9

BOXES: AN EXPERIMENT IN ADAPTIVE CONTROL

D. MICHIE and R. A. CHAMBERS
DEPARTMENT OF MACHINE INTELLIGENCE AND PERCEPTION
UNIVERSITY OF EDINBURGH

BOXES is the name of a computer program. We shall also use the word boxes to refer to a particular approach to decision-taking under uncertainty which has been used as the basis of a number of computer programs.

Fig. 1 shows a photograph of an assemblage of actual boxes—matchboxes to be exact. Although the construction of this Matchbox Educable Noughts and Crosses Engine (Michie 1961, 1963) was undertaken as a 'fun project', there was present a more serious intention to demonstrate the principle that it may be easier to learn to play many easy games than one difficult one. Consequently it may be advantageous to decompose a game into *a number of mutually independent sub-games* even if much relevant information is put out of reach in the process. The principle is related to the method of subgoals in problem-solving (see Newell *et al.* 1960) but differs in one fundamental respect: subgoals are linked in series, while sub-games are played in parallel, in a sense which will become apparent.

DECOMPOSITION INTO SUB-GAMES

The motivation for developing algorithms for small games (by a 'small' game we mean one with so few board positions that a boxes approach is feasible) needs explanation, since small games are generally too trivial to be of intellectual interest in themselves. The task of *learning* a small game by pure trial and error is, on the other hand, not trivial, and we propose that a good policy for doing this can be made useful as a component of a machine strategy for a large game. The point is that the board states of a large game may be mapped

on to those of a small game, in a many-to-one mapping, by incomplete specification. This is what the chess player does when he lumps together large numbers of positions as being 'similar' to each other, by neglecting the strategically irrelevant features in which they differ. The resultant small game can be said to be a 'model' of the large game. He may then, in effect, use his past experience *of the model* to select broad lines of play, and in this way guide and supplement his detailed analysis of variations in the large game. To give a brutally extreme example, consider a specification of chess positions so incomplete as to map from the viewpoint of White the approximately 10^{50} positions of the large game on to the seven shown in Fig. 2. Even this simple classification may have a role in the learning of chess. A player comes to

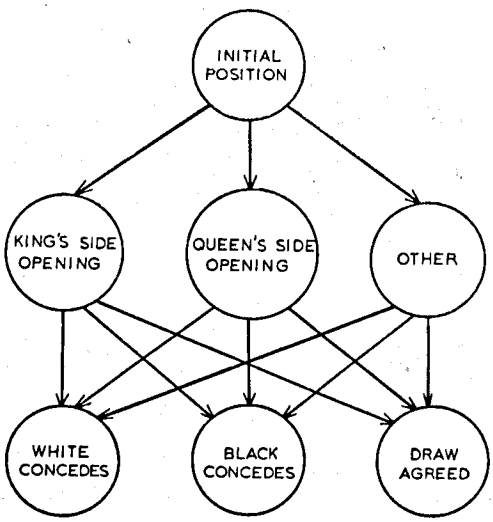


FIG. 2. A 'model' of chess.

realise from the fruits of experience that for him at least it pays better to choose King's side (or Queen's side, or other, as the case may be) openings. The pattern of master play, it may be added, has also changed, over the past hundred years, in ways which would show up even in this model, for example through decrease in the frequency of King's side openings and increase in the frequency of draws. Before leaving the diagram we shall anticipate a later topic by observing that it has the structure of a three-armed bandit with three-valued pay-offs.

In brief, we believe that programs for learning large games will need to have at their disposal good rules for learning small games. In the later part of this paper we apply the idea to an automatic control problem viewed as a 'large game'. As will appear, the approach is crude but the results to be reported show that decomposition into sub-games, each handled in mutual isolation by the same simple decision rule, is in itself sufficient to give useful performance in a difficult task.

As an example of the 'boxes' approach, consider again the matchbox machine of Fig. 1. This consists of 288 boxes, embodying a decomposition of the game of tic-tac-toe (noughts and crosses) into 288 sub-games, this being the number of essentially distinct board positions with which the opening player may at one time or another be confronted. Each separate box functions as a separate learning machine: it is only brought into play when the corresponding board position arises, and its sole task is to arrive at a good choice of move for that specific position. This was implemented in MENACE by placing coloured beads in each box, the colour coding for moves to corresponding squares of the board. Selection of a move was made by random choice of a bead from the appropriate box. After each play of the game the value of the outcome ('win', 'draw' or 'lose') was fed back in the form of 'reinforcements', i.e., increase, or decrease, in the probability of repetition in the future of the moves which in the past had led to the good, or bad, outcome. Echoing the terminology of Oliver Selfridge (1959), whose earlier 'Pandemonium' machine has something of the 'boxes' concept about it, one may say that the decision demon inhabiting a given box must learn to act for the best in a changing environment. His environment in fact consists of the states *of other boxes.* He cannot, however, observe these directly, but only the outcomes of plays of the game which have involved his box.

How should each demon behave?

PROBABILISTIC DECISION BOXES

In MENACE, and its computer simulation, choices among alternative moves were made initially at random, and after each play the probabilities of those moves which the machine had made were modified by a reinforcement function which increments move-probabilities following a win, leaves them unchanged following a draw and diminishes them following a defeat. In an alternative mode the value of the outcome is measured not from a standard outcome as baseline but relative to the past average outcome. With this sliding origin feature, a draw is reckoned a good result when defeat has been the rule, but a bad result when the machine is in winning vein. Before leaving the topic we shall make two remarks about adaptive devices based on the reinforcement of move-probabilities:

(i) such devices cannot be optimal;
(ii) it may nevertheless be possible in some practical contexts, to improve the best deterministic devices that we know how to design by incorporating a random variable in the decision function.

The basis of the second remark is connected with the fact that a move has an information-collecting role, and that a trade-off relation exists between expected gain of information and expected immediate payoff. The existence of this relation becomes apparent as soon as we try to devise optimal deterministic decision rules to replace *ad hoc* reinforcement of move-probabilities.

DETERMINISTIC DECISION BOXES

The task then is to draw up optimal specifications for the demons in the boxes.

In the present state of knowledge we cannot do this, as can be seen by considering the simplest learning task with which any demon could possibly be faced. Suppose that the demon's board position is such that his choice is between only two alternative moves, say move 1 and move 2. Suppose that his opponent's behaviour is such that move 1 leads to an immediate win in a proportion p_1 of the occasions on which it is used and move 2 wins in a proportion p_2 of plays. p_1 and p_2 are unknown parameters, which for simplicity we shall assume are constant over time (i.e., opponent does not change his strategy). The demon's task is to make his choices in successive plays in such a way as to maximise his expected number of wins over some specified period.

Under these ultra-simplified conditions, the problem is equivalent to the 'two-armed bandit' problem, a famous unsolved problem of mathematics. The difficulty can be expressed informally by saying that it can pay to make a move which is, on the evidence, inferior, in order to collect *more evidence* as to whether it really is inferior. Hence the problem can be formulated as that of costing information in the currency of immediate gain or loss: how much is the choice of a given move to be determined by its cash value for the current play of the game (we assume that all games are played for money) and how much by its evidence-collecting value, which may be convertible into future cash? We propose now to illustrate this formulation by exhibiting the behaviour of a game learning automaton designed by D. Michie to be optimal in all respects except that the evidence-collecting role of moves is ignored. R. A. Chambers' program which simulates the automaton is known as GLEE (Game Learning Expectimaxing Engine).

The learning automaton starts with a full knowledge of the moves allowed it and the nature of the terminal states of the game, but it is initially ignorant of the moves available to the opponent and only discovers them as they are encountered in play. As they are encountered, new moves are recorded and thereafter frequency counts are kept to record their use by the opponent.

Each terminal node of the game-tree represents a result of the game; in the application of GLEE to the game of Noughts and Crosses, a terminal node is given a utility value of $+1$ if the result is a win for the automaton, -1 for a loss and 0 for a draw. Non-terminal nodes are assigned scores which estimate the expected outcome value of the game. The automaton assigns these scores in the light of its previous experience and revises them as its experience is increased. Initially the scores are zero for all non-terminal nodes.

The graph is considered by levels, all nodes on the same level being produced by the same number of moves. Thus nodes at level one represent the states of the game produced by the opening move. Positions which can be equated by symmetry are represented by the same node (as is also the case with the MENACE automaton).

The revision of scores, i.e., the update, is done by backward analysis starting from the terminal node where the last game ended. The process moves back through the graph one level at a time. For levels where the automaton is on play each node is given a score equal to the maximum of the scores of nodes that can be reached by one legal move. Where the opponent is on play, his previous behaviour from the given node is considered and the node is assigned a score equal to the *expected value* of the outcome. We call this updating process 'expectimaxing'. The expected value calculation is derived as follows:

Consider a node, N, at which the opponent is to move and let him have used k different moves, each n_i $(i=1, \ldots, k)$ times in the past. The total number of alternative plays from N is unknown so we assume that c additional moves exist and reach nodes of score zero (the value 2 was taken for c in the experimental runs referred to later). The other moves end in nodes with scores s_i $(i=1, \ldots, k)$.

By a development of Laplace's Law of Succession we can determine the probability, p_i, that the next opponent move from N will use the ith alternative, i.e.,

$$p_i = \frac{n_i+1}{\text{Estimated number of alternatives} + \text{total number of past plays from } N.}$$

$$= \frac{n_i+1}{(k+c)+\sum_{i=1}^{k} n_i}$$

Knowing those values of p_i $(i=1, \ldots, k)$ and the scores reached by each alternative we can calculate the quantity.

$E=\sum_{i=1}^{k} p_i s_i$. This defines the score associated with the node N.

To make a move the automaton examines all the legal alternatives and chooses the move leading to the position having the highest associated score, ties being decided by a random choice. It thus seeks to optimise the expected outcome of the *current* play of the game only. Fig. 3 shows the results of three trials each of 1000 games in which the opening player was a random move generator and the second player was the learning automaton. The automaton's score is shown as the number of wins minus the number of losses over each 100 consecutive games. By analysis of the optimal move trees associated with each type of opening play, the score level for an optimal strategy was calculated. It can be seen from Fig. 3 that the performance of the learning automaton levels out below this optimal level. This is due to an inherent weakness in the automaton's aim of short-term optimisation. The weakness is that as soon as the automaton finds, in a given position, a move with a positive expected outcome, then since other tried moves have had negative outcomes and unknown moves have zero expected outcomes it will continue to use that move as long as its expected outcome stays just greater than

zero. In this way it may be missing a better move through its lack of 'research'. Thus we see that neglect of evidence-collecting can lead to premature decision taking. This can be demonstrated in detail by reference to the very simple game of two-by-two Nim.

In this trivial case of the game there are two heaps of two objects and each of the two players in turn removes any number of objects from any one heap. The winner, according to the version considered here, is the player to remove

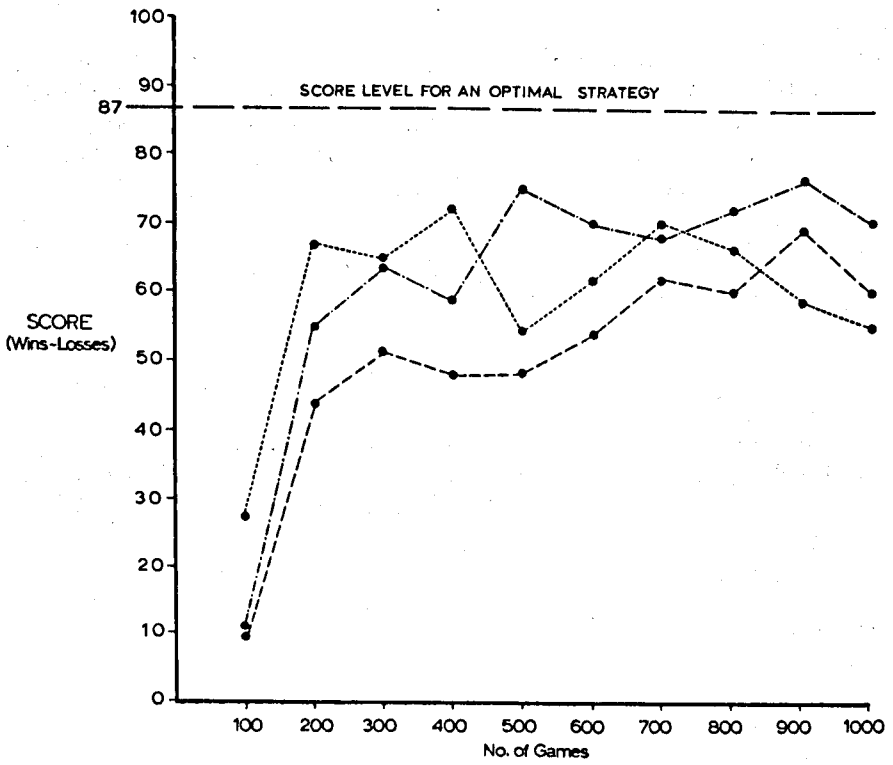


FIG. 3. Some representative results of GLEE playing Noughts and Crosses as second player, the opening player being a random move generator. The vertical scale gives hundred game totals.

the last object. Let the automaton be the opening player, then symmetry reduces the choice of opening moves to two. Against a good opponent the automaton will always lose the game, and both openings will be equally bad. But consider the case of a random opponent. Fig. 4 shows a graph representation of the game. For his first reply the opponent has three alternative moves from node (A) and two from node (B). As the opponent plays randomly, games proceeding from (A) will on average give the automaton two wins for every loss while those games proceeding from (B) will yield equal numbers of wins and losses. Thus in the long term the automaton's best opening move is to select position (A). Consider now a sequence of four games as follows:

the automaton chooses (A) and loses, (B) and wins, then (B) twice more losing each time. Fig. 5 shows the state of the automaton's knowledge with the assumed and as yet unknown opponent moves. Applying expectimax, as

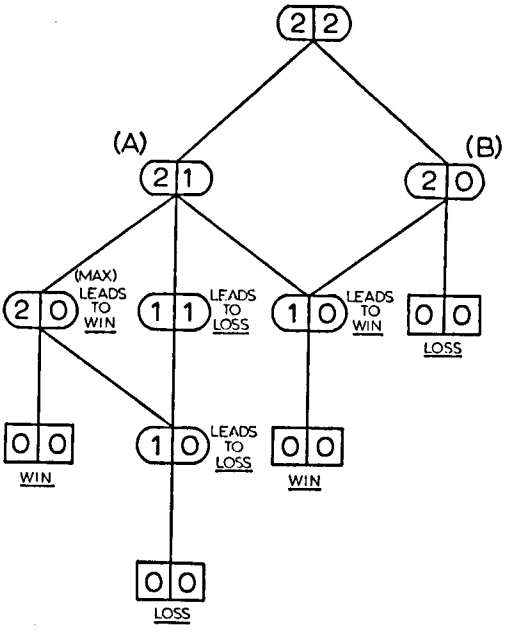


FIG. 4. 2 × 2 NIM showing winning and losing moves. The integer pairs at nodes indicate the state of play by showing the number of items in each of the two heaps.

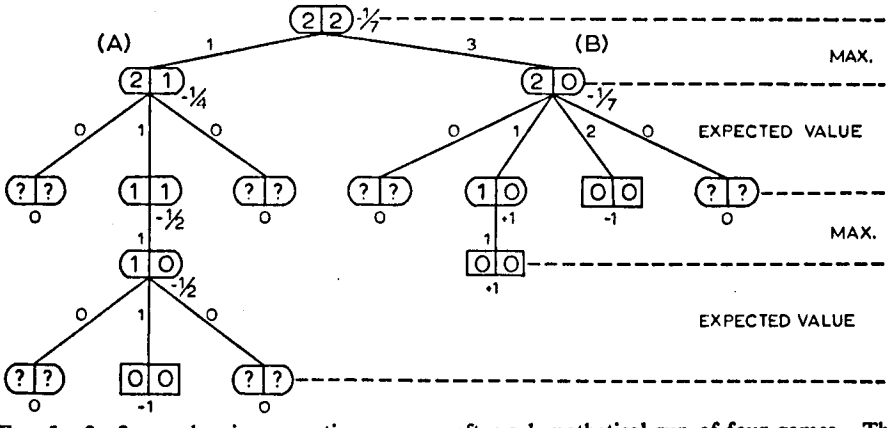


FIG. 5. 2 × 2 NIM showing expectimax scores after a hypothetical run of four games. The numbers against nodes are scores and the integers against graph connections show move frequencies.

explained, shows that due to the premature decision taking, brought on by losing at (A) and winning first time at (B), the automaton will still choose (B) despite the last two losses.

This feature of premature decision taking was also very evident in the trials of the automaton at noughts and crosses against a random opponent. Table 1 illustrates a series of games in which the automaton was second player and shows all the games in the first 100 of the series in which the random opponent opened in the centre. Analysis of this opening shows that by symmetry there

TABLE 1

Some selected results from the GLEE game-learning program using the game of Noughts and Crosses (tic-tac-toe). The automaton is the second player against a random opponent. Only those plays are shown in which the opponent opened to the centre of the board. For further explanation see text.

No. of each game in which random opponent opened in centre	Automaton's first reply: M=middle of side C=corner	Result for automaton
7	C	Lost
12	C	Lost
23	C	Lost
26	M	Won
27	M	Won
28	M	Won
31	M	Drew
37	M	Won
39	M	Lost
40	M	Won
44	M	Won
55	M	Lost
56	M	Won
61	M	Won
62	M	Lost
65	M	Lost
66	M	Lost
67	M	Lost
69	M	Won
74	M	Lost
77	M	Won
80	M	Lost
83	M	Lost
87	M	Won
89	M	Drew
90	M	Lost
92	M	Lost
95	M	Won
98	M	Drew
100	M	Lost

are only two distinct replies, a corner, C, or the middle of a side, M, the first being superior to the second. Indeed against an optimal strategy the second player always loses after playing M. However against a random opponent the automaton chanced to lose three games using the first reply, C, and then to win three using the second, M. This was sufficient to cause it to play the second, poorer, move throughout a trial of 2000 games. In the first 1000 games

the random player opened 336 games with the centre move with the automaton replying by moving into the middle of a side, M. In a similar series of 1000 there were 320 games with centre opening but this time the automaton always played in the corner, C. The results as percentages were as follows:

Automaton replying M 62% wins 18% losses 20% draws
Automaton replying C 75% wins 16% losses 9% draws

A remedy for this fault, which illustrates again the information-versus-payoff dilemma of trial-and-error learning, could be developed by use of a method which we have incorporated in our application of the boxes idea to a problem in adaptive control. The program is called BOXES and the method is called the 'target' method.

ADAPTIVE CONTROL AS A STATISTICAL GAME

BOXES is based on formulating the adaptive control problem in terms of the 'game against nature'. A valuable and detailed survey of formulations of this type has recently been published by Sworder (1966), who restricts himself in the main to theoretical questions of optimality. The boxes algorithm was devised by D. Michie for tasks for which optimal policies cannot be specified, and implemented in 1961 by Dean Wooldridge, junior, as a FORTRAN II program. It is incomplete in a number of ways. Yet it illustrates clearly the idea, expressed above, of reducing a large game to a small, model, game and then handling each separate board state of the model as a separate sub-game (box). In the adaptive control situation, where the state variables are real numbers, the large game is infinitely large, so that the sacrifice of information entailed in the boxes approach is correspondingly extreme. In spite of this, the algorithm actually does master the simulated task, which is one which severely taxes conventional methods.

BALANCING A POLE

The model task chosen for experimentation was that of balancing a pole on end. For the purpose in hand, this system has a number of merits:

(1) it has been thoroughly studied by Donaldson (1960) in an illuminating exercise on a related, but different, theme—namely the design of an automaton to learn a task by sensing the control movements made by a second automaton already able to perform this task (e.g., a human being);
(2) Widrow and Smith's (1964) subsequent and independent approach to Donaldson's problem used a design reminiscent of the boxes principle;
(3) a recent study in automatic control (Schaefer and Cannon 1966) has shown that the pole-balancer problem generalises to an infinite sequence of problems of graded difficulty, with 1, 2, 3, . . . , etc., poles balanced each on top of the next. There is thus ample scope, when needed, for complicating the problem;

(4) even the 1-pole problem is extremely difficult in the form for which adaptive methods are appropriate, i.e., physical parameters of the system specified in an incomplete and approximate way, and subject to drift with the passage of time. No optimal policy is known for such problems in general.

Fig. 6 shows the task. A rigid pole is mounted on a motor-driven cart. The cart runs on a straight track of fixed length, and the pole is mounted in such a way that it is only free to fall in the vertical plane bounded by the track.

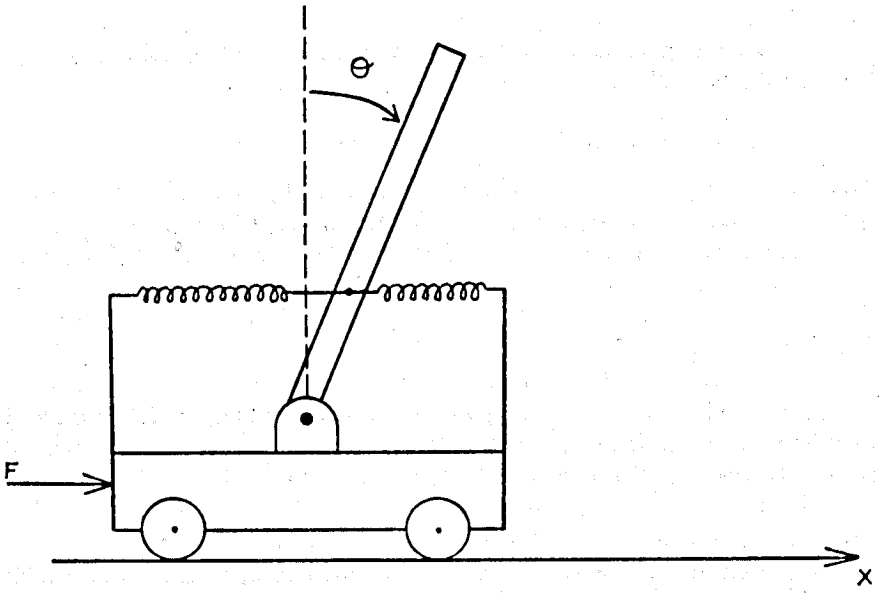


FIG. 6. A simple representation of the system being controlled. The spring is used to reduce the effective force of gravity. This feature was incorporated in the original laboratory apparatus so as to bring it within the range of human powers of adaptive control.

The cart's motor applies a force which is constant except for the sign. The sign is controlled by a switch with only two settings, + and −, or 'left' and 'right'. The problem is thus one of 'bang-bang' control. For the experimental runs reported here the cart and pole system was simulated by a separate part of the program, not built in hardware. The interval from sense to control action was set to zero and the interval from control action to sense to 0·05 sec. Various other parameters of the simulation program corresponding to such quantities as frictional resistance, length of pole, the mass of the pole and of the cart, and the spring clamping of the pole, were adjusted until the behaviour of the simulated system approximated to that of a real physical system.

BLACK BOX

The form in which we pose the problem is this. The adaptive controller must operate without prior knowledge of the system to be controlled. All it knows

is that the system will emit signals at regular time intervals, each signal being *either* a vector describing the system's state at the given instant *or* a failure signal, indicating that the system has gone out of control. After a failure signal has been received, the system is set up afresh and a new attempt is made. On the basis of the stream of signals, and this alone, the controller must construct its own control strategy. No prior assumptions can safely be made about the nature of the system which emits the signals, nor of the time-invariance of its parameters. In this sense, the task is never finished, since parameters of the system might drift into a new configuration requiring further modification of the controller. Most 'black box' problems permit some time-invariance assumptions. Our program must be ready for an even blacker box.

STATE SIGNALS

The state of the system at any instant can be represented by a point in an n-dimensional space of which the axes correspond to the state variables, n in number. In our case it is convenient to recognise four such variables,

(i) x, the position of the cart on the track,
(ii) θ ,the angle of the pole with the vertical,
(iii) $\dot{x}$, the velocity of the cart, and
(iv) $\dot{\theta}$, the rate of change of the angle.

(iii) and (iv) could be estimated by differencing (i) and (ii) with respect to time interval, but the program has these sensed separately. The state signal thus consists of a 4-element vector.

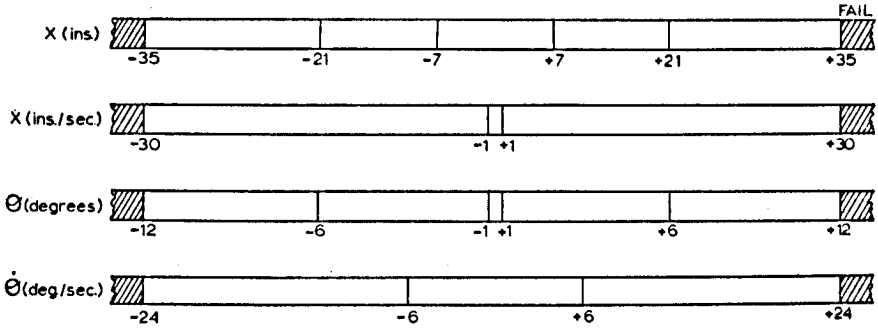


FIG. 7. Thresholds used in quantizing the state variables.

The 'model' was constructed by quantising the state variables by setting thresholds on the four measurement scales as shown in Fig. 7: thus, only 5 grades of position, x, were distinguished, 5 of angle, 3 of velocity and 3 of angel-change. It can be seen that this quantisation cuts the total space into a small number of separate compartments, or boxes; in our implementation the number was $5 \times 5 \times 3 \times 3 = 225$.

DECISION RULES

In order to envisage how the rest of the algorithm works it is easiest to imagine each one of the 225 boxes as being occupied by a local demon, with a global demon acting as a supervisor over all the local demons, according to the

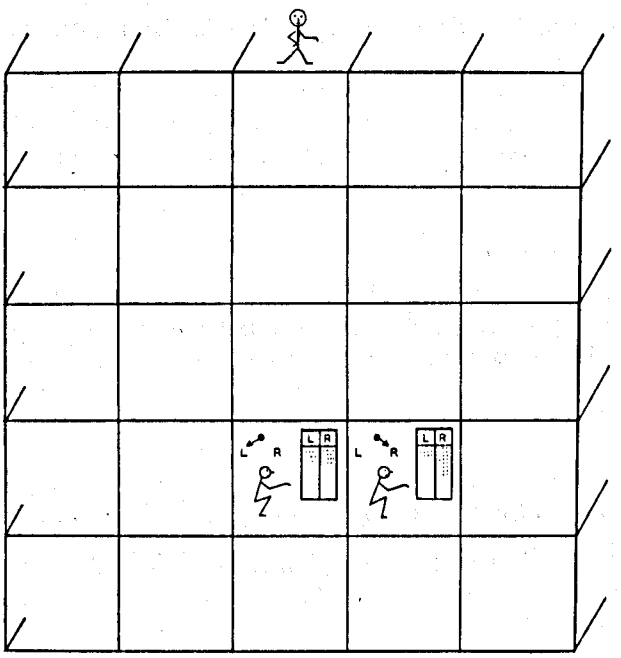


FIG. 8. A simple representation of the independent decision mechanisms controlling each box in the state space.

general scheme of Fig. 8. Each local demon is armed with a left-right switch and a scoreboard. His only job is to set his switch from time to time in the light of data which he accumulates on his scoreboard. His only experience of the world consists of:

(i) *LL* the 'left life' of his box, being a weighted sum of the 'lives' of left decisions taken on entry to his box during previous runs. (The 'life' of a decision is the number of further decisions taken before the run fails.)

RL the 'right life' of his box.

LU the 'left usage' of his box, a weighted sum of the number of left decisions taken on entry to his box during previous runs.

RU the 'right usage' of his box.

(ii) *target* a figure supplied to every box by the supervising demon, to indicate a desired level of attainment, for example, a constant multiple of the current mean life of the system.

(iii) $T_1, T_2, \ldots, T_i, \ldots, T_N$ times at which his box has been entered during the current run. Time is measured by the number of decisions taken in the interval being measured, in this case between the start of the run and a given entry to the box.

The demon's decision rule gives S, the setting of his switch, as a function of (i) and (ii). We can express this informally by saying that when the system state enters his box for the first time, the demon sets S by applying his decision rule, and notes the time of entry, T_1; if his box is re-entered during the run, the demon notes the new entry time, T_i, but does not change his switch, so that the same decision, S, is taken at every entry to his box during the course of that run. When the run fails, the global demon sends a failure message to all the local demons, indicating the time of failure, T_F, and also giving a new value of *target*. Each local demon then updates the 'life' and 'usage' totals for his box so that with these new values and the new value of *target* he can make a new decision if his box is entered during the next run.

The rules for updating and resetting adopted in the present case were as follows, using the convention that, say, X' means 'the previous value of X':

Consider a box in which the decision setting, S, was *left*, during the run which has just terminated.

Let N = the number of entries to the box during the run;
DK = constant multiplier less than unity which performs the function of weighting recent experience relative to earlier experience;
K = another multiplier weighting global relative to local experience;

then the local demon updates his totals, defined under (1) above, using:

$$LL = LL' \times DK + \sum_{i=1}^{N} (T_F - T_i);$$

$$LU = LU' \times DK + N;$$

$$RL = RL' \times DK;$$

$$RU = RU' \times DK.$$

The global demon has similar totals, 'global life', GL, and 'global usage', GU, and updates them using:

$$GL = GL' \times DK + T_F;$$
$$GU = GU' \times DK + 1;$$

From this the demon calculates *merit* as $\frac{GL}{GU}$,

and then computes *target* as $C0 + C1 \times merit$,
where $C0 \geqslant 0$ and $C1 \geqslant 1$.

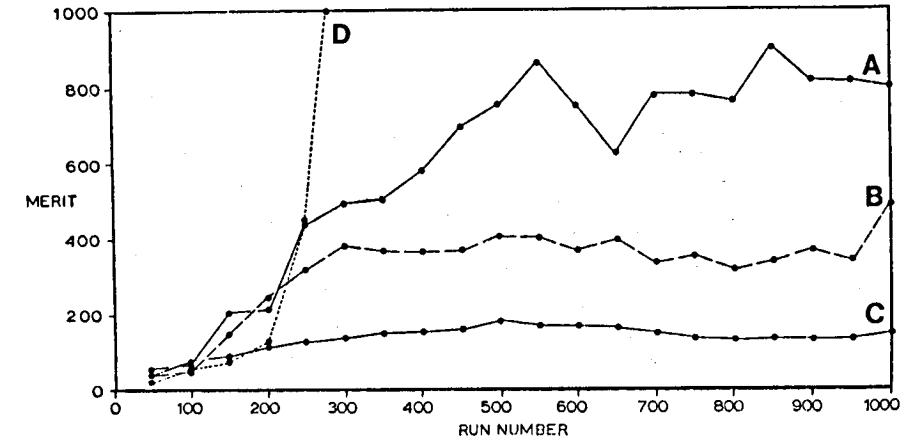
From *target* and his local totals, the local demon calculates the 'left value' of his box from

$$value_L = \frac{LL + K \times target}{LU + K}$$

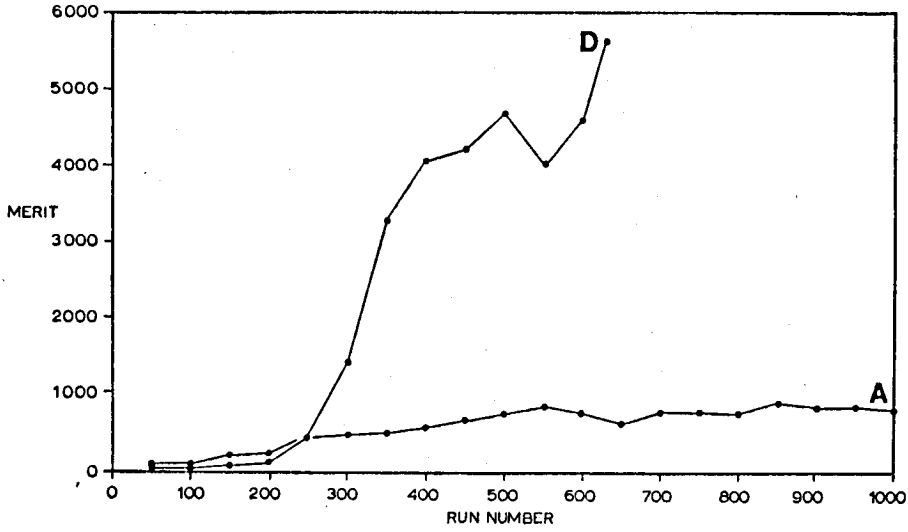
and similarly the 'right value', $value_R$. The control action, S, is selected as *left* or *right* according as $value_L$ is greater or less than $value_R$.

PERFORMANCE TESTS

After some preliminary trials the adjustable parameters *DK* and *K* were set at 0·99 and 20·0 respectively for the tests. The values of *C0* and *C1* were 0 and 1 respectively, throughout all the trials. Time has not allowed



(*a*)



(*b*)

FIG. 9. Duplicate runs of BOXES. Run A is shown on both graphs to facilitate comparisons.

any systematic attempt at optimisation since the decision algorithm has only recently been developed to the stage reported here. A test run consisted in a succession of attempts by the program, each attempt being terminated by receipt of the failure signal. For each new attempt, or 'control run', the

system state was set up by choosing a random point from a uniform distribution over the central region of the space defined by $x = \pm 21{\cdot}0$ in, $\dot{x} = \pm 1{\cdot}0$ in/sec, $\theta = \pm 6{\cdot}0$ deg and $\dot{\theta} = \pm 6{\cdot}0$ deg/sec.

Figs. 9 (*a*) and (*b*) show duplicate test runs A, B, C and D. In these test runs the value of *merit* (explained in the previous section) was recorded after the update which followed the failure of each control run, and the graphs show these values for every fiftieth control run. For each test run the same values were kept for all the adjustable parameters, but different starting values were given to the pseudo-random number generators which were used for resolving control decisions with equal left and right values and for setting up the new starting conditions of the system for each control run.

The wide limits within which starting conditions could be generated caused considerable variation in the duration of control runs. This is not obvious from the individual graphs because of the pronounced smoothing effect of the weighted-average calculation by which *merit* is computed, but in fact the length of extreme runs was a factor of 10, or more, greater or less than *merit*. Thus test run C produced a run of 1368 decisions at a point where *merit* was only 146, and test D, at a point with a *merit* value of 4865, produced a run of over 72 000 decisions which is equivalent to an hour of real-time control.

The use of different random number sequences in test runs, combined with this variation in the duration of control runs, means that the sequences of boxes entered by the system and the early decisions taken in those boxes differed greatly for each test run. Since the 'success' of any box depends on the decisions taken in neighbouring boxes, it is easy to see how the early decision settings for the whole collection of boxes were different for different test runs, whereas the low level of experience caused the control runs to be short and thus produce similar values of *merit* for all the tests. However, because of premature decision-taking fixing some of these early decision settings, the later behaviour of the tests varied considerably as can be seen in the graphs of Fig. 9, but the influence of the *target* term in the decision rule is noticeable even without optimisation. Graphs B and C are comparable, respectively, to the best and average performances of earlier, non-*target*, versions of BOXES. Graph A is better than any previous performance and D indicates what we hope to achieve when the target method has been improved.

'LEVEL OF ASPIRATION'

Earlier in this paper we referred to the fact that a move which on the evidence is disadvantageous may none the less be worth making if the expected information-gain associated with it is sufficiently high, and we emphasised that this fact should be reflected in the design of a trial-and-error automaton.

The problem of optimal information trade-off, as stated earlier, still awaits solution even in the simplest case—the 'two-armed bandit' problem. We have, in the meantime, evolved an approximate solution to this family of problems, and have found that it shows promise of being able to keep the 'premature

fixation' hazard at bay. It can moreover be made to yield, in some contexts at least, a rather close approach to optimality as we have verified by comparison with a limited range of optimal values computed for the two-armed bandit problem. The essential idea is to use a 'target' level of performance to calculate for the available alternative moves *not* their expected payoffs but 'optimistic' values of these expectations. Choice between moves is then made by comparing the optimistic values. These are calculated by taking a weighted mean between the expected value of the move and the target value. The weight of the expected value is taken as the amount of information associated with it, while the weight of the target value is an adjustable parameter of the program. The higher the target value and the greater its weight, the more 'research minded' is the behaviour of the automaton, tending to try new possibilities rather than to consolidate partial successes. Its 'level of aspiration', in anthropomorphic terms, is higher.

The idea is readily generalised, and can be immediately applied, for example to the GLEE automaton, or to the two-armed bandit itself. We have only recently started to explore this line: even the results of Fig. 9 were obtained before any optimisation of the parameters of the target method was attempted. We can, however, say in summary that the target method, when grafted on to the 'boxes' approach, has given good performance for adaptive control problems inaccessible to standard procedures.

REFERENCES

Donaldson, P. E. K. (1960). Error decorrelation: a technique for matching a class of functions. *Proc. III International Conf. on Medical Electronics.* pp. 173-178.

Michie, D. (1961). Trial and error, in *Science Survey, 1961* (eds. S. A. Barnett and A. McLaren) part 2, pp. 129-145. Harmondsworth: Penguin.

Michie, D. (1963). Experiments on the mechanisation of game-learning. 1. Characterisation of the model and its parameters. *Computer Journal*, 6, 232-236.

Newell, A., Shaw, J. C., & Simon, H. A. (1960). A variety of intelligent learning, in a general problem solver in *Self-organizing Systems* (eds. Marshall C. Yovits and Scott Cameron) pp. 153-189. London: Pergamon.

Schaefer, J. F., & Cannon, R. H. Jr. (1966). On the control of unstable mechanical systems. *Proc. IFAC 1966.* Paper 6C.

Selfridge, O. G. (1959). Pandemonium: a paradigm of learning, in *Mechanization of Thought Processes*, Vol. 1. N.P.L. Symposium No. 10, pp. 511-531.

Sworder, D. D. (1966). *Optimal Adaptive Control Systems.* Academic Press.

Widrow, B., & Smith, F. W. (1964). Pattern recognising control systems, in *Computer and Information Sciences* (eds. J. T. Tou and R. H. Wilcox). Clever Hume Press.