

[This is a copy of the original Report UC-DSE/1(1972), which was the first of a series later given the International Standard Serial Number ISSN 1188. Any additions to or comments on the original wording are enclosed in square brackets [...]. Every care has been taken in the copying so as to obtain a word-perfect copy with identical page numbering and contents, but freedom has been taken with typeface, layout, and minor format details of diagrams. Copy made by the editor, John H. Andreae, who may be contacted on jandreae@xtra.co.nz]

Man-Machine Studies

Progress Report UC-DSE/1(1972)

Department of Electrical Engineering
University of Canterbury, Christchurch[, New Zealand].

Ministry of Defence Research Contract No. 106

“Pattern Recognition Studies having Applications in Underwater Acoustics.

UNCLASSIFIED - UNLIMITED

Copyright

The editor and contributors reserve the right to publish some or all of the material in this Report at a later date in theses, journals etc.

No part of the Report may be copied without permission of the editor: J.H.Andreae.

MAN - MACHINE STUDIES GROUP

Department of Electrical Engineering
University of Canterbury,
Private Bag, Christchurch[,N.Z.].

PROGRESS REPORT UC-DSE/1(1972)

to the Defence Scientific Establishment on relevant work of the Man-Machine Studies Group under Ministry of Defence research Contract No. 106:-
“Pattern Recognition Studies having Applications in Underwater Acoustics.”

December 1972

Editor: - J.H.Andreae

Contributors:- J.G.Cleary
G.A.Cox
P.M.Heffernan
R.W.Platts
A.B.Robson
M.R.Sloane

Unclassified - Unlimited

0.0 Preface

In this, the first report of the Man-Machine Studies Group (MMSG) in the Department of Electrical Engineering at the University of Canterbury, the area of learning machines has been selected for detailed exposition. Other areas of study by the group will be mentioned briefly, but detailed expositions will be deferred to later reports that will follow at six-monthly intervals. [In fact, 40 reports would be issued over a period of 20 years, so this rate was maintained!]

Comments, criticism, discussion and suggestions will be most welcome from anyone. The MMSG holds small, informal, weekly discussions at which D.S.E. members, passing through Christchurch, would be most welcome. (Telephone: Christchurch 517-414).

0.1 Learning Machines

The description of work on learning machines is written for the reader without knowledge of the state-of-the-art in general and without knowledge of the work of the MMSG in particular. In this way it is hoped to interest members of D.S.E. and others in a research area which is fast becoming one of the major research fields of engineering and mathematics.

The problem tackled formally in the design of learning machines are primitive, theoretical examples of the more complex situations encountered in the system-modelling and computer-interaction projects of the group. This is further reason for describing the learning machine studies before considering the other applications of the man-machine approach.

0.2 CONTENTS

- 1.0 Introduction p.2..
 - 1.1 Linear System and SRA p.3.
 - 1.2 A Stochastic Environment p.4.
 - 1.3 A Bayes Perceptron p.8.
 - 1.4 Omitted!
 - 1.5 Heuristics p.10.
 - 1.6 Appendix for section 1. p.11.

- 2.0 The STELLA Learning Machine p.12.
 - 2.1 Competition for Space p.13.
 - 2.1.1-5 n-Ladder p.14.
 - 2.1.6-9 2^m-Ladder p.15.
 - 2.1.10 Conclusions p.17.
 - 2.2 Look and Act STELLA p.18.
 - 2.2.1 The Skeleton of a Proposal p.18.
 - 2.3 CLUMPS Predictor p.20.
 - 2.3.1 Definition of a CLUMPS Predictor p.20.
 - 2.3.2 Discussion p.22.
 - 2.4 TREEPLE Memory and String STELLA p.23.
 - 2.4.1 Illustration of TREEPLE Memory p.24.
 - 2.4.2 String STELLA p.26.
 - 2.5 Monologue p.27.
 - 2.5.1 Monologue STELLA as an FSM p.27.
 - 2.5.2 HOI Program for a simple STELLA p.30.
 - 2.5.2.1 Part 5 Decision from ST-I p.30.
 - 2.5.2.2 Part 9 Updating ST-I Control Policy p.33.
 - 2.5.3 HOI Program for Monologue STELLA p.35.
 - 2.5.3.1 Part 1 Supervisory part of program p.35.
 - 2.5.3.2 Part 7 Problem Environment, Reward and Print-Out p.36.
 - 2.5.3.3 Part 20 Memory Print-Out p.36.
 - 2.5.3.4 Computer Simulation Results p.37.
 - 2.6 A Predictor for STELLA p.43.
 - 2.6.1 Action Matrices p.43.
 - 2.6.2 String Correlator p.44.
 - 2.6.3 Modified Action Matrices p.45.
 - 2.6.4 Constructing an Automaton from Input/Output Behaviour p.45.
 - 2.6.4.1 Wrap Unwrap Network (WUN) p.46.
 - 2.6.4.2 WUN Storage p.48.
 - 2.6.4.3 WUN Heuristics p.49.
 - 2.6.4.4 How near did WUN get? p.51.
 - 2.6.4.5 Quintuple Monologue Predictor (QMP) p.52.
 - 2.6.4.6 Cleary's Compatibility Rules for Automaton States p.55.
 - 2.6.4.7 Predictor Using Slide and Strings (PUSS) p.57.
 - 2.6.4.7' PostScript to PUSS pp.64a-c. [At end of Contents in original.]
 - 2.6.5 Future Work on Predictors for STELLA p.65.

- 3.0 Miscellaneous Topics p.67.
 - 3.1 maxEman p.67.
 - 3.1.1 maxEman hypothesis p.67.
 - 3.1.2 maxEman by Platts p.69.
 - 3.1.3 A Bean and Bowl Task p.70.

- 4.0 References p.71.

1.0 INTRODUCTION

A “learning machine”, or LM for short, can be thought of as a man-made device which interacts with an unknown environment so as to learn about that environment, so as to obtain reward of some kind, and possibly to avoid punishment of some kind.

In order to leave the term learning machine with as broad a connotation as possible, we define a sub-class of learning machines, called STELLAs, with which we shall be particularly concerned.

Definition (1.1)

A STELLA is an automaton (P,A,R,S,f,g) operating in discrete time (t=0,1,2,...), comprising

- a set P of input patterns,
- a finite set A of output actions,
- a binary reward input R(t), 0 or 1,
- a finite set of states S,
- a function f which determines the next state $s' \in S$, given the current input pattern $p \in P$ and current state $s \in S$,
i.e. $s' = s(t+1) = f(p(t),s(t))$ or $f: P \times S \rightarrow S$
- a function g which selects an action $a \in A$. given the current input pattern $p \in P$ and current state $s \in S$,
i.e. $a = a(t) = g(p(t), s(t))$ or $g: P \times S \rightarrow A$,

and designed to increase the “frequency of reward R(t)”. The definition is obscure with respect to the meaning of “frequency of reward” beyond requiring that an “average frequency of reward” be measured by a sum

$$\sum_{t=m}^{m+N} a_{t-m}R(t)$$

for some m and N (>0) corresponding to the starting time and length of period over which the average is being measured, and for at least one of the (N+1) non-negative weights a_{t-m} being non-zero. The weights are not necessarily independent of time.

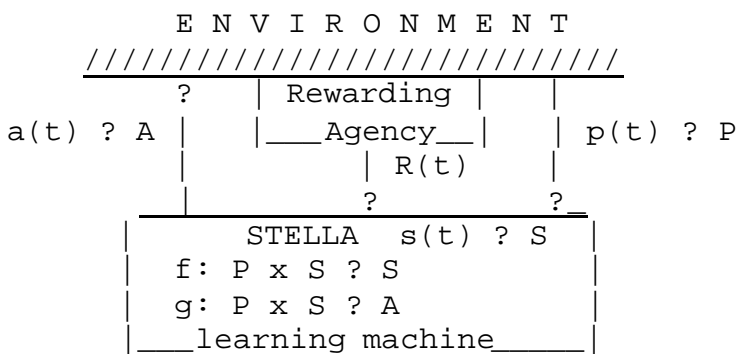


Figure 1.1

A simpler and less formal definition of a STELLA learning machine, which is almost as general as definition (1.1), was given in (Andreae and Cashin, 1969):-

“The problem of the learning machine can be appreciated more readily if it is reformulated for a human problem-solver: *There is a row of lights in front of you (input pattern), each light being ON or OFF at any instant. Below the lights a row of push-buttons (actions) lie within your reach and these may be pressed, one at a time, in any sequence. There is nothing else but a single light (reward) below the buttons and your task is to cause this light to flash on as often as possible.*

That the input pattern is a binary word (row of lights) in this definition is more of a convenience than a restriction. In nearly all of the environments used in our simulations of the STELLA-environment situation, the input pattern p has been a binary word. (NB. The name STELLA was derived from the name of the laboratories in which early forms of the machine were devised.)

1.1 Linear System and SRA

A great deal depends upon the nature of the environment and upon how much can be assumed about the environment by the designer of the learning machine.

If, for example, the environment is known to be a linear system and reward is a quadratic performance index, then the full power of modern control theory can be applied without recourse to learning heuristics. (e.g. Athans, 1971).

For a second example, if the environment is a finite automaton with state-describing output (i.e. there is a one-to-one correspondence between the state of the environment and the input pattern to the learning machine), and if reward $R(t)$ is always 1 for a correct action and 0 for an incorrect action, then the learning machine can be [the] stimulus-response automaton (SRA) as described by Suppes (1969). A correct action is defined by a reward function r , which determines the correct action a^* for each environmental state (= SRA input pattern) p . Thus, correct $a(t) = a^* = r(p(t))$.

An appropriate design for the SRA is a table T of pattern-action pairs (p,a) , which is initially empty, and a set of heuristics H , where

- H1 : If $p(t)=p_i$ and $(p_i, a_i) \in T$ for some i , then $a(t) = a_i$;
- H2 : If $p(t) \notin p_i$ for all $(p_i, a_i) \in T$, then $a(t) = \text{RANDOM}(A)$;
- H3 : If $R(t)=1$, then $T(t) = T(t-1) \cup (p(t), a(t))$.

With this SRA, as $t \rightarrow \infty$,
$$S = \frac{1}{N+1} \sum_{t=t'-N}^{t'} R(t) \rightarrow 1.$$

In words, the heuristics say:-

- (1) If there is a pattern-action pair in the table such that the pattern is the same as the current input pattern, then the SRA should perform the action of that pair.
- (2) If not, the SRA action should be chosen at random.
- (3) If a new pattern-action combination is rewarded, add it to the table.

Since the table starts with no pairs in it, since only correct pairs can be added to it, since a correct pair is always added if it occurs, and since the random selection of actions guarantees that an appropriate action will eventually be found for every pattern which continues to be received, the table must eventually hold all the correct pairs necessary for the SRA to obtain reward on every step.

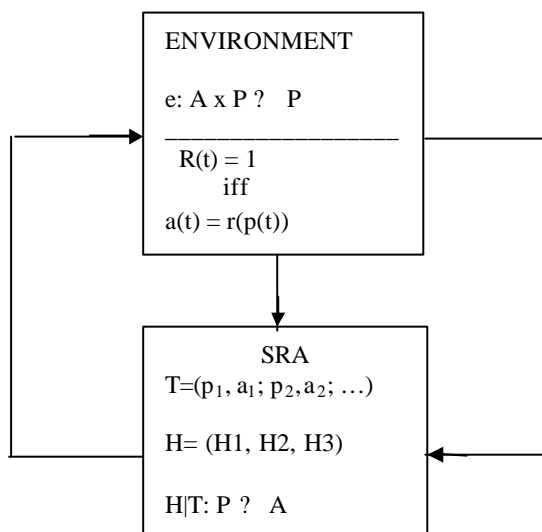


Figure 1.2

1.2 A Stochastic Environment

A third example of an environment for which a theoretically optimal controller can be prescribed is a Markov stochastic environment with Howard's (1960) policy improvement procedure (PIP). This may be seen as a stochastic version of the SRA problem in which the environment mapping e becomes a collection of stochastic matrices.

The computational situation considered by Howard is converted into a learning problem as follows:-

- (i) There are $|P|$ input patterns $p = 1, 2, 3, \dots, |P|$,
 $|A|$ actions $a = 1, 2, 3, \dots, |A|$,
 $|P|$ values v_i $i = 1, 2, 3, \dots, |P|$,
 and a discount factor β $0 < \beta < 1$.

(ii) A control policy, or table, T comprises a list of actions, one for each possible input pattern: $T = (a_1, a_2, a_3, \dots, a_{|P|})$

(iii) A 3-dimensional matrix E has elements e_{ijk} which are estimates of the probabilities e_{ijk} that input pattern $p(t) = i$ followed by action $a(t) = k$ will be followed by input pattern $p(t+1) = j$.

(iv) A reward matrix R has elements r_{ik} which are estimates of the probabilities r_{ik} that input pattern $p(t) = i$ followed by action $a(t) = k$ will earn reward $R(t) = 1$.

(v) There is a training sequence during which actions $a(t)$ are chosen at random and elements of the matrices E and R are estimated from the actual sequences of input patterns $p(t)$ and rewards $R(t)$. See Appendix.

(vi) At the end of the training sequence, Howard's PIP is initiated and a control policy is constructed iteratively:

PIP(1) Set values $v_i = 0$ ($i = 1, 2, 3, \dots, |P|$)

PIP(2) Choose actions for T at random. Call them $a_1^1, a_2^1, a_3^1, \dots, a_{|P|}^1$

PIP(3) Perform the following iteration until $a_i^n = a_i^{n+1}$ for all i on step $n+1$:-

PIP(4) Calculate a new set of values by solving the equations

$$v_i^n = r_{ix} + \beta \sum_{j=1}^{|P|} e_{ijx} v_j^n \quad [\text{where } x = a_i^n \text{ and } i = 1, 2, 3, \dots, |P|]$$

using the actions $[x =] a_i^n$ in the table.

PIP(5) Compute a new control policy by setting, for each input pattern i , the action a_i^n in the table equal to a_i^{n+1} , where a_i^{n+1} maximises over the set A the expression

$$r_{iy} + \beta \sum_{j=1}^{|P|} e_{ijy} v_j \quad [\text{where } y = a_i^{n+1}].$$

PIP(6) $n \leftarrow n+1$ and return to PIP(4) subject to PIP(3).

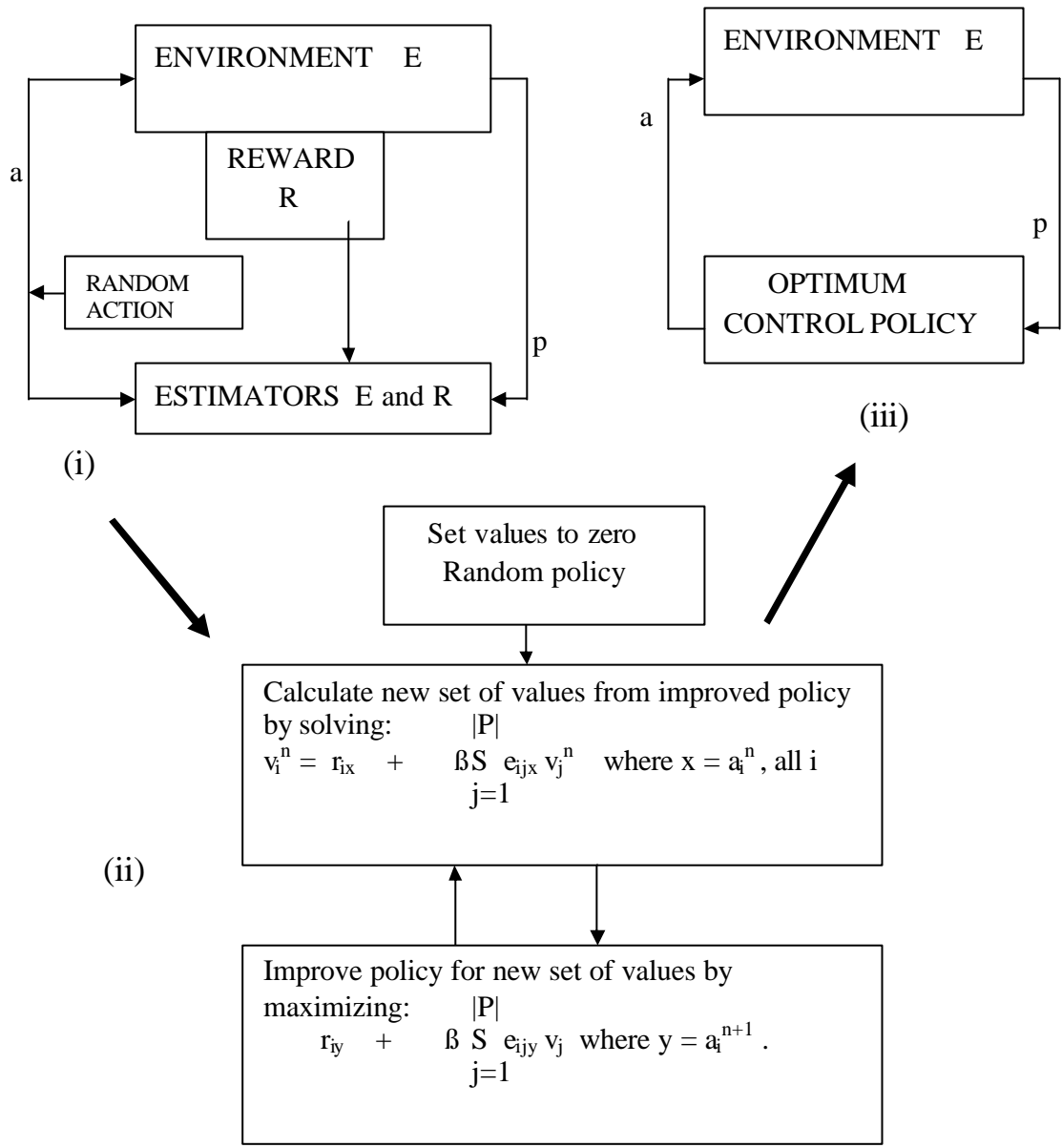


Figure 1.3 (i) Training sequence for estimators
 (ii) PIP computation of optimum policy
 (iii) Application of optimum policy

Though somewhat artificial, the training-sequence-policy-improvement-procedure (TS-PIP) arrangement illustrates a number of aspects of the general learning problem.

- (1) The learning system has to gather information as well as apply control. Stage (i) of the TS-PIP in Figure 1.3 can be looked upon as an information gathering stage during which the environment and the rewarding agency are modelled by the estimating matrices E and R. In stage (iii) this information is applied in the form of an optimum policy calculated in the brief computation stage (ii).
- (2) It is only the assumption of stationary environment and reward conditions (constant E and R) which allows the stages (i), (ii) and (iii) to be separated. If the environment or reward conditions were liable to change, then it would be useless to separate stages (i) and (iii) since the optimum policy would soon become suboptimum. In a non-stationary case, the three stages must be merged into one so that estimates are constantly being updated.
- (3) If estimators are being updated with information from a non-stationary environment, they need to be able to forget old information which is no longer relevant.
- (4) It is assumed in the Markov assumption of the TS-PIP scheme, that the states of the environment are numbered by the input patterns or, which is equivalent, that there is a one-to-one correspondence between states and patterns. To assume that a learning system is given the structure of its environment to this detail is unreasonable. Normal engineering design techniques would be more appropriate than a learning system for tackling such a well-understood problem environment.
- (5) The matrices E and R represent simple models of the environment and of the rewarding agency. It is likely in a more complicated situation that a learning system would have to learn different reward schedules in the same environment, i.e. it would have different tasks to learn in the same environment. Knowledge of the environment gained in learning one task should be useful in learning another task in the same environment.
- (6) The Markov assumption that the input patterns number the states of the environment can fail in two ways to hold in more general cases. The input pattern may over-specify or under-specify the environment. The former case is treated in the example of the perceptron, where many patterns correspond to the same condition, while the latter case is given special attention in our Monologue schemes.
- (7) Immediate reward is, in general, of more value than deferred reward. For each step that the reward is deferred its value is reduced by β , the discount factor. Thus, reward of value 1 now, has value β if we have to wait one step for it, β^2 if we have to wait 2 steps for it, and so on.
- (8) A feature of the TS-PIP scheme, which may be lost if stages (i) and (iii) are merged, is the way the environment is learned while the machine is exploring randomly. If the environment is learned while the machine is operating under some control policy, then it may be impossible for the machine to learn those aspects of the environment which would enable it to find a better policy. This emphasises the need for a learning machine to have periods of random exploration ("play") to maintain a broad view of its environment.

1.3 A Bayes Perceptron

The fourth example of a theoretical environment and optimum learning machine requires the LM to be a classifier of input patterns $p(t) \in P$, each action $a(t) \in A$ of the LM being rewarded ($R(t)=1$), if given in response to an input pattern from the appropriate class of patterns.

The Bayes Perceptron learning system (BPLS) outlined here is based on the four concepts of
independent evidence
conditional probability
maximum likelihood
stochastic estimation.

For a thorough and most illuminating treatment of Perceptrons, the reader is referred to Minsky and Papert (1969). Here it will suffice to follow a narrow path to our goal.

Suppose that the input patterns $p(t)$ from the environment to the BPLS are binary words of m bits, such that each bit describes by 1 or 0 the presence or absence of some feature or property of the environment. The input pattern can be expanded in terms of binary coefficients b_j which are 0 or 1 according to whether the corresponding bit of the input pattern is 0 or 1:-

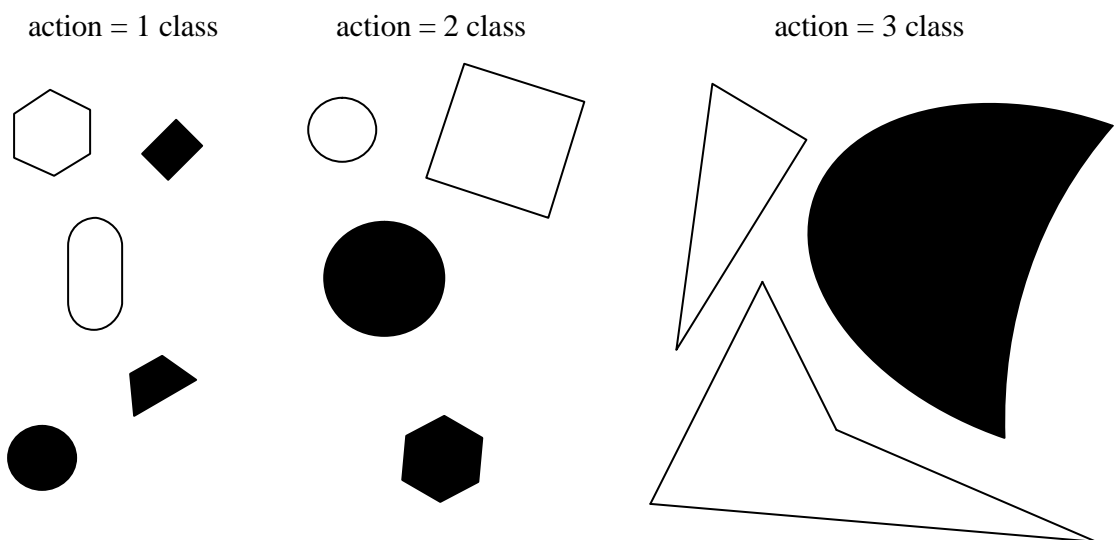
$$p(t) = b_0 + b_1 \cdot 2 + b_2 \cdot 2^2 + \dots + b_{m-1} \cdot 2^{m-1} = \sum_{j=0}^{m-1} b_j \cdot 2^j$$

Example. The input pattern classifies simple figures according to the four features

- large/small 1st bit
- all sides straight/not all 2nd bit
- dark/light 3rd bit
- symmetrical/asymmetrical 4th bit

Thus the input pattern 1010 would describe a large, dark, asymmetric pattern with not all sides straight. There might be three actions appropriate to the three classifications

- 1) small figures
- 2) symmetrical figures
- 3) asymmetric large figures.



The drastic assumption of independent evidence enables us to express the probability of an input pattern as the product of the probabilities of the individual features contained in it:-

$$\text{Probability of } p(t) : Pr_{p(t)} = \prod_{j=0}^{m-1} (Pr_j)^{b_j} \cdot (Pr_{\bar{j}})^{b_{\bar{j}}} \quad \text{INDEPENDENT EVIDENCE}$$

where $b_{\bar{j}} = 1 - b_j$ is 1 if the j^{th} feature is not present, and $Pr_{\bar{j}} = 1 - Pr_j$ is the probability that the j^{th} feature is not present.

Bayes' Rule gives the following relationship between conditional probabilities:-

$$Pr_{ij} \cdot Pr_j = Pr_{ji} \cdot Pr_i \quad \text{BAYES' RULE}$$

where Pr_{ij} is the probability of i given that feature j is present.

With the independent evidence condition and Bayes' Rule, an expression can be derived (Appendix) for the logarithm of the probabilities that the i^{th} action is and is not appropriate for input pattern $p(t)$:-

$$\log\left(\frac{Pr_{ip(t)} \cdot Pr_i}{Pr_{\bar{i}p(t)} \cdot Pr_{\bar{i}}}\right) = \sum_{j=0}^{m-1} (b_j \cdot \log\left(\frac{Pr_{ij} \cdot Pr_{\bar{i}}}{Pr_{\bar{i}j} \cdot Pr_i}\right) + (b_{\bar{j}} \cdot \log\left(\frac{Pr_{i\bar{j}} \cdot Pr_{\bar{i}}}{Pr_{\bar{i}\bar{j}} \cdot Pr_i}\right))$$

which is of the form

$$w_{ip} = \sum_{j=0}^{m-1} (b_j \cdot w_{ij} + b_{\bar{j}} \cdot w_{i\bar{j}})$$

According to the principle of maximum likelihood, we ought to choose the action i which gives the maximum w_{ip} , i.e. the action for which the ratio of the probabilities of action i being right and wrong is highest. However, since the probabilities from which the weights w_{ip} are calculated will be estimates in practice, we must choose the sub-maximum actions sometimes in order to improve the accuracy of the estimates. It could well be true that the maximum is a false one arising from the short time of estimating and that the true maximum will be found only when all of the probability estimates have been improved by further trials. Cashin (1970) has shown the desirability of maintaining estimators of estimator variances and has proposed the BANDIT algorithm to ensure effective convergence. If storage is limited, we are likely to be forced to use a less efficient but safe stochastic policy of choosing an action with probability proportional to its likelihood. In other words, the action estimated to be the most likely has the greatest chance of being selected.

A method for estimating the weight is given in the Appendix. As in the TS-PIP scheme, we can only consider the BPLS scheme to be optimal if we can provide a sufficiently long training sequence, during which actions are performed at random, for the accuracy of the estimates to be established.

1.5 Heuristics

In each of the theoretical examples given above, unrealistic assumptions had to be made about the environment for an optimal learning system to be formulated. In practice, it is of prime importance that we design systems which find solutions, regardless of whether those solutions are optimal in some sense. Very often it is impossible to define conditions for optimality, let alone devise a system which would meet them. This point appears implicitly in the definition (1.1) of a STELLA, where the values of the crucial parameters a_{t-m} had to be left open. Their values are a matter of choice, opinion or judgment. Thus there is no way of deciding in an absolute way whether a certain amount of reward soon is better than more reward later. Heuristic arguments can often be used to achieve a good compromise.

This does not mean that heuristics are necessarily vague, illogical, subjective or irrational. Usually they are good heuristics only if they are the opposite of these.

The word "heuristic" is commonly used both for the principle on which an algorithm or 'rule-of-thumb' is based and also for the algorithm itself. The practice is continued here, distinction being made only when necessary.

Some of the most powerful heuristic principles are the following:-

- a) Make use of all the information available.
- b) Try the heuristic rules on ideal situations.
- c) Good rules tolerate errors and noisy data.
- d) Look for situations in which the algorithm might get stuck.
- e) Check whether all the arbitrary parameters used are necessary.
- f) Test algorithms under a wide variety of conditions.
- g) Try to interpret heuristic algorithms as generalizations of theoretically optimal but restricted procedures.

The strategy for designing STELLA has been given (Andreae and Cashin, 1969) as the following set of heuristic rules:

- (i) Try out a design.
- (ii) Find classes of problem that it cannot tackle which, as humans, we feel it ought to be able to tackle.
- (iii) Modify the design so that it can tackle one or more of these additional classes of problem.
- (iv) With this new design, return to (i).

1.6 APPENDIX for Section 1

1.6.1 Estimation of probabilities (c.f. Section 1.2 (v))

Let $\hat{p}(t)$ be an estimate of the mean \bar{x} of a random sequence $x(t)$ of ones and zeroes. t is an integer representing discrete time at which samples of $x(t)$ are received.
EWPA estimator (exponentially-weighted-past-average: e.g. Brown, 1962)

$$\hat{p}(t+1) = (1 - b)\hat{p}(t) + b.x(t)$$

or

$$\hat{p}(t+1) = \hat{p}(t) + b.(x(t) - \hat{p}(t))$$

Notice that the average rate of increase of $\hat{p}(t)$ per time unit is amount of increase times probability of increase minus amount of decrease times probability of decrease, or $\bar{x} . \beta(1-p(t)) - (1 - \bar{x}) . \beta . \hat{p}(t)$, which equals zero when $\hat{p}(t) = \bar{x}$.

$\hat{p}(t)$ tends to rise when it is less than \bar{x} and it tends to fall when it is greater than \bar{x} .

For proper proofs of the convergence of $\hat{p}(t)$ on \bar{x} , see Minsky and Papert (1969) or Witten (1970), where its application to pattern recognition and perceptron problems is discussed at length.

1.6.2 Derivation of the BPLS formula of section 1.3

$$\begin{aligned} Pr_{i|p} . Pr_p &= Pr_{p|i} . Pr_i, \quad \text{where we are writing } p \text{ for } p(t). \\ &= Pr_i . \left(\prod_{j=0}^{m-1} (Pr_{ij})^{b_j} . (Pr_{\bar{j}i})^{b_{\bar{j}}} \right) \\ &= Pr_i . \left(\prod_{j=0}^{m-1} (Pr_{ij})^{b_j} . Pr_{\bar{j}/Pr_i} . (Pr_{\bar{j}i})^{b_{\bar{j}}} . Pr_{\bar{j}/Pr_i} \right) \\ &= Pr_i^{1-m} . Pr_p . \prod_{j=0}^{m-1} (Pr_{ij})^{b_j} . (Pr_{\bar{j}i})^{b_{\bar{j}}} \end{aligned}$$

Repeat with \bar{i} in place of i and divide the first equation by the second. Then take logarithms of each side to obtain the equation quoted in section 1.3, page 9.

B.R.Gaines' estimation procedure is used for the log probability ratios. For w estimating $\log(Pr_x/Pr_{\bar{x}})$, $x = (0,1)$, $0 < w < 1$

$$\begin{aligned} x = 0, w \geq 0.5: & w = w - a \\ x = 0, w < 0.5: & w = -bw^2 + w \\ x = 1, w > 0.5: & w = b(1-w)^2 + w \\ x = 1, w < 0.5: & w = w + a \end{aligned}$$

Typically, $a = 0.0125$ and $b = 0.045$ for a 16:1 range in $Pr_x/Pr_{\bar{x}}$.

2.0 The STELLA learning machine

A number of basic theoretical learning systems were described in section 1 as an introduction to the research which will be outlined in this section.

We have not been concerned recently with the computer simulation of complete learning systems. The relatively complete learning system STELLA was simulated extensively from 1963-1966 (Andreae, 1966; Gaines and Andreae, 1966). Modifications to the 1966 STELLA, which have been proposed (Andreae and Cashin, 1969) and are proposed here, require separate testing and development before being incorporated in a full system. The complete STELLA system is approaching a size for which computer simulation will be too slow and expensive. However, the building of a hardware model severely restricts development and innovation because of the difficulty of introducing significant changes in a partly-built model. Further, there is little justification for constructing hardware unless the result can be seen to be economically worthwhile in terms of practical application. The learning system is still too small and primitive to be applied to significant, real-world problems. We are forced to resort to computer simulation and theoretical analysis of partial structures and separate heuristics.

In the following sub-sections, we consider a number of structures and heuristics which are currently in question or under development. In particular,

- (i) Competition for space
- (ii) Look and Act STELLA
- (iii) Clumps predictor
- (iv) Treeple memory and String STELLA
- (v) Monologue

By far the greatest amount of work has been done on the Monologue concept, so the other four will be discussed briefly first.

2.1 Competition for Space

A significant improvement in the performance of STELLA was realized when “forgetting” in the original design (Andreae, 1963) was replaced by competitive storage. Forgetting was used to get rid of old information. The famous tortoises of Grey Walter (1951) gradually lost their conditioned reflexes as a result of charge leaking away from storage capacitors. Samuel (1959), in one of the most successful of all learning programs, used a clever method of forgetting in which usefulness was offset against age. Nevertheless, information was discarded when its age without use exceeded a certain threshold, regardless of demands on storage space.

Ross Ashby (1961) has discussed the dangers of over-writing earlier information with later information. It is suicidal, for instance, to replace earlier information by later information which actually depends upon that earlier information.

In STELLA, information is discarded only when space has to be found for new information, not yet tested. Since new information is always in a tentative state of having produced reward just once, it must be given a chance to show whether or not it is worth keeping. We are not interested in the impractical ideal of a machine with an elastic memory, so some existing information must be discarded to make room for the new information. STELLA employs three different parameters to assess the value of each ‘chunk’ of information in its store:

- (i) The expectation parameter assesses the relevance of the information.
- (ii) The age parameter assesses the reliability of the information in terms of the duration of its successful use.
- (iii) The success fraction parameter assesses the current usefulness of the information.

In this sub-section we are not concerned with the details of usage and interaction of these parameters. Instead, we shall look at two simple ways of using a limited amount of storage for holding learned information. Simple conclusions will be drawn from the exercise. In particular, it will be argued that the cost of using parameters like those listed above is only justified if the information which they preserve is sufficiently valuable and difficult to obtain. Otherwise the space which they occupy would be better used in storing more information.

In both of the storage methods considered below an environment is postulated which produces an input pattern $p(t)$ randomly from a set P at integer times t . Actions $a(t) \in A$ from the learning machine are rewarded with probability β if the pattern-action pair $(p(t), a(t))$ belongs to a set B of preferred responses. Otherwise the actions are rewarded with probability $(1 - \beta)$:-

$$R(t) = \begin{cases} \beta & \text{if } (p(t), a(t)) \in B \text{ (preferred)} \\ 1 - \beta & \text{if } (p(t), a(t)) \in P \times A - B, \quad \frac{1}{2} \leq \beta \leq 1 \end{cases}$$

Each pattern-action pair (p, a) requires a binary word location of m bits for storage.

2.1.1 n-Ladder

Pattern-action pairs are stored on a "ladder" of n ordered m-bit locations numbered 1,2,3,...n.

2.1.2 n-Ladder Heuristics

If p(t) is in a pair stored on the n-ladder, perform the associated action;
 If the action is rewarded and the pair is below the top of the ladder, exchange the pair with the next pair up the ladder;
 If the action is not rewarded and the pair is not at the bottom of the ladder, exchange the pair with the next pair down the ladder;
 If p(t) is not in a pair stored on the n-ladder, perform an action at random;
 If the action is rewarded, exchange the pattern and random action for the pair at the bottom of the ladder.

2.1.3 n-Ladder Case (1) $\beta=1$

Any pattern-action pair is either always rewarded or never rewarded. Only preferred responses (i.e. rewarded pattern-action pairs) can be stored on the ladder. The n-ladder will gradually fill with preferred pairs. When full, preferred pairs can only be exchanged for preferred pairs. The n-ladder is optimal.

2.1.4 n-Ladder Case (2) $\frac{1}{2} = \beta < 1, n = |B| = |P|, P? B$ is 1-to-1

Each pattern has one and only one preferred action.

Let p_i be the probability of a preferred pair being stored on the i^{th} rung of the ladder. We consider the probability that a preferred pair on the $(i+1)^{th}$ rung is exchanged for an unpreferred pair on the i^{th} rung, namely

$$(1/|P|) p_{i+1} (1 - p_i) (1 - \beta + 1 - \beta),$$

and the probability that a preferred pair on the i^{th} rung is exchanged for an unpreferred pair on the $(i+1)^{th}$ rung:

$$(1/|P|) p_i (1 - p_{i+1}) (\beta + \beta),$$

A stationary statistical state will be reached when these probabilities are equal for all pairs of adjacent rungs:

$$p_{i+1} (1 - p_i) (1 - \beta) = p_i (1 - p_{i+1}) \beta$$

or

$$p_{i+1} / (1 - p_{i+1}) = (\beta / (1 - \beta)) p_i / (1 - p_i)$$

Thus all probabilities are given in terms of β and p_i , the probability of a preferred pair on the bottom rung. Since we have assumed the number of patterns $|P|$ to be equal to the number of rungs, n, the ladder will eventually fill with a response for every pattern. The proportion, a, of preferred responses will remain constant from then on. It follows that

$$\sum_{i=1}^n p_i = a.n$$

2.1.5 n-Ladder Case (3) $\frac{1}{2} = \beta < 1, n < |B| = |P|, P? B$ is 1-to-1

This is similar to the previous $n=|P|$ case, except that there will always be patterns ($|P| - n$ in number) for which responses are not stored. When one of these input patterns is generated by the environment, an action is selected at random. The possibility of the action being the preferred one for the pattern in question is $1/|A|$ (one out of the number of possible actions). Thus, the choice of a random action for a pattern not on the ladder can be treated as a “zero rung” of the ladder for which $p_0 = 1/(|A|)$.

A stationary statistical state will be reached for the ladder and the process of random selection of actions (rung 0) if the previous relations

$$p_{i+1} / (1 - p_{i+1}) = (\beta / (1 - \beta)) p_i / (1 - p_i)$$

hold for rungs 1 to n, and the average rates of transfer of preferred and unpreferred responses to and from the zero rung balance:

$$((|P| - n) / |P|) \cdot p_1 (1 - p_0) \cdot (1 - \beta) / \beta = ((|P| - n) / |P|) \cdot p_0 (1 - p_1)$$

or

$$p_1 / (1 - p_1) = (\beta / (1 - \beta)) p_0 / (1 - p_0) = (\beta / (1 - \beta)) \cdot 1 / (|A| - 1)$$

Example. $|A| = 8, \beta = 0.75: p_0 = 1/8; p_1 = 0.3; p_2 = 0.6; p_3 = 0.8; p_4 = 0.9; p_5 = 0.9; \dots$

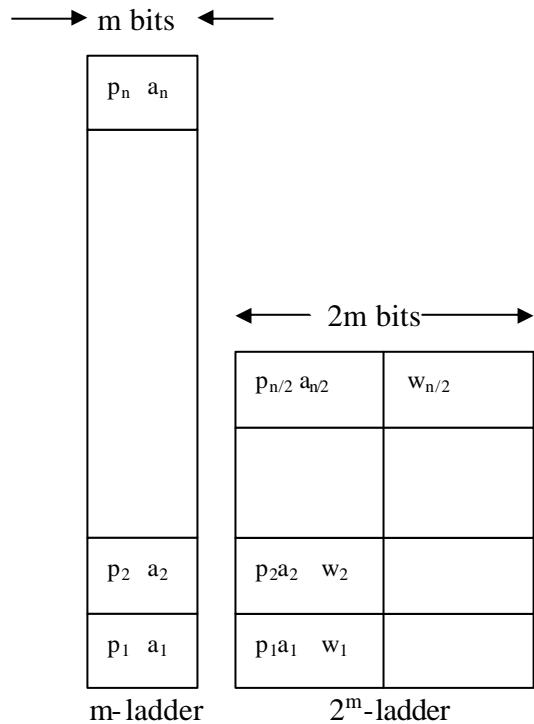
Therefore, under these conditions, there are, on average, only

$(1-0.3) + (1-0.6) + (1-0.8) + (1-0.9) + \dots \simeq 1.4$ unpreferred responses on the ladder, however long it is.

2.1.6 2^m -Ladder

Up to $n/2$ pattern-action pairs are stored on a ladder with 2^m ordered positions. This is the same as having $n/2$ pairs stored with $n/2$ associated weights, each m-bit weight giving the position of the pair on the 2^m ladder.

Notice that the amount of storage used by the 2^m -ladder is the same (mn bits) as for the n-ladder, so we can compare their efficiencies as stores directly.

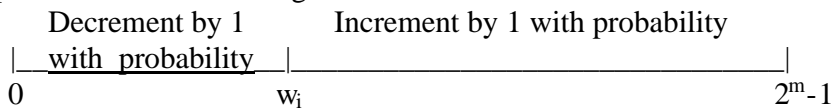


m-ladder

2^m -ladder

2.1.7 2^m -Ladder Heuristics(c.f. the ADDIE of Gaines (1967))

If $p(t)$ is in a pair (p_i, a_i) stored on the 2^m -ladder, perform the associated action a_i ;
 If the action is rewarded, increment the associated weight w_i by one with probability $(2^m - 1 - w_i)/(2^m - 1)$;
 (Remember that the values of the weights range from 0 to $2^m - 1$).
 If the action is not rewarded, decrement the associated weight w_i with probability $w_i/(2^m - 1)$;
 If $p(t)$ is not in a pair stored on the 2^m -ladder, perform an action at random:
 If the action is rewarded, exchange the pattern and action $(p(t), a(t))$ for the stored pair with the lowest weight.



2.1.8 2^m -Ladder Case (1) $\beta = 1$

As in the n -ladder case (2.1.3), the ladder fills with preferred pairs and stays full of preferred pairs.
 The n -ladder is clearly preferable since it holds twice as many pairs. The weights of the 2^m -ladder are redundant and represent waste space.

2.1.9 2^m -Ladder Case (2) $\frac{1}{2} = \beta < 1, n = |B| = |P|, P? B$ is 1-to-1

The 2^m -ladder heuristics ensure that the weights of preferred responses have mean (expected) values of $\beta(2^m - 1)$ and the weights of unpreferred responses have mean values of $(1 - \beta)(2^m - 1)$. They maintain these means with a variance $\beta(1 - \beta)2^m$. (Appendix and Gaines (1965)). This corresponds to a minimum standard deviation of $2^{(m/2) - 1}$.

The 2^m -ladder separates the preferred and unpreferred responses by two standard deviations when m is only 4 and β is no greater than $5/8$. Larger values of m and β will make the separation even more definite.

Example. $|P| = 64$ patterns, $|A| = 8$ actions, $m=6+3=9$ bits, $\beta=0.75$.

6 bits are required to store any one of the 64 different patterns and 3 bits are required to store any one of the 8 actions. Strictly, we should reserve a zero pattern to indicate a vacant location, but this is only needed during the filling up stage, so we forget it.

The weights of preferred responses will have a mean of $0.75(2^9-1)$. The weights of the unpreferred responses will have a mean of $0.25((2^9-1))$. The standard deviation will be the square root of $0.75(1-0.75) 2^9$. Computing these simple expressions

weight	mean	standard deviation
preferred responses	383	10
unpreferred responses	128	10

With much smaller standard deviations on the means, it is readily seen that the 2^m -ladder will reach a stationary statistical state with all but one of the responses stored as preferred responses. The remaining one will be in a continuous state of interchange with new randomly selected responses. The fraction of the new responses which will be preferred responses will be, on average,

$$(\beta/|A|)/((\beta/|A|) + ((|A|-1)/|A|)(1-\beta)) = 0.3$$

To compare the behaviour of the 2^m -ladder with the n -ladder, we recall the example in subsection 2.1.5 and repeat the figures given there together with corresponding figures for the 2^m -ladder. In each case, $n = |P| = 64$:-

	p_1	p_2	p_3	p_4	p_5	$p_{64} \sim 1$
n -ladder	0.33	0.60	0.82	0.93	0.975	$p_{64} \sim 1$
2^m -ladder	0.30	~ 1	~ 1	~ 1	~ 1	$p_{32} \sim 1$

The n -ladder is clearly superior since it stores on average nearly 32 more preferred responses. The n -ladder is sufficiently selective for the 2^m -ladder to gain a negligible half a preferred response while losing 32 preferred responses.

2.1.10 Conclusions

The sacrifice of storage space for weights is not justified on the basis of simple arguments relating to the proportion of preferred responses which are stored.

It may be inferred that the use of weights will only be justified when the probability of a preferred response being stored and recognized as a preferred response is so small that loss of preferred responses must be minimized. The virtues of the 2^m -ladder would show up in the situation where the value of some responses depended upon the presence of others. Thus STELLA must construct sequences of responses, or "paths to reward". In such cases the indiscriminate discarding of preferred responses by the n -ladder could be catastrophic.

A broader theory is needed to justify the use of weights.

2.2 Look and Act STELLA

There are three main reasons for wanting to complicate further the STELLA learning machine structure by separating the machine into “look” and “act” parts:-

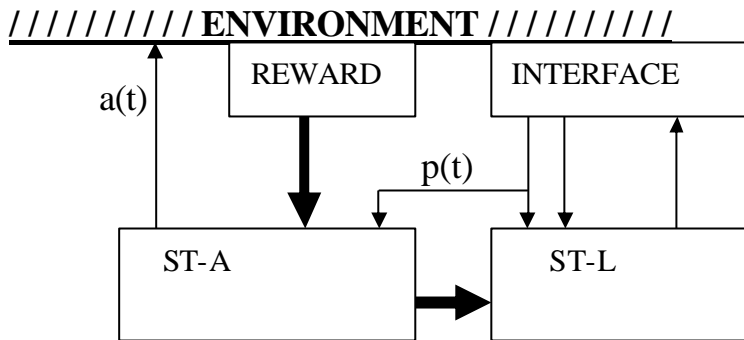
- (A) When the machine makes a successful decision on the basis of input pattern recognition and action selection, credit can be assigned to both processes. However, if the decision is unsuccessful, it is not known whether the pattern recognition or the action selection or both are to blame.
- To date, the problem has been evaded in STELLA by restricting the action selection procedures. The most elaborate form of action synthesis used in STELLA produced action sequences by means of “post-actions” (Andreae, 1965). Even with this minor elaboration, action synthesis had to be sufficiently slow in comparison with the pattern recognition process for the latter to be well established before the former appeared.
- The “look” and “act” separation is intended primarily to resolve the credit assignment problem between pattern recognition and action synthesis.
- (B) Look and Act correspond roughly to state estimation and optimal control in the linear systems control problem. Even in the linear system, where they are related by Kalman’s (1960) duality principle, state estimation and optimal control remain as two separate and essential tasks.
- State estimation assumes that the actions are given, while the optimal control problem assumes that the input patterns (in STELLA terminology) are given. In the Look and Act partition we go farther and allow a subset of actions to be performed in the information-gathering process of looking. This is, perhaps, more analogous to Feldbaum (1960)’s Dual Control Theory, in which he attempts to optimize the division of effort between information gathering and productive output.
- (C) The importance of the Look function is emphasised in our human experience by the elaborate mechanisms of eye, ear and hand with which we are endowed (e.g. Noton, 1971). Little is known of the processes involved in the synthesis of muscular actions, but the pianist, for example, illustrates the complexity and precision which can be achieved.

2.2.1 The Skeleton of a Proposal

The Look and Act parts of STELLA will be called STELLAment-L and STELLAment-A, or just ST-L and ST-A, respectively.

ST-A receives external reward, the reward of STELLA as a whole.

ST-L receives reward from ST-A, when the latter recognizes an input pattern and associates with it an increase of expectation, i.e. ST-A makes a successful step.



LOOK and ACT STELLA

ST-L receives the input pattern from an interface (e.g. a controllable eye) together with proprioceptive information from the interface.

ST-A receives the input pattern from the interface.

ST-L's actions transform the input pattern received by acting on the interface (e.g. controlling eye muscles).

ST-A's actions act on the environment to effect a rewardable task.

Time sequence of events.

.....

ST-A performs a single action.

A new input pattern $p(t)$ is received.

ST-L performs actions changing $p(t)$ into $p'(t)$ into $p''(t)$

until either ST-A recognizes one of the transformed $p(t)$
or ST-L has done one random action.

ST-A performs a single action.

....

....

A Problem Environment

It is planned to try out the Look/Act partition with the French Military Game problem, for which we already have several simple STELLA programs. ST-L would rotate and reflect the board positions so that ST-A could take advantage of the symmetries of the board without having to learn them.

2.3 CLUMPS Predictor

The most difficult part of a learning machine to design is the adaptive model with which the machine represents its view of the environment. For such a model to start with no information at all about the environment and to grow so as to be able to represent a dynamic environment containing moving and possible purposeful entities like itself, the structure must be more flexible than anything which has been considered so far. It remains a marvel how the human brain achieves this, although year by year evidence accumulates to show that a great deal more of the human is pre-programmed than was expected. We may yet find that our world view is predominantly determined by genetic coding. Be this as it may, the design strategy of STELLA requires that the environment model or "predictor" should be able to start *carte blanche*.

The original predictor of STELLA was a set of correlation matrices (Andreae, 1963), one for each action. These were later given an adaptive logic ability called the String Correlator. The main trouble with the String Correlator, as with many other model synthesis techniques, is that they can easily "blow up" into an unmanageable size requiring more storage than is available. Because of this, an alternative idea was proposed in 1965, called the CLUMPS predictor, to make best use of strictly limited storage (memory) space. It was the subject of a patent application in 1966, but nothing has been heard of that since 1968 when it was making its way through the British Patent Office! The results of a hand simulation were given in an internal memorandum (Andreae, 1968). A Report by I.H.Witten (1970) takes up the idea and subjects it to scrutiny, extension and computer simulation.

Here we shall do no more than define the concept in preparation for further work.

2.3.1 Definition of a CLUMPS Predictor

A CLUMPS Predictor comprises n CLUMPS, 4 operations and a set of heuristic rules:-

If c_k is a CLUMP, $c_k \in C$, the set of numbered CLUMPS and k is the number or index of the CLUMP;

If $p(t)$ is the input pattern at time t , then $p(t) \in P$, the set of input patterns or input space.

If $a(t)$ is the action at time t , then $a(t) \in A$, the set of actions.

The i^{th} CLUMP, c_i , comprises a distance function $d_i(p)$ and a list, L_i , of triples; the j^{th} triple, t_{ij} , on the list L_i comprises an action, $a_{ij} \in A$, a CLUMP number, k_{ij} , and a transition probability estimator, u_{ij} ; thus $t_{ij} = (a_{ij}, k_{ij}, u_{ij})$; triples are updated in standard fashion according to the CLUMP to CLUMP transitions which occur;

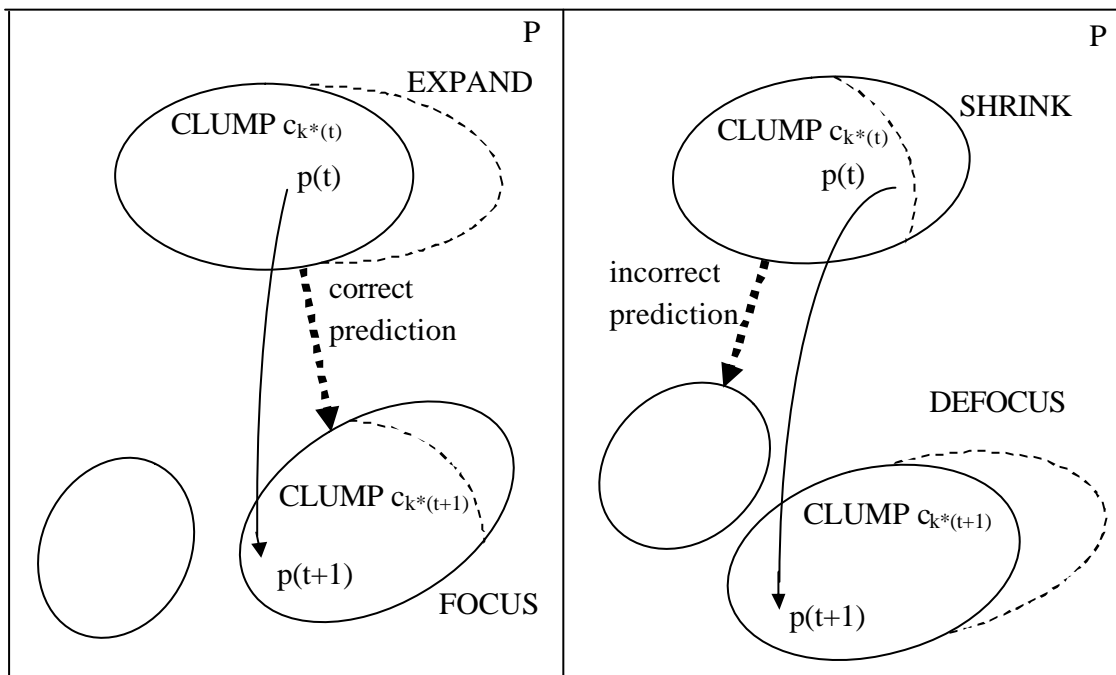
Each pattern $p \in P$ is "nearest" to a CLUMP c_k such that

$$d_k(p) < d_j(p) \quad \text{for all } j \in k;$$

FOCUS(d,p), DEFOCUS(d,p), EXPAND(d,p) and SHRINK(d,p) are four operations which have the effect of broadening and narrowing the distance functions $d(\bullet)$ of CLUMPs, acting as evidence for predictions and as predictions, with respect to a pattern p. Examples of such functions are given in Andreae (1968) and Witten (1970);

The heuristic rules, summarized in the table below, apply to two CLUMPs, numbered $k^*(t)$ and $k^*(t+1)$, which are the nearest to the consecutive input patterns $p(t)$ and $p(t+1)$; the asterisk * is used to denote nearest, i.e. $k^*(t) = \min_k d_k(p(t))$.

	New CLUMP Prediction, if correct	Last CLUMP Evidence for prediction
Correct Prediction $k^*(t+1) = (k_{k^*(t),j} a_{k^*(t),j} = a(t))$	FOCUS($d_{k^*(t+1)}, p(t+1)$)	EXPAND($d_{k^*(t)}, p(t)$)
Incorrect Prediction $k^*(t+1) ? (k_{k^*(t),j} a_{k^*(t),j} = a(t))$	DEFOCUS($d_{k^*(t+1)}, p(t+1)$)	SHRINK($d_{k^*(t)}, p(t)$)



2.3.2 Discussion

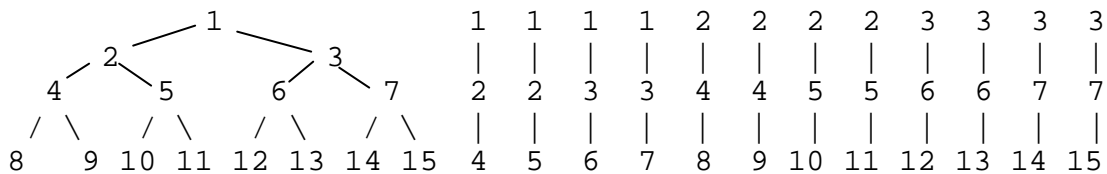
The CLUMPS predictor is a limited storage arrangement intended to make the best use of the available storage space. It is formulated as a Markov process via a fixed number, n , of states, the CLUMPs. As a predictor, these "states" have to apply to every pattern of the input pattern space even though n may be much smaller than $|P|$, the number of patterns. As a limited storage device, there is need for some process to optimize adaptively the use of the storage. In the predictor the sizes of the pattern subsets covered by each CLUMP are determined by a dynamic equilibrium between the demands of evidence and prediction. When successful, each CLUMP is first a prediction and the evidence for the next prediction. Broad evidence and narrow predictions are desirable, so the two conflicting demands are set against each other. The CLUMP sizes are continually being extended and restricted as a result of the operations embodied in the CLUMP heuristics.

2.4 TREEPLE Memory and String STELLA

There has always been criticism of the way STELLA adds new policy elements (i.e. pattern-action pairs) one at a time. This means that if a sequence of steps (pattern followed by action) earns reward, only the last step is remembered, which seems wasteful. However, as argued in Gaines (1966), "storage of policy elements one at a time favours shorter paths and, by on-line iteration of this procedure, STELLA builds up near-optimal paths terminating in reward."

Since the single step is the unit of learning in STELLA, it is important to ensure the relevance and reliability of this unit, the policy element. The three weight parameters mentioned in section 2.1 are designed for this purpose. Also transition indicators and estimating probabilities are provided to hold sequential information when this is necessary. Here we consider an alternative way of storing the sequential and branching information needed to describe trees of paths to reward.

Branching information can be stored implicitly in straight sequences. Thus, triple step sequences are sufficient to represent the branching of a binary tree with distinguished nodes:-

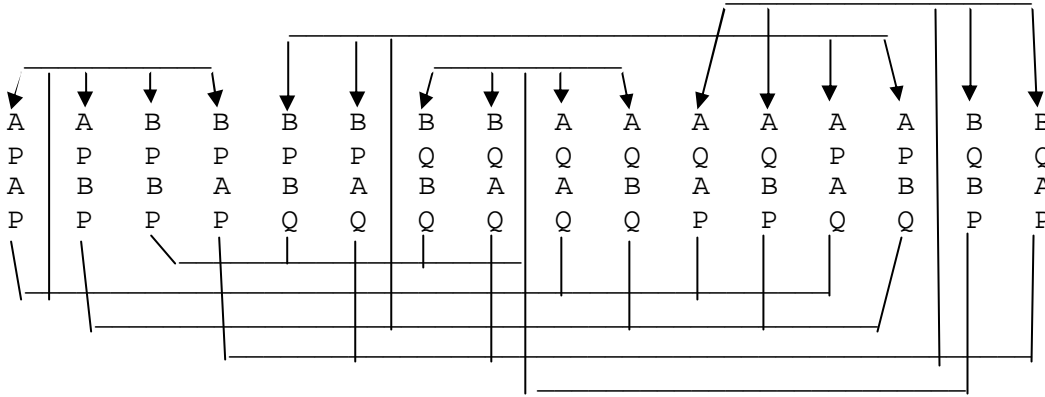


There are three other pieces of evidence which seem to favour short sequences for the unit of learning:-

- A. There is a considerable amount of evidence (Miller, 1967) for the span of human short term memory being about seven entities. Treating short term memory as a temporary store for sequences going into more permanent storage, we can treat this as evidence suggestive of the size of the unit of learning---sequences of length less than seven if the entities are received sequentially.
- B. The human cortex is a large convoluted sheet with a thickness of about 50 neurons in roughly 6 layers. Activity takes place mainly through the thickness of the sheet, suggesting a large parallel store of short sequences. Detailed suggestions of how the cortical structure might be related to the STELLA structure have been given (Andreae, 1962, 1965).
- C. Computer simulation of STELLA with a variety of problem environments showed that she was less successful at learning long paths to reward than expected. This is explained by saying that a chain of many unit steps is more likely to have a weak link than a chain of fewer multiple steps (sequences).

There is another way of looking at the TREEPLE Memory concept. If the input pattern from the environment does not by itself determine the state of the environment, then a sequence of input patterns will be needed in order to observe the environmental state. The SRA of section 1.1 demonstrated the advantages of state-determining patterns.

The same information can be given in a form more reminiscent of the structure of the cortex:-



The second 8-state automaton environment is not described satisfactorily by double pattern-action pairs as can be seen by the excess of transitions in this description:-

- a) APAP --> a, b, c, d, i, l, m
- b) APBP --> a, b, d, m, o, p
- c) BPBP --> a, b, d, m, o, p
- d) BPAP --> a, b, c, d, i, l, m
- e) BPBQ --> e, f, g, h, m, n
- f) BPAQ --> c, e, f, i, j, n
- g) BQBQ --> c, e, f, h, i, j, k, o
- h) BQAQ --> g, h, j, k, m, n, o
- i) AQAQ --> g, h, j, k, m, n, o
- j) AQBQ --> c, e, f, h, i, j, k, o
- k) AQAP --> a, b, g, k, l, o, p
- l) AQBQ --> c, d, g, i, k, l, p
- m) APAQ --> c, e, f, i, j, n
- n) APBQ --> e, f, g, h, m, n
- o) BQBP --> c, d, g, i, k, l, p
- p) BQAP --> a, b, g, k, l, o, p

This excess of transitions is due to the fact that the double pattern-action pairs are not state-determining, while in the case of the first 8-state automaton they were. For the second automaton triples resolve the ambiguities but, of course, the number required increases.

Both of the 8-state automata used have symmetries which allow simpler descriptions than would have been necessary in the worst case. On the other hand, in real life environments it is unlikely that the input patterns would be as uninformative as the set P. It is possible, therefore, that the degree of sequential complexity involved in these 8-state automata is representative of most real life problems.

Consider now the representation of the first 8-state automaton by triples without associated transitions. So long as we require matching of double pattern-action pairs at all times, the triples will determine the allowed transitions without specially stored transition information.

A A A A B B B B B B B B B B B B A A A A A A A A A A A A B B B B
P P P P P P P P P P P P P Q Q Q Q Q Q Q Q Q Q Q Q P P P P Q Q Q Q
A A B B B B A A B B A A B B A A A A B B A A B B A A B B B B A A
P P P P P P P P Q Q Q Q Q Q Q Q Q Q Q Q P P P P Q Q Q Q P P P P
A B A B A B A B A B A B A B A B A B A B A B A B A B A B A B
P P P P Q Q Q Q Q Q Q Q Q Q Q Q P P P P P P P P P P P P Q Q Q Q

Exercise: Use this TREEPLE Memory to predict the next input pattern at each step of the behavioural sequence given at the beginning of the section.

This is as far as we can go at the present time to support the TREEPLE Memory proposal.

2.4.2 String STELLA

Following suggestions of John Cleary, a program was written in the TRAC string-processing language for a STELLA in which each learning unit or policy element is an unlimited string of behaviour. With suitable heuristics, not yet perfected, the strings should attain stable lengths according to the demands of the environment. Thus, a short string has wider applicability but less logical power, and vice-versa. A balance analogous to the CLUMP size balance is envisaged.

An n-ladder type of storage was employed in the program because of the arithmetical limitations of our TRAC implementation. It is planned to change this to a 2^m -ladder type of storage with weights for estimating expectation, age and success fraction for each string. This is unlikely to be attempted until John Cleary has perfected his Eleusis player, which uses elaborate string-transforming functions. (Note. J.G.Cleary is currently completing an M.Sc. thesis on theorem-proving, so there will be no report on his work this time.)

2.5 Monologue

The first detailed proposal for the addition of Monologue to STELLA was made in a letter to Dr Hans Zschirnt of the Air Force Cambridge Research Laboratories (Andreae, 1966'). Reference was made to the concept and also to the alternative possibility of a Note-book in Andreae (1969'). The first publication with detailed illustrations of the idea was made with the help of P.M.Cashin in Andreae and Cashin (1969). It was not until this year, however, that time was found to subject the proposal to computer simulation.

Peter Heffernan first pointed out that Monologue could be looked upon as representing the state of STELLA. Since other views of Monolgue have been given in the earlier references, we shall use the state point of view to introduce Monologue here.

It was shown in section 1.1 that a learning machine could be a simple SRA (stimulus response automaton) if the input patterns described the states of the environment unambiguously. When the input patterns are not state-describing, the learning machine must have the additional ability of an FSM (finite state machine) in order to "observe" the states of the environment, like the Luenberger (1964) observer for linear systems. A simple SRA does not have the sequential complexity for this.

To show that Monologue converts STELLA from being an adaptive SRA to being an adaptive FSM is the purpose of the next subsection. It should be pointed out that a broader and more versatile role is planned for Monologue than that indicated by the FSM equivalence. For instance, the possible decoupling of the Monologue STELLAment from the main STELLAment was discussed carefully in Andreae (1966').

2.5.1 Monologue STELLA as an FSM (finite state machine)

First we recall the SRA from subsection 1.1 and re-present Figure 1.2 in a form suitable for our purposes:-

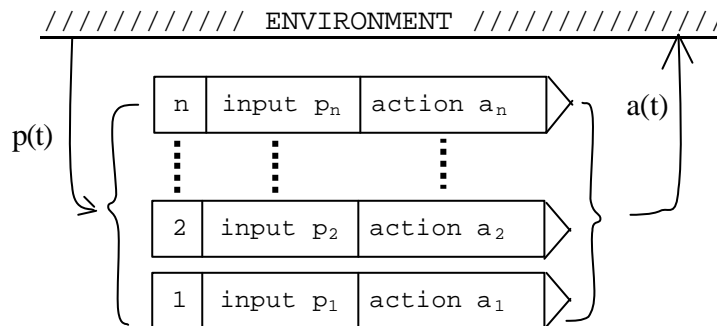


Figure 2.5.1 SRA or simple STELLA

Secondly, the FSM is distinguished by means of a diagram of the same form:-

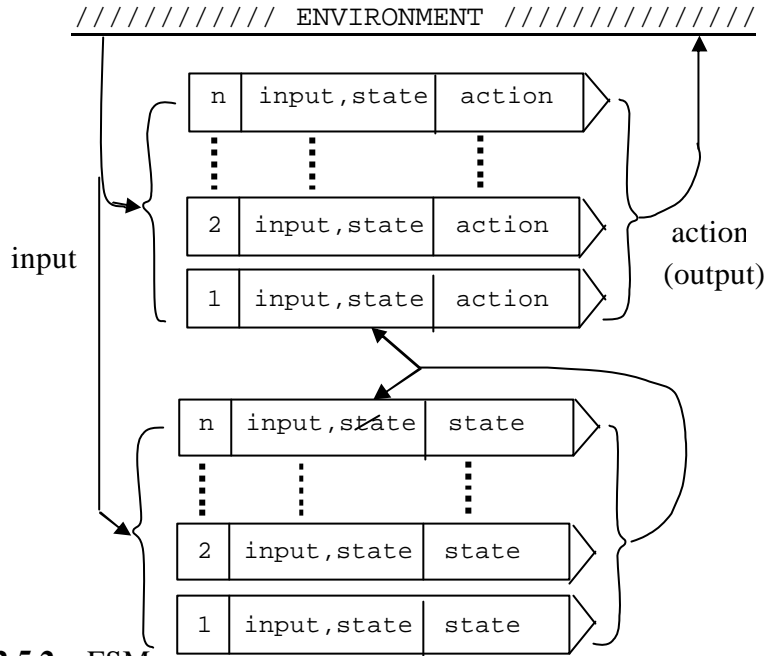


Figure 2.5.2 FSM

As a reminder, the FSM is a quintuple (P,Q,A,d,?). We have been using the set of inputs (or input patterns) P and the set of outputs (or actions) A. Q is the set of states. ? and d are two functions which map input-state pairs into the next action and next state, respectively:-

$$\begin{aligned} \text{Output function,} & \quad ? : P \times Q \rightarrow A \\ \text{Next state function,} & \quad d : P \times Q \rightarrow Q \end{aligned}$$

In Figure 2.5.2 the two parts of the FSM can be seen to correspond to the output function ? (above) and to the next-state function d (below). Notice the duplication of input-state pairs in the two parts and, also, how the state (i.e. the next state) is fed back to both parts from the output of the lower part.

The FSM can be looked upon as two separate SRAs coupled by output feedback from the lower one. This is sufficient to convert the SRA, which is no more than a “conditioned reflex” machine, into an FSM, which can respond to the present and past behaviour of the environment with the maximum variety possible to a finite machine.

It can now be shown how Monologue is intended to do the same for a simple STELLA, which like the SRA is no more than an elaborate conditioned reflex machine endowed with learning. The simple STELLA is assumed not to have transition information and a predictor.

Thirdly, then, we can draw a diagram to show how two STELLAments are coupled to form a Monologue STELLA. Some of the duplication apparent in our diagram of the FSM disappears in the Monologue arrangement because the lower or Monologue STELLAment (ST-J) receives the number of the last policy element (PE) used by the main STELLAment (ST-I) instead of the input pattern from the environment. This provides it with as much information but allows for more flexibility in the way of decoupling between the STELLAments.

We are not yet in the position to talk about optimal arrangements for Monologue STELLA, but the one presented seems to make sense from several points of view. For instance, monologue can be looked upon as augmenting the input of the main STELLAment to make its input state-determining; in this case there would seem to be an analogy between ST-J and a Luenberger observer for automata. For a second example, ST-J can be seen as a means of holding important information about past environmental behaviour for use by ST-I; in this case, it provides a kind of note-book facility.

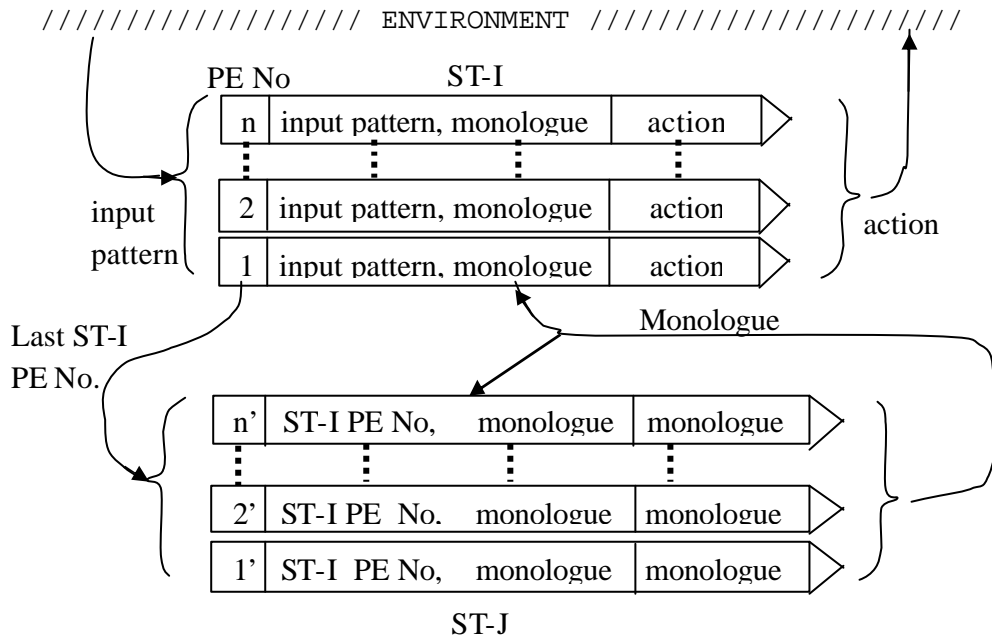


Figure 2.5.3 Monologue STELLA

No mention has been made of reward. Both STELLAments receive reward together, but since they operate alternately, reward is assumed to be given after ST-I's action and before the next monologue from ST-J.

2.5.2 HOI Program for a simple STELLA

The two simple STELLAments used in the simulated Monologue STELLA are separate parts of an HOI (Hybrid Operations Interpreter) interactive language program. The parts 5 and 9 for ST-I and parts 6 and 10 for ST-J differ only in the substitution of I or J in the names of variables. In this subsection, parts 5 and 9 of the HOI program are discussed in detail. Parts 2 and 3 which initialize the variables and parameters are of little importance to a machine which depends so little upon its starting state. Some of the effect of parts 2 and 3 will be mentioned while discussing parts 5 and 9. Part 1 is a supervisory part which controls the sequence of operations between the two STELLAments. It will be discussed with the other small linking parts of the full program.

Part 5 effects the choice of an action by ST-I.

Part 9 effects the updating of the memory (control policy) of ST-I.

2.5.2.1 Part 5 Decision from ST-I

```
5.100)RA,L;ACTI=IP(1+RAND*NAI)
5.200)LPEI=CPEI,CPEI=0
5.300)K=0,SPE=0,SXP=0
5.310)K=K+1,K>NI?5.4.
5.320)QI[K]<=0?5.31.
5.330)PEI[K]<>PTNI?5.31.
5.340)XP=EXI[K]*(1-RAND+RAND*SFI[K],RA,L;
5.350)SXP<XP?SPE=K,SXP=XP
5.360)5.31.
5.400)SPE==0?5.9.
5.500)RA,L;SXP<RAND+NI?5.9.
5.600)CPEI=SPE,ACTI=AI[CPEI]
5.700)K=1,1,NI!QI[K]=>1?QI[K]=QI[K]+1
5.800)QI[CPEI]=1,EXIT;
5.900)K=1,1,NI!CHI[K]=1,QI[K]>0?QI[K]=0.5
5.910)K=1,1,NI!QI[K]==0?2.91;2.92;"---I",K:
```

Explanation

5.100

All instructions in an HOI program are numbered to allow easy insertion of additional instructions at any time. The number of this instruction is 100 in part 5. RA,L; calls an Assembler subroutine which updates the random number generator producing RAND in the range 0<RAND<1.

ACTI holds the action selected by ST-I. It is the purpose of part 5 to choose a new integer value for ACTI between 1 and NAI, the number of actions allowed ST-I.

In case the control policy of ST-I cannot recommend a choice of action, this instruction puts a new integer part, IP(), of 1+Random Number x NAI.

5.200

The current policy element selected now becomes the last as we prepare to take another step forward.

5.300

Initialize to zero the index K, the selected policy element SPE and the selected expectation product SXP.

These are three temporary variables to be used in the following computing loop.

5.310

Increment the index K by one. If K is larger than NI, the number of policy elements in ST-I, then jump out of the loop to instruction 5.400.

In the HOI language ; means execute, : means print out, . means jump, and ! means iterate.

5.320

Each policy element in ST-I has a queue index QI[].

Values of the queue index have the following significance:-

-1 if the policy element is empty

0 if the policy element is tentatively stored, but not yet confirmed by reward.

.5 if properly stored but not used since the last unsuccessful step, i.e. ready to be used.

n if n successful steps since the policy element was used.

If the policy element PEI[K] is only tentatively stored then this instruction causes it to be skipped since it cannot be used until confirmed by reward. Also skips if it is empty.

5.330

If the pattern, PEI[K], stored in the Kth policy element is different from the input pattern, PTNI, for this step, then the policy element is inappropriate and is skipped.

5.340

Compute the product of the expectation, EXI[K], of the Kth policy element and (1-random number times (1-the success fraction SFI[K] of the Kth policy element)). In this way, a policy element with a high success fraction is sure to be credited with most of its expectation in the selection process, but one with a lower success fraction is likely to be penalized. Update the random number by linking to the subroutine RA.

5.350

Is the computed expectation product, XP, larger than the best so far, SXP? If so, make the new value the best and record the number of the policy element responsible in SPE.

5.360

Jmp back to the beginning of the computing loop.

5.400

If no policy element has produced an expectation product, then exit with a random action via instructions 5.900 and 5.910, which deal with unsuccessful steps.

5.500

The same applies if the best expectation product is below a random limit corresponding to the expectation of the last policy element in a chain of all NI policy elements, where each has an expectation equal to RAND times that of the next one up the chain. This is a heuristic, random, worst-case measure.

5.600

The current selected policy element of ST-I, CPEI, is put equal to the result, SPE, of the stochastic selection process. The action ACTI of ST-I becomes the action of the policy element. (c.f. the heuristic H1 of the SRA in subsection 1.1)

5.700

This instruction updates the step count in the queue index of all properly stored policy elements which have been used since the last unsuccessful step.

5.800

The queue index is set to one for the currently selected policy element.

5.900

Another parameter of each policy element of ST-I is the change index CHI[K].

Values of the change index have the following significance:-

- 1 if no change is intended
- <1 if it holds an expectation tentatively.

This instruction following a failure of the control policy of ST-I to select an action, cancels all tentative changes in expectation waiting for confirmation by reward.

5.910

The other consequence of an unsuccessful step is the deletion of tentatively held policy elements, waiting for confirmation by reward. The instructions 2.910 and 2.920 in part 2 of the program delete the Kth policy element of ST-I. After this deletion there is a print out of the message ---I,K , where K is a number, to record the fact that the Kth policy element of ST-I has been deleted.

The main effect of part 5 of the program is to select the action of the matching policy element with the largest expectation product.

The main steps are:-

- Select an action at random
- ?
- Find the largest expectation product of policy elements which are properly stored and have a pattern matching the input pattern
- ?
- If no policy element is appropriate or the expectation product is too low, exit ? with a random action after cancelling tentatively held expectations and policy elements
- ?
- A policy element and action is selected for the current step.
- Queue indices are updated.
- ?

2.5.2.2 Part 9 Updating ST-I Control Policy

In view of the longer length of this part of the program, it will be taken in sections.

```

9.100)REWD=0?9.3      If no reward, go to instruction 9.300
9.110)K=0             Set counter to zero for computing loop
9.120)K=K+1,K>NI?9.18.  Jump out of loop if K exceeds NI
9.130)CHI[K]<1?QI[K]>1?9.7;9.8; Tentatives confirmed
9.140)CHI[K]<1?QI[K]==0?EXI[K]=INAG*CHI[K] store new expectation
9.150)QI[K]>0?QI[K]=0.5,CHI[K]=1      store properly
9.160)9.12.          Return to beginning of loop

```

INAG is an initial value for the age weights in ST-I. It is usually set to 0.5.

The symbol == should be read as "is it equal to", in contrast to the usual assignment meaning of a single "=".

```

9.180)CPEI>0?EXI[CPEI]=1-AGI[CPEI]*(1-EXI[CPEI]),9.3.
9.200)K=1,1,NI!PEI[K]==PTNI?AI[K]==ACTI?CPEI=K,9.18.

```

If a policy element was selected for this step, then increment its expectation because of the reward received.

If no policy element was selected but one did have a matching pattern and had the action which was selected, then make this policy element the selected policy element and return to instruction 9.180 to update its expectation.

```

9.210)K=1,1,NI!PEI[K]==0?SPE=K,9.25.  Select the first vacant PE
9.220)X=1,SPE=0             Find PE with least expectation
9.230)K=1,1,NI!Y=EXI[K]*SFI[K],Y<K?SPE=K,X=Y  success fraction
9.250)CPEI=SPE,"***",SFI,CPEI:      product and output message.

```

A new PE to record the rewarded step is now stored in the PE which was vacant or had the smallest expectation x success fraction.

```

9.260)PEI[SPE]=PTNI,AI[SPE]=ACTI      Store pattern and action
9.270)EXI[SPE]=INEX,AGI[SPE]=INAG,SFI[SPE]=INSF

```

INEX, INAG and INSF are initial values for expectation, age and success fraction.

```

9.300)CPEI==0?LPEI==0?EXIT;          No PE selected this time or last
9.310)CPEI==0?SXFL=0,9.34.          PE selected last step, but not this
9.320)LPEI==0?9.9.                  PE selected this step but not last

```

If no PE was selected this step nor last, no updating is needed.

If PE was selected on last step, but not this step, the last PE needs to be weakened.

If PE was selected this step but not last, then a new PE needs to be stored tentatively.

If none of these three possibilities apply, a PE was selected last time and this, so updating depends on whether the step was successful or not.

9.330) SXFL=1, EXI[LPEI]=>EXI[CPEI]?REWD==0?LRWD==0?SXFL=0
 9.340) SXFL==0?EXI[LPEI]=AGI[LPEI]*EXI[LPEI], 9.6.

A step is successful (SXFL=1) if (a) Reward is received, (b) Reward was received last time and a PE is selected this step, or (c) expectation has increased. If the step is unsuccessful, the expectation of the last PE to be selected is decremented and tentatively held PE's are deleted. Tentative changes of expectation are cancelled.

9.380) K=LPEI, EXI[CPEI]<=EXI[K]?9.45. If expectation has not fallen, hold
 9.400) CHI[K]=EXI[CPEI], REWD==0?9.45. tentative expectation.
 9.420) QI[K]=2, 9.7; QI[K]=0.5 Not tentative if reward.
 9.450) 9.8; Update age.

If reward has been received, then instruction 9.420 temporarily sets QI[K]=2 so that when instruction 9.700 is executed the new expectation will be calculated as though the successful step were one step ago. The QI[K] is set to 0.5 immediately afterwards. This is a programming convenience only.

In the following two instructions all success fractions are decremented, except that of the successful policy element.

9.500) K=1, 1, NI!SFI[K]=AGI[K]*SFI[K]
 9.510) SFI[LPEI]=SFI[LPEI]+1-AGI[LPEI], EXIT;
 9.600) 5.9; 5.91; EXIT; Execute instructions in part 5
 9.700) EXI[K]=EXI[K]+(1-AGI[K])*(CHI[K]*(QI[K]-1)/QI[K]-EXI[K])

This last instruction affects a tentatively held expectation, taking into account the number of steps since it was reserved.

9.800) AGI[K]=1/(2-AGI[K]) Update age

The next 7 instructions store a new policy element in a manner analogous to the instructions 9.200 to 9.270.

9.900) K=1, 1, NI!PEI[K]==LPTI?AI[K]==LAI?LPEI=K, 9.33.
 9.910) K=1, 1, NI!PEI[K]==0?SPE=K, 9.95.
 9.920) X=1, SPE=0
 9.930) K=1, 1, NI!Y=EXI[K]*SFI[K], Y<X?SPE=K, X=Y
 9.950) LPEI=SPE. "***", SFI, LPEI:
 9.960) PEI[SPE]=LPTI, AI[SPE]=LAI
 9.970) CHI[SPE]=EXI[CPEI], AGI[SPE]=INAG, SFI[SPE]=INSF
 9.980) QI[SPE]=0, REWD==1?EXI[SPE]=INAG*CHI[SPE], QI[SPE]=0.5

In the last instruction, the queue index is set to 0 to indicate tentative storage, unless the new policy element is being stored because of reward.

2.5.3 HOI Program for Monologue STELLA

To complete the description of a computer simulation program for Monologue STELLA, we have to

- (1) Outline the main steps of the supervisory part 1.
- (2) Remember that parts 6 and 10 for ST-J are identical to the parts 5 and 9 for ST-I, except for the substitution of J for I in variable names.
- (3) Describe the problem and reward part 7.
- (4) Discuss some results from the program.

2.5.3.1 Part 1 Supervisory part of program

The first few instructions are concerned with housekeeping and initialization. The main computing cycle is as follows:-

```

1.600) LAJ=ACTJ , LPTJ=PTNJ           Update ST-J action and input pattern
1.602) CPEI==0?PTNJ=MONL , 1.61.     If no PE selected by ST-I.
1.604) PTNJ=CPEI+MONL                Form ST-J pattern from ST-I PE and monologue
1.610) 6 ;                           Execute part 6: ST-J decision
1.620) MONL=0.1*ACTJ , LAI=ACTI , LPTI=PTNI , PTNI=CELL+MONL   CELL is the
1.630) 5 ;                           Execute part 5: ST-I decision   input from environment
1.640) 7 ;                           Execute part 7: Test for reward and print results.
1.650) 9 ;                           Execute part 9: Update ST-I
1.660) LPEI>0?PTNJ=LPEI+FP (PTNJ)   ST-I PE stored during updating?
1.670) 10 ;                          Execute part 10: Update ST-J
1.690) STEP=>RUN?STEP , RUN:RUN+HALT; Control of run length
1.700) $SSW?20 ;                    If sense-switch depressed, print memory (part 20)
1.800) $SSW?HALT ;                 If sense-switch depressed, halt for alterations.
1.900) 1.6 .                        Return to beginning of main computing cycle.

```

Notice the order:-

- ST-J decision
- ?
- ST-I decision
- ?
- Reward?
- ?
- Update ST-I
- ?
- Update ST-J

Input patterns:-

	ST-I	ST-J
Integer part:	CELL from environment	Last PE selected by ST-I
Fractional part:	0.1 x Monologue action for both	

(NB. It is assumed for convenience that fewer than 10 Monologue actions are employed.)

2.5.3.2 Part 7 Problem Environment, Reward and Print-out

It will soon become apparent why such a simple problem environment is being described. We started with more complex and more significant problem environments, but the results did not demonstrate the limitation of Monologue with the clarity which the simple environment does.

The problem is no more than the following:-

The input pattern from the environment (CELL) is 1 or 2.

We shall find it convenient in diagrams to write P and Q in place of 1 and 2.

The actions of ST-I are 1 and 2. We shall find it convenient in diagrams to write A and B in place of 1 and 2.

The Monologue actions of ST-J are 1,2,3,..6. We shall find it convenient in diagrams to write M1,M2,...,M6 for the Monologue actions.

Each STELLAment is allowed 6 policy elements.

Reward is given if, after input pattern P, ST-I's action differs from the last action, and if, after input pattern Q, ST-I's action is the same as the last action.

(NB. A simple STELLA could tackle this problem easily if she were shown her last action in her input pattern. This was discussed in Andreae (1966'))

7.110)STEP=STEP+1	Increment step counter
7.120)\$NI,\$FI,STEP,ACTI,ACTJ,\$FD,PTNI,PTNJ:	Print step, actions and patterns
7.130)CPEI>0?CPEJ>0?\$FI,CPEI,CPEJ:7.2.	Print PEs selected
7.140)CPEI>0?\$FI,CPEI:7.2.	by ST-I only
7.150)CPEJ>0?\$FI,CPEJ:	by ST-J only
7.200)RA,L;	Update random number RAND
7.300)LCLL=CELL	Update contents of last cell
7.400)CELL=1+IP(2*RAND)	New random value for cell, 1 or 2.
7.500)LRWD=REWD,REWD=0	Preserve whether reward last step or not
7.600)LCLL==2?ACTI<>LAI?7.9.	No reward condition
7.700)LCLL==1?ACTI==LAI?7.9.	No reward condition
7.800)REWD=1,"REWARD":	Set reward and print message
7.900)"*",\$FI,CELL:	Print new contents of cell.

2.5.3.3 Part 20 Memory Print-Out

20.100)"STI",\$NT,\$FI,STEP:	Print step count
20.200)K=1,1,NI!20.6;20.61;	Iterate through each ST-I PE
20.300)"STJ":	Print "STJ"
20.400)K=1,1,NJ!20.7;20.71;	Iterate through each ST-J PE
20.500):1.5;EXIT;	Print heading and exit
20.600)\$NI,\$FI,K,AI[K],\$FD,PEI[K]:	Print PE No., action, pattern
20.610)\$NI,EXI[K],AGI[K],SFI[K]:	Print expectation, age, success fraction
20.700)\$NI,\$FI,K,AJ[K],\$FD,PEJ[K]:	Print PE No., action, pattern
20.710)\$NI,EXJ[K],AGJ[K],SFJ[K]:	Print expectation, age, success fraction

2.5.3.4 Computer Simulation Results

The following results are taken from a run dated 25-10-1972. Other results will be given in diagram from from earlier runs.

Initial values:- CELL=2, NI=6, NAI=2, NJ=6, NAJ=4, INEX=1, JNEX=1, INAG=.5, JNAG=.5, INSF1, JNSF=1.

First few steps with empty memory:-

STEP	ACTI	ACTJ	PTNI	PTNJ	
1	2	4	2.4	.4	Remember ST-I actually follows ST-J
	CELL=1				
2	2	2	1.2	.4	
	CELL=2				
3	2	2	2.2	.2	
	REWARD CELL=1				
	***, CPEI = 1				Policy elements stored as a
	***, LPEI = 2				result of reward
	***, CPEJ = 1				
4	2	4	1.4	1.2	
	CELL=1				
5	2	2	1.2	.4	
	CPEI = 2				ST-I PE selected
	CELL=1				
	***, LPEI = 3				Tentative PE
6	2	2	1.2	2.2	
	CPEI = 2, CPEJ = 1				ST-I and ST-J select PEs
	CELL=1				
	---I, K=3				Tentative PE abandoned
	***, LPEJ = 2				
7	2	2	1.2	2.2	
	CPEI = 2, CPEJ = 1				
	CELL=2				
	---J, K=2				
8	2	2	2.2	2.2	
	CPEI = 1, CPEJ = 1				
	REWARD CELL=2				Sense-switch depressed
	STI, 8				to cause memory print-out.
	1	2	2.2		
	.75	.5	.5		
	2	2	1.2		K AI[K] PEI[K]
	.125	.667	1.0		EXI[K] AGI[K] SFI[K]
		
	STJ,				
	1	2	2.2		K AJ[K] PEJ[K]
	.625	.667	1.0		EXJ[K] AGJ[K] SFJ[K]
	2	0	0		
	0	0	0		
		

The gradual learning of the task is illustrated by the change in memory over the next 700 or so steps. The memory will be given in a more compact form so that it can be shown for more stages:

K	AI	PEI	EXI	AGI	SFI	AJ	PEJ	EXJ	AGJ	SFJ	step
1	1	2.2	.458	.750	.056	2	2.2	.181	.667	.005	31
2	2	1.2	.426	.750	.032	2	3.1	1.00	.750	.053	
3	1	2.1	.533	.833	.236	1	1.2	1.00	.667	.059	
4	1	1.3	.736	.857	.294	3	5.1	1.00	.800	.254	
5	2	1.1	.833	.750	.185	3	4.3	.667	.800	.245	
6	1	2.3	.833	.800	.304	1	6.3	1.00	.800	.390	
1	2	2.4	.336	.667	.333	2	2.1	.500	.500	.500	69
2	2	2.1	.500	.500	1.00	2	3.2	1.00	.500	1.00	
3	2	1.2	.375	.500	1.00	3	1.4	1.00	.667	.296	
4	1	1.3	.576	.955	.411	2	5.2	.250	.500	1.00	
5	2	2.2	.750	.500	1.00	3	4.3	.596	.933	.378	
6	1	2.3	.840	.900	.221	1	6.3	1.00	.909	.272	
1	1	2.1			.002	3	3.4				step 100
2	1	1.1				2	6.2				
3	2	2.4			.004	1	2.1				
4	1	1.3				2	5.2				
5	2	2.2	.961	.917	.530	3	4.3				
6	2	1.2				2	4.3				
1	1	1.2				2	1.2				step 194
2	1	1.1			.000	2	6.2				
3	2	2.4			.000	1	2.1			.000	
4	1	1.3				2	5.2				
5	2	2.2	.992	.984	.665	3	4.3				
6	2	1.2				2	4.3			.000	
1	1	1.2				2	1.2				step 280
2	1	1.1			.006	2	6.2				
3	2	2.1				2	4.4				
4	2	1.4				2	5.2				
5	2	2.2	.995	.990	.600	4	1.2				
6	2	1.2			.009	2	3.1				
1	1	1.3	.500	.500	1.00	4	1.3				step 323
2	2	2.4				2	2.4				
3	2	1.1				1	3.1				
4	1	1.2				2	5.2				
5	2	2.2	.986	.991	.510	3	4.2				
6	2	1.4	1.00	.500	1.00	0	0				
1	1	1.3	.733	.909	.188	4	1.3	.839	.875	.222	
2	1	2.3				3	2.3				step 352
3	1	2.4	1.00	.899	.303	4	2.3				
4	1	1.2				2	5.2				
5	2	2.2	.986	.991	.409	4	3.4	1.00	.857	.167	
6	2	1.4	.852	.875	.223	3	6.4	.604	.917	.298	

K	AI	PEI	EXI	AGI	SFI	AJ	PEJ	EXJ	AGJ	SFJ	step
1	1	1.3	.757	.975	.203	4	1.3	.953	.960	.200	434
2	1	2.3	.741	.963	.140	3	2.3	1.00	.938	.111	
3	1	2.4	1.00	.971	.426	4	2.3	1.00	.900	.009	
4	1	1.2	1.00	.800	.000	2	5.2	.878	.991	.199	
5	2	2.2	.986	.991	.195	4	3.4	1.00	.970	.413	
6	2	1.4	.962	.968	.236	3	6.4	.700	.979	.264	

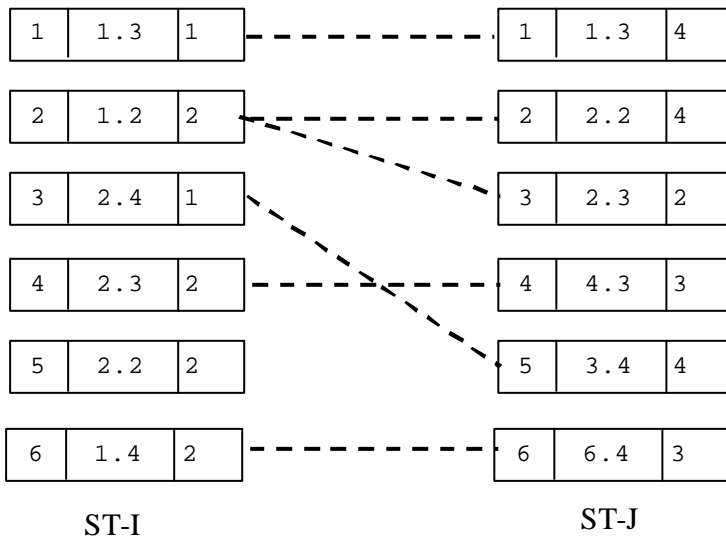
1	1	1.3				4	1.3				step 484
2	1	2.3				4	4.3				
3	1	2.4				4	2.3				
4	2	2.3	1.00	.857	.357	2	5.2				
5	2	2.2				4	3.4				
6	2	1.4				3	6.4				

1	1	1.3				4	1.3				step 512
2	2	1.2	1.00	.667	1.00	4	4.3				
3	1	2.4				2	2.3	1.00	.500	.500	
4	2	2.3	1.00	.909	.092	2	5.2				
5	2	2.2	.986	.991	.097	4	3.4				
6	2	1.4				3	6.4				

Every step rewarded after step 513.

											step
1	1	1.3	.808	.993	.255	4	1.3	.987	.989	.269	722
2	2	1.2	1.00	.750	.000	4	2.2	.245	.667	.000	
3	1	2.4	1.00	.990	.227	2	2.3	.500	.500	.000	
4	2	2.3	1.00	.984	.240	3	4.3	1.00	.981	.249	
5	2	2.2	.986	.991	.014	4	3.4	1.00	.990	.230	
6	2	1.4	.980	.991	.275	3	6.4	.788	.994	.272	

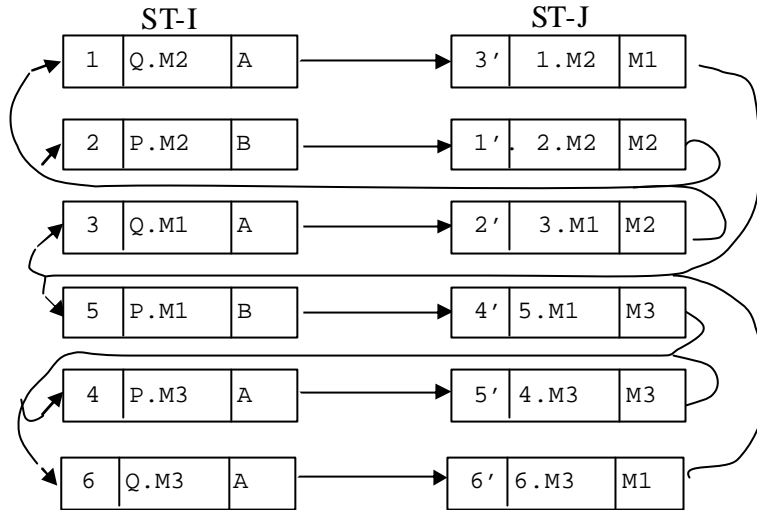
The final control policy at step 722 is shown diagrammatically below with patterns and actions given as numbers corresponding to the computer print-out.



The following diagrams show the growth of the optimum policy. The symbols given in subsection 2.5.3.2 are used in place of numbers.

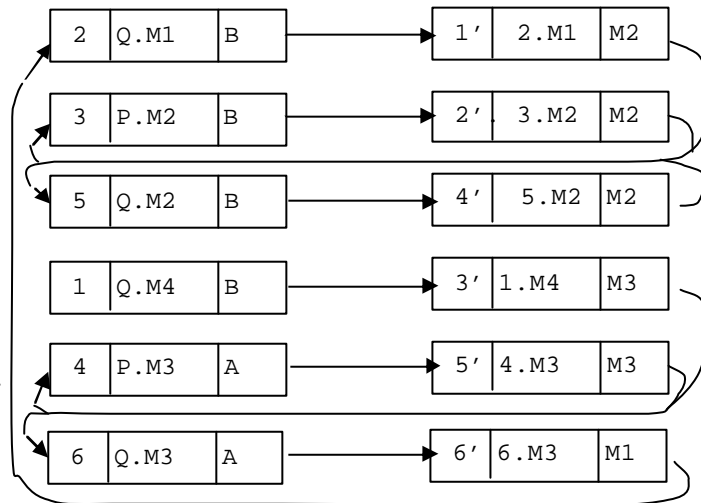
step 31
Confusion!
M1 always follows A,
but M2 and M3 follow
A and B arbitrarily.
Parts of an optimal
policy in 3,5,3',6'.

None of this survives.



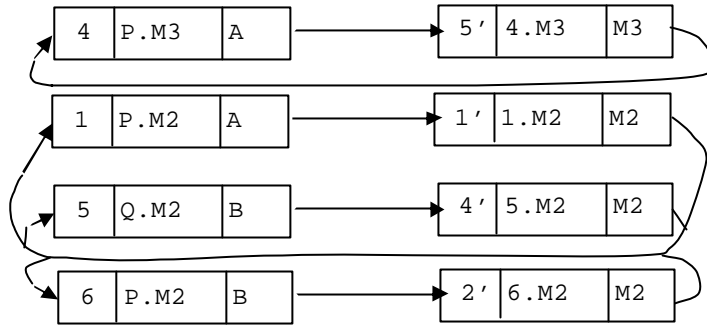
step 69
M1 always follows A,
M2 always follows B,
but M3 is ambiguous.
2,5,6,1',2',4',5' are
consistent with an
optimal policy.

Only 5 survives to
step 722 and it is not
then in the optimal policy.

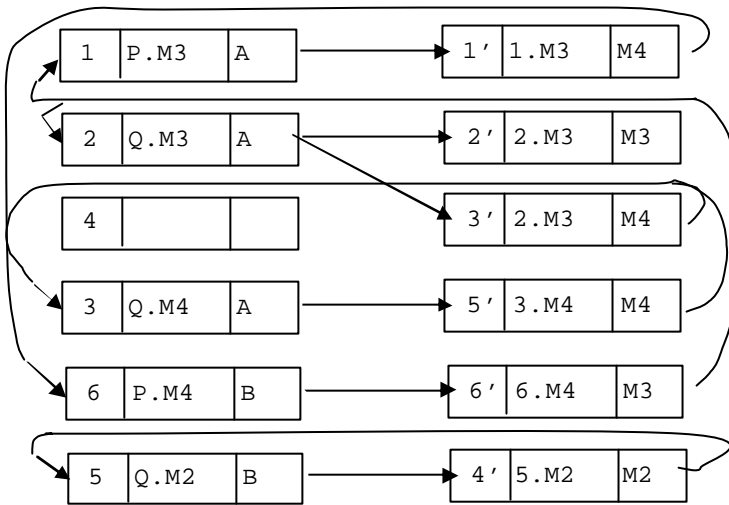


NB. The connecting arrows show the links formed by Monologue actions and by the ST-J patterns containing the policy element number of ST-I.

step 194
 The policy elements not shown have zero success fractions. 1 and 1' conflict with the partial optimality of the others.



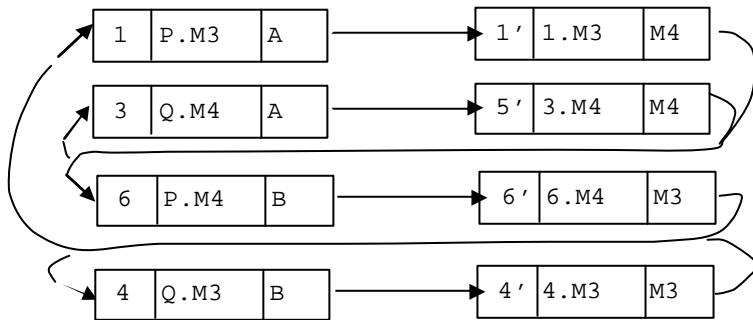
step 352
 1,3,6,1',5',6' will survive to form part of the optimal policy. 2,2' and 3' conflict with the reward conditions.



step 722
 optimal policy found

M4 always follows A
 M3 always follows B

Monologue provides the last action as state information for the current decision.

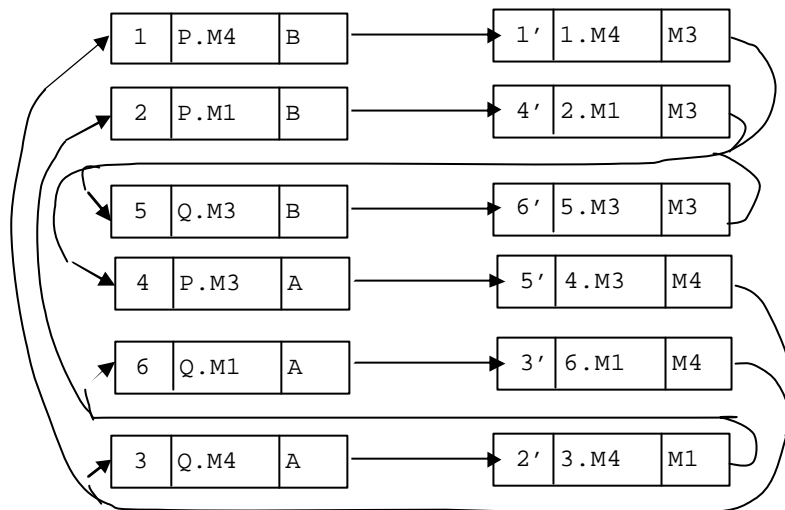


We are impelled to ask ourselves why it took over 500 steps for Monologue STELLA to find the optimal policy given on the last page.

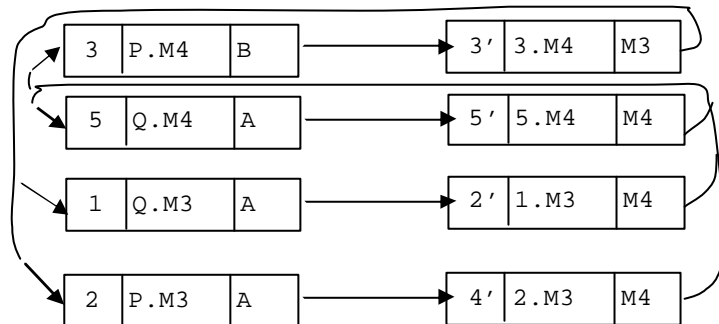
IS THIS A VERY DIFFICULT PROBLEM IN VERY SIMPLE FORM,
OR ARE WE THROWING AWAY AVAILABLE INFORMATION IN OUR
HEURISTICS?

To make matters worse, we show below from other runs an optimal policy which is unstable because it leaves no spare memory, and a suboptimal policy which is stable.

step 275
(23-10-1972)
This policy is lost
by step 355 even
though its responses
are correct.



step 611
(19-10-1972)
Policy elements 1,2'
are wrong but they
still lead to reward
on the next step.
How should the machine
know this is not optimal?



It can be strongly argued that the Predictor should be provided with Monologue. The Predictor receives reward in the sense of confirmation of its predictions at every step, so it has a simpler task in some ways than the Control Policy. However, the above task also makes reward available at every step, so we cannot deduce that a Predictor would be more succesful at using Monologue.

2.6 A Predictor for STELLA

In this section, we ask “What kind of predictor or internal model should be provided in STELLA?”

The question of prediction has already been considered implicitly in section 1.3 (BPLS) and explicitly in 2.3 (CLUMPS). Here we shall make comparison between ideas which have been tried or proposed earlier and some new possibilities which are showing promise. We are not concerned here with environments which can be treated as linear systems, for which powerful and well-known methods are available (e.g. Morrison, 1969 and Widrow, 1971) even though difficulties remain in that case as pointed out by Kalman (1971).

In our context, a good predictor should be able to:-

- P1) model a variety of environments from on-line information provided in a sampled, discrete or quantized, form.
- P2) operate within the confines of a strictly limited memory size.
- P3) learn ‘from scratch’ and in a changing environment.
- P4) tolerate errors and noise in its input.
- P5) provide estimates of its own reliability.
- P6) represent a deterministic environment deterministically if this can be done with the memory available.
- P7) model ‘active’ environments containing moving objects and, possibly, purposeful entities like itself.

It is evident that the last of these is out of reach at the present time and that the others are only partly attainable. Before considering a reasonable compromise, some of the alternatives must be described.

2.6.1 Action Matrices

The original predictor for STELLA comprised one matrix for each action. The matrix element of the i^{th} row and j^{th} column of the k^{th} matrix predicted the j^{th} bit of the input pattern following action k . In fact, the j^{th} column of a matrix was a BPLS (c.f. section 1.3) with a binary action declaring whether the j^{th} bit of the next input pattern would be in the class 0 or in the class 1. The weights were estimated as outlined in sections 1.3 and 1.6. Unconditional weights, predicting a bit of the pattern-after-action regardless of the pattern-before-action, were also included as required for the $\log(P_i/P_r)$ of the BPLS.

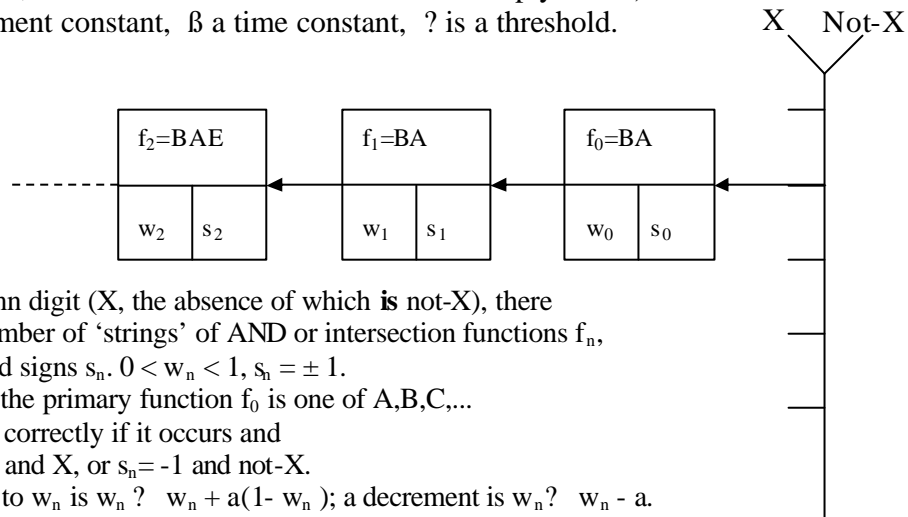
It was assumed that the input pattern was a binary word, that binary digits of the word corresponded to features of the environment, that these features provided independent information, and that the input pattern was state-describing so that no information was needed from earlier input patterns in order to make the predictions.

This list of severe assumptions is reason enough to look for something better, although the BPLS is a rugged system satisfying P3 and P4 and not too sensitive to departures from the independence assumption. Reliability estimators were provided for each matrix column to satisfy P5. Simple deterministic environments soon drove some of the weights to their extremes, causing the matrices to predict in a deterministic fashion as required for P6.

In an attempt to improve the action matrices with regard to P1 and P2, modifications were introduced which reduced the uneconomic use of memory by weakly predicting weights and, at the same time, allowed interdependent evidence from the digits of the 'pattern-before-action'. These modifications derived from a system called the String Correlator which is transcribed, for the record, from the original notes in the next section (16 Jan. 1964). This is done because of the relevance of this early idea to some of the techniques which we are exploring currently.

2.6.2 String Correlator

A,B,C,... are digits of the pattern-before-action and X is a digit of the pattern-after-action. (C might be not-A, but the absence of A is not taken to imply not-A) α is an increment/decrement constant, β a time constant, θ is a threshold.
 $0 < \alpha, \theta < 1$



1. To each column digit (X, the absence of which is not-X), there are a finite number of 'strings' of AND or intersection functions f_n , weights w_n and signs s_n . $0 < w_n < 1$, $s_n = \pm 1$.
2. In each string the primary function f_0 is one of A,B,C,...
3. An f_n predicts correctly if it occurs and either $s_n = +1$ and X, or $s_n = -1$ and not-X.
4. An increment to w_n is $w_n + \alpha(1 - w_n)$; a decrement is $w_n - \alpha$.
5. If a decrement would change the sign of w_n , the subtraction is carried out and then the signs are reversed of both w_n and s_n .
6. If f_n predicts correctly, w_n is incremented.
7. If f_n occurs without predicting correctly, w_n is decremented unless both f_m ($m > n$) predicts correctly, and $w_n > \alpha$.
8. If $w_n < \theta$, f_{n+1} is emptied.
9. If $w_n < \theta$ for a period greater than β since f_n was last filled, f_n is emptied.
10. If $w_n > \theta$ and f_{n+1} is empty, it is filled with the intersection (AND) of f_n with an f_n^* occurring in the pattern which led to the rise of w_n above θ . The f_n^* is selected from the allowed f_n 's by random choice weighted according to the w_n 's.
11. Let f_n be the f_n having $w_n > \theta$ and the highest n in its string.
12. The FINAL PREDICTION of X or not-X is given by the s_n of the f_n with highest w_n .
13. θ is here conceived as the minimum useful frequency of occurrence of an infallible function.

2.6.3 Modified Action Matrices

In the modified action matrices each matrix element has an associated pair of computer words, of equal length to the input pattern, which hold the current logic for that element. The 1's in one of the words indicate which bits are to be ANDed of the bits in the other word:-

e.g.	First word (which bits)	10110001	}	implies	--
	Second word (logic)	01101111			ACDH
	Corresponding features	ABCDEFGH			

In contrast to the variable number of f's of the String Correlator, the modified action matrices only have one logic function per element (corresponding to the f_n). The logic is changed when the element is predicting only weakly (probability of a 1 is near 0.5). Tests for weakly predicting elements are made every, say, 500 steps. As in the String Correlator, new logic is derived from the last occurring input pattern so that non-existent conditions cannot be included. Features are included in a logic combination according to their probability of occurrence, estimated from the unconditional (no logic) weights for each column digit.

The main reason for discussing the modified action matrices, even though they have been fully described in Andreae (1966), is to suggest that they could still provide an effective and economic predictor for binary word input patterns and small finite action sets if the logic capability were allowed to extend back to earlier input patterns than the pattern-before-action. Where the action set is more complex and separate action matrices cannot be used, action conditions can be incorporated with the patterns-before-action logic in a single "matrix"/ (The word "matrix" has been used loosely for a set of fixed-length strings of elements, one for each bit of the pattern-after-action).

2.6.4 Constructing an Automaton from Input/Output Behaviour

Recently we have been exploring another direction under the simplifying assumptions

- (1) the environment is a finite state automaton;
- (2) the input patterns and actions are noise-free.

The predictor problem now reduces to an automaton synthesis problem:

Given a string of input/output behaviour on-line from a finite state automaton, construct a minimal state automaton which would reproduce the input/output behaviour up to that time.

There is no solution to the problem known for the general case (to our knowledge) even if computation time (i.e. time between consecutive state transitions of the automaton) and information storage are unlimited.

The following discussion is exploratory. Much of the motivation and inspiration has come from John Cleary, who expects to give an account of his work in the next Report to DSE after submission of his MSc thesis. Several research evenings have been devoted to this problem.

2.6.4.1 Wrap-Unwrap Network (WUN)

Although the Wrap-Unwrap Network is not offered as a practical solution to the automaton-synthesis problem, we have found it useful as a basis for discussing more powerful heuristics.

A list-processing terminology will be used as follows:

The name of a list is underlined, e.g. NODES, LIST; the name of an item on a list is not. The first item on a list is obtained by the function FIRST(list), whose value is the first item on the list named. The next item after a given item on a list is obtained by the function NEXT(item). e.g. the third item on a list NODES of nodes can be given a name THIRD-NODE by the assignment instruction:

THIRD-NODE ← NEXT(NEXT(FIRST(NODES)))

where the backward pointing arrow can be read “becomes”.

The last item on a list is obtained by the function LAST(list).

An item can be selected randomly from a set by means of the function RANDOM(set).

The random selection is made anew each time the function is applied.

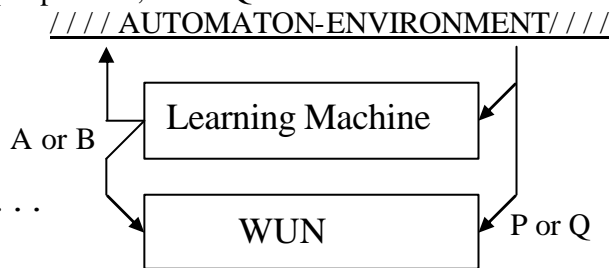
The value of an undefined item or empty list is \emptyset , zero or null.

The Wrap-Unwrap Network is a state-output model of the automaton environment, each node of the WUN corresponding to a state of the model in which a particular output is emitted. From each node there can be one transition to another node for each action possible.

For simplicity, we illustrate the WUN system with input/output behaviour comprising only 2 actions, A and B, and only 2 input patterns, P and Q.

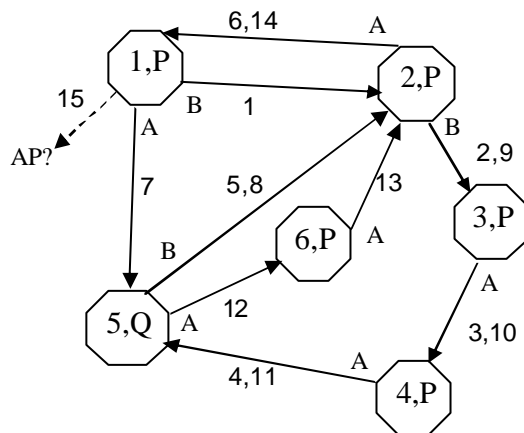
Below is reproduced the first 15 action/input pattern pairs of the sequence given in section 2.4.1.

-P BP BP AQ AQ BP AP AQ
BP BP AQ AQ AP AP AP AP



A WUN model which satisfactorily reproduces the behaviour up to, but not including, the last action/pattern pair is shown below. An action/pattern pair will also be called a “step”.

Step numbers and actions are written alongside each arrow showing a transition from one state (node) to another. The number of a node and the output associated with it are given inside each node hexagon. Notice how the 15th step AP disagrees with the A-transition from node 1 to node 5, an AQ step.



The principle of the WUN system is that, when the network exhibits an incompatibility with the behaviour string, the network is unwrapped as far as the last choice point and is then re-formed with an alternative from that point.

In the example shown, step 15 is incompatible, so steps 14 and 13 are unwrapped. It is not sufficient to unwrap step 14 alone because there is no alternative to the already established transition from node 2 to node 1 under action A for step 14. However, there were a number of choices at step 13:-

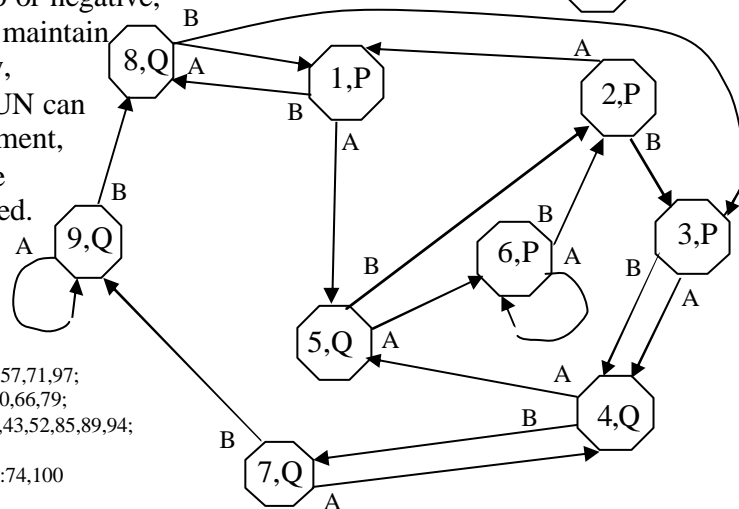
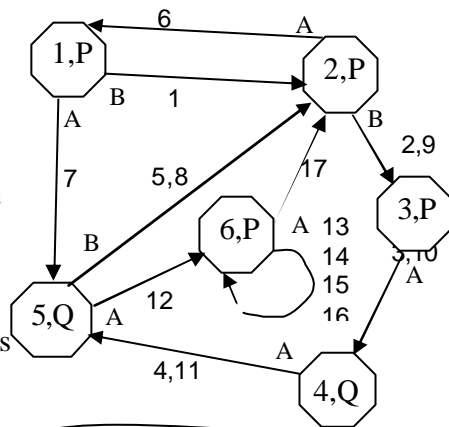
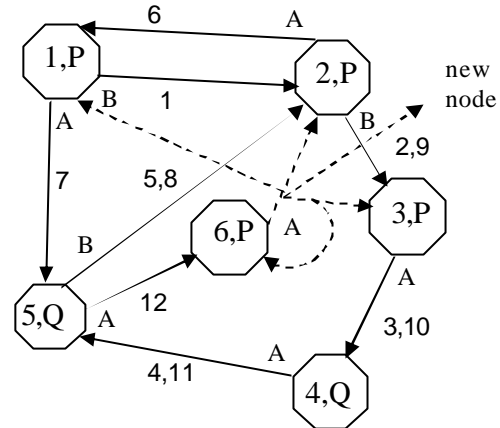
In making the choice now, we have the unwrapped sequence AP AP AP to match, so the connection from node 6 to nodes 1, 2 or 3 are incompatible. A simple extra rule, that random choice is made between maximum length matches, determines the 13th transition as being from node 6 to itself. This transition not only allows step 14 and 15 but continues to be satisfied up to step 17 where a choice occurs.

```
-P BP BP AQ AQ BP AP AQ BP BP AQ
 0 1 2 3 4 5 6 7 8 9 10
AQ AP AP AP AP AP BP BP BQ ...
11 12 13 14 15 16 17 18 19 ...
```

Continuing with this wrapping, unwrapping and random selection among maximum length alternatives, we proceed until the network models 100 steps. Even the liberty taken with the start of the input/output sequence (step 1) raises interesting possibilities, although a computer program would have missed this opportunity!

By decrementing all step numbers on transitions by one after each step from now on and by removing any which go zero or negative, the network can be made to maintain a memory of 100 steps only, in detail. In this way the WUN can adapt to a changing environment, although at times it might be almost completely unwrapped.

STATE ACTION:steps;
 1A:7,36,78,87; 1B:34,54,62,68,76;
 2A:6,33,53,61,67,86;
 2B:2,9,18,29,39,44,60,90,95;
 3A:3,10,30,40,45; 3B:19,56,70,96;
 4A:4,11,22,31,41,46,59,65,82,92; 4B:20,57,71,97;
 5A:12,23,37,42,47,83,88,93; 5B:5,8,32,60,66,79;
 6A:13-16,24-27,48-51,84; 6B:1,17,28,38,43,52,85,89,94;
 7A:21,58; 7B:72,98;
 8A:35,75,77; 8B:55,63,69; 9A:73,99; 9B:74,100



Example. WUN model corresponding to the diagram at the bottom of page 46. Assume that we have NODEMAX = 10.

Nodes	1	2	3	4	5	6	7	...	10								
Output	P	P	P	Q	Q	Q	∅	...	∅								
TO-NODEs	5	2	1	3	4	∅	5	∅	6	2	2	∅	∅	∅	...	∅	∅
Step	7	1	6	2	3	∅	4	∅	12	5	13	∅	∅	∅	...	∅	∅
Lists	∅	∅	14	9	10	11	∅	8	∅	∅	∅	∅	∅	∅	∅	∅	∅

2.6.4.3 WUN Heuristics

WUN(1) (Initialize)

All lists and variable ? ∅

STEP ? ∅

REAL-TIME points to the gap between FIRST(ENVMNT) and NEXT(FIRST(ENVMNT)).

This signifies zero or starting time.

THIS-PAIR ? FIRST(ENVMNT)

This is a pattern only,

i.e. FIRST(ENVMNT) = ∅/pattern

FROM-NODE ? 1

First node is identified with first pattern.

OUTPUT(FROM-NODE) ? PAT(THIS-PAIR) PAT(THIS-PAIR) and

ACT(THIS-PAIR) extract the action

and pattern from the action/pattern pair.

Apply WUN(2)

WUN(2) (Main Cycle)

STEP ? STEP + 1

STEP-ON(THIS-PAIR)

As soon as REAL-TIME advances to or past the next gap, THIS-PAIR is identified with the next

pair on the ENVMNT list. Meanwhile the computation is held up.

RESTEP ? STEP

PAIR ? THIS-PAIR

ACTION ? ACT(THIS-PAIR) New action is identified

PATTERN ? PAT(THIS-PAIR) New pattern is identified

If STEP-LIST(FROM-NODE,ACTION) = ∅, apply WUN(3)

If OUTPUT(TO-NODE(FROM-NODE,ACTION)) ? PATTERN, apply WUN(4)

NEXT(LAST(STEP-LIST(FROM-NODE,ACTION))) ? STEP

Put step number on step number list.

FROM-NODE ? TO-NODE(FROM-NODE,ACTION)

Apply WUN(2)

move to next node

WUN(3) (Select a new TO-NODE)

Define the set of compatible alternatives as ?

COMPATIBLE includes each node j for which OUTPUT(j) = PATTERN
and also the first empty node (k, say) for which
OUTPUT(k) = \emptyset .

(This means that one new node is a possibility in the random selection
of alternative possibilities)

If COMPATIBLE ? \emptyset ,

TO-NODE(FROM-NODE, ACTION) ? RANDOM(COMPATIBLE)

and OUTPUT(TO-NODE(FROM-NODE, ACTION)) ? PATTERN.

? A more elaborate definition involving maximum length matching sequences must
be used when WUN(3) is applied from WUN(4). The definition given is appropriate
when the network is not unwrapped.

Put unselected possibilities of COMPATIBLE on a list called POSS(STEP).

(This creates a POSS() list for every step which is quite impracticable.
It would be tidier and much more economical, but somewhat more
complicated, to use a stack. The unselected possibilities would then be
pushed onto the stack with a heading which indicated to which step
they belonged.)

End of WUN(3)

WUN(4) (Unwrap)

RESTEP ? RESTEP - 1

STEP-BACK(PAIR)

Find NODE and ACTION such that STEP-LIST(NODE, ACTION) contains

Delete RESTEP from STEP-LIST(NODE, ACTION) RESTEP.

If STEP-LIST(NODE, ACTION) = \emptyset , apply WUN(5),

else apply WUN(4).

WUN(5) (Wrap)

If POSS(RESTEP) = \emptyset , apply WUN(4),

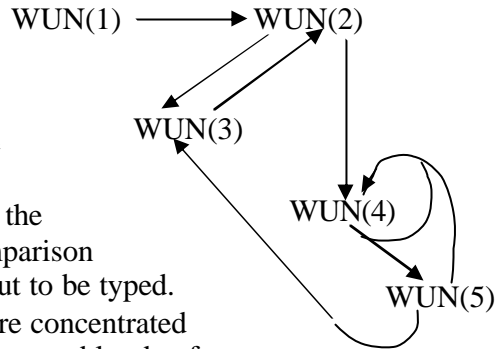
else test each possibility on the POSS(RESTEP) list to find which continue farthest
around the network before coming to a choice point or the current value of STEP.

Set STEP ? RESTEP, THIS-PAIR ? PAIR.

Define the set COMPATIBLE as these maximum length possibilities and apply
WUN(3). (No further change in POSS(STEP) is caused by the last clause of
WUN(3)).

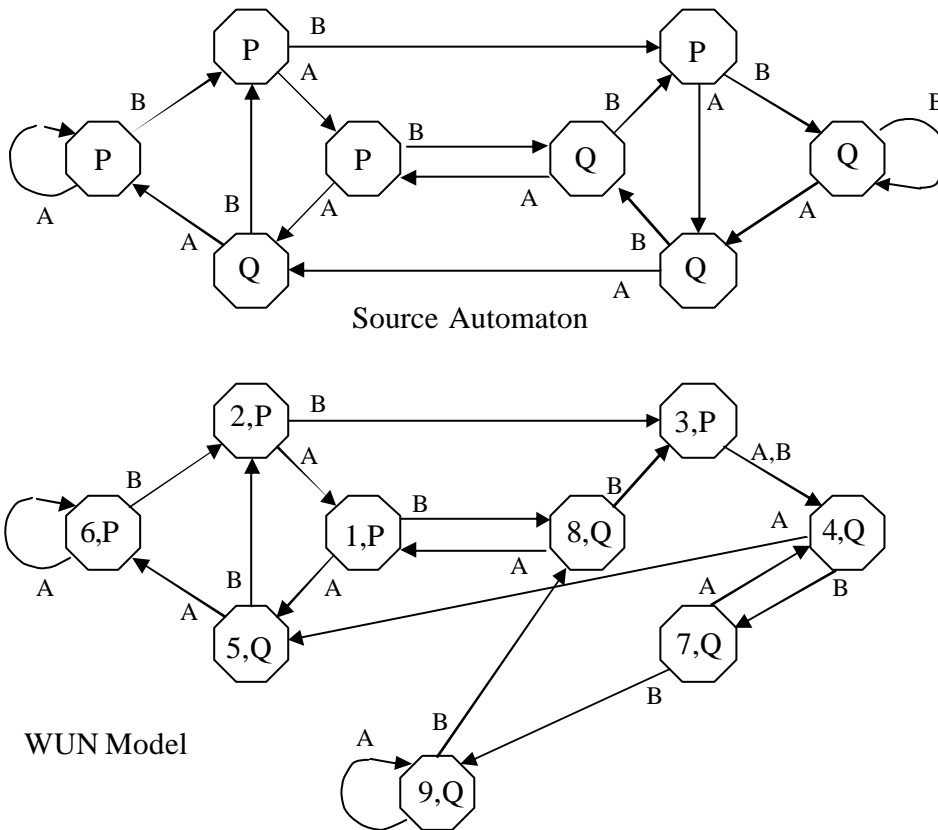
The lack of detail in WUN(5) is intentional. This is the part of the heuristics where more powerful heuristics are most likely to be substituted. It is also the part which requires the most complicated set of describing rules. The additional effort needed to formalize it is not justified at present, since the informal description given appears to be ambiguous.

The interlinking of the WUN heuristics is indicated in the diagram below



2.6.4.4 How near did WUN get?

The reader who gets this far deserves to be told how close the network at the bottom of page 47 is to the automaton from which the behaviour sequence was derived. This comparison was not made until this paragraph was about to be typed. It is shown below. Notice how the errors are concentrated in the later part of the network which is, presumably, due for some more unwrapping and wrapping.



2.6.4.5 Quintuple Monologue Predictor (QMP)

Looking closely at the error in the WUN model of 2.6.4.4, it can be seen that the B-transition from state 4 to state 7 is a key fault. Since this transition has a step number of 19 associated with it, the network would have to unwrap to step 19 in order to get rid of it. The decrementing procedure, suggested at the bottom of page 47, is unlikely to help because, by the time it has eliminated what is now called step 19, other step numbers will have been added to the fault. The real trouble is that the transition is correct in itself. Another case is the A-transition from state 3 to state 4, which has step numbers back to 3.

There appears to be a need for something more flexible. The Quintuple Monologue Predictor (QMP) was intended to provide the extra flexibility, but it goes too far in this direction. It is described briefly and without too much attention to detail because of the possibility of combining its principle with that of the WUN.

Notice that the WUN model at the bottom of page 51 (or any finite state machine) could be described unambiguously by the quintuples (indeed quadruples would do):

1PAQ5 2PAP1 3PAQ4 4QQAQ5 5QAP6 6PAP6 7QQAQ4 8QAP1 9QQAQ9
1PBQ8 2PBP3 3PBQ4 4QBQ7 5QBP2 6PBP2 7QBQ9 8QBP3 9QBQ8

The state numbers 1..9 are quite arbitrary and play the role of Monologue (section 2.5). If the patterns were state-determining, the additional numbers would be unnecessary. In the QMP we allow the Monologue, or arbitrary state numbers, to be altered as the action/pattern sequence from the environment moves the system from quintuple to quintuple.

The initial motivation for the QMO was the observation that Monologue STELLA failed to cope with tasks in which reward was given only at the end of a sequence of actions. In the case of prediction, confirmation or rejection of a prediction comes at each step (this is only a half-truth because of uninformative input patterns) so it seemed to be the better place to us Monologue.

The reason for including the before-action and after-action patterns in the quintuples (abbreviated to 5tpls) is to ensure that the QMP will work precisely when the patterns are state-determining without interference from the Monologue process.

QMP Heuristics will be illustrated by an example with comment.

A maximum of 10 Monologue state will be allowed. We do not discuss the question of what to do when all are used up. Ideally, the model should be reduced to a simpler form. Sometimes it is possible to delete 5tpls because they are disconnected from the other 5tpls or only connected in one direction to them. Standard reduction methods do not seem to be of much use in the more difficult situations.

QMP Heuristics

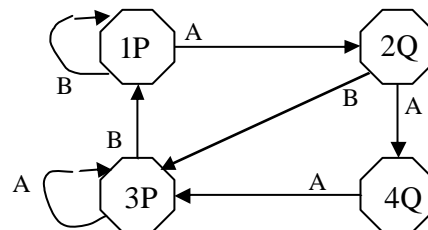
Behaviour of the environment is received as a series of p/a or pattern/action pairs.

No two 5tpls may have the same initial three symbols (e.g. 1PA-)

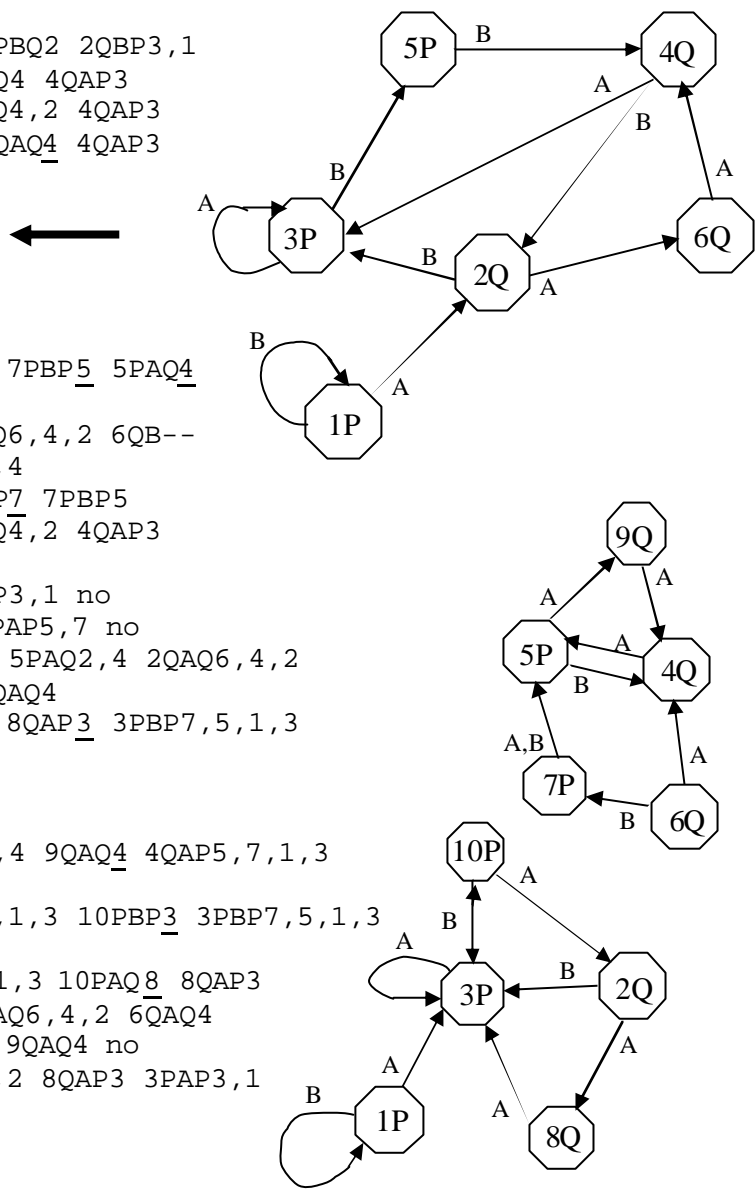
No two 5tpls may have the same first symbol but different second symbol (e.g. 3P--- and 3Q--- are not allowed)

The first four symbols may only be changed by being stored anew or by being deleted as a whole. (e.g. may not change 1PAP- to 1PAQ-).

Step	p/a	Quintuples	
0	PB	1PB--	Prepare to store first 5tpl with 1PB-- matching input pair..
1	PB	1PBP1 1PBP1	First 5tpl was completed with P from next p/a and Monologue 1 because only 5tpl with 1P--- is the one being stored. This leads immediately to itself to match the second p/a.
2	PA	1PA--	Prepare to store another 5tpl with 1PA--
3	QA	1PAQ2 2QA--	There is no 5tpl beginning with -Q---, so a new one is formed.
4	QB	2QAQ2 2QB--	Only 5tpl with -Q--- is 2QA-- so it adds Monologue addressing itself. Form new 5tpl with 2QB--
5	PA	2QBP1 1PAQ2	2QB-- is given the only Monologue (1) with P in a 5tpl This leads to the matching 5tpl
6	PA	2QBP3,1 3PAP <u>1</u> 1PAQ2	A clash occurs since 1PAQ2 leads to a 5tpl beginning with 2Q--, but the input p/a is PA. We hve to go back to the last correct 4tpl 2QBP- where a change in Monologue can be made. The Monologue is 1 at present, so it is changed to 3 the only other preceding P in a 5tpl. Now completion of a 5tpl 3PA-- is needed. We know that the next pattern is P so it becomes 3PAP-. Random choice of Monologue between 1 and 3 (because 1P--- and 3P--- are the only 5tpls with -P---) gives 1, underlined to show that it was obtained by random choice. The last 5tpl on the line follows to match the most recent p/a. Notice how the old Monologue is retained in 2QBP3,1 to prevent alteration back to 1 before other possibilities have been tried.
7	QB	2QBP3,1	Follows directly
8	PB	3PB--	New 5tpl
9	PA	3PBP <u>3</u> 3PAP1	Monologue 3 randomly selected from 1 and 3
10	QA	3PBP1,3 1PAQ2	2QAQ2 Monologue changed on 3PBP3 to 3PBP1,3
11	QA	2QAQ2	
12	PA	2QAQ4,2 4QAP <u>3</u> 3PAP1	
13	PA	1PAQ2	
14	PA	3PAP3,1 3PAP3,1 3PAP3,1	
15	PA	3PAP3,1	



- 16 PA 3PAP3,1
- 17 PB 3PBP1,3
- 18 PB 1PBP1
- 19 QB 3PBP5,1,3 5PBQ2 2QBP3,1
- 20 QA 5PBQ4,2 4QBQ4 4QAP3
- 21 QA 4QBQ2,4 2QAQ4,2 4QAP3
- 22 QA 2QAQ6,4,2 6QAQ4 4QAP3
- 23 PA 3PAP3,1
- 24 PA 3PAP3,1
- 25 PA 3PAP3,1
- 26 PA 3PAP3,1
- 27 PB 3PBP5,1,3
- 28 PB 5PBQ4,2
- 29 PA 3PBP7,5,1,3 7PBP5 5PAQ4
- 30 QA 4QAP3
- 31 QB 5PAQ2,4 2QAQ6,4,2 6QB--
- 32 PA 6QBP5 5PAQ2,4
- 33 PB 6QBP7,5 7PAP7 7PBP5
- 34 QA 7PAP5,7 5PBQ4,2 4QAP3
- 35 PA 3PAP3,1
- 36 QA 4QAP1,3 1PAP3,1 no
4QAP7,1,3 7PAP5,7 no
4QAP5,7,1,3 5PAQ2,4 2QAQ6,4,2
- 37 PB 5PAQ6,2,4 6QAQ4
5PAQ8,6,2,4 8QAP3 3PBP7,5,1,3
- 38 PB 7PBP5
- 39 PA 5PAQ8,6,2,4
- 40 QA 8QAP3
- 41 QA 5PAQ9,8,6,2,4 9QAQ4 4QAP5,7,1,3
- 42 PB 5PBQ4,2
- 43 PB 4QAP10,5,7,1,3 10PBP3 3PBP7,5,1,3
- 44 PA 7PAP5,7
- 45 QA 3PBP10,7,5,1,3 10PAQ8 8QAP3
- 46 QA 10PAQ2,8 2QAQ6,4,2 6QAQ4
- 47 PA 2QAQ9,6,4,2 9QAQ4 no
2QAQ8,9,6,4,2 8QAP3 3PAP3,1
- 48 PA 3PAP3,1
- 49 PA 3PAP3,1
- 50 PA 3PAP3,1



It can be seen that the QMP has disintegrated into two parts at this stage. The small amount of sequential memory used in the QMP (compared with the long memory used in the step numbers of the WUN model) makes it unable to cope with an environment as difficult as the one used above (8 state automaton on page 51). However, if one or two of the outputs are made state-determining, the QMP has much less difficulty.

A full investigation of the QMP would require a computer simulation, but it is not considered justified at the present time, especially in view of the promising technique discussed in the next section.

2.6.4.6 Cleary's Compatibility Rule for Automaton States (CRAS)

It will be recalled that, when the WUN network was unwrapped, a sequence of behaviour was available for the rewrapping, but the only use made of the information in the sequence was to select between maximum length alternatives to the first choice point.

In this section we consider the constraints on state assignment which are implicit in a sequence of behaviour: _

$$p_1 a_1 \quad p_2 a_2 \quad p_3 a_3 \quad p_4 a_4 \quad \dots \quad p_i a_i \quad \dots \quad p_n a_n$$

We shall write $p_i = p_j$ to mean that both "pattern nodes" p_i and p_j stand for the same input pattern. Behavioural sequences beginning and ending with an action will be denoted by a "z" with or without a suffix, e.g. $z_1 = BPAPA$, $z_j = BPAPBQAPA$, and $z_5 = APBPBPAQA$.

Expressions of the form $p(z_1, z_2, z_3, \dots, z_m)^*$ will be used to denote any non-empty string constructed by concatenating any number of the strings $pz_1, pz_2, pz_3, \dots, pz_m$ taken in any order. e.g. $p(z_1, z_j, z_5)^* = pBPAPA$ or $= pBPAPBQAPA$ or $= pAPBPBPAQA$
 or $= pBPAPApAPBPBPAQA$ or $= pAPBPBPAQApBPAPA$
 or $= pBPAPApAPBPBPAQApBPAPApBPAPApBPAPA$ or = etc.

$C_m(p,q,r,\dots)$ and $\bar{C}_m(p,q,r,\dots)$ denote m-fold compatibility and m-fold incompatibility, respectively, of the m pattern nodes p,q,r,... .

Definition. m pattern nodes are m-fold compatible (incompatible) if they can (cannot) all be assigned together to the same state of a connected state-output automaton which will reproduce the given sequence of behaviour.

CRAS:- (m-fold compatibility)

If $p_i z_0 p_{i+1} z_1 p_{i+2} z_2 \dots z_{m-2} p_{i+m-1}$ occurs in the sequence of behaviour, and if for every occurrence of $p_i(z_0, z_1, \dots, z_{m-2})^* p$ starting from node p_i and for arbitrary p it is true that $p = p_i$, then $C_m(p_i, p_{i+1}, p_{i+2}, \dots, p_{i+m-1})$.

(m-fold incompatibility)

If $p_i z_0 p_{i+1} z_1 p_{i+2} z_2 \dots z_{m-2} p_{i+m-1}$ occurs in the sequence of behaviour, and if for some p ? p_i , $p_i(z_0, z_1, \dots, z_{m-2})^* p$ occurs from p_i , then $\bar{C}_m(p_i, p_{i+1}, p_{i+2}, \dots, p_{i+m-1})$.

(First Corollary)

m-fold compatibility of an m-element set of pattern nodes implies (m-1)-fold compatibility of an (m-1) element subset of that set.

(Second Corollary)

(m-1)-fold incompatibility of an (m-1)-element set of pattern nodes implies m-fold incompatibility of any n-element superset containing the set.

Examples of the use of CRAS

Two pattern nodes having different patterns are incompatible. (Remember that we are only concerned here with state-output automata (e.g. Arbib, 1969, p.58), sometimes called a Moore machine)

Let the input sequence be $p_1a_1p_2a_2p_3a_3 = PAPAQB$

The $\epsilon_2(p_1, p_3)$ because $p_1z_0p_3$ occurs with $z_0 = ARA$ and $p_3 = Q \neq p_1$.

Two pattern nodes followed by the same sequence up to a difference in pattern are incompatible. i.e. They cannot be the same state if the same sequence after them leads to different states.

$\epsilon_2(p_1, p_2)$ because $p_1z_0p_2$ occurs with $z_0 = A$ and $p_1z_0p_1z_0Q = PAPAQB$ occurs from p_1 .

Three pattern nodes are incompatible if it is a consequence of their being the same state that a node with a different pattern must also be the same state.

Let the input sequence be $p_1a_1p_2a_2p_3a_3p_4a_4p_5 = PBPAPAPBQ$

$C_2(p_1, p_2)$ since no occurrence of $p_1(a_1)^* = PB$ or $PBPB$ or ... starting from p_1 is followed by Q . In other words, since these two nodes are followed by different actions, we cannot tell whether they are different states or not from the evidence.

$C_2(p_1, p_3)$ since no occurrence of $p_1(BPA)^* = PBPA$ or $PBPAPBPA$... starting from p_1 is followed by Q .

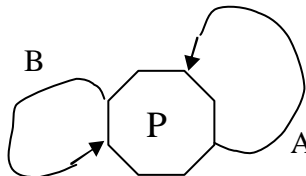
$C_2(p_2, p_3)$ since no occurrence of $p_2(A)^* = PA$ or $PAPA$ or ... starting from p_2 is followed by Q .

But,

$\epsilon_2(p_1, p_2, p_3)$ since $PBPAPAPBQ = p_1(z_0, z_1)^*Q$ occurs where $z_0 = B$ and $z_1 = A$.

In other words, if p_1 and p_3 are the same state, then p_4 must be the same state. But if p_1 and p_4 are the same state, then p_1 and p_5 must be the same state, which cannot be since they have different patterns.

The implication of p_1, p_2 and p_3 representing the same state is shown in diagrammatic form below.



2.6.4.7 Predictor Using Slide and Strings (PUSS)

A compromise predictor has been devised which seems to combine the advantages of most of the ideas discussed so far, viz. TREEPLE memory, String STELLA, WUN, QMP and CRAS.

PUSS was developed from an automaton-building system of John Cleary's, which uses a tree to store compatible strings. He expects to formalize this shortly. At present, automaton construction is avoided in PUSS because of the possibility of inconsistencies. However, a network representation of the PUSS strings by LINKNET (Cashin, 1970) appears feasible, even though it may not correspond to a strictly consistent automaton at any one time.

PUSS has the particular advantage of being able to tolerate slightly noisy input (i.e. errors in pattern or action) and non-stationary environments. Also it is a limited storage system.

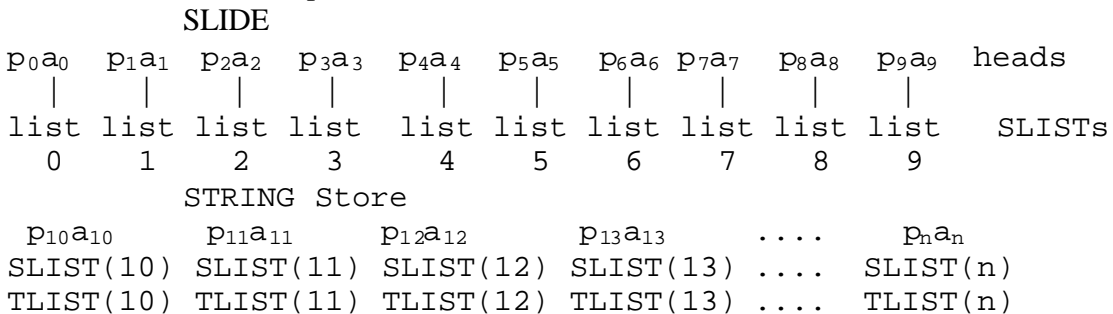
Storage for PUSS

The m most recent pa 's from the pattern/action sequence of behaviour are held in the SLIDE. Each entry (pa) in the SLIDE is the first item, or "head", of a variable-length list, or string, called the SLIST(index). p_0a_0 , the head of the first SLIST, SLIST(0), is the most recent pa to have been received from the environment. In the following we shall assume that $m = 10$, so that strings cannot exceed 10 in length.

As each new pa is received from the automaton-environment, the entries in the SLIDE are right-shifted (SLIST indices incremented) one place into the STRING store and the new pa becomes p_0a_0 . p_9a_9 becomes $p_{10}a_{10}$, the head of the first SLIST in the STRING store. SLIST(10) is given TLIST(10), the first of the transition lists. SLISTS in the SLIDE donot have TLISTS.

The STRING store is a limited-length store which holds up to $n-m+1$ strings (i.e. SLISTS) together with their transitions (i.e. TLISTS). The value of n will determine the amount of memory needed. Both m and n can be treated as incrementable, but additional flexibility of this kind would be liable to reduce the system's ability to cope with noise and non-stationarity. For the sake of illustration, suppose that $n = 100$. The ratio of n/m would have to be much larger if the numbers of patterns and actions were more than assumed here, (P,Q; A,B), in the examples. The heuristics are general.

Predictions are made directly from the SLISTS and TLISTS. A certain amount of model network construction is possible.



Definitions

A string is of length k if it comprises k pa's.

A string extension of SLIST(i) ($0 \leq i \leq m$) is formed by concatenating the heads of SLIST(i), SLIST($i-1$), SLIST($i-2$),...,SLIST(0) to a length greater than the initial length of SLIST(i).

A string extension of SLIST(i) ($m \leq i \leq n$) is formed by concatenating the heads of SLIST(i), SLIST(j), where j is any one of the entries in TLIST(i), SLIST(k), where k is any one of the entries in TLIST(j), and so on, beyond the initial length of SLIST(i) and, if required, until an empty TLIST is found.

A string equals another string if the two strings have the same length and have the same entries in the same order.

A string includes another string if the two can be made equal by removing pa's from the end of the first.

A substring of a string is any non-null string obtained by removing none, one or more pas's from the end of the string.

Two strings are compatible if neither includes the other and if their shortest unequal substrings of equal length k (i.e. their substrings of length $k-1$ are equal) differ only in the actions of the final pa's.

Two strings are incompatible if neither includes the other and if their shortest unequal substrings of equal length k (i.e. their substrings of length $k-1$ are equal) differ in the patterns of the final pa's.

Examples

PB PA QB and PB PA QB are equal, do not include each other, are not compatible and not incompatible.

PB PA QB and PB PA QB QA are not equal (are unequal), but the second includes the first; they are neither compatible nor incompatible.

PB PA QA and PB PA QB QA are compatible and unequal and neither includes the other.

PB PA QA and PB PA PB QA are unequal and incompatible.

PB, PB PA, PB PA QB and PB PA QB QA are all substrings of the string
PB PA QB QA.

PB PA QB QA is a string of length 4.

Heuristics for PUSS

PUSS(1) (Initialize)

- 1.1 Clear SLIDE and STRING Store.
- 1.2 Read the first m pa's into the SLIDE.
- 1.3 Apply PUSS(2)

PUSS(2) (Main Cycle)

- 2.1 If a string extension of any SLIST string in the SLIDE (but not the unextended string) is incompatible with a string extension of any other SLIST string, then the string in the SLIDE is extended by the least number of pa's needed to show the incompatibility.
- 2.2 If any SLIST string in the SLIDE is included in another SLIST string, and if a string extension of the string in the SLIDE is compatible with the other string (not an extension of it), then the least extension of the string in the SLIDE which shows the compatibility is made.
- 2.3 Delete SLIST(n), TLIST(n) and all entries "n" in TLISTS.
- 2.4 "Right-shift" SLIDE and STRING store, i.e. increment by one all indices of all lists and all entries of TLISTS.
- 2.5 Accept new pa as p0a0, the head of SLIST(0).
- 2.6 Add the entry "m" to TLIST(m+1).
- 2.7 If any string in the SLIDE, SLIST(j) ($0 \leq j < m$), equals SLIST(m), replace TLIST(m) by TLIST(j), substitute m for j in every TLIST where j occurs and apply PUSS(3) to delete SLIST(j) and TLIST(j).
- 2.8 If SLIST(m) includes any other string, SLIST(j) ($m < j \leq n$), in the STRING Store, apply PUSS(3) to delete SLIST(j) and TLIST(j).
- 2.9 If a prediction is required, apply PUSS(4).
- 2.10 Apply PUSS(2).

PUSS(3) (Delete SLIST(j) and TLIST(j))

- 3.1 Delete SLIST(j) and TLIST(j).
- 3.2 Delete all entries "j" in TLISTS.
- 3.3 Decrement by one all SLIST and TLIST indices greater than j and all entries greater than j in TLISTS.

PUSS(4) (Prediction)

- 4.1 If a prediction is required from a string of pa's, called BASE, then continue to remove the head of BASE until the remainder, of length k, equals an SLIST string, SLIST(i) where $k < i \leq n$. Predictions from the "present" use the SLIDE as the BASE string.
- 4.2 The prediction is the set of alternatives provided by all string extensions of SLIST(i). Often it will only be necessary to predict one pattern ahead, so that the prediction will be the set formed by the union of patterns in the first pa's added to SLIST(i) in each string extension of SLIST(i).

Example

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21...
0	PA																					
1	QA	PA																				
2	PA	QA	PA																			
3	QB	PA	QA	PA																		
4	QA	QB	PA	QA	PA																	
5	PB	QA	QB	PA	QA	PA																
6	PA	PB	QA	QB	PA	QA	PA															
7	QA	PA	PB	QA	QB	PA	QA	PA														
8	PB	QA	PA	PB	QA	QB	PA	QA	PA		m											
9	PA	PB	QA	PA	PB	QA	QB	PA	QA	PA?	?											
10	QB	PA	PB	QA	PA	PB	QA	QB	PA	QA	PA											
11	QA	QB	PA	PB	QA	PA	PB	QA	QB	PA	QA	PA										
												10										
													10									
12	PA	QA	QB	PA	PB	QA	PA	PB	QA	QB	PA	QA										
													11	10								
13	QB	PA	QA	QB	PA	PB	QA	PA	PB	QA	QB	PA	QA									
14	QA	QB	PA	QA	QB	PA	PB	QA	PA	PB	QA	QB	PA									
15	PB	QA	QB	PA	QA	QB	PA	PB	QA	PA	PB	QA	QB	PA								
16	PA	PB	QA	QB	PA	QA	QB	PA	PB	QA	PA	PB	QA	QB								
17	QA	PA	PB	QA	QB	PA	QA	QB	PA	PB	QA	PA	PB	QA								
18	PB	QA	PA	PB	QA	QB	PA	QA	QB	PA	PB	QA	PA	QB								
19	PB	PB	QA	PA	PB	QA	QB	PA	QA	QB	PA	PB	QA	QB								
20	QB	PB	PB	QA	PA	PB	QA	QB	PA	QA	QB	PA	PB	QA								
		QB	PB			PA																
21	PA	QB	PB	PB	QA	PA	PB	QA	QB	PA	QA	QB	PA	PB								
		PA	QB	PB			PA		QA													
22	QA	PA	QB	PB	PB	QA	PA	PB	QA	QB	PA	QA	QB	PB								
			PA	QB	PB			PA		QA												

PUSS(1)
Filling up SLIDE

PUSS(2) begins

? TLIST(11)
PUSS(3) USED
TLIST(12) added to TLIST(m)

SLIST(1), SLIST(2) and
SLIST(5) extended to
show incompatibilities,

It should be pointed out that even at this stage the network constructed contains a basic discrepancy. The environment-automaton does not have A and B transitions from a P state to the same Q state. The fault has arisen because some of the strings are not state-determining.

Ideally, each string in the STRING Store prevents the head of the string from corresponding to more than one state of the environment-automaton. A string, whose head can only correspond to one state of the environment-automaton, will be called a state-determining string. In general, there must be strings which are not state-determining in the STRING Store.

Suppose that a string s can correspond to two state sequences of the environment-automaton beginning in different states but ending in the same state. In this case, no string extensions sz of s can distinguish the states corresponding to the head of the string. However, a pre-extension xs of the string s , formed by adjoining the string x ahead of the string s , may be incompatible with another pre-extension ys , thus distinguishing the states corresponding to the beginning of s . This prompts the following heuristic:-

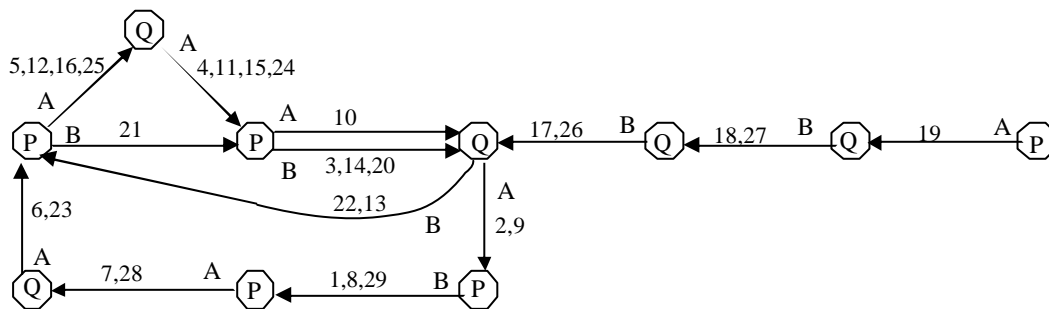
Heuristic

Try assigning the same state to the heads of equal-length compatible strings (i.e. strings which differ only in the final action), which do not appear as parts of other strings.

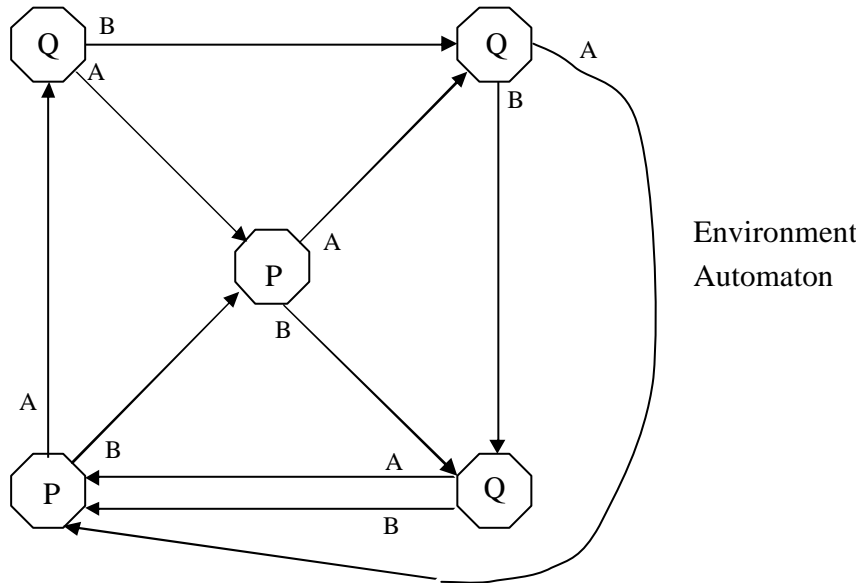
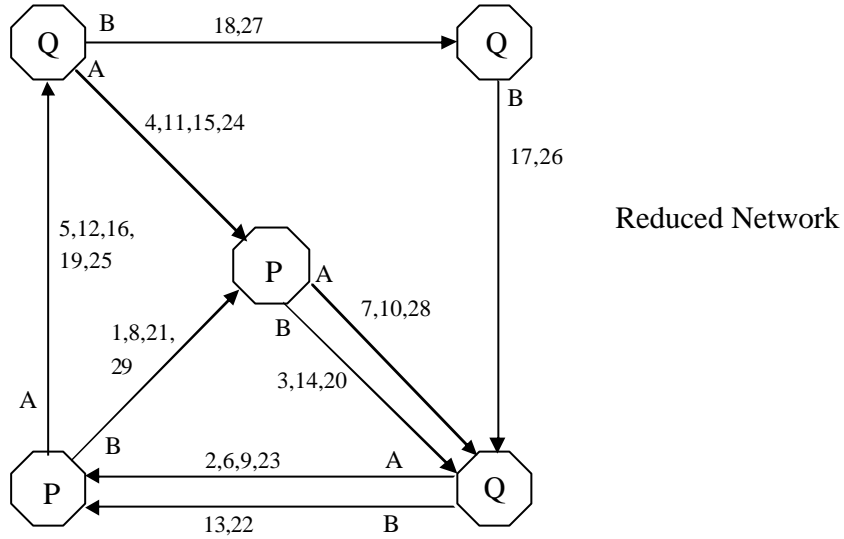
In the example we are considering, there are two pairs of strings to which the heuristic applies:-

16 PA QA PB QB 15	and	18 QB QB QA 17
25 PA QA PB QA 24		27 QB QB QB 26

The effect of identifying the heads of strings 16 and 25, and 18 and 27, is to reduce the network considerably:-



Further reduction of the network follows by inspection and enables us to make a direct comparison with the environment automaton. Notice that this final network is entirely consistent with the strings in the SLIDE and STRING Store.

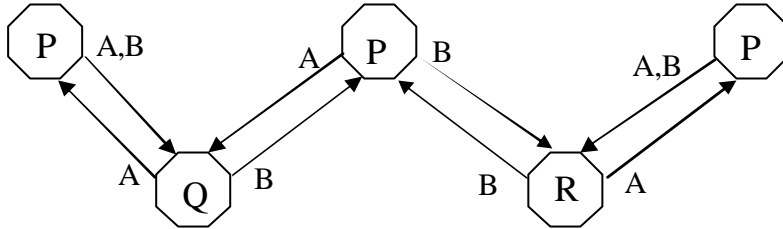


Computer simulation of PUSS will be needed to explore the limitations of the predictor.

It may be desirable to store an additional "predicted pattern" with each SLIST string, this predicted pattern being the pattern following the final action. In this way predictions will only be made by strings which are not parts of other strings corresponding to the behaviour sequence (BASE string).

2.6.4.7' Postscript to PUSS

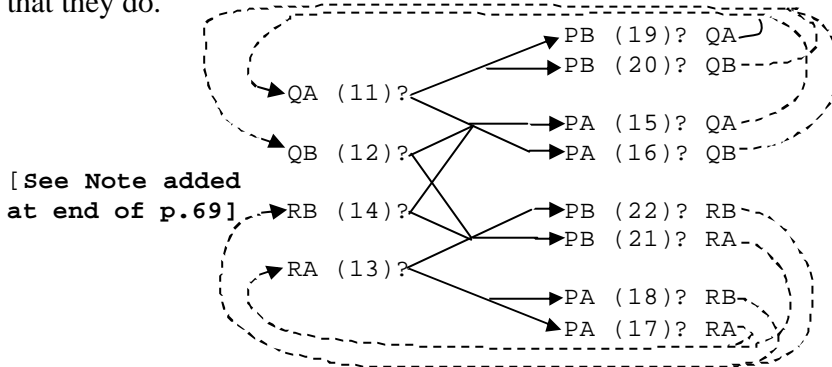
John Cleary has proposed the following environment-automaton as a counter-example to PUSS. This inspired the following addition of Monologue to PUSS.



With the ball of string behaviour from this automaton, PUSS would collect the strings and transitions listed below in her STRING Store:

11	12	13	14	15	16	17	18	19	20	21	22
QA	QB	RA	RB	PA	PA	PA	PA	PB	PB	PB	PB
15	15	17	15	QA	QB	RA	RB	QA	QB	RA	RB
16	16	18	16	11	12	13	14	11	12	13	14
19	21	21	21								
20	22	22	22								

If a network is constructed from these, an inconsistency becomes apparent, e.g. transitions 15 and 17 cannot refer to the same P-state and yet transitions 13 and 14 imply that they do.



Two ways of resolving the inconsistency come to mind. In the first, we would use higher-fold incompatibilities in building up strings in the STRING Store. In the second, we would use monologue state assignments to distinguish otherwise identical strings in the STRING Store. Since the second method would only operate when inconsistencies appeared and could be directed to remove them, while the first method would fail if the SLIDE were not long enough, the second method looks more promising.

Here we suggest using monologue only when a prediction is required and only as a temporary addition to the STRING Store, the basic PUSS algorithm and strings remaining unaffected. Later it may be found profitable to use the monologue assignments to induce the formation of new or extended (head or tail extensions) strings.

When monologue assignments have been made, string extensions for prediction will be constrained by the monologue. In other words, strings can only be extended in a manner consistent with the assignment of monologue. Since monologue is interpolatory in PUSS, while it was extrapolatory in STELLA, it can be expected to be more powerful and less arbitrary here.

The new heuristic given below is used in place of PUSS(4), but PUSS(4) is reverted to if a consistent monologue assignment is not found.

PUSS(4) (Prediction with Monologue Assignment)

- 4.1 Make next monologue assignment according to the monologue assignment tree (MATREE). If there is an inconsistency, record the rejection of this monologue assignment on the MATREE and, if no computation time or depth limit has been exceeded, repeat 4.1.
- 4.2 If a computation time or depth limit has been exceeded, exit with unsuccessful monologue assignment and apply old PUSS(4).
- 4.3 If a prediction is required from a string of pa's, called BASE, then continue to remove the head pa of BASE (the head pa is the least recent pa on the string) until the remainder, of length k, equals a string extension (subject to monologue) of some string SLIST(i) where $k < i \leq n$.
- 4.4 The prediction is the set of alternatives provided by extension of the string SLIST(i) beyond the length k over which it corresponds to the (truncated) BASE string.

Organization of the MATREE is a crucial problem which has not been worked out yet. Here we shall use a simple organization in which monologue assignments are made relative to a minimal monologue assignment obtained by:-

Minimal Monologue Assignment

Starting with the first string in the STRING Store, and continuing to the last, assign monologue k to the next string,

- where $k = 1$ for the first string;
- $k = k' + 1$ for a string which is incompatible with all previous strings and k' is the highest monologue assignment so far;
- $k = k^*$ for a string, if the first string with which it is most compatible^a has monologue k^* assigned.

Other monologue assignments are organized in the MATREE according to the splitting of the strings in the minimal monologue assignment. We have not yet ordered the alternatives within each splitting. If a consistent monologue assignment is found for a particular splitting, any of the string splits which are not addressed by any transition lists are treated as deleted.

An ordering of the splitting of the strings in the minimal monologue assignment will be made by a numbering scheme, in which the i^{th} element of an r-tuple for r unsplit strings, indicates how many identical strings (string splits) the i^{th} string has been split into.

^aTwo strings are more compatible if they are equal over a longer length. Two strings are compatible here if they are 2-fold compatible with each other and with any other string assigned the same monologue as either.

Example of MATREE numbering scheme for case of only 4 strings in the minimal monologue assignment (i.e. 4 strings in STRING Store, and r=4):-

	4-tuple	
Minimal Monologue Assignment		1,1,1,1 2,1,1,1 1,2,1,1 1,1,2,1 1,1,1,2
Double Splits		3,1,1,1 2,2,1,1 2,1,2,1 2,1,1,2 1,3,1,1 1,2,2,1 1,2,1,2 1,1,3,1 1,1,2,2 1,1,1,3
Triple Splits		4,1,1,1 3,2,1,1 3,1,2,1 3,1,1,2 2,3,1,1
.....	

A minimal monologue assignment for Cleary's counter-example would be

1,1,1,1,1,1,1,1,1,1,1,1,1,1 =

11(1)	12(1)	13(2)	14(2)	15(3)	16(3)	17(4)	18(4)	19(3)	20(3)	21(4)	22(4)
QA	QB	RA	RB	PA	PA	PA	PA	PB	PB	PB	PB
15(3)	15(3)	17(4)	15(3)	QA	QB	RA	RB	QA	QB	RA	RB
16(3)	16(3)	18(4)	16(3)	11(1)	12(1)	13(2)	14(2)	11(1)	12(1)	13(2)	14(2)
19(3)	21(4)	21(4)	21(4)								
20(3)	22(4)	22(4)	22(4)								
	?		?								

inconsistencies

The inconsistencies in the minimal monologue assignment point to the following splittings:-

- (i) Splitting 12(1) and 14(2) 1,2,1,2,1,1,1,1,1,1,1,1
- (ii) " 15(3) and 16(3) 1,1,1,1,2,2,1,1,1,1,1,1
- (iii) " 21(4) and 22(4) 1,1,1,1,1,1,1,1,1,1,2,2

A consistent assignment can be obtained by making both (ii) and (iii) together, but heuristics are needed to guide a search through the MATREE. One strategy would be to apply all the indicated splittings together and then delete the unnecessary ones.

Not too much significance should be given to the fact that the consistent assignment obtained from (ii) and (iii) recovers the original automaton. further research is needed and looks promising. (postscript: 12.1.1973)

2.6.5 Future Work on Predictors for STELLA

In section 1.1 brief mention was made of a linear environment, but since then we have been concerned only with deterministic and stochastic automata.

Both the linear continuous system of the control theorist and the deterministic or stochastic automaton of the computer theorist seem inadequate to deal with the kind of environments which humans handle daily. Neither the state space of linear systems nor that of automata seem appropriate for an environment in which distinguishable entities or objects “move around” in some sense. In such an environment, continuity can be a local, rather than a global property, so that we can talk about discrete objects with internally continuous properties, interacting with, say, a continuous background.

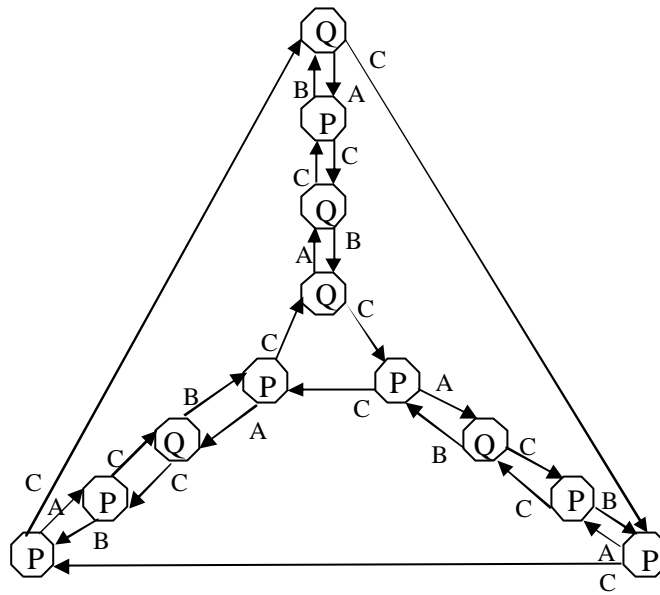
The mathematical basis for such an environment probably exists already in terms of module theory (e.g. interesting discussions in Kalman, 1969), tolerance automata (e.g. Arbib in Kalman, 1969) and category theory (Freyd, 1964), but my understanding of them is still too meagre to appreciate the possibilities fully. While getting to grips with the new techniques, we can still tackle the problem at a simpler level.

When talking about “an object in an environment” we mean, particularly, that the object can be removed from the environment and can be brought back. It has an existence of its own. In control theory terms, the object is controllable and observable. It must have a zero state relative to the background environment. There must be controls which will cause it to take that zero state, and there must be controls which will cause it to be observed again. The additivity of an object and the environment in which it exists is essential to the idea.

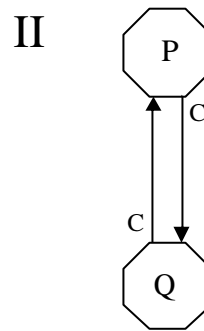
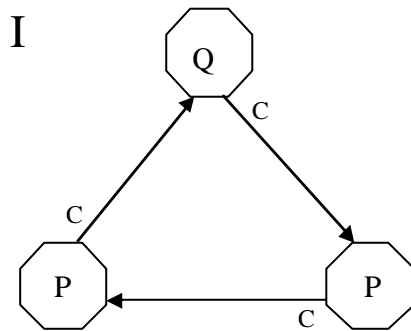
In pattern recognition, there has long been discussion of the ways and means of identifying distinctive features (e.g. Hill, 1969, discussing speech). These distinctive features are required to be independent as far as possible so that any one feature can be present or absent irrespective of the others. It is desirable for simplicity, that the whole be interpreted as a sum of its parts. Complex interactions make interpretation difficult or impossible. It pays to treat a visual environment as an additive collection of objects on a background, especially if they have a natural tendency to separate existence and progress.

In the case of sight and, to a lesser extent, in the case of hearing, the human modelling system (i.e. predictor) seems to be strongly dependent for success on its possession of controls, such as eye and head movements, which enable it to reduce objects or distinctive features to their zero states and to recover them again.

We can tackle the problem in a simple form by attempting to provide the Look and Act STELLA of section 2.2 with a predictive model. But how do we model an environment-automaton with a state space like the following?



What practical system will prefer to model a sequence of behaviour, which could have come from the automaton shown above, as coming from the two separately-excited automata below?



A switches II ON and I OFF
 B switches I ON and II OFF

3.0 Miscellaneous Topics

It was intended to have accounts of the work of other members of the MMSG in this section, but the editor has already occupied too many pages on learning machines. The second Report will consider topics not mentioned here. There have been interesting developments in the School Timetabling Project of Robert Platts, in the Miniviews Display Project of Maurice Sloane, in the NIDDA computer modelling project also of Maurice Sloane, and in the plant modelling of Graeme Cox. The editor will complete the Report with a short discussion of maxEman.

3.1 maxEman

In man-machine studies the part played by the man is usually described in loose terms because man is such a complex system and because so little is known of his behaviour; an exception is to be found in the operator models for specific control tasks (e.g. Bekey, 1965, and Gaines, 1968), but it is difficult to extrapolate these to tasks where control concepts like the transfer function are inapplicable.

A proposal, made a year ago and reported with Robert Platts in Andreae (1972), attempts to identify a limiting and, to some extent, optimal human behaviour pattern. This behaviour pattern is that which would be expected of a human operator interacting with an effective console/display interface at maximum effort.

maxEman, which is short for maximum-Effort-man, refers to a human operator performing a task with maximum mental effort without pause or interruption. It is a name for a collection of hypotheses which we hope to improve and refine.

For the record and for comparison with the suggestions that follow, we reproduce the original memorandum proposing maxEman:

3.1.1 maxEman hypothesis

1. Man's adaption to a task is a procedure by which he organizes a task to match his somewhat limited recognition-decision-action abilities.
2. For maximal efficiency, man (maxEman) requires a task to appear in the following form and time-scale:-

abcabcabc.....

where a is a recognition/observation stage in which environmental stimuli are received and up to almost 7 entities distinguished;
b is a decision step in which some ordering of or preference for the distinguished entities is made;
c is an action stage in which output to the environment is synthesised;
and the minimum time for a/b/c is greater than 0.2/0.001/0.2 second.

3. There may be no advantage in distinguishing a and b.
4. The time 0.2 second is the minimum human reaction time approx.
5. The time 0.001 second is the minimum time which would be required for an afferent neuron to fire and stimulate an efferent neuron, approx.
6. Part of the process of adaption to a task will involve a pattern recognition development in which more appropriate (and, if necessary, more complex) entities are distinguished in stage a.
7. Part of the process of adaption to a task will involve an action synthesis development in which more appropriate actions are established.
8. A more appropriate recognition entity in (6) above may be seen as one which allows through step b the most drastic action to be effected in stage c without loss of control of the task.
9. A more appropriate action in (7) above may be seen as one which advances the task to a point at which further decision is necessary.
10. It should be remarked that the word "environment" in (2) above can refer to the human's internal "imagination-model", but that when it does the time for stages a and c may decrease.
11. Although there does not seem to be anything new in what is said above, it could form the basis for assessing machine assistance in man-machine studies (timetable program, plant model display) and it might be a step towards designing a learning machine which could "hurry".
12. Key questions which remain to be answered are (i) how does man distinguish success achieved through an improvement of recognition from that achieved through an improvement of action, (ii) does the optimal number of entities depend on the task, and (iii) how is distance from maxEman related to boredom, frustration, etc.?
13. Steps towards answering (12)(i) are being made in a Look and Act STELLA, comprising two STELLAments, the Act one rewarding the Look one for finding it an appropriate input pattern 3-12-1971
- PS. The E in maxEman should, perhaps, stand for effort rather than efficiency. How about a maxCman (C for comfort) etc.?! 13-12-1971

M.R.Mayson supported the change from efficiency to effort and suggested quantification of the hypothesis in terms of stages/time, stages/task and effort/rest. 14-12-71.

An alternative pattern for maxEman was suggested by Robert Platts in a memo reproduced here:-

4.0 References

- Andreae, J.H.(1962): "STELLA V" Internal Memorandum, Standard Telecommunication Laboratories, Harlow, Essex, 10th May.
- Andreae, J.H.(1963): "STELLA: A Scheme for a Learning Machine". 2nd IFAC Congress, Basle. Published in 1964 in Automation and Remote Control, ed. Broida. Proc.2nd IFAC Congress, p.503. Butterworths.
- Andreae, J.H. (1965) and Joyce, P.L.: "Learning Machine". Brit.Patents 1,011,685-7
- Andreae, J.H. (1966): "Computer Simulation of the Learning Machine STeLLA." Standard Telecommunication Labs. Internal memorandum (copies available from the author).
- Andreae, J.H. (1966'): Letter to Air Force Cambridge Research Laboratories. 19th August.
- Andreae, J.H. (1968): "STeLLA." Internal Memorandum, University of Canterbury, 21st April.
- Andreae, J.H. (1969) and Cashin, P.M.: "A Learning Machine with Monologue." Int.J.Man-Machine Studies 1 1-20 (Feb).
- Andreae, J.H. (1969'): "Learning Machines: A Unified View". (Submitted in 1966 to) Encyclopaedia of Linguistics, Information and control, ed. Meetham, Pergamon Press.
- Andreae, J.H. (1972), Gale, N.J., Henderson, K.D. and Platts, R.W.: "Developing an Interactive Aid for School Timetabling." Electrical Engineering Report No. 7 (May), University of Canterbury.
- Arbib, M.A. (1969): "Theories of Abstract Automata." Prentice-Hall Inc.
- Ashby, W.R. (1961) and Riguet, J.: "The Avoidance of Over-writing in Self-Organizing Systems." J.Theoret. Biol. 1 431-9
- Athans, M. (1971): "The Role and Use of the Stochastic Linear-Quadratic-Gaussian Problem in Control System Design." IEEE Trans. AC-16(6)529-552(Dec)
- Bekey, G.A. (1965): "Description of the Human Operator in Control Systems." in "Modern Control System Theory", ed.Leondes, McGraw-Hill Book Co.
- Brown, R.G. (1962): "Smoothing, Forecasting and Prediction of Discrete Time Series." Prentice-Hall Inc.
- Cashin, P.M. (1970): "Computing Procedures for a Learning Machine." Ph.D. Thesis, University of Canterbury.
- Feldbaum, A.A. (1960): "Dual Control Theory I-IV." Reprinted articles in "Optimal and Self-Optimizing Control", ed. Oldenburger, MIT Press.
- Freyd, P. (1964): "Abelian Categories." Harper and Row, N.Y.

- Gaines, B.R. (1966) and Andreae, J.H.: "A Learning Machine in the Context of the General Control Problem." Proc.3rd IFAC Congress, London.
- Gaines, B.R. (1967): "Stochastic Computing" AFIPS Conference Proc.30 SJCC.
- Gaines, B.R. (1968): "Training the Human Adaptive Controller." Proc.IEE 115, 1183.
- Hill, D.R. (1969): "Man-machine Interaction Using Speech." Advances in Computers 11, 165.
- Howard, R.A. (1960): "Dynamic Programming and Markov Processes." MIT Press.
- Kalman, R.E. (1960): "On the General Theory of Control Systems." First IFAC Congress, Basle. Published in Automatic and Remote Control, 1963, ed. Coales, Butterworths, London.
- Kalman, R.E. (1969), Falb, P.L. and Arbib, M.A. "Topics in Mathematical System Theory." McGraw-Hill Book Co.
- Kalman, R.E. (1971 and deClaris, N.: "Aspects of Network and System Theory." p.407, Holt, Rinehart and Winston Inc.
- Luenberger, D.G. (1964): "Observing the State of a Linear System." IEEE Trans. MIL-8(2) 74-80 (also AC-16(6) 596).
- Miller, G.A. (1967): "The Psychology of Communication." Penguin Books.
- Minsky, M. (1969) and Papert, S: "Perceptrons." MIT Press.
- Morrison, N. (1966): "Introduction to Sequential Smoothing and Prediction." McGraw-Hill Book Co.
- Noton, D. (1971) and Stark, L.: "Eye Movements and Visual Perception." Sci.American 224(6) 34 (June).
- Samuel, A.L. (1959): "Some Studies in Machine Learning Using the Game of Checkers." IBM.J. 3(3) 210-229 (July).
- Sperling, G. (1966): "Successive Approximations to a Model for Short Term Memory." reprinted in "Information Processing Approaches to Visual Perception." ed.Haber. Holt, Rinehart,Winston Inc, p.32
- Suppes, P.(1969): "Stimulus-Response Theory of Finite Automata." J.Math Psychology 6 327-355.
- Walter, W.G. (1951) "A Machine that Learns." Sci.American 185(2) 60-63.
- Widrow, B. (1971): "Adaptive Filters." in Kalman (1971) p.563
- Witten, I.H. (1970): "Performance of Some Old and New Adaptive Decision Strategies in Practical Machines." M.Sc. Thesis, University of Calgary, Alberta.
- Witten, I.H. (1970): "The STeLLA Predictor in a 'Clumps' Situation: An Approach to Implicit Categorization." Project Report No.CPSC 569, Dept of Mathematics, University of Calgary.

- - - - -

- 8th January 1973