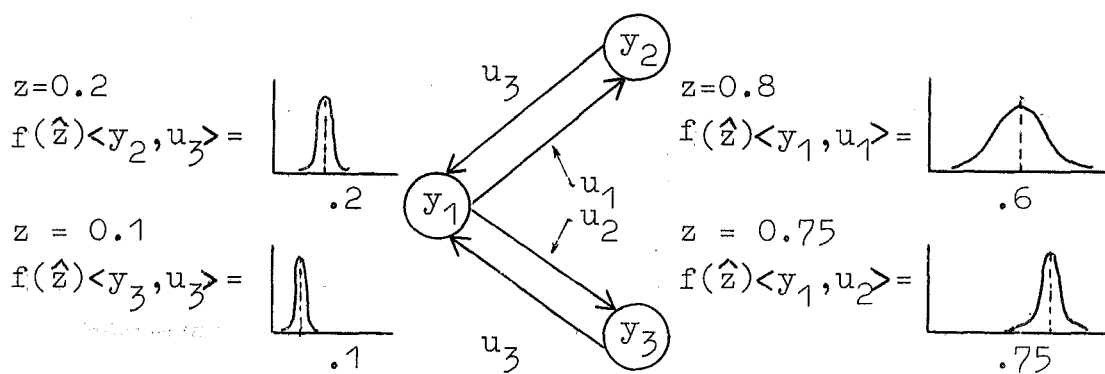


Figure 9.1

10.1. Example of Expectance Calculation

The same example used in section 4-5(2) to illustrate value-iteration and policy-iteration will now be used to illustrate expectance calculation by simulation, with the added complexity of probability density functions for the estimators.



As in section 4-5(2) the transition probabilities, given any action will be 1 or 0. For simplicity all probability density functions will be assumed to be normal distributions, so that mean and standard deviation can be used to fully describe them.

Assumed state of rote learning table:

$((y, \text{-----} (u, f(\hat{z}), \text{-----} (y, f(\hat{p}) \quad)))$

$y_1 \begin{cases} u_1, \mu=0.6, \sigma=0.3 \text{ --- } y_2, \mu=1.0, \sigma=0 \\ u_2, \mu=0.75, \sigma=0.05 \text{ --- } y_3, \mu=1.0, \sigma=0 \end{cases}$
 $y_2 \text{ --- } u_3, \mu=0.2, \sigma=0.05 \text{ --- } y_1, \mu=1.0, \sigma=0$
 $y_3 \text{ --- } u_3, \mu=0.1, \sigma=0.05 \text{ --- } y_1, \mu=1.0, \sigma=0$

As an example consider the simulated steps (as they may occur in the TRIP algorithm of section 4-8) -

$$\begin{aligned} y &= y_1, & u &= u_1 \\ y' &= y_2, & u' &= u_3 \end{aligned}$$

Assume that the expectance probability densities are initialized to the normal distributions $N(1.0, 0.5)$. This corresponds to an apriori assumption and the standard deviation must not be made smaller than the information available allows. The values used here assume a little knowledge of the situation. One way of gaining this knowledge is to solve the problem by the policy-iteration method (without probability densities) and use the present values to estimate the expectance in a manner similar to that used in section 4-7 to obtain the BANDIT decision from the present values. There are several alternatives along the same lines but we will show that the apriori values are not very critical and it is probably better to use very conservative apriori expectance estimates, for example $N(0, 100)$ and apply the simulation method for a larger number of steps. This problem will be made easier by normalizing the expectance into the range 0 to 1 as will be done in later work.

Notice that $u = D(y)$ becomes a random decision when all expectance probability densities are the same,

$N(1.0, 0.5)$. Using the BANDIT algorithm for $D(y)$ in section 4-6(7) -

$D(y_1)$: Samples x_q of $g(\hat{h}|y_1, u_q) = S(N(1.0, 0.5))$
 say $x_1 = 1.2$, $x_2 = 0.7$
 then $D(y_1) \leftarrow u_1$. (This time through.)

Since $g(\hat{h}|y, u) = N(\mu, \sigma)$ let
 $g(\hat{h}|y, u) \leftarrow \text{UPDATE}(g(\hat{h}|y, u), \text{sample})$ be
 $\mu \leftarrow \alpha \cdot \mu + (1-\alpha) \cdot \text{sample}$
 $\sigma \leftarrow (\alpha \cdot \sigma^2 + (1-\alpha) \cdot (\mu - \text{sample})^2)^{\frac{1}{2}}$

where α = a smoothing constant, $0 \leq \alpha \leq 1$. Let $\alpha = 0.8$, and

$$\text{sample} = S(g(\hat{z}) \langle y, u \rangle) + \lambda S(g(\hat{h}|y', u'))$$

Let $\lambda = 0.5$, then for simulated step $y_1, u_1 \dashrightarrow y_2, u_3$

$$\begin{aligned} \text{sample} &= S(N(0.6, 0.3)) + 0.5 \cdot S(N(1.0, 0.5)) \\ &= \text{say } 0.71 + 0.5 \cdot 0.86 \\ &= 1.14 \end{aligned}$$

$$\mu \leftarrow 0.8 \cdot 1.0 + 0.2 \cdot 1.14 = 1.03$$

$$\text{and } \sigma \leftarrow (0.8 \cdot 0.25 + 0.2(1.03 - 1.14)^2)^{\frac{1}{2}} = 0.41$$

hence $g(\hat{h}|y_1, u_1)$ now is $= N(1.03, 0.41)$.

Results from a simulation run of 100 steps:

step	0	50	1000
$g(\hat{h} y_1, u_1)$	$N(1.0, 0.5)$	$N(0.99, 0.36)$	$N(0.92, 0.32)$
$g(\hat{h} y_1, u_2)$	$N(1.0, 0.5)$	$N(1.14, 0.14)$	$N(1.07, 0.09)$
$g(\hat{h} y_2, u_3)$	$N(1.0, 0.5)$	$N(0.83, 0.24)$	$N(0.73, 0.14)$
$g(\hat{h} y_3, u_3)$	$N(1.0, 0.5)$	$N(0.74, 0.14)$	$N(0.57, 0.20)$
$\Pr(D(y_1)=u_1)^*$	0.5	0.35	0.33

A sketch graph of the convergence for this simulation run is given in figure 9.2.

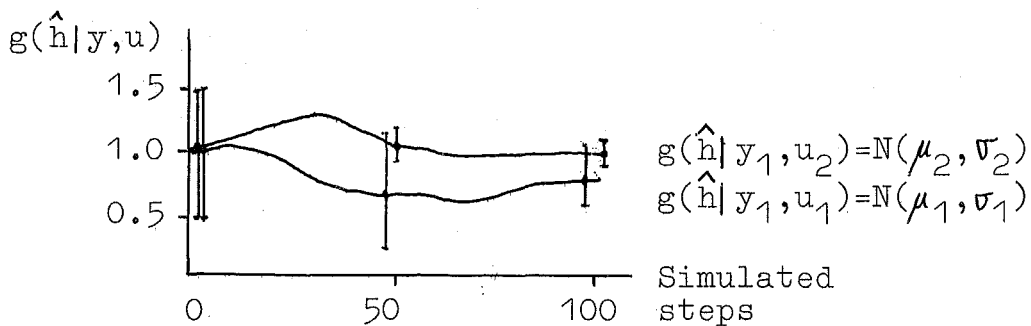


Figure 9.2

To illustrate that the apriori values are not critical a subsequent run was started with the values;

$$g(\hat{h}|y_1, u_1) = N(1.5, 0.5)$$

$$g(\hat{h}|y_1, u_2) = N(0.5, 0.5)$$

*See Appendix C.

after a simulation of 100 steps these estimates 'crossed over' to reach the values;

$$g(\hat{h}|y_1, u_1) = N(0.93, 0.28)$$

$$g(\hat{h}|y_1, u_2) = N(1.05, 0.10),$$

a sketch of this run is shown in figure 9.3

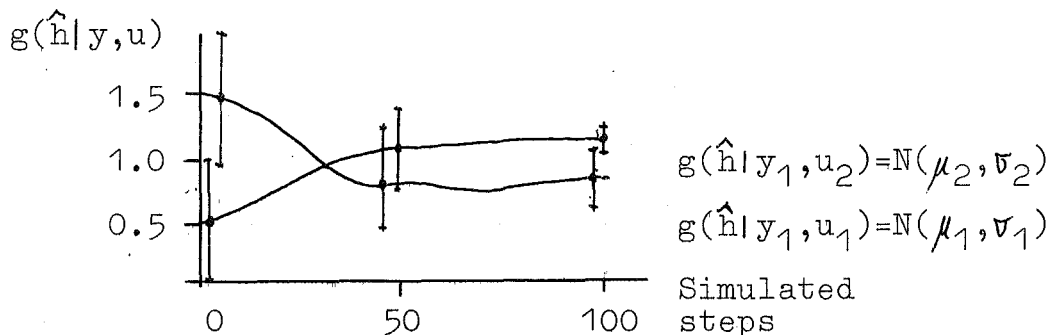


Figure 9.3

With this simple example it is a straight-forward matter to compare the present values produced by the policy-iteration method, with the expectance values as produced by the simulation method. Remember that the present value $\hat{v}(y_i)$ is

$$\hat{v}(y_i) = \hat{z}\langle y_i, d(y_i) \rangle + \gamma \sum_j \hat{p}\langle y_i, d(y_i), y_j \rangle \cdot \hat{v}(y_j)$$

and for this example there are only two possible policies namely $d(y_1) = u_1$ or $d(y_1) = u_2$, from section 4-5(2) we saw that

$$\text{if } d(y_1) = u_1, \text{ then } \hat{v}(y_1) = 0.93;$$

$$\text{if } d(y_1) = u_2, \text{ then } \hat{v}(y_1) = 1.07.$$

For the expectance (and ignoring the fact that probability densities are to be considered),

$$\hat{h}(y_i, u_k) = \hat{z}_{y_i, u_k} + \gamma \sum_{\forall j} \hat{p}_{\langle y_i, u_k, y_j \rangle} \cdot \sum_{\forall l} \text{Pr}(u_l | y_j) \cdot \hat{h}(y_j, u_l).$$

Now for the example,

$$\hat{h}(y_1, u_1) = \hat{z}_{\langle y_1, u_1 \rangle} + \gamma \cdot \hat{h}(y_2, u_3)$$

$$\hat{h}(y_1, u_2) = \hat{z}_{\langle y_1, u_2 \rangle} + \gamma \cdot \hat{h}(y_3, u_3)$$

$$\begin{aligned} \hat{h}(y_2, u_3) &= \hat{z}_{\langle y_2, u_3 \rangle} + \gamma \cdot \text{Pr}(u_1 | y_1) \cdot \hat{h}(y_1, u_1) \\ &\quad + \gamma \cdot \text{Pr}(u_2 | y_1) \cdot \hat{h}(y_1, u_2) \end{aligned}$$

$$\begin{aligned} \hat{h}(y_3, u_3) &= \hat{z}_{\langle y_3, u_3 \rangle} + \gamma \cdot \text{Pr}(u_1 | y_1) \cdot \hat{h}(y_1, u_1) \\ &\quad + \gamma \cdot \text{Pr}(u_2 | y_1) \cdot \hat{h}(y_1, u_2) \end{aligned}$$

By inspection it is seen that, if

$\text{Pr}(u_1 | y_1) = 1$, i.e. $D(y_1) = u_1$ always, then

$\hat{h}(y_1, u_1) = \hat{v}(y_1)$ with $d(y_1) = u_1$; similarly for

$\text{Pr}(u_1 | y_1) = 0$ then $\hat{h}(y_1, u_2) = \hat{v}(y_1)$ for $d(y_1) = u_2$.

$$\text{Pr}(u_1 | y_1) = 1.0: \quad \hat{h}(y_1, u_1) = \hat{v}(y_1) = 0.93$$

$$\text{Pr}(u_2 | y_1) = 1.0: \quad \hat{h}(y_1, u_2) = \hat{v}(y_1) = 1.07$$

From the simulation run for finding the expectance values;

$$\text{Pr}(u_1 | y_1) = 0.33: \quad \hat{h}(y_1, u_1) = 0.92$$

$$\text{Pr}(u_2 | y_1) = 0.67: \quad \hat{h}(y_1, u_2) = 1.07.$$

4 - 10 EXPECTANCE ENTRY IN THE ROTE LEARNING TABLE

Up to here the rote learning table has had only two forms of information stored in it -

1. Basic recordings of machine environment interaction,
2. Statistics measured over a number of interactions and continually updated.

The estimated expectance $\hat{h}(y,u)$ or more generally the probability density function for this expectance $g(\hat{h}|y,u)$, has been calculated from the rote learning table alone. By this fact the estimated expectance is a 'summary' or 'interpretation' of the rote learning table contents. However, the expectance is used to enable each operator selection to be made by the machine, and to recalculate it each time by the stochastic simulation method previously considered would be a very time consuming procedure. With this motivation for putting the estimated expectance values into the rote learning table we will now consider the implications of doing so.

The form of the rote learning table will now be:

$$(y,(u,f(\hat{z}),g(\hat{h}),(y,f(\hat{p})))) \quad (38)$$

We now have the current value of the estimated expectance available at every step and can make the small but very important step of saying that the expectance estimates can be updated not only by simulated steps, but also by actual machine environment interaction

steps. As for a simulated step:

$\hat{h}(y,u)$ can be updated with the value $z(y,u) + \gamma \cdot \hat{h}(y',u')$, where y',u' follow y,u .

Notice that $z(y,u)$ may be the best estimate from the rote learning table $\hat{z}\langle y,u \rangle$, or it may simply be the observed valuation z resulting from the step.

From this we can write -

$$\hat{h}(y,u) \leftarrow \text{UPDATE} (\hat{h}(y,u), z + \gamma \cdot \hat{h}(y',u'))$$

which may be done as -

$$\hat{h}(y,u) \leftarrow \alpha \hat{h}(y,u) + (1-\alpha)(z + \gamma \hat{h}(y',u')) \quad 0 \leq \alpha \leq 1,$$

where y',u' follows y,u and z is observed at y' . With $g(\hat{h}|y,u)$ in the rote learning table the above equations can be extended to -

$$\hat{h}_m \langle y,u \rangle \leftarrow \alpha \hat{h}_m \langle y,u \rangle + (1-\alpha)(z + \gamma \hat{h}_m \langle y',u' \rangle)$$

$$\hat{h}_v \langle y,u \rangle \leftarrow \alpha (\hat{h}_v \langle y,u \rangle^2 + (1-\alpha)(\hat{h}_m \langle y,u \rangle - (z + \gamma \hat{h}_m \langle y',u' \rangle))^2)^{\frac{1}{2}}$$

where $g(\hat{h}|y,u)$ is assumed to be normal - $N(\hat{h}_m, \hat{h}_v)$.

The important point about these equations is that:

1. The expectance estimates are being maintained 'on line' with the machine environment interaction, using some straightforward UPDATE procedure, and

2. None of the 'statistical' entries in the rote learning table are involved. That is the expected earnings $\hat{z}\langle y, u \rangle$ and the estimated conditional transition probabilities $\hat{p}\langle y, u, y \rangle$ are not required in the UPDATE procedure.

From section 4-9 we have the operator selection procedure

$D(y_i)$: Let $x_q = S(g(\hat{h}|y_i, u_q))$, $\forall u_q$. (samples)
 for $x_k \geq x_q$, $\forall q$
 $D(y_i) \leftarrow x_k$.

...

Since this procedure does not require the 'statistical' entries in the rote learning table either, we may choose to eliminate them leaving the new rote learning table -

$$(y, (u, g(\hat{h}))) \quad (39)$$

As a refinement the expectance can also be normalized into the range 0 to 1 -

$$\hat{h}(y, u) \leftarrow \alpha \hat{h}(y, u) + \beta (z + \gamma \hat{h}(y', u'))$$

where $\beta = (1 - \alpha) / (1 + \gamma)$.

4 - 11 BANDIT-EXPECTANCE MACHINE

With the expectance estimates in the rote learning table, $(y, (u, g(\hat{h})))$, as discussed in the last section, we are in a position to define a basic learning machine. This machine is based on the concepts developed in the previous sections but as will be seen it takes an extremely simple form.

BANDIT-EXPECTANCE Machine Algorithm

1. [Observation.] $y \leftarrow$ observed environment state,
 $z \leftarrow$ observed valuation.
2. [Operator selection, $D(y)$.]
 $x_q = S(g(\hat{h})\langle y, u_q \rangle), \forall u_q$ (S - sample)
 if $x_k \geq x_q, \forall q$
 then $u \leftarrow u_k$.
3. [Perform action.] Output operator $u = u_k$ to the
environment, and observe the response:
 $y' \leftarrow$ observed environment state,
 $z' \leftarrow$ observed valuation.
4. [Check if new state is 'known'.]
 if $\{g(\hat{h})\langle y', u \rangle\} \neq \text{null}$, then go to step 6.
5. [Create new entry.] Put all entries of the form:
 $(y', (u', g(\hat{h})))$,
 into the rote learning table.
 Where - u' = all possible operators.
 $g(\hat{h})$ = an apriori probability density for
 the expectance, say $N(0.5, 0.5)$.
6. [Select operator, $D(y')$.]
 $x_q = S(g(\hat{h})\langle y', u_q \rangle), \forall u_q$ (S - sample)
 if $x_1 \geq x_q, \forall q$
 then $u' \leftarrow u_1$.
7. [Update expectance probability density.]
 $g(\hat{h})\langle y, u \rangle \leftarrow \text{UPDATE}(g(\hat{h})\langle y, u \rangle, \beta(z + \beta g(\hat{h})\langle y', u' \rangle))$
 $(\hat{h}$ normalized to the range 0 to 1).
8. [Step from y to y' .] $y \leftarrow y'$
 $z \leftarrow z'$
 $u \leftarrow u'$, go to step 3.

.....

It should be noted that by starting with a rote learning table, and considering its contents at any given time as defining a Markov process, we have eventually ended up with the same algorithm that was presented (rather intuitively) in Chapter 3, as a logical extension of stochastic learning automata schemes. Although the automata scheme presented in Chapter 3 can be considered as a presentation of the results of this chapter without the background detail and foundation, this is not the full picture since although the BANDIT-EXPECTANCE algorithm is essentially the same, it has now been developed as an algorithm working on a rote learning table - a concept quite different in approach to that of the stochastic learning automata work.

While the BANDIT EXPECTANCE machine uses a minimal rote learning table, $(y, (u, g(\hat{h})))$, in the sense that no transition probabilities or z valuations are explicitly recorded, it has not attempted to 'generalize' entries to enable each to cope with a number of 'similar' states. This important area is a main part of the STELLA scheme and the concepts will be discussed in section 4-13.

Some idea of the performance of the basic BANDIT EXPECTANCE machine may be obtained from the example described in the next section.

4 - 13 FOX AND DOGS GAME

The 'French Military Game' or 'Fox and Dogs Game' [9] was used to demonstrate the BANDIT-EXPECTANCE machine. This game is played on a board as illustrated in figure 13.1. The 'F' indicates the position occupied by the Fox and the 'D's indicate the three Dogs.

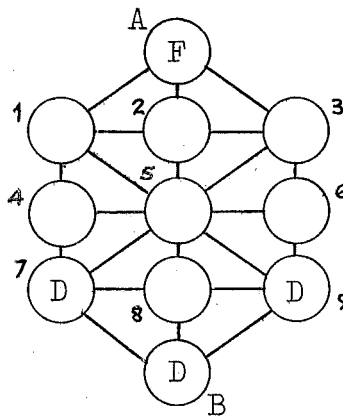


Figure 13.1

The Dogs (one at a time) and the Fox take alternate moves, a legal move being from the current position (circle) to any adjacent position joined by a line on the board. The Dogs have the additional constraint of only being allowed to move 'up' or 'across' relative to figure 13.1.

The aim for the Dogs is to surround the Fox so that he has no legal move, and thus win the game. For example, the position with the Fox at A and the Dogs at 1, 2 and 3 is a win to the Dogs if the Fox is to move, but a win to the Fox if the Dogs are to move. This

position will be written as A-123; other winning positions for the Dogs (with the Fox to move) are 4-157 and 6-359. If the Fox can 'evade' the Dogs, the Dogs will eventually have to concede defeat since they are unable to move 'down' the board after the Fox. An example is the position 5-468, which is an effective win for the Fox regardless of who is to move, although the Dogs may not concede defeat until several moves later in the game (assuming the Fox has not made a silly move in the meantime - which before the machine has learned very much is quite probable).

This game was programmed (using the LINKNET technique described in Appendix A) to be played interactively with an operator, using a CRT display (Appendix B) to show the current board positions. The BANDIT-EXPECTANCE machine can play either the Fox or the Dogs or both, but we will assume that the machine is to play the Fox for simplicity. It was left entirely up to the operator to concede defeat giving a win to the Fox, while a win by the Dogs was detected by the Fox finding no legal moves available, or at any other time by an input from the operator.

For this game the machine environment interaction (all within the computer program) is via:

Observed state $Y = \{F-D1D2D3 : F, D1, D2, D3 \text{ are mutually exclusive members of } \{A, 1, 2, \dots, 8, 9, B\}\}$

Valuation $Z = \{-1, 0, +1\}$ $-1 = \text{a loss}$
 $0 = \text{no indication}$
 $+1 = \text{a win.}$

Operators $U = \{\text{Up to 6 possible moves computed by a legal move generator for a given state } y \in Y.\}$

Internal to the BANDIT-EXPECTANCE machine program the rote learning table took the form:

$$(y, (u, g(\hat{h})))$$

with $y = \text{the observed states,}$

$u = \text{the possible actions for each state,}$

$g(\hat{h}) = \text{the expectance probability density, taken as a normal distribution } N(\mu, \sigma), \text{ with mean normalized into the range } -1(\text{loss}) \text{ to } +1(\text{win}).$

Apriori density used was $N(0, 0.5)$ corresponding to a "don't know" condition.

Points of interest about this formulation:

- . There are 165 possible states since symmetry is not recognized.
- . A loss may occur after several state-move combinations.
- . A win is not clearly defined (being at the operator's

discretion), and may occur after a large number of different state-move combinations.

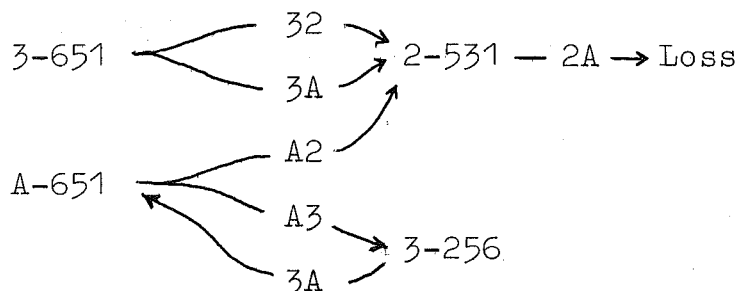
- . The number of operators available varies from state to state (between 1 and 6). Although the operators are given in the rote learning table, their effect on the board position is definitely not known to the machine. (They must be known in a game playing program based on a look ahead tree search program like most chess and checkers machines.)
- . The game can always be won by the Dogs if they play an optimal game [9]. This means that tree search programs would have trouble with this game since although they could possibly plan a complete game they could not then choose the best move without learning the operator's most likely errors (non optimal moves).

An example of some of the contents of the rote learning table for the BANDIT-EXPECTANCE machine is given below (after about 40 games experience):

y	u	$g(\hat{h})=N(\mu, \sigma)$	Comments
2-531	2A	-.99, .00	Almost certain loss
3-652	3A	+.30, .02	Cycles with A-651
3-651	3A	-.61, .04	Probable loss with either move
	32	-.63, .05	
A-651	A2	.00, .50	Not updated yet
	A3	+.48, .03	Good, cycles with 3-652
3-621	3A	-.20, .13	Poor move, small evidence
		+.74, .04	Good move and better evidence

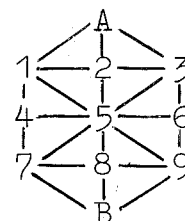
Remembering that each expectance estimate is up-dated from the next expectance to be 'seen', the build up of the rote learning table illustrated may be better understood by the following diagram of probable board positions assuming a reasonable level of play by the Dogs -

Positions Moves Positions



(Eventual win)

Board Layout



The quantitative assessment of the learning ability of the machine is extremely difficult. The machine can

not be played against an operator for such assessment since it is virtually impossible for him to maintain a consistent level of performance - he also is learning! A player with consistent performance can be built from a BANDIT-EXPECTANCE machine by having it learn to play to some level of performance and then fixing its rote learning table so that no further improvement (or degradation) can occur. By playing this 'fixed memory' BANDIT-EXPECTANCE machine against another 'free' one, a learning curve can be compiled. This was tried but, although interesting, the results were of little significance since among other things there is no way to indicate the performance level of the 'fixed-memory' machine.

As an indication of the performance of the BANDIT-EXPECTANCE machine, it was found that (after less than 100 games experience) a novice player would often query, after a few games, whether it was possible for him to win at all. An experienced player (but not knowing the optimal strategy) could nearly always beat the machine, but often after quite a long game. The teaching operator usually admitted defeat as soon as the Fox had clearly evaded the Dogs. Because of this the machine, after evading the Dogs, had little aversion to going back up the board amongst the Dogs again! (This 'stupidity' in the early stages of learning would be

exploited by opponents to recover from their blunders, and thus was self correcting.) It is interesting to note that the machine assumed a 'personality' (HIM) to the operator, in particular during the early stages of learning when changes in performance could be observed. "Oh good he's learnt not to make that silly move", or "H'm, he's getting tricky now!"

Important game strategy such as - move to position 5 if at all possible - require generalization over states, which is not possible with the simple BANDIT-EXPECTANCE scheme. After learning all the possibilities, the result may be the same as knowing the rule - but the method is far less attractive. However, it is not possible to generate these 'higher level' heuristics without the evidence from performance of a sound 'low level' machine. It is hoped that the BANDIT-EXPECTANCE machine can assume such a role. The next section ends this chapter by considering further extension to the rote learning table concept to reduce the memory required and to cope with 'generalization' over states.

4 - 14 EXTENSIONS TO THE ROTE LEARNING TABLE

The rote learning table has been developed from a simple record of past events; through a record of estimators over past events; to a control strategy for the learning machine. Expectance has been shown as looking forward from a given state-operator combination, to assess the long term earnings of this particular operator, by taking into account future earnings, transition probabilities, and operator selection probabilities. We are now concerned with looking 'across' the state-operator combinations to see if any of them are essentially 'saying the same thing'. By finding groups of state-operator pairs that can be condensed into a single composite entry in the rote learning table we may achieve the two benefits:

1. A rote learning table memory space reduction,
2. An ability to select an operator for a previously unseen state on the basis that this state probably belongs to a known composite. This avoids starting it as a new entry with apriori $g(\hat{h})$ and hence random operator selection.

The first point can be achieved to some extent by simply eliminating 'poor' entries. This was tried (and worked) for the BANDIT-EXPECTANCE machine playing the NIM game. The procedure was to overlay a new entry (required by a state not currently in the rote learning

table) on top of the current entry with the smallest maximum (over operators) expectance. This is a crude method of reducing memory and it does not have the benefit of point 2 - it throws away hard earned information rather than generalizing over it.

Point 2 is 'generalization' over states, and has been established as a workable scheme by STeLLA [1,6] and in a slightly different way by Doran's robot [4]. The method for generalization over states is very wide, it includes a large class of pattern recognition schemes.

The change in the rote learning table is that several entries, $(y, (u, g(\hat{h})))$, become a single entry $(y\text{-rule}, (u, g(\hat{h})))$. Where $y\text{-rule}$ is a pattern template or procedure for determining if some observed state y is 'similar' or 'belongs' to this rote learning table entry. At this stage the term 'control-policy' as used by Gaines and Andreae [6] becomes a more meaningful term than 'rote learning table entry'.

It can be noted that if there is a maximum of N possible operators for any possible state y , then given a powerful enough pattern recognition system, only N control policies are required by a fully 'mature' learning machine. This is not to say that it is possible to learn these N policy elements without utilizing a much larger number.

The problems of state generalization have not been covered in this thesis work on the grounds that this is a higher level function that can be reasonably grafted on to the rote learning table without any drastic alteration to the underlying algorithms that build, update and utilize the contents of this table. The concept is of a pattern recognition procedure observing the rote learning table all the time trying to condense several entries $(y, (u, g(\hat{h})))$ into a composite entry $(y\text{-rule}, (u, g(\hat{h})))$, or even condense several composites into a single more powerful composite. This idea is illustrated in figure 14.1.

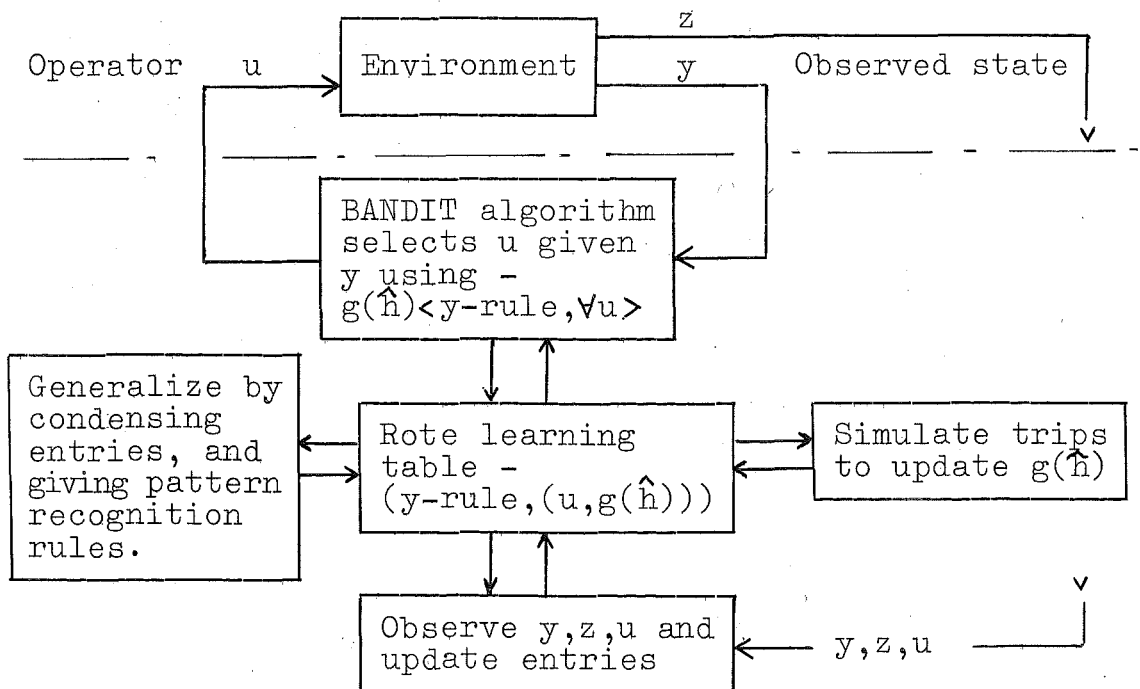
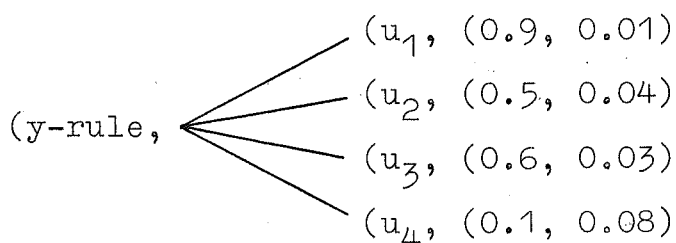


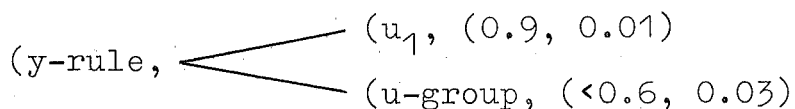
Figure 14.1

Another idea is to have a predictor that can make use of the rote learning table, predict ahead from the current state y , and hence select an operator y , even though the current state y is not itself in the rote learning table. This method is in some ways similar to the idea of a composite entry (y -rule) since the information that allows prediction is the same information available in forming the y -rule composite entries.

A second mechanism for reducing the memory requirement of the rote learning table is to condense operators for a given state or state composite entry. For example, the composite -



could well be condensed into -



without any significant change in performance since the probability that the BANDIT algorithm will choose any operator other than u_1 is very small. To illustrate, the probability of a sample from a normal distribution being over 3 standard deviations away from its mean is less than 0.0001, and u_1 is 100 standard deviations

above u_2 , u_3 , or u_4 ! (This is an extreme example.) If the u -group does get selected than u_2 , u_3 , u_4 could be selected among by uniform random selection.

All the preceding comments are intended merely to indicate the fields that are open to extend the basic rote learning table, and BANDIT-EXPECTANCE machine. In fact STeLLA has extended well into these areas on a heuristic basis. The advantages are not that the BANDIT-EXPECTANCE machine has progressed further, but it has, consolidated the basic system by -

- . Linking the basic heuristics to stochastic learning machine theory, and the theory of Markov processes,
- . Extending the linear-weighted - random-choice to the more powerful BANDIT selection algorithm,
- . Extending 'expectation' into the more general form of the expectation function (accepting any range of z) which is linked to the 'present value' of Markov process theory.
- . Developing an on-line method for keeping expectance updated, which allows (if desired) an elimination of transition probabilities and valuation expectation estimators.

REFERENCES

- 1 Andreae, J.H. Learning Machines: A Unified View.
 Encyclopaedia of Linguistics Information and
 Control, Pergamon Press, 1969
- 2 Bellman, R. A Problem in the Sequential Design of
 Experiments. SANKHYA Vol.116, pt 3 and 4 1965,
 pp 221-229.
- 3 Bellman, R. and Kalaba, R. Dynamic Programming and
 Modern Control Theory. Academic Press 1965.
- 4 Doran, J.E. Planning and Generalization in an
 Automaton/Environment System. Machine Intelligence
 4, Ed. Meltzer, B. and Michie, D. Edinburgh
 University Press 1969. pp 433-454.
- 5 Howard, R.A. Dynamic Programming and Markov
 Processes. MIT Press 1960.
- 6 Gaines, B.R. and Andreae, J.H. A Learning Machine in
 the Context of the General Control Problem, 3rd
 IFAC Congress, London, June 1966. (Inst. Mech. Eng.)
- 7 Kemeny, J.G. and Snell, J.L. Finite Markov Chains.
 D. Van Nostrand Co. 1960.
8. Scientific American, Mathematical Games, by Martin
 Gardiner. Vol.209, No.4, October 1963 and Vol.209,
 No.5, November 1963.

CHAPTER FIVE

CONCLUSION

CHAPTER 55 - 1 REINFORCEMENT LEARNING

Throughout this thesis we have essentially been considering 'learning by reinforcement'. The 'values' that have been adapted or reinforced have been based on arc-costs (Chapter 2), reward probability (Chapter 3), and expectance (Chapters 3 and 4).

Now we shall consider these processes from a higher view point, and talk of 'value' reinforcement to cover all these cases. By the phrase "the 'value' of an alternative" will be meant the measure of this alternative relative to other alternatives, the measure being based on reinforced 'values'.

The major decision process that has been used to select alternatives given their 'values' has been the BANDIT algorithm. However, this algorithm is a decision maker - not a reinforcement learner. The decision has been separated out from the adaptive 'value' assignment process.

Reinforcement learning can also be referred to as 'incremental' or 'statistical' learning. The following section presents something of the viewpoint opposed to reinforcement learning as a central part of an intelligent machine.

5 - 2 AGAINST REINFORCEMENT LEARNING

Arguments against the use of reinforcement learning have been positively given by M. Minsky (M.I.T.). These views are clearly expressed, and carry the authority of his prominence and experience in the field of artificial intelligence. The view point will thus be presented by direct quotation:

M. Minsky 1963 [6] in "Computers and Thought".
"... I am not convinced that such 'incremental' or 'statistical' learning schemes should play a central role in our models. They will certainly appear as components of our programs but, I think, mainly by default. The more intelligent one is, the more often he should be able to learn from an experience something rather definite; e.g. to reject a hypothesis, or to change a goal. (The obvious exception is that of a truly statistical environment in which averaging is inescapable. But the heart of the problem is always, we think, the combinatorial part that gives rise to searches, and we should usually be able to regard the complexities caused by 'noise' as mere annoyances, however irritating they may be.)"

More recently in the introduction to 'Semantic Information Processing, 1969 [7] ...

"... Consider the qualitative effect, upon the subsequent performance of Bobrow's STUDENT [7], of telling

it that 'distance equals speed times time'! That one experience alone enables it to handle a large new portion of high-school algebra: the physical position-velocity-time problems. It is important not to fall into the habit, suggested by so much modern work in psychology, of concentrating only on the kinds of 'learning' that appear as slow-improvement-attendant-upon sickeningly-often-repeated experience!"

"Bobrow's program does not have any cautious statistical devices that have to be told something over and over again, so its 'learning' is too brilliant to be called so. ... It seems that as we incorporate more and more sophisticated heuristic methods, the need for senseless sources of variation in behaviour become less and less necessary.

"Of course I do not suggest that there is no use in having cautious evidence-assessing mechanisms. I only want to present a sufficiently positive view to set the negative view in perspective..."

5 - 3 LEARNING BY BEING TOLD

The idea of 'telling' a machine some facts which will help it, is an important idea that appears to clash with the idea of reinforcement learning. However, it is the 'value' assignment that is influenced by 'telling', not the decision making. It seems quite

possible that a BANDIT decision algorithm could be used in a scheme involving both reinforcement and telling.

Consider the use of the variance of a 'value' estimator. The variance is a measure of the confidence in the 'value'. In this light there is no reason to calculate variance from the sum of the deviations squared if there is a richer source of information available that can give a direct assessment of the confidence that can be placed in a 'value' estimator.

Following this up we see that 'telling' can hold more information than simply an observation; it says not only that the 'value' is such-and-such but it may also specify a confidence in this. We may well be able to 'tell' the machine that the value of an alternative is 1.0, wishing to inform it not that the mean value of samples will be 1.0 but simply that the 'value' always will be 1.0 - a perfect action for the particular situation.

Telling from this view point is rather like assigning an apriori 'value' - it forces the 'value' to some condition, rather than simply contributing an observation or sample. The methods for fully using subjective assessment as apriori data are not yet completely formulated, although there is considerable interest in such techniques for decision analysis [4,5].

Once 'values' have been assigned, regardless of whether they were arrived at by analysis of a large number of observations or by a 'telling' or 'forcing' process, the BANDIT decision algorithm is still appropriate. If the 'values' are absolutely known with zero variance (maximum confidence) the BANDIT algorithm reduces to simply selecting the maximum (or minimum).

As a point of interest, it may well be that even a poor confidence in some 'values' may be sufficient to almost completely separate them out as far as the decision process (BANDIT) is concerned. For example, let us 'tell' the machine that alternative A has a 'value' of 5.0 with a confidence assessed as a variance of 0.5, while alternative B is known to be 5.75 exactly. With this information the probability of the BANDIT algorithm choosing alternative A (when trying to choose a maximum value) is less than 0.00001 - alternative B is for all practical purposes always chosen. We are still assuming normal distributions since that is what a large number of observations would lead to. However, it is reasonable to alter this to cater for some desired performance or for computational convenience; the criteria being that the variance specifies the confidence in the 'value'.

The complexity needed to be able to 'tell' the machine more 'meaningful' (and useful) facts in a straightforward manner is of course a major problem. It is not the intention to side-step this problem (which is behind Minsky's comments) but simply to point out that a road into at least some form of compatibility between 'telling' and reinforcement learning ('observing') may be available.

Although the approach of semantic modelling [7] has shown impressive progress it is not the only way to develop language for man-machine communication. An entirely different approach is to try and design a machine that is capable of generating its own internal language to describe its experience. A feature of such a language is that the semantics are not predetermined and imposed onto the machine, but are developed as relevant to the machine's experience. Tentative steps in this direction are presented in a paper describing 'monologue' for STeLLA, Andreae and Cashin (1969) [2].

5 - 4 WHY THE GAP?

It is the author's opinion that the gap between reinforcement learning and semantic modelling is an area deserving much greater research effort than is currently evident. It is not an area that is likely to produce spectacular progress. but the eventual results seem to be invaluable to both camps. It does not seem reasonable to follow one line or the other simply because the common ground between the two is almost non-existent at present. The important question is, surely: Why is there no common framework?

To better the current question-answering systems or robot manipulators there is certainly evidence to suggest that starting with a reinforcement learning scheme is not likely to lead to success. On the other hand there is a definite but hard to define attraction in the general idea of learning from experience, and of simulating the human brain, that are absent from the more successful achievements in matching the human's 'external' intelligent abilities. Also there is little evidence to show that designing a machine that can match a human at some task (thought of as requiring intelligence) has necessarily achieved more in advancing the mechanization of intelligence than designing a low level machine that can learn to develop its own limited model of its environment - although there is no doubt which could be more useful in action.

An attempt was made at one stage during this thesis work, to embed a STeLLA type scheme [1] into the framework of the Stanford Research Institute's Robot project [8]. This attempt showed clearly the gap between such projects, and how difficult it is to establish common ground to fill the gap. We know these schemes are in many ways incompatible, but we do not clearly know the basic reasons why this should be so.

5 - 5 GRAFTING LEARNING ABILITY ONTO A PROGRAM

The idea that an intelligent program need not learn to be intelligent, and that learning potential can be realized through 'telling', has certainly led to a number of successful programs (e.g. SIR, STUDENT, ANALOGY [7]). However the pitfalls of updating the information data-base for such programs have not been clearly stated. Put another way, the programs seem to be designed to do the best they can on the basis of their current data, but do nothing to try and attain the best data.

Heuristic search programs illustrate this point well since they have been called "the central paradigm of artificial research" (Feigenbaum 1938 [3]). The search strategy aims to find a path through a problem tree, the 'values' involved being supplied from the information data-base. Now the search technique is

designed to find the best 'value' alternative (path), and most important, to do so in the most efficient manner. It may well be that finding any solution (path) is a sufficient goal. However, if several possible solutions can be found then the best (maximum or minimum 'value') is chosen. More basic than this the search itself is directed to try the most promising or best 'value' alternative at each point through the search.

It is tempting once having an efficient heuristic search program, to simply say: Use this on the best information data-base (e.g. arc costs) available, and as more information comes to light simply improve the data-base accordingly. The pitfall is that unless it can be established that the information comes to light entirely independently of the performance of the search program, then the problem is in the class of 'on-line' problems discussed in Chapter 2. As shown in Chapter 2 the 'on-line' problem needs more than an effective search strategy - it also needs a BANDIT (type) decision maker.

The idea that a program which does the 'best possible' with a given data-base can simply embody learning by having its data-base updated with new information is not restricted to heuristic search problems. Any program that acts on a data-base which is updated in a

way that depends on how it has acted in the past needs to have its decision process elaborated (BANDIT'ised!) to successfully cope with the 'learning' situation.

5 - 6 SUMMARY OF MAIN POINTS

A review of the introduction in Chapter 1 may be fruitful, in retrospect, as an 'over view' of the material presented.

The points to be emphasised in this thesis are:

- . The BANDIT algorithm, as a fundamental mechanism for any scheme involving the choice of the best alternative based on valuation using a current (incomplete) set of data: Data collected being dependent on the choice made.
- . The expectance function, particularly the way it is developed from the idea of 'present value', and the on-line implementation that results from its recursive definition.
- . The need for the BANDIT algorithm (or another with the same purpose) in a class of problems called 'on-line' path finding that occur not only in artificial intelligence work but also in the operations research area.
- . The linking together of stochastic automata theory and a class of heuristic programming work. Also the extension of the stochastic learning automata

machine-environment interaction in doing this.

- . The linking together of Markov process theory with the BANDIT algorithm, expectance function and rote learning tables.
- . The computational tools, LINKNET and the display system philosophy, given in the appendices.

More detailed lists of points have been given at the end of each chapter.

REFERENCES

- 1 Andreae, J.H. Learning Machines: A Unified View.
 Encyclopaedia of Linguistics Information and
 Control, Pergamon Press, 1969.
- 2 Andreae, J.H. and Cashin, P.M. A Learning Machine
 with Monologue. Int. J. of Man-Machine Studies,
 Vol.1, No.1, 1969, pp 1-20.
- 3 Feigenbaum, E.A. Artificial Intelligence: Themes
 in the Second Decade. Invited paper IFIP68,
 Stanford Artificial Intelligence Project Memo AI-67.
- 4 Howard, R.A. The Foundations of Decision Analysis.
 IEEE Trans on Sys. Sci. Cybernetics, Vol. SSC-4,
 No.3, September 1968, pp 211-219.
- 5 Jaynes, E.T. Prior Probabilities. IEEE Trans on
 Sys. Sci. Cybernetics, Vol. SSC-4, No.3, September
 1968, pp 227-240.
- 6 Minsky, M. Steps towards Artificial Intelligence.
 Computers and Thought, Ed. Feigenbaum, E.A. and
 Feldman, J. McGraw Hill 1963, pp 406-452.
- 7 Minsky, M. Ed. Semantic Information Processing.
 MIT Press, 1968.
- 8 Nilsson, N.J. A Mobile Automaton: An application of
 artificial intelligence techniques. Proc. Int.
 Joint Conf. on Artificial Intelligence. Ed.
 Walker, D.E. and Norton, L.M., A.C.M. 1969, pp 509-520.

APPENDIX A

LINKNET -

A Structure for Computer Representation and Solution of
Network Problems

ABSTRACT

LINKNET is an information structure for representing any network of nodes and interconnecting arcs. The structure applies linked lists and enables list-processing techniques for problem solving with networks. The LINKNET structure has provided a concise implementation of algorithms arising in a wide variety of network problem solving, such as power system analysis, game playing programs, minimum cost path finding and the determination of certain trees and meshes in a network.

The work reported in this appendix was carried out in cooperation with M.R. Mayson and R. Podmore. M.R. Mayson has been applying the LINKNET technique to power system load flow studies and particularly the problem of finding 'clumps' of 'tightly connected' nodes in a power supply network. R. Podmore has also been applying LINKNET to power system network problems; with special attention to short circuit studies and transient stability. Separate publications are being prepared on these applications of LINKNET. This appendix serves as a more general 'over view' of the technique itself.

1. INTRODUCTION

The methods of linked list-manipulation of data structures as used in this work are well known to computer scientists; unfortunately the use of these techniques in the general run of application problems is not so common. Experience with a number of application programs, both in assembler language and FORTRAN, has shown that the representation of networks and the implementation of network problem solving algorithms is a particularly fruitful area.

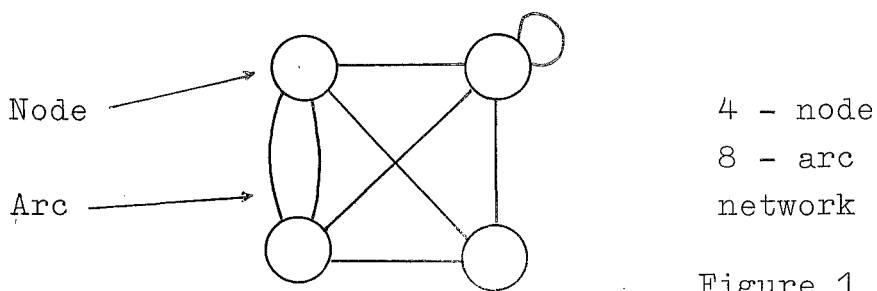
As Knuth [1] has noted: Although List-processing systems such as IPL-V, LISP, and SLIP are useful in a large number of situations, they impose constraints on the programmer than are often unnecessary; it is usually better to use the methods of List-processing (as described by Knuth [1]) in one's own programs, tailoring the data formats and processing algorithms to the particular application. LINKNET follows this philosophy even though it is concerned not with a particular application but with a whole class of problems - those involving problem solving with networks. The LINKNET scheme is not being described as a rigid protocol for network problems, but rather it is described to direct attention to the applications of List-processing to network problems by showing a framework that has proved itself very effective.

The LINKNET structure was developed for two problems, one of power system analysis and the other a game playing program. The common element between these two problems is that they have both to deal with networks, one in the form of bus bars and power lines, the other in the form of positions and legal moves. It quickly became apparent that the LINKNET structure was useful not only because it gave a way to represent and manipulate the networks, but also it facilitated the computational procedures used. Further applications confirmed that the LINKNET structure is well suited to a general class of problems involving such operations as searching, iterative scanning, modification and organization of any network and its attributes.

2. THE BASIC STRUCTURE

2.1 Graphs

A network, or graph, consists of 'nodes' which are the junction points for the 'arcs' or interconnecting lines. An example network with 4 nodes and 8 arcs is shown in figure 1. Notice that, except as may be dictated by particular applications, there are no restrictions on loop cross overs or parallel arcs.

Figure 1

The nodes and arcs may each have a set of attributes; for example the arcs may have associated flow rates as in the case of a network of water pipes, or they may have directions as in the case of a directed graph with arrows on the arcs.

2.2 LINKNET Elements

The network is represented in the computer by the LINKNET structure and although this can be implemented in machine language or in a high level language such as FORTRAN or ALGOL it is easier to develop the ideas and notation at the machine language level. Thus it will be assumed that groups of words can be assigned as contiguous blocks or 'elements', and that the addresses of these words are available to enable any of them to be accessed.

Each node of the network is represented in the LINKNET structure by a 'node-element' which holds the identity of the node and all its attributes, and in a similar manner each arc is assigned an 'arc-element'. In addition the structure has 'bead-elements' that are used to give the topology of the network by connecting

the node-elements to appropriate arc-elements in a manner to be described shortly.

The elements (node - arc - or bead-elements) can be thought of as one or more consecutive words of computer memory, with subdivision into fields, each field holding one or more attributes of the entity (node or arc) being represented. Some of the fields contain addresses rather than attributes. The address of an element, also called a link pointer, or reference to that element, is the memory location of its first word. Figure 2 shows a node-element, an arc-element and a bead-element with their fields and the names given to the fields; all the fields shown are basic to the LINKNET structure and additional fields may be specified for the purposes of particular problems.

Node-element:

NUMBER
LIST
NODE-DATA

Field Descriptions

NUMBER - This field holds a number (or characters) to identify the node.

LIST - This field holds a link to a bead-element at the top of a list of beads specifying the arcs connected to this node.

NODE-DATA - This field is a composite of one or more fields holding the attributes of the node.

Arc-element:

NAME
ARC-DATA

NAME - This field holds a number (or characters) to identify the arc.

ARC-DATA - This field is a composite of one or more fields holding the attributes of the arc.

Bead-element:

NEXT
ARC
END

NEXT - This field holds a link to the next bead in a list of bead elements.

ARC - This field holds a link to an arc-element.

END - This field holds a link to the node-element of the node at the other end of the arc specified by the ARC field.

Figure 2

2.3 Lists

As specified in figure 2 the LIST field of a node-element is a link to a list of bead-elements; the NEXT field of the bead-element gives a link to the next bead-element on this list, or is 'null' if there are no more bead-elements on the list. To display a list of bead-elements a diagram like that in figure 3 is used. Figure 3 also introduces a link-variable (or pointer variable), NODE, which is a computer variable whose value is a link, in this case pointing to node 1.

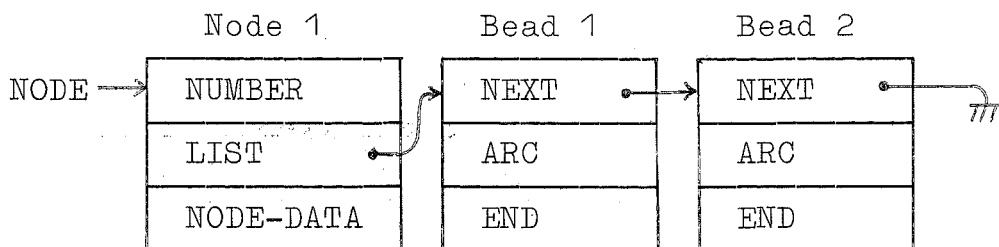


Figure 3

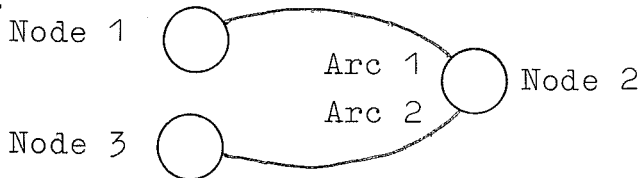
To refer to a field within an element the name of the field is given, followed by a link to the desired element in parentheses; for example in figure 3:

LIST(NODE) = The address of bead 1, and
 NEXT(LIST(NODE)) = The address of bead 2.

Notice that the fields LIST, NEXT, END, NODE-DATA and so on only have values when qualified by a link-variable (or a link-constant), they are not themselves variables.

We are now in a position to consider the LINKNET representation of a simple network, as illustrated in figure 4.

NETWORK:



LINKNET representation:

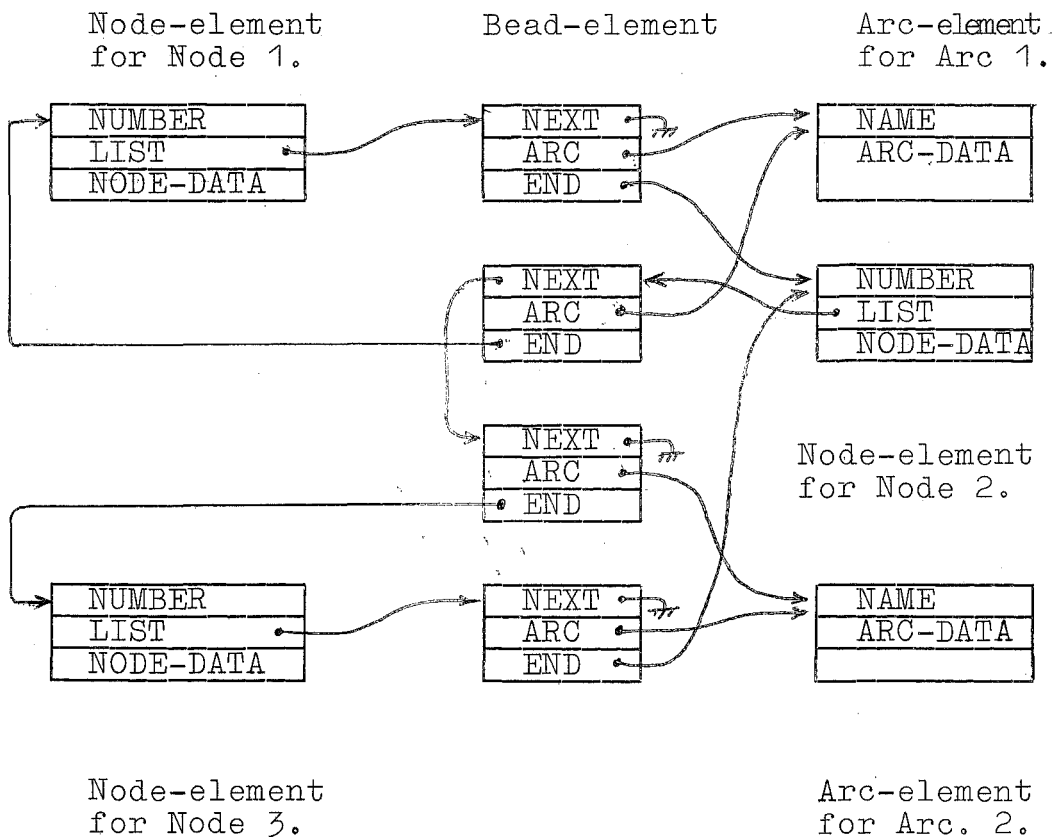
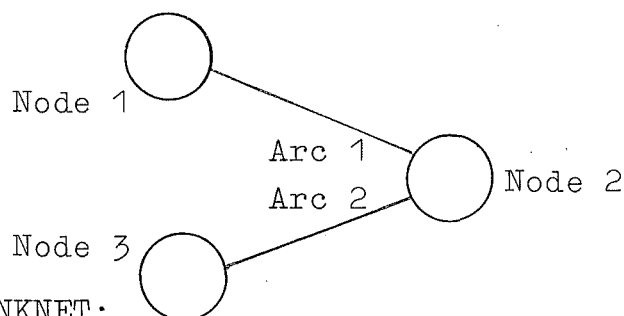


Figure 4.

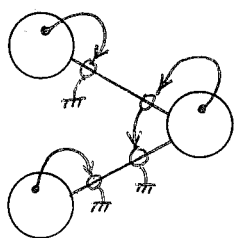
Figure 5 uses the simple network used in figure 4 but shows the pointers (with less detail) in separate diagrams in order to bring out the way LINKNET gives a

'direct' representation of the network. It is an important point that LINKNET attempts to represent the network rather than any data structures that arise during problem solving. This representation primarily of network, with data structures overlaid onto it as needed, has given the programmer a good 'feel' for what a program is doing and has contributed directly to the success of LINKNET applications.

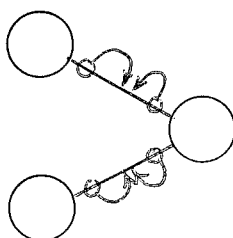
NETWORK



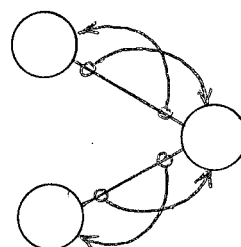
Elements of LINKNET:



LIST(NODE)



ARC(BEAD)



END(BEAD)

NEXT(BEAD)

Figure 5

Two simple data access operations will now be used to illustrate the notation and show how the LINKNET structure facilitates certain procedures.

2.4 Access to arc attributes

The attributes of all the arcs connected to any chosen node of the network can be accessed in turn by 'scanning' the list of bead-elements attached to the node and using the link in the bead-element's ARC field:

arc attribute = ARC-DATA(ARC(BEAD))

where $BEAD \leftarrow LIST(NODE)$ (NODE points to the
chosen node; \leftarrow indicates the value replacement operation)
and then $BEAD \leftarrow NEXT(BEAD)$
until $BEAD = NULL$.

2.5 Access to node attributes

The attributes of all the immediate neighbours to any chosen node can be accessed in a similar manner:

neighbour node attribute = NODE-DATA(END(BEAD))

where $BEAD \leftarrow LIST(NODE)$
and then $BEAD \leftarrow NEXT(BEAD)$
until $BEAD = null$.

3. CREATION OF LINKNET

3.1 Construction of a LINKNET structure

The LINKNET construction algorithm will be written in such a way as to be directly related to a high level language such as FORTRAN or ALGOL. The basic

difference from the machine language implementation considered up to now is that the fields of each element are assigned as separate arrays or vectors, thus the fields of each element are not consecutive memory storage locations (although it may still be easier to think of them as consecutive). Also since storage for the elements is assigned as a set of arrays, identification of a node or arc can be made by specification of an index value, index i being a link or pointer to the i th node or arc element.

Let us assume that the input data that describes the network is arranged in two parts; the first giving a description of the nodes of the networks, and the second giving a description of the arcs of the network, and which nodes they connect.

A1 [Allocate storage.] Assign as single dimensional arrays; NUMBER, LIST, NODE-DATA, NAME, ARC-DATA, NEXT, ARC, END.

A2 [Input information about a node.]

Read in the values; index, identity, data

NODE \leftarrow index

NUMBER(NODE) \leftarrow identity

NODE-DATA(NODE) \leftarrow data

If more node information exists go to A2.

A3 [Initial bead-elements.]

BEAD1 \leftarrow bead-index (BEAD1 & BEAD2 are indices
BEAD2 \leftarrow BEAD1 + 1 for two beads)

A4 [Input information about an arc.]

Read in the values; index, identity, node1, node2,
data ARC1 \leftarrow index, NAME(ARC1) \leftarrow identity,
NODE1 \leftarrow node1, ARC-DATA(ARC1) \leftarrow data,
NODE2 \leftarrow node2,

(NODE1 & NODE2 are indices to the node elements at each
end of the new arc).

A5 [Attach a bead-element to NODE1.]

If LIST(NODE1) = null, LIST(NODE1) \leftarrow BEAD1;
Else BEAD \leftarrow LIST(NODE1)
loop: If NEXT(BEAD) = null, NEXT(BEAD) \leftarrow BEAD1,
Else BEAD \leftarrow NEXT(BEAD), go to loop.

A6 [Attach a bead-element to NODE2.]

Repeat step A5 but for NODE1 read NODE2 and for
BEAD1 read BEAD2.

A7 [Set bead-element links.]

NEXT(BEAD1) \leftarrow null, ARC(BEAD1) \leftarrow ARC1,
END(BEAD1) \leftarrow NODE2,
NEXT(BEAD2) \leftarrow null, ARC(BEAD2) \leftarrow ARC1,
END(BEAD2) \leftarrow NODE1.

A8 [Repeat for next arc information.]

If more arc information exists, BEAD1 \leftarrow BEAD1 + 2,
BEAD2 \leftarrow BEAD1 + 1, go to step A4.

4. APPLICATIONS OF LINKNET

4.1 Minimum length path finding

As an example of the use of a LINKNET structure a minimum length path finding algorithm will be described. In this problem the arcs each have a given length and the object is to find the minimum length path from a given start node to a given goal node. The path finding algorithm to be used aims to try as few paths as possible in the course of finding the minimum length path; the particular algorithm to be used is a simplified version of the powerful A* algorithm of Hart, Nilsson and Raphael [2]:

- B1 Mark the start node 'open' and all the other nodes 'closed'. Set the distance attribute of all nodes to zero.
- B2 Select the 'open' node, n, with minimum 'distance'.
- B3 If n is the goal node terminate the algorithm. The length of the minimum length path is the 'distance' of n, and the path can be traced back to the start by use of the predecessors or 'parents' marked at each node.
- B4 Set the distance of each node m that is adjacent to node n to be the minimum of its current 'distance' value or the 'distance' of node n plus the length of the arc from node n to node m.

B5 Mark all the nodes that have had their distances altered by step B4 'open' and note their predecessor was n. Mark node n as 'closed' and go to step B2.

To implement the above algorithm it will be assumed that the LINKNET structure for the network has been set up and that in addition to the fields shown in figures 2 and 4 there are the following fields initialized to the values given:

DISTANCE(NODE) = 0, this field will hold the distance value of each node as used in the B-procedure.

PARENT(NODE) = null, this field will hold the predecessor of the node.

STATE(NODE) = closed, this field will indicate if the node is 'open' or 'closed'.

OTHER(NODE) = null, this field holds a link to the next node in a list of 'open' nodes.

LENGTH(ARC) = length of the arc.

C1 [Put the start node on the 'open' list.]

OPEN-LIST \leftarrow start-node (OPEN-LIST is a pointer.)

C2 [Find the minimum distance 'open' node.]

NODE \leftarrow OPEN-LIST, MIN-DIST \leftarrow large-value,

Loop: If MIN-DIST > DISTANCE(NODE),

Then MIN-DIST \leftarrow DISTANCE(NODE),

and MIN-NODE \leftarrow NODE:

In any case continue with $NODE \leftarrow OTHER(NODE)$,

If $NODE \neq \text{null}$, go to Loop.

C3 [Check if algorithm terminates.]

If $MIN-NODE = \text{goal-node}$, terminate algorithm.

C4 [Update distance of nodes.]

$NODE \leftarrow MIN-NODE$, $BEAD \leftarrow LIST(NODE)$,

Begin: $NEW-DIST \leftarrow DISTANCE(NODE) + LENGTH(ARC(BEAD))$,

$NEW-NODE \leftarrow END(BEAD)$,

If $DISTANCE(NEW-NODE) > NEW-DIST$,

Then $DISTANCE(NEW-NODE) \leftarrow NEW-DIST$,

and $PARENT(NEW-NODE) \leftarrow NODE$,

and if $STATE(NEW-NODE) = \text{closed}$,

then $STATE(NEW-NODE) \leftarrow \text{open}$,

and $OTHER(NEW-NODE) \leftarrow OPEN-LIST$,

and $OPEN-LIST \leftarrow NEW-NODE$:

In any case continue with $BEAD \leftarrow NEXT(BEAD)$,

If $BEAD \neq \text{null}$, go to Begin.

C5 [Delete node from 'open' list.]

If $OPEN-LIST = NODE$, then $OPEN-LIST \leftarrow OTHER(NODE)$,

Else $N \leftarrow OPEN-LIST$, (N is a link variable)

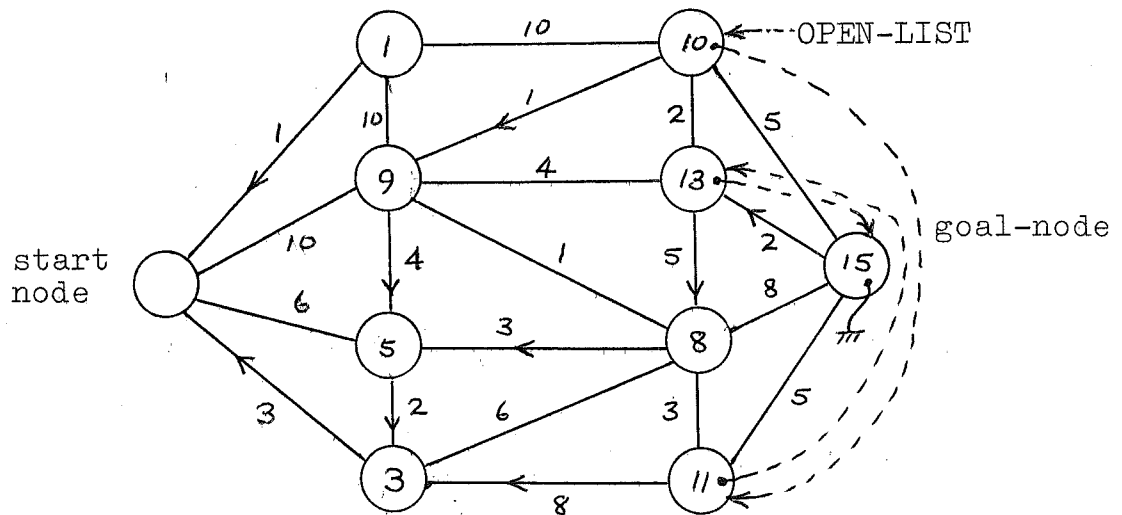
Step: If $OTHER(N) = NODE$,

then $OTHER(N) \leftarrow OTHER(NODE)$;

Else $N \leftarrow OTHER(N)$,

After this go to step C2.

Figure 6 shows an example minimum length path finding problem with an indication of the distance values and 'open' list members at the stage when the algorithm has been through step C5 six times and has three times more to execute step C5 before terminating. The arrows on the arc indicate the parent nodes for each node, on termination a path could be traced back from the goal node to the start node using these pointers. The minimum length path, and distance of the goal node on termination of the algorithm, is 14 units.



The numbers on the arcs are the arc lengths.
 The numbers at the nodes are the distances.
 The arrows on the arcs are the parent pointers.

Figure 6

This implementation of the minimum length path algorithm shows that the programming can very closely follow the algorithm's descriptive terms. This is a common feature in algorithms that apply to networks since very frequently they 'look at' adjacent nodes or expand out from a node; with the LINKNET structure this is simply a matter of scanning down the bead-element list attached to the node being considered. Admittedly the algorithm is quite simple and the implementation given is slightly clumsy but extension to having an ordered list of 'open' nodes (thus eliminating the need for the STATE field of the node-elements) and adding the extra function that is used in the A* algorithm [2], are refinements that do not destroy the basic simplicity of implementation.

The example used in figure 6 appears in Berge [3], along with some different algorithms for minimum length path finding.

4.2 Finding Meshes and Spanning Trees

A problem will now be considered that has several different applications. The applications will be mentioned briefly later but for now the problem will be treated just as a network manipulation problem. Starting with any network, for example, the network in figure 7, the aim is to label a set of arcs that

connect up all nodes of the network without forming any loops. Such a tree is called a spanning tree and the arcs that form this tree are to be labelled 'branches' while all the remaining arcs are to be labelled 'links'. Once the spanning tree is set up the meshes of the network can be defined as loops containing one link and a path along tree branches, thus there are the same number of meshes as links. The particular spanning tree found by any algorithm may be quite important, research is still underway [4] to find efficient algorithms to find the best 'root' node or starting node for an algorithm to produce a spanning tree that is minimal in some sense. The algorithm given here simply finds a spanning tree starting from a given root node.

Briefly the algorithm 'expands' out from the root node marking all arcs as 'branches' and all nodes at the ends of these branches as 'tree members'. Subsequent expansions occur from nodes that are members of the tree, and if any arc leads to a node also in the tree then this arc is marked as a 'link' rather than a 'branch'. The algorithm terminates after all nodes that are members of the tree have been expanded once.

Assume that the LINKNET structure has been set up for the network with the following extra fields:
TAG(NODE) = 0 initially and 1 after it has been found.
PARENT(NODE) will receive a pointer to the node from

which this node was reached, i.e. its predecessor in the spanning tree.

TYPE(ARC) will be set to 'branch' for an arc in the spanning tree or to 'link' for all other arcs.

Three stacks (push-down lists) will also be used: STACK holds nodes to be expanded from, STACK1 and STACK2 hold the nodes at the ends of each 'link', one in each stack.

D1 [Put root node onto working stack.] $STACK \leftarrow \text{root-node}.$

D2 [Get next node to expand from.]

$NODE \leftarrow STACK,$

If $NODE = \text{null}$ terminate algorithm.

D3 [Tag nodes and classify arcs.] Scan down the list of bead-elements pointed to by LIST(NODE) and for each bead on this list:

$NODE1 \leftarrow \text{END}(\text{BEAD}), \text{ARC1} \leftarrow \text{ARC}(\text{BEAD}),$

If $\text{TYPE}(\text{ARC1}) = \text{branch or link}$, step on to next BEAD.

Else if $\text{TAG}(\text{NODE1}) = 0$, then $\text{TAG}(\text{NODE1}) \leftarrow 1,$

and $\text{TYPE}(\text{ARC1}) \leftarrow \text{branch},$

and $\text{PARENT}(\text{NODE1}) \leftarrow \text{NODE},$

and $STACK \leftarrow \text{NODE1};$

Else if $\text{TAG}(\text{NODE1}) = 1$, then $\text{TYPE}(\text{ARC1}) \leftarrow \text{link}$

and $STACK1 \leftarrow \text{NODE},$

and $STACK2 \leftarrow \text{NODE1};$

After all beads are done go to step D2.

Figure 7 shows a simple network after the algorithm has specified the spanning tree, the branches are shown as solid lines with an arrow indicating the parent pointer, while the links are marked as dotted arcs. It should be clear that from this structure the meshes can be found by tracing back from each end of each link (STACK1 and STACK2 hold these nodes); after the trace reaches the root node from each end of the link, common branches (if any) can be eliminated.

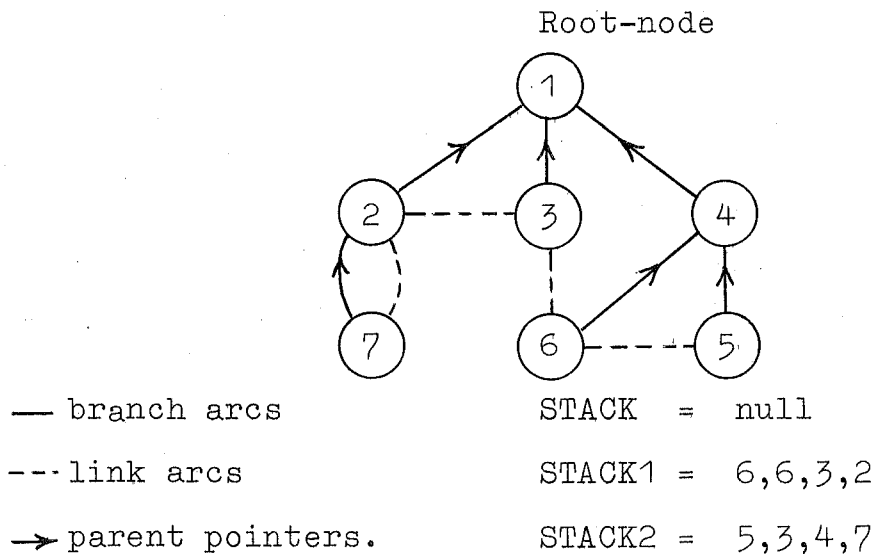


Figure 7

Use of Trees and Meshes

One application for this algorithm is to find mesh loops in an electrical network so that a set of simultaneous equations can be set up to solve for the currents flowing in each mesh loop. In practice there

may be the additional complication of mutual coupling between arcs by way of magnetic fields. The network now consists not only of nodes interconnected by arcs but also of arcs interconnected by 'mutuals'. However the LINKNET structure can be expanded to cope with this complication in the manner indicated by the diagrams in figure 8.

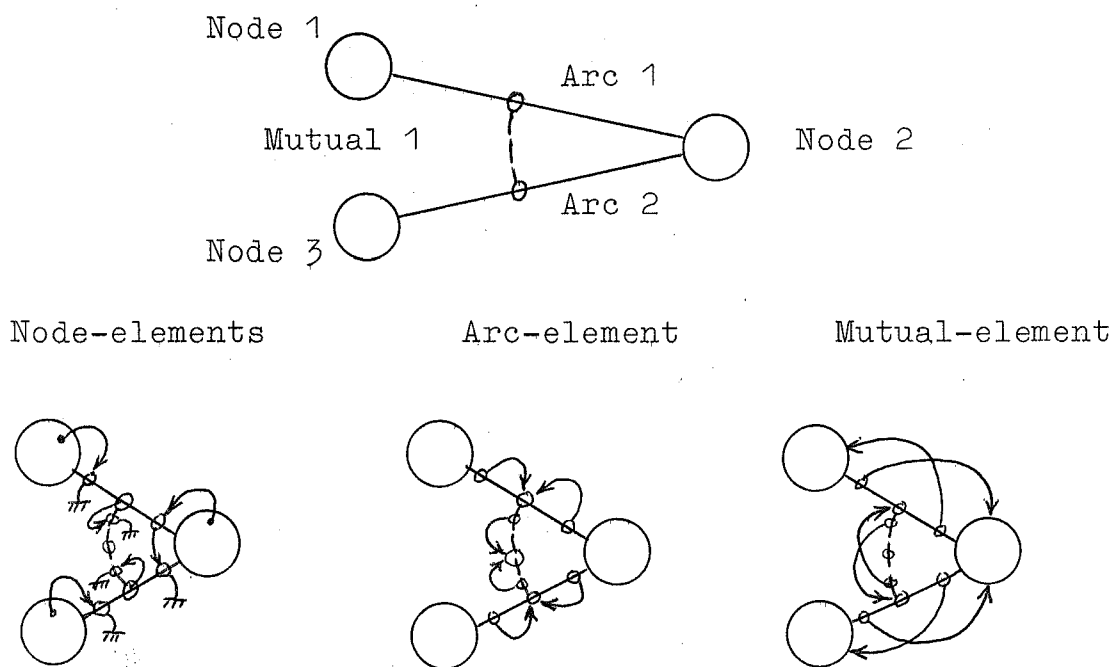


Figure 8

The mesh loop current equations can themselves be thought of as a new network, where the nodes are mesh loops, and the arcs are mesh loop interactions. A program has been written (in FORTRAN) that after finding the meshes of an electrical network, constructs a second LINKNET structure to represent the mesh loop

equations. Further than this it is possible to solve these equations by an iterative procedure that utilizes this new LINKNET structure.

5. CONCLUSIONS

The LINKNET structure for representing networks by the use of linked lists has been described in some detail. The structure represents the network in the computer without any immediate regard to processing that may occur on the network or on attributes of network elements. The structure is put forward as a basic form that can be elaborated as required to include features that are special to any particular network.

After the LINKNET structure is established the implementation of several procedures is considered. It is shown that for particular problem procedures the LINKNET structure can be easily extended to facilitate the desired manipulations of the network or its attributes. More than this it is often the case that the LINKNET structure can be used to guide the course of the procedures from node to node and arc to arc in the required manner.

The saving of memory space that can be achieved by the use of a linked list data structure rather than a matrix method have not been stressed in this presentation. It is often the case however, that networks are

very large; power systems or transportation networks for example may have several hundred nodes. Not only are such networks often large but they are far from fully interconnected giving rise to very sparse entries in a matrix representation. In such cases the saving in memory by use of a LINKNET type structure may be extremely important.

The main aim of this appendix has been to bring attention to the ease of applying linked-lists and List-processing methods to network problems. It is hoped that the LINKNET structure and network procedures demonstrated here will give more programmers an incentive to consider this attractive alternative to matrix methods for network problem solving.

REFERENCES

- 1 Knuth, D.E. The Art of Computer Programing
Volume 1, Fundamental Algorithms, Chapter 2, Data
Structures, Addison-Wesley, 1968.
- 2 Hart, P.E., Nilsson, N.J. and Raphael, B. A Formal
Basis for the Heuristic Determination of Minimum
Cost Paths. I.E.E.E. Trans on System Science &
Cybernetics, Volume SSC - 4, No.2, July 1968,
pp 100-107.
- 3 Berge, C. The Theory of Graphs, Methuen and Co.
London, 1962.
- 4 Snow, C.R. and Scoins, H.I. Towards the Unique
Decomposition of Graphs. Machine Intelligence 4
Ed. Meltzer, B. and Michie, D. Edinburgh University
Press 1969, pp45-55.

APPENDIX B

GRAPHICS DISPLAY SYSTEM

APPENDIX B1. INTRODUCTION

This appendix briefly covers some of the main points in the overall organization of a graphical display system that was designed and constructed in the course of this thesis work in cooperation with M.R. Mayson. It is only intended to give the basic concepts of system organization that emerged from the work, with little implementation detail and only a functional outline of the hardware that is involved.

The display system runs on an EAI640 computer. This is a conventional 16 bit word, 1.6 μ s access, 8K machine with high speed paper tape and a fixed head disc. The machine is installed in the Electrical Engineering Department as part of an 640/580 (590) hybrid computer system.

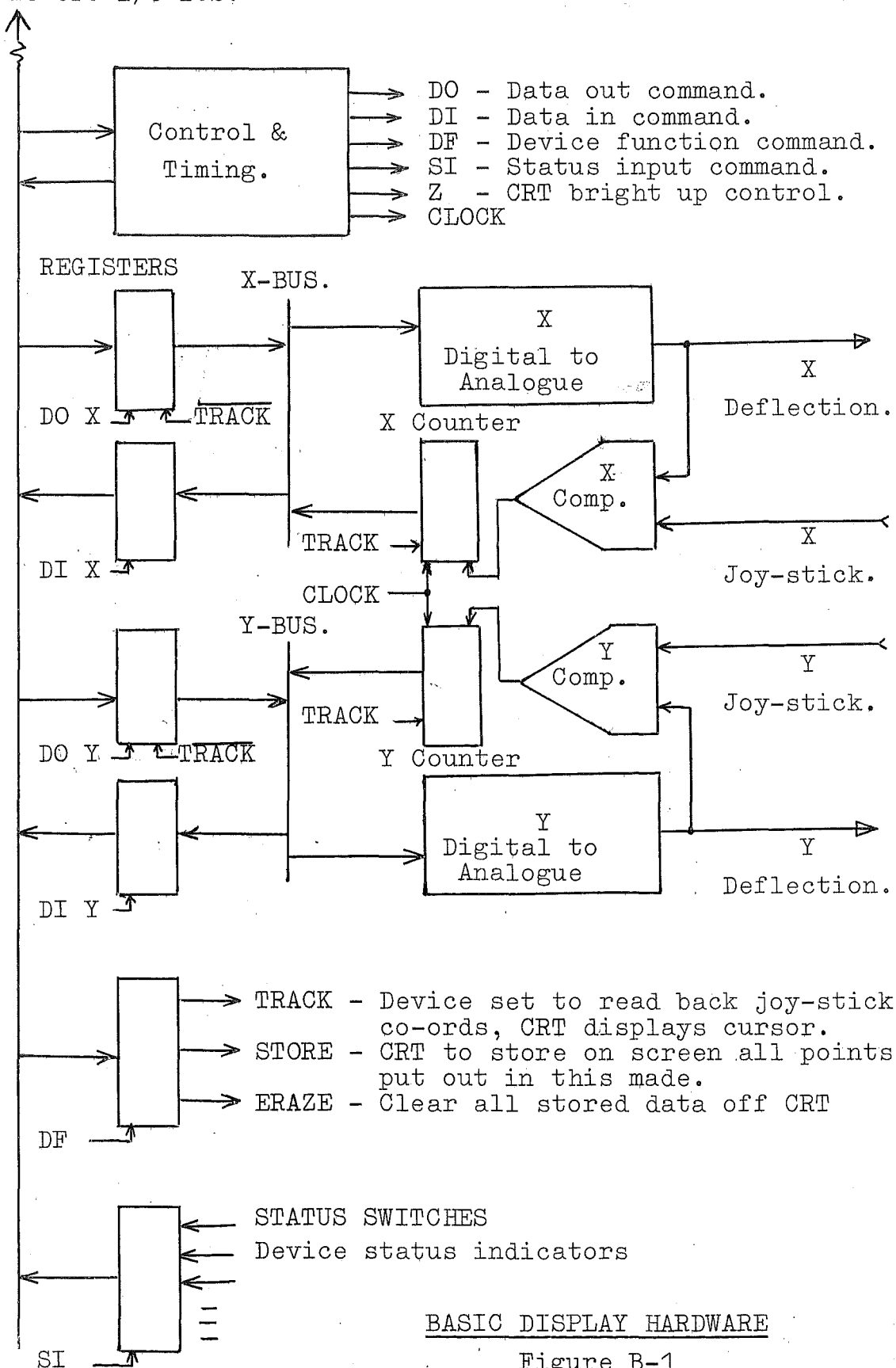
The CRT display chosen was a Tektronix 611 (11") storage display-scope; a storage system being selected almost entirely on a cost basis. This type of storage CRT can retain an image after a single write operation, or it can display a repetitively written image as done by a normal CRT. An additional feature is the 'write-through' mode which enables information to be displayed by repetitive writing without storage, while other previously written information is retained as a display

image. This was one of the first 611's produced (before commercial use in computer terminals, etc.) and it has been in use for almost two years. It is interesting (and important) to note that the use of this facility is almost always for text output - disc map dumps, text editing and so on - very seldom is the full graphical ability required.

Another basic decision (that has proved reasonable) was to put in a minimal amount of hardware and cope with as much as possible in software. The only functions built into the hardware are for the output of points, CRT mode control (store, write-through, erase), together with control for inputs from function buttons, status switches, and a joy stick for moving a write-through 'crusor' with ability to read in the co-ordinates. Line drawing, character generation and so on are all executed by the software; the speed is normally very well matched between hardware ($\geq 20\mu\text{s}/\text{dot}$ required) and the software (about 6 instructions/dot).

The basic functions of the hardware are illustrated in figure B-1.

To CPU I/O BUS.



BASIC DISPLAY HARDWARE

Figure B-1

The hardware has been built twice, the first was a 'lash-up' using RTL logic, and the current version was built commercially (to our design) using CTuL circuitry.

The software is into its second major version. The experience gained from the mistakes made with the first software package is the main reason for this appendix. The software effort has been very large (almost too much for the two of us part time) and like most software it always remains an open ended job.

2. Basic System Requirements

A list of the features that were considered necessary in roughly their order of implementation is:

- 1) Character generator.
- 2) Teletype simulator, allowing text to be put out on the teletype, the display, or both, at any time by use of the status switches.
- 3) Line drawing routine (between any two co-ordinate pairs).
- 4) Display options for standard system programs - particularly the text editor.
- 5) Display data file interpreter to draw points, lines and text from information in a picture data file (figure B-2).
- 6) A graphics editor to create and edit the display data files in an interactive manner (figure B-2).

Items 1) through 4) are quite straightforward and will not be referred to again. Items 5) and 6) involve the vast majority of the effort and will be considered in more depth.

First we will define two distinct activities - An Editing phase - at this time display data files are edited interactively before preparation as relocatable data files (modules).

A run-time phase - at this time memory is restricted since the display must co-exist in core with a user's application programs.

Figure B-2 shows the interpreter, the editor, and their relationship.

3. The Edit Phase

The editing problems can be tackled in two different ways - although these two approaches are not necessarily incompatible or mutually exclusive.

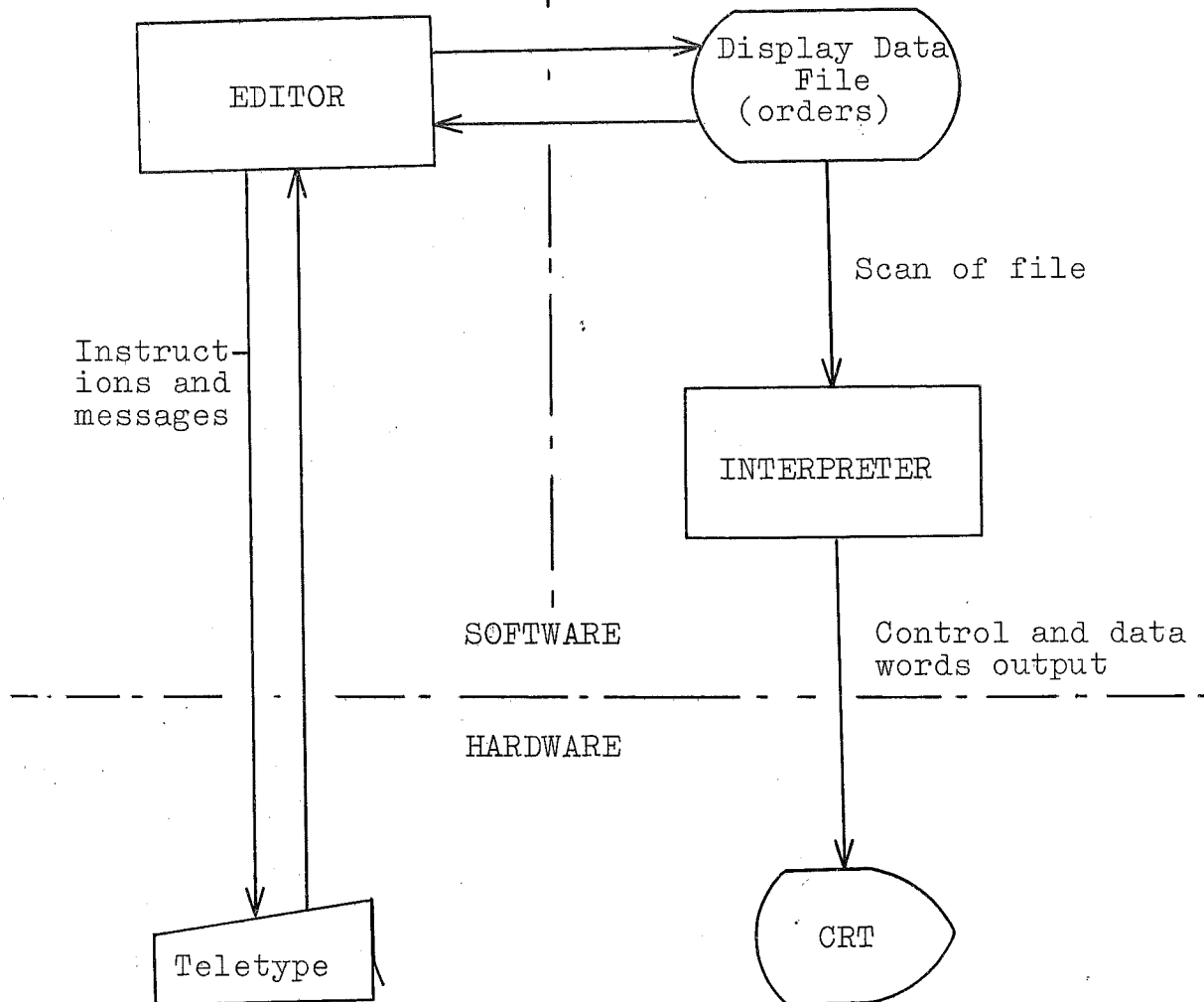
1) Direct Draw Facility

The idea here is to allow the user to create and edit a picture directly on the display screen. For example to draw a line he could position the cursor with the joy stick and press 'BEGIN POINT' (function button) then move the cursor and press 'DRAW LINE'. In

In core for Edit
Phase only.

In core for both
Edit Phase and
run-time phase.

B - 6



Also: Function buttons,
Status switches,
Joy-stick.

BASIC FUNCTIONS OF DISPLAY EDITOR AND DISPLAY INTER-
PRETER

Figure B-2

a similar manner 'FIND', 'CHANGE', 'DELETE', 'INSERT', 'TEXT' and so on can be used to create and edit a data file.

This philosophy was embodied in our first display system of software and it is undoubtedly attractive and very easy for an operator to use. Unfortunately (as will be discussed) it leads to some very difficult software problems unless the data files are of quite a simple form.

Direct draw is probably the best approached as an eventual extension of approach 2) -

2) Display Language Facility

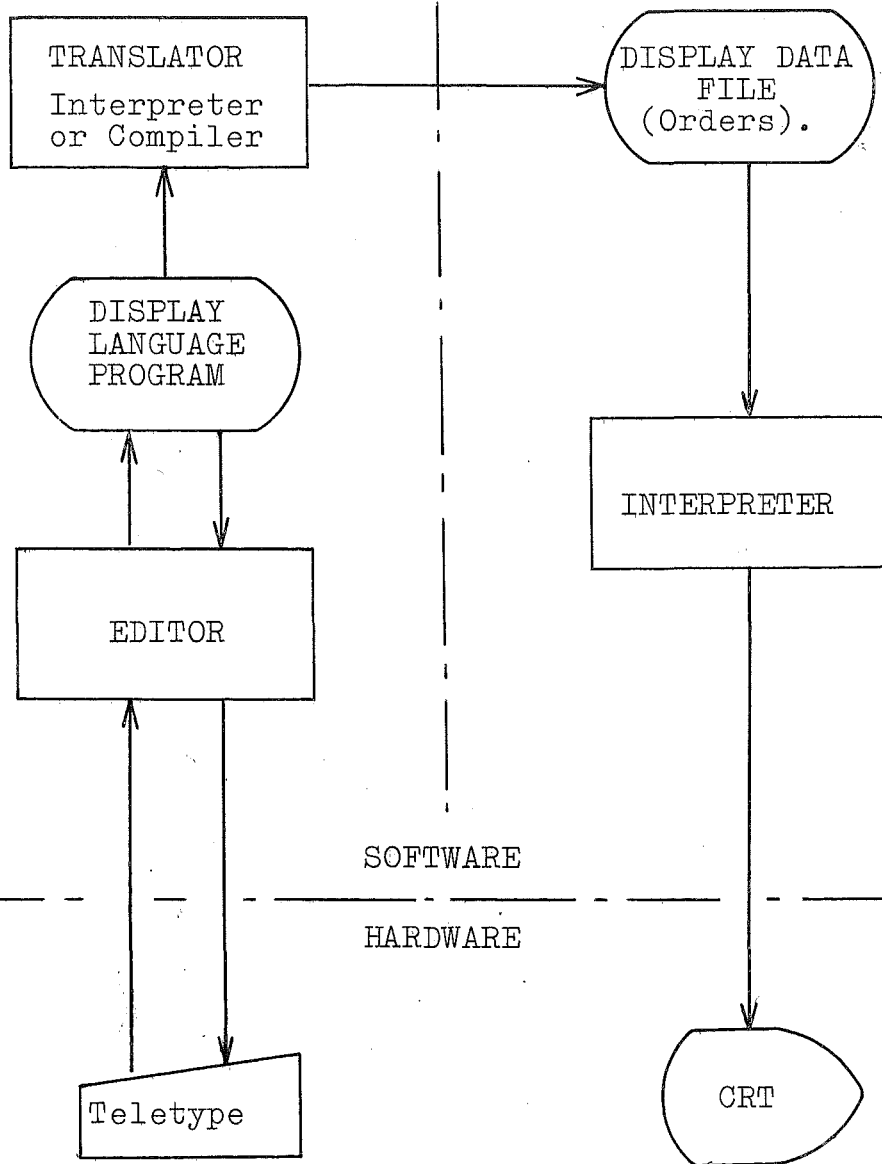
A high level display language is a far more satisfactory system from the software point of view since it is possible to design a modular and extensible system. For anything approaching the direct draw facility the display language must be reasonably rich (with pseudo operations and macro generation). A compiler or interpreter could be used to generate the basic display file data from the high level language instructions. Figure B-3 illustrates this.

A subtle and interesting point is that even if direct draw facilities are created the operator still has the ability to edit the picture program rather than the picture itself as it appears on the display. The

In core only for edit
phase.

In core for both the
edit phase and the
run-time phase.

B - 8



Also: Function buttons,
Status switches,
Joy-stick.

USE OF A DISPLAY LANGUAGE

Figure B-3

importance of this type of editing is for operations like naming a picture, inserting a call to a sub-picture, linking run-time variables into a place in a picture, and so on. In fact for all the operations associated with picture editing that do not necessarily appear directly as parts of the image. These features are mentioned again under Run-time Phase, below.

Rather than define a high level language directly, the approach we have chosen is to embed the low level data file information into a higher level interpreter system so that together they form a self-extensible higher level display language. The user can employ pre-programmed high level commands or he can create new ones of his own or he can create even higher level instructions that employ the original high level instructions.

The higher level interpreter (higher level than the display file processor to be discussed later) used in this case is TRAC (Text Reckoning And Compiling) [1], [2]. This is a String processing and macrogenerator language, similar to GPM [3]. The features that make TRAC suitable for this work are:

- a) It is designed as an interactive language.
- b) It is able to define text macro forms with formal parameter creation and substitution.

- c) It has the ability to embed itself into higher level interpreters written in TRAC itself.

4. The Run-Time Phase

At run-time the display programs co-exist in core with a user's non display programs. The user's programs have the ability to request the display of a display data file, or to take an active role in the display if so desired. It is desirable to have a modular set of display programs so that if only the basic display features are required the core overhead can be kept accordingly low.

It has been found that it is not the direct display of display data files which causes difficulties, but rather the communication between display files and run-time programs, and the flow of program control. Some of these problems are generated by the following requirements:

- 1) The display data files need to be available as relocatable modules or subroutines that can be named and loaded along with standard relocatable modules produced by assembler and FORTRAN programs.
- 2) Display files should be able to call other data files as subroutines (subpictures).
- 3) Display files should be able to request the value of run-time (computed) variables for inclusion in the display.

- 4) Display files should be able to initiate computational programs as subroutines during picture construction.
- 5) Run-time programs should be able to control the execution of a display by modification of display files and also by control over the programs that display the display data files.

Rather than go through the display development chronologically the development given here is our latest approach. This is (overall) considered to be the most fruitful approach for any small machine, storage tube display system, in order to meet the sort of requirements outlined in the sections above. Comments will be interspersed at points where the approach is considered to have particular advantages over alternative approaches (and our previous attempts!).

Display Order Interpreter

The basic software and first requirement for implementation is the display order interpreter. The function of this program is to interpret and execute the display orders from the display data file. This can be looked on as a simulation of the hardware used in refresh CRT systems to display from code in memory. The display data files instructions are called 'orders' to distinguish them from the CPU 'instructions' of the normal

machine code. It is worth considering designing the interpreter to handle orders with the same format as some particular refresh system hardware. In this way the same display files could be used either on the storage display via the interpreter or on a refresh CRT (perhaps on another computer) using the hardware. However, if such hardware is not likely to be available it is not worth restricting the order set in this way.

If the order set is only for use by an interpreter the set is easily left open-ended for future extensions and alterations. More interesting the orders can be made considerably more complex and specialized with very little extra software overhead. For example, different orders can be made 1, 2, 3 or more words long, and the x and y coordinates can be given as memory base and displacement addresses rather than actual values (relative or absolute). It seems well worth while taking advantage of the storage mode of the storage tube CRT by allowing more complex display orders than is possible with refresh CRTs.

A critical factor that has become apparent is that the display order set should include not only display instructions (point, line etc.), but also procedural orders including at least the following:

- 1) Jump and link (to subroutine) orders.
- 2) An end order that may serve as a subroutine return.

- 3) A change or link order that allows for a change or branch to normal machine CPU (non interpreter) instruction execution.

Notice that an order to go to machine instruction mode (point 3)) eliminates the need for a fuller set of interpreter orders to handle arithmetic, logical, and test operations.

In the design of the interpreter itself the following points should be considered:

- 1) The interpreter must be re-entrant.
- 2) All registers used by the interpreter that may alter the interpretation of orders (the 'state' of the interpreter) should be made available external to the interpreter (as global names or absolute locations). Preferably the complete stack of registers for each entry (activation) should be available.
- 3) There should be no flags counters, etc. that are not either clearly defined as parts of the registers (point 2)), or reset prior to any exit from the interpreter or the completion of an order. (This may be obvious in view of point 1) but it is vital to the success of the interpreter).
- 4) On exit from a picture subroutine (or computational subroutine) the 're-entry' of the interpreter should

allow the two possibilities:

- a) Retaining the interpreters registers (the X,Y coordinates in particular), as they are on exit from the subroutine, or
- b) Restoring the registers to their condition before the last interpreter entry (before the subroutine was called, i.e. the normal activation record 'pop up' operation of a re-entrant program).

5) All input and output from the interpreter to the display CRT function buttons and so on should be routed through a common I/O routine. The purpose of this is:

- a) This enables the same interpreter and display files to be routed to different device controllers for example the CRT display could be put out to a X-Y plotter. The only restriction on this is the available hardware (the I/O routine should take care of device peculiarities).
- b) The output can be intercepted in the I/O routines so that the output of a display file can be 'simulated' at high speed. This enables us to find points in the display that are not explicitly in the display data file but are computed during the course of the display output. The simplest example of this is points in the middle of a line which is drawn with a single display order.

- 6) The 'instruction fetch' operation of the interpreter that sequentially fetches display orders for execution, should be made as an accessible program module. The purpose behind this is to enable data files to be executed even though the orders are not sequential in core storage. A particular use of this facility (to be mentioned again later) is to enable the 'order fetch' program to fetch a string of characters off a linked list, assemble a group into a binary word and return this to the interpreter as the next instruction.

Figure B-4 summarizes the main points in the interpreter construction. Notice that the design looks very much like a CPU organization. This is a good way to look at the interpreter architecture, and will serve as a sound design guide.

Display File Editing.

With a display order set established and the associated interpreter working we can move on (both in this description and in implementation!) to a higher level interpreter for compiling and editing the display orders. Because of the requirement to have the display data files (orders) in the form of standard relocatable modules (standard object program format) there are

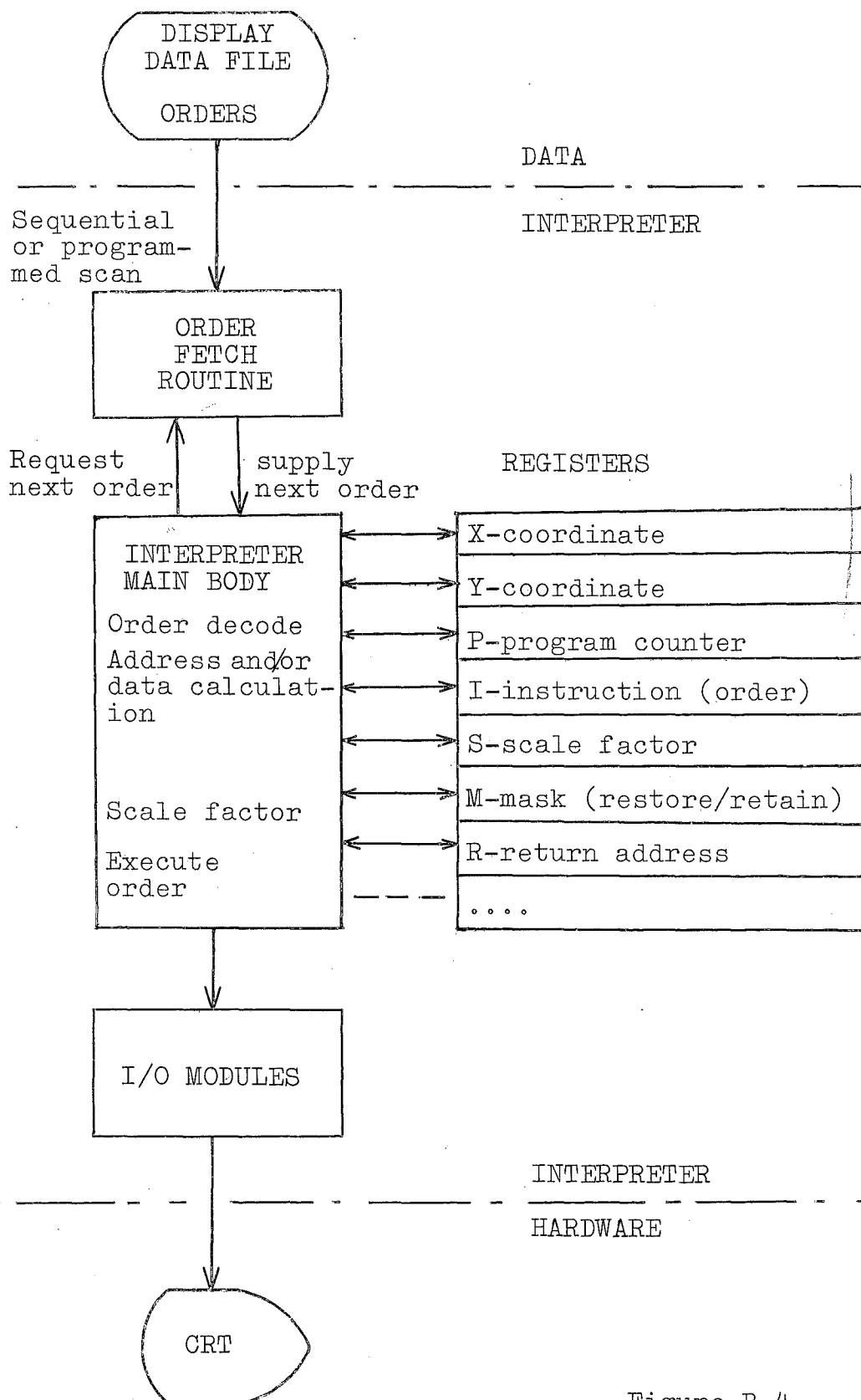


Figure B-4.

several advantages in having a program that accepts mnemonics for the display orders and rather than producing the display data file itself producing these mnemonics in terms of standard assembler mnemonics. That is, the picture 'source' file is translated into an assembler source program in such a way that the assembler will process this into relocatable program module which when loaded will have the desired display orders, address references and so on. Figure B-5 shows the overall process involved. The particular advantages of this two stage process all relate to making the higher level interpreter (or compiler) easier to write:

- 1) Mnemonics for standard display orders such as absolute points or lines, can be translated into assembler 'data' type statements.
- 2) All resolution of symbolic names and relative addressing can be resolved by the assembler in the normal way.
- 3) Global and local symbols, including a global (external) name for the display data file as a whole, can be coped with by inserting standard assembler language pseudo-operations into the source code.

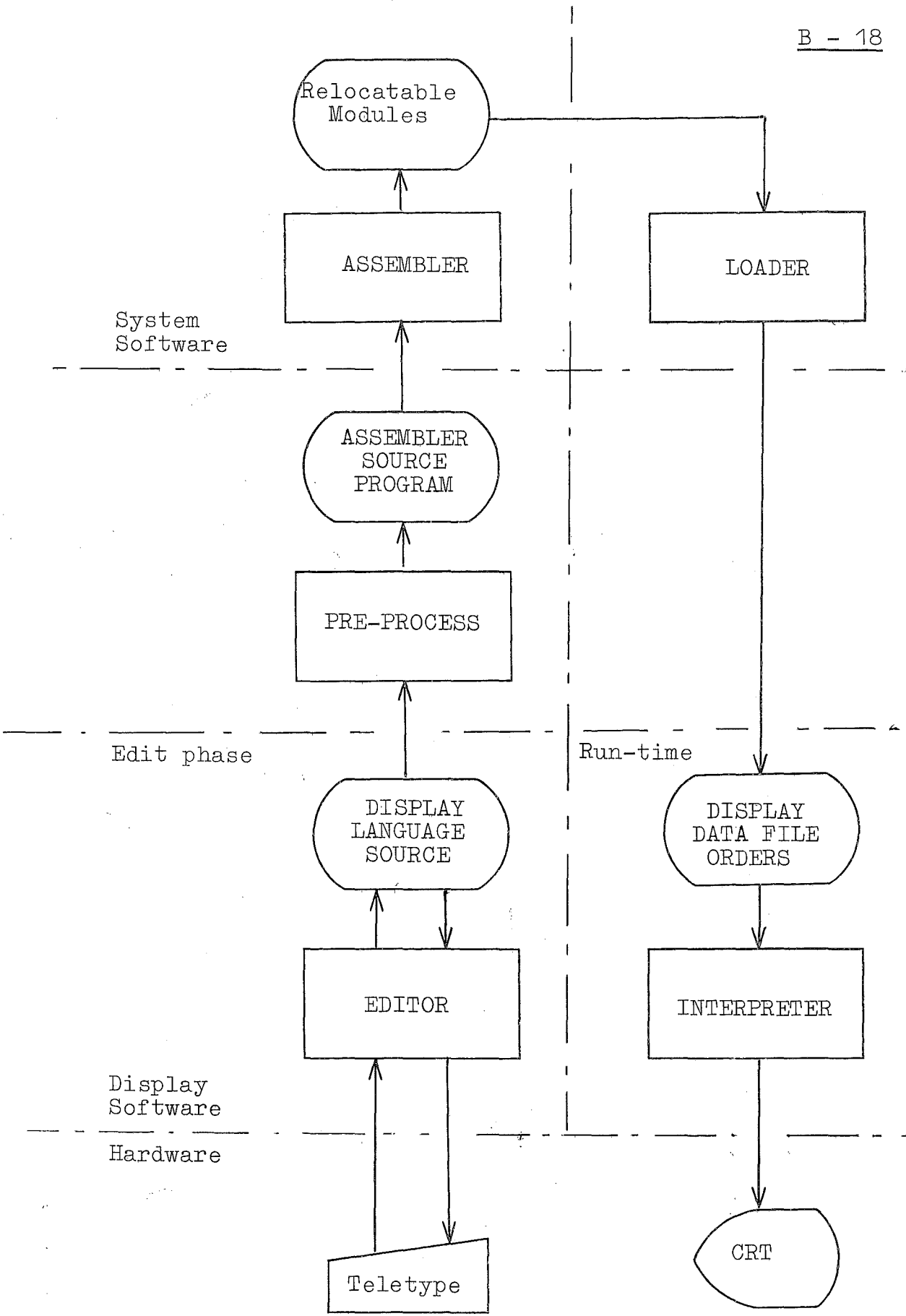


Figure B-5

If the standard assembler has the ability to have additional mnemonics added to it, it may be possible to use the (modified) assembler directly on the picture 'source' program. In this case no additional precompiler program would be needed, this possibility was not available (easily) with the EAI 640 assembler.

There is one real disadvantage of a scheme requiring an assembly process before the display data file is in a suitable condition to be loaded and displayed - that is, the lack of editing ability in the direct draw vein. The editing of a picture takes on the same form as the editing of a computational program. The round of source editing, assembly, loading, and debugging is far from the interactive ideals of the direct draw facility. Figure B-5 shows the overall process involved.

A far more powerful and flexible scheme that can be built up towards a direct draw capability is the use of a high level interpreter as discussed in the next section. See also figure B-6.

5. Editor-Interpreter

The high level interpreter used in our implementation was TRAC, as mentioned previously. Rather than talk of a higher level interpreter in general, TRAC will be referred to, even though some of the comments apply to

other interpreters that may be used in place of TRAC.

The key feature that makes TRAC a more attractive system than a simple pre-processor is the ability to draw a picture onto the CRT direct from a character string that can be defined using TRAC. This is best described by a simple example:

A display data file of orders to draw a square 100 by 100 (octal) would appear in core -

Display orders in a binary file (octal)	Interpreter action
70000 30000	Output a point to the CRT at coordinates 0,0.
70100 130000	Draw a line from the current position for $\Delta x=100, \Delta y=0$ (horiz.)
70000 130100	Line $\Delta x=0, \Delta y=100$.
73700 130000	Line $\Delta x=-100, \Delta y=0$.
70000 133700	Line $\Delta x=0, \Delta y=-100$.
177777	End of picture

In TRAC this list of display orders could be defined in the form of a string of (ASCII) characters called, say, BOX. This is done by typing -

```
*(DS,BOX,(70000,30000,70100,130000,...,133700,177777))'
```

The TRAC primitive DS defines a string with name BOX. To fetch this string from memory the following can be typed -

 *(CL,BOX)' resulting in a returned value
that is the string of characters listed above.

An additional primitive to the basic TRAC set can be defined as *(DR,list) where 'list' is a string of octal-digit characters; the function of DR being to issue each set of digits as a binary word to an external program. The program used in our case is the fetch order routine for the display order interpreter. Thus the command -

 (DR,(CL,BOX))' results in the drawing via the interpreter of a box (100 x 100) on the CRT, starting at position X,Y = 0,0.

TRAC has the ability to place formal parameters into a string of characters by the use of the SS (segment string) primitive. The commands

*(DS,BOX,(X,Y,70100,130000,...,133700,177777))'
*(SS,BOX,X,Y)'

would result in a string BOX being defined as above but with formal parameters in place of the X and the Y.

The parameters can be substituted for by a call to the string BOX -

 *(CL,BOX,70000,30000)'

which would result in the same string of characters as

in the first definition of BOX given previously.

The primitive BU forms the boolean union of its two arguments, for example -

※(BU,100,201)' would result in the value 301.

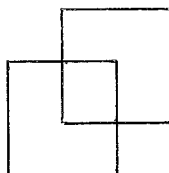
We can now define BOX as a string of characters as before but with formal parameters for the X and Y position of the first corner of the box -

```
※(DS,BOX,(※(BU,70000,X) (BU,30000,Y),
          70100,130000,...,133700,177777))'
※(SS,BOX,X,Y)'
```

Now a command -

```
※(DR,※(CL,BOX,0,0)※(CL,BOX,50,50))'
```

would result in the drawing on the CRT of two boxes something like this:



This ability can be embedded into higher level functions to allow a simpler set of commands to be typed. The TRAC programs to do this are shown in figure B-6 as 'TRAC DRAW PROGRAM', with an arrow indicating the use of the DR primitive to activate the interpreter.

Also shown in figure B-6 is a TRAC program called 'PRE-PROCESS' program. This program allows TRAC to produce a standard format assembler language source

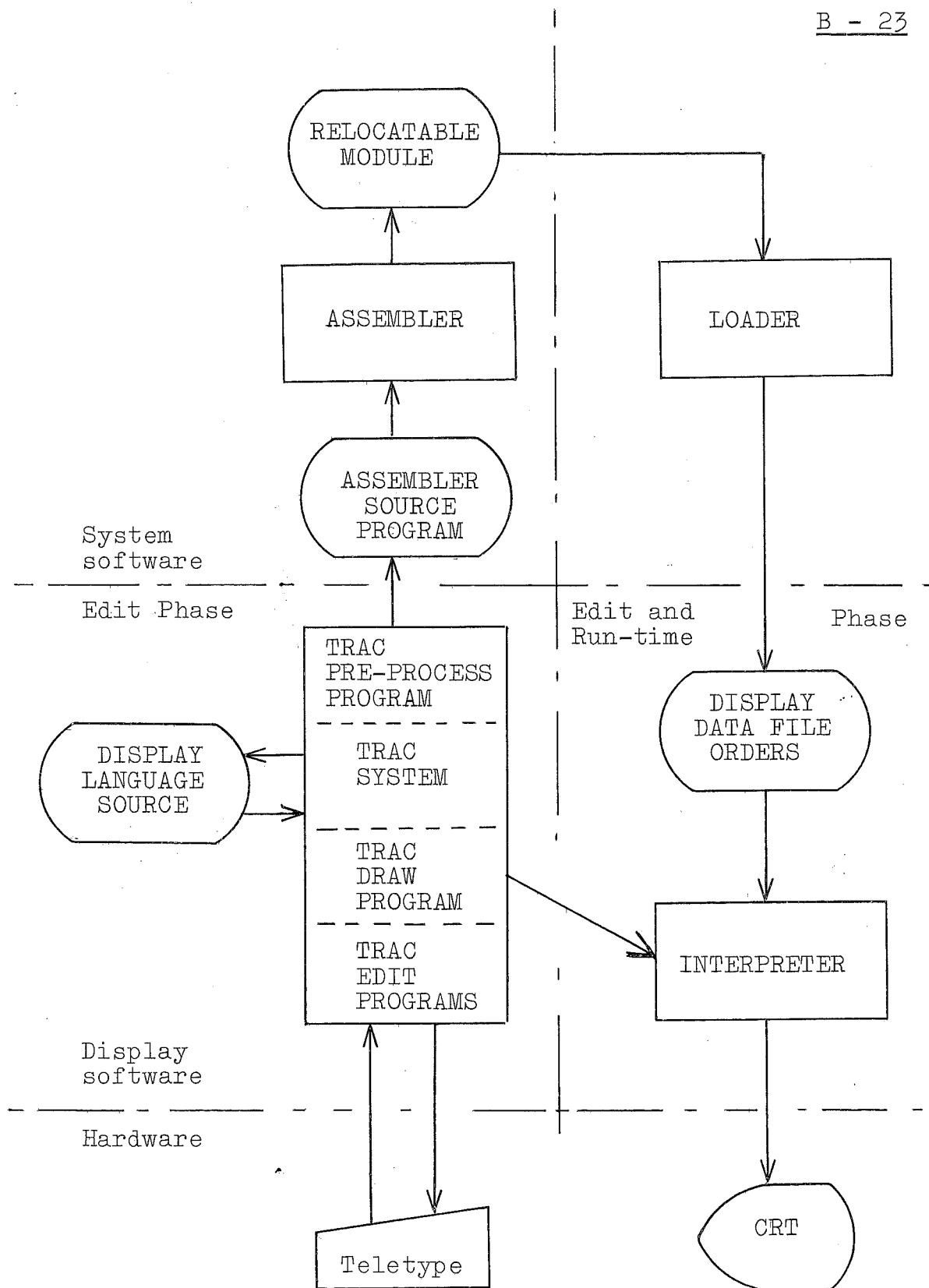


Figure B-6

file for the production of relocatable modules for use at run-time. The operating procedure would be to create and edit a picture as a TRAC source program, with the effect of this picture being viewed on the CRT with the TRAC DRAW PROGRAM. After the picture is finished a source program may be saved for future editing and an assembler source program produced by execution of the TRAC PRE-PROCESS PROGRAM.

To give some idea of how PRE-PROCESS can work consider the following illustrative example:

※(DS,PROGRAM,(*	Notes:-
*ASSEMBLER SOURCE OF TRAC STRING FRED.	
*	*marks
REL 0	comments.
NAME FRED	Relocatable
	form.
	External
	name.
	OCT =
*FRED OCT ※(SS,FRED,(,))※(CL,FRED,(assembler
OCT))	mnemonic for
	octal data.
OCT 177777	Display end
END 0	order.
))※(SS,PROGRAM,FRED)'	Assembler
※(DS,BOXES,(※(CL,BOX,0,0)※(CL,BOX,50,50)))'	end.
※(DS,BOXES,※(SS,BOXES,177777)※(CL,BOXES))'	Delete end
	orders

The following order should result in a value that is a text string for a complete assembler source program

for a display data file to draw the two boxes illustrated previously.

*(CL,PROGRAM,BOXES)'

BOXES replaces
FRED each time.

REFERENCES

- 1 Mooers, C.N. TRAC a procedure-describing language for the interactive typewriter. Comm.ACM 1965 p.215.
- 2 Mooers, C.N. How some fundamental problems are treated in the design of the TRAC language. Symbol Manipulation Languages and Techniques, Ed. Bobrow, D.G., North Holland 1968. From Proc. IFIP working conference on Symbol Manipulation, Pisa, 1966.
- 3 Strachey, C. A general purpose macrogenerator. Computer J. 1965, p.225.

Other material not referenced directly in the text:

- . Interactive Graphics in Data Processing. IBM System Journal, Vol.7, No.3 and 4, 1968.

This volume contains more than a dozen papers and has a large number of references.

APPENDIX C

ANALYTIC CALCULATION OF BANDIT SELECTION PROBABILITY

FOR NORMAL PROBABILITY DENSITIES

APPENDIX CBANDIT Selection Probability for Normal Distributions

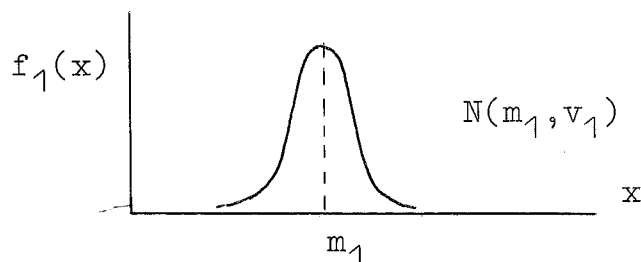
The BANDIT algorithm makes a decision on the basis of a set of samples; one from each of the 'value' probability densities of each of the alternatives involved. The probability that any particular alternative will be taken is not explicitly calculated. There is some interest in knowing this probability while observing the performance of the BANDIT algorithm and it may be calculated in two ways:

1. By repeated application of the BANDIT algorithm the probability of selecting each of the possible alternatives can be assessed by counting the proportion of the total decisions assigned to each alternative (Monte Carlo method).
2. By direct calculation from the (assumed known) analytic form of the 'value' probability densities. This method is useful as a check against method 1, and in obtaining more accurate results with less computation.

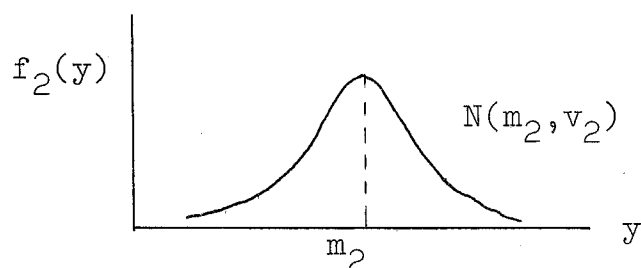
This appendix is concerned with the details of method 2 for the case of Normal distributions for the 'value' probability densities.

First consider the choice between two normal probability densities

Alternative A1



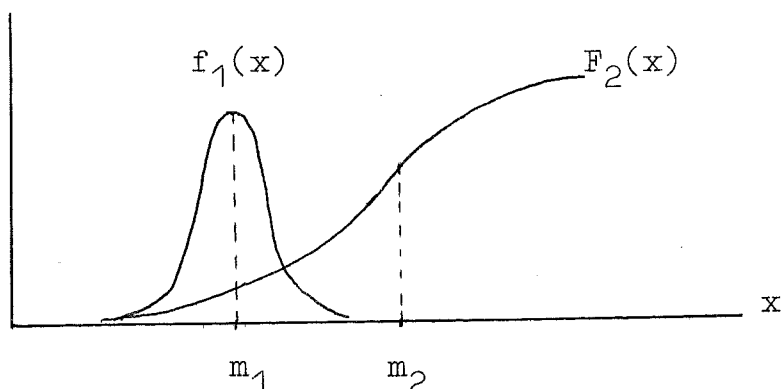
Alternative A2



For the BANDIT algorithm, selecting for a maximum:

$$\begin{aligned} \Pr(A1) &= \int_{-\infty}^{\infty} f_1(x) \cdot \int_{-\infty}^x f_2(y) \cdot dy \cdot dx \\ &= \int_{-\infty}^{\infty} f_1(x) \cdot F_2(x) \cdot dx \end{aligned}$$

where $F_2(x)$ is the cumulative distribution for $f_2(x)$.



$$\begin{aligned}
\Pr(A1) &= \int_{-\infty}^{\infty} \frac{1}{v_1 (2\pi)^{\frac{1}{2}}} \exp(-(x-m_1)^2/2v_1^2) \cdot \\
&\quad \cdot \int_{-\infty}^x \frac{1}{v_2 (2\pi)^{\frac{1}{2}}} \exp(-(y-m_2)^2/2v_2^2) dy \cdot dx \\
&= \int_{-\infty}^{\infty} \frac{1}{v_1 (2\pi)^{\frac{1}{2}}} \exp(-(x-m_1)^2/2v_1^2) \cdot \\
&\quad \cdot \frac{1}{2}(1 + \operatorname{erf}((x-m_2)/v_2 2^{\frac{1}{2}})) \cdot dx
\end{aligned}$$

where $\operatorname{erf}(x)$ is the error function*.

Now substitute,

$$t = (x-m_1)/v_1, \quad dt = dx/v_1.$$

$$\Pr(A1) = \int_{-\infty}^{\infty} \frac{1}{(2\pi)^{\frac{1}{2}}} \exp(-t^2/2) \left[\frac{1}{2}(1 + \operatorname{erf}\left\{\frac{t \cdot v_1 + m_1 - m_2}{v_2 2^{\frac{1}{2}}}\right\}) \right] \cdot dt$$

now $\operatorname{erfc}(x) = 1 - \operatorname{erf}(x)$

$$\Pr(A1) = \int_{-\infty}^{\infty} \frac{1}{(2\pi)^{\frac{1}{2}}} \exp(-t^2/2) \left[\frac{1}{2} \operatorname{erfc}\left\{\frac{t \cdot v_1 + m_1 - m_2}{v_2 2^{\frac{1}{2}}}\right\} \right] \cdot dt$$

By extension, for N alternatives:

$$\Pr(Ai) = \int_{-\infty}^{\infty} \frac{1}{(2\pi)^{\frac{1}{2}}} \exp(-t^2/2) \left[\prod_{\substack{j=1, N \\ j \neq i}} \frac{1}{2} \operatorname{erf}\left\{\frac{t \cdot v_i + m_i - m_j}{v_j \cdot 2^{\frac{1}{2}}}\right\} \right] \cdot dt$$

*Handbook of Mathematical Functions. Ed. M. Abramowitz and I.A. Segun. Dover 1964, p.298.

APPENDIX D

THE BANDIT ALGORITHM IN HEURISTIC

SEARCH ALGORITHMS

APPENDIX D

This appendix shows how the BANDIT algorithm may be embedded into a heuristic graph search algorithm to allow for updating of the heuristic function for information gained by traversal of a previously found path (the 'on-line' path finding problem of Chapter 2).

The Heuristic Path Algorithm (HPA), Pohl 1969 [3], is used here because it is typical of heuristic graph search algorithms, and particularly similar to the Graph Traverser of Doran and Michie 1966 [1], and the A* algorithm of Hart, Nilsson and Raphael 1967 [2].

The problem space for heuristic search is a directed graph G , which is a set of nodes X and edges E (arcs in Chapter 2) which are ordered pairs from the node set.

$$G: \quad X = \{x_1, x_2, \dots, x_n\}$$

$$E = \{(x_i, x_j) \mid x_i, x_j \in X, x_j \in \Gamma(x_i)\}$$

Γ is the successor mapping. In using directed graphs to characterize problem domains the node x_i has a data structure that specifies a state of the problem. The mapping $\Gamma(x_i)$ represents the set of possible states (nodes) resulting from one move or operator applied to state (node) x_i .

The heuristic function $h(x)$ is a measure of the estimated distance from node x to the goal node. For a transportation problem the heuristic function may be the distance from location (node) x to the desired destination via a direct line rather than the true (unknown) distance.

First the HPA algorithm by itself:

s = start node, t = terminal node (goal).

$g(x)$ = the number of edges from s to x , as found in the search.

$h(x)$ = an estimate of the number of edges between x and t , the heuristic function.

$f(x) = (1-w)g(x) + w.h(x) \quad 0 \leq w < 1.$

S = set of nodes already visited. Also called the expanded nodes.

S' = set of nodes directly reachable (in one edge) from S , also called the candidate nodes.

1. Place s in S and calculate $\Gamma(s)$ placing them in S' .
If $x \in \Gamma(s)$ then $g(x) = 1$ and
 $f(x) = (1-w) + w.h(x).$
2. Select $n \in S'$ such that $f(n)$ is a minimum.
3. Place n in S and $\Gamma(n)$ in S' (if not already in S')
and calculate f for the successors of n ($\Gamma(n)$).
If $x \in \Gamma(n)$ and $x \notin S$ then $g(x) = g(n) + 1$ and
 $f(x) = (1-w).g(x) + w.h(x).$
4. If n is the goal state then halt, otherwise go to step 2.

.....

The HPA algorithm builds a tree; as each node is reached a pointer to its predecessor is maintained. Upon termination the solution path is traced back from the goal node through each predecessor.

The Graph Traverser [1] is similar to the HPA if only $h(x)$ and not $g(x)$ is used ($W=1$); while the A* algorithm [2] uses $f = g + h$ ($w=\frac{1}{2}$).

After a path has been found it is traversed, and at this stage the $h(x)$ for nodes along the traversed path can be updated with observations of their value on this traversal of this particular path.

The update process can be written

$$q(h(x)) \leftarrow \text{UPDATE } (q(h(x)), h(x))$$

where $q(h(x))$ = the probability density distribution for
the heuristic function h of node x .

$h'(x)$ = the observed value of $h(x)$ on a
traversal.

BANDIT-HPA

1. Place s in S and calculate $\Gamma(s)$ placing them in S' .
If $x \in \Gamma(s)$ then $g(x) = 1$ and
 $f(x) = (1-w) + w.\text{sample}(q(h(x)))$.
2. Select $n \in S'$ such that $f(n)$ is a minimum.
3. Place n in S and $\Gamma(n)$ in S' (if not already in S')
and calculate f for the successors of n ($\Gamma(n)$).
If $x \in \Gamma(n)$ and $x \notin S$ then $g(x) = g(n) + 1$ and
 $f(x) = (1-w).g(x) + w.\text{sample}(q(h(x)))$.
4. If n is not the goal node go to step 2.
5. Trace back to find the solution path, and then
traverse this path.
6. During the traversal -
$$q(h(x)) \leftarrow \text{UPDATE } (q(h(x)), h'(x))$$
7. At the end of the traversal go to step 1 to prepare
for the next traversal.

.....

REFERENCES

- 1 Doran, J.E. and Michie, D. Experiments with the
 Graph Traverser program. Proc. R. Soc. A, 294,
 1966, pp 235-259.
- 2 Hart, P., Nilsson, N. and Raphael, B. A Formal
 Basis for the Heuristic Determination of Minimum
 Cost Paths. IEEE Trans. Sys. Sci. Cybernetics,
 July 1968, SSC-4, No.2, pp 100-107.
- 3 Pohl, I. First Results on the Effect of Error in
 Heuristic Search. Machine Intelligence 5,
 Edinburgh University Press 1969, pp 219-236.