Experiments on the Mechanization of Game-learning
2—Rule-Based Learning and the Human Window

Donald Michie
Machine Intelligence Research Unit, Hope Park Square, Edinburgh EH8 9NW, Scotland, UK

Machine learning involves the modification or creation by program of stored information structures, so that machine-deliverable information becomes more accurate, larger in amount, or cheaper or faster to deliver. A further desideratum, concerned with intelligibility to the user, is reviewed in the light of recent work on computer induction.

INTRODUCTION

The first successful learning programs were developed in the 1950s and belonged to a general category which was at that time commonly known as 'hill-climbing'. Global mathematical models of system performance were typically constructed in forms permitting multi-dimensional representation in systems of orthogonal co-ordinate axes. Numerical parameters were then automatically tuned at run time in response to sensed deviations from computed criteria of optimality. This scheme embraces classical adaptive control, along with many studies of machine learning in games and game-like situations. How the problem appeared to AI workers of that epoch can be gleaned from the proceedings of a celebrated symposium held in 1957 under the title: *Mechanisation of Thought Processes* (HMSO, London).

The present paper is the second of a series. The first appeared nearly 20 years ago and proposed a new principle for attaining the above purposes,[1] now known as 'rule-based' learning. The idea was to partition the problem-domain into a mosaic of smaller sub-domains and to associate a separate rule of action with each. In the pre-learning state a rule may be stochastic or even vacuous. In the learning mode, entry into a given sub-domain invokes a global procedure for collecting data on the sensed consequences of executing the associated rule and for up-dating the rule's content in the light of these. A worked toy example was supplied in the form of a computer simulation of a machine for learning to play Noughts and Crosses (Tic-Tac-Toe). Distinct equivalence classes into which the positions can be grouped were taken as the separate sub-domains. These were represented as separate 'boxes', as in a card-filing system.

Successful tests of the 'boxes' principle were subsequently made on a hard dynamical problem, namely automatically controlling the support point of an inverted pendulum within a bounded space. The adaptive pole-balancer was required to deliver 20 left-right decisions per second to a motor-controlled cart on which was balanced a pole free to move in the vertical plane defined by a straight bounded track (Fig. 1).[2] The task of the BOXES program was to acquire by trial and error, or from being shown by a human tutor, or from a combination of both, the ability to control the cart within the bounds set by the ends of the track without permitting the pole's angular deviation from the vertical to exceed a pre-set tolerance. For experimental runs the precise specification was: the system fails if any of four monitored variables (position on track, velocity, pole angle, angular velocity of pole) pass outside fixed bounds. Initially decisions were taken randomly, and fail-free periods were measured in seconds. After learning, the fail-free periods lasted half an hour or more.

BOXES was the first system to be driven by a set of independently modifiable production rules, and thus foreshadowed today's 'expert systems'. The mode of learning was primitive, being confined to revising the action-recommendations associated with stored situation-patterns, the latter being fixed. The next series of experiments at Edinburgh, involving computer-coordination of hand-eye robots, focused on the situation-perception component of machine learning.

In the FREDDY robot work, the learning module built new patterns in memory as the basis of adaptive perception of situation-categories. The stored structures were descriptions in semantic net form of the visual appearances of various objects such as cup, spectacles, axle, wheel, and so forth. From these the system was required at run time to recognize instances from images sampled from the television camera acting as the robot's eye. In robot vision 'programming by example' becomes a necessity. The infeasibility of programming in the ordinary sense is apparent if one compares the police task of identifying a culprit from photographs with identification purely on the basis of verbal description. The Edinburgh versatile assembly program was thus operating in the foothills of a domain now commonly

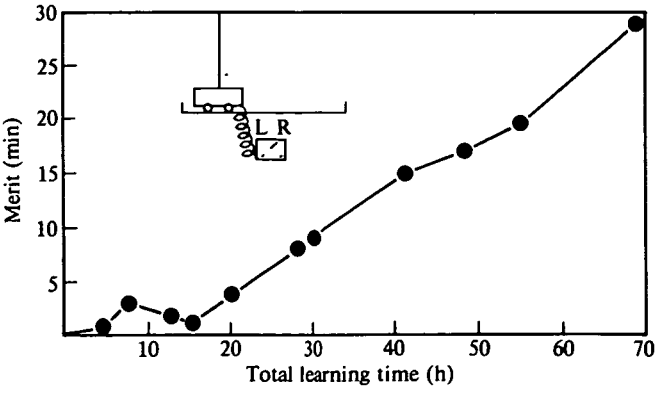

Figure 1. An illustration of the BOXES program to control a motor-driven cart running on a track of fixed length and balancing a pole by a set of independently modifiable production rules.

CCC-0010-4620/82/0025-0105 $04.50

termed 'computer induction'. Although the method for updating the stored relational structures was crude, the descriptive power of these structures and of the algorithms for manipulating and matching them was worthy of note.

Quinlan's ID3 program, developed from Hunt's earlier proposals,[3] builds internal descriptive structures in the form of decision trees. They are executed as PASCAL code. Tests were made using patterns from the King-Rook-King-Knight end-game in chess which are straightforward for chess-masters to recognize but effectively impossible to program. The induced decision trees exemplify the *cheaper information* category. Non-learning solutions are possible, and range from exhaustive search to prior computation and storage of a look-up table of 428 distinct feature-vectors paired with decision-classes.

Comparative economics are given in Table 1. The first information structure is a calculation-intensive program, hand-coded along conventional lines. The second is a look-up table of attribute combinations obtained from an exhaustive computation, together with hand-coded routines for table-access. The third is a decision tree synthesised by Qunilan's ID3 program by program controlled sampling from the above table. A fourth representation, not shown here, was Quinlan's own best effort at a hand-coded pattern-directed search. This proved to be about 30% less economical than the machine-made representation. For a more complex problem the differential rose to 400%.

Table 1. Three information structures for guiding a computer in performing a difficult classification in chess

Classification method	Execution time (msec)	Memory required (× 1K words)	Time × memory
Minimax search	7·67	2.1	16.1
Look-up	1.12	67.7	75.8
Machine-synthesized decision tree	0.96	2.5	2.4

RULE-BASED SYSTEMS

The stored information structures through which machine learning is mediated cannot usually be described unequivocally as either program or data. They lead a double life, being in fact *rules*. During learning, a rule is treated as a data structure. The process is then akin to editing. When a rule is fired at runtime, it is executed as program. For further learning to occur, some trace must be saved of the result.

A rule is something which says

if ⟨condition⟩ **then do** ⟨action⟩.

Satisfaction of the condition part is detected by matching against a database which models the ongoing state of the task environment. Execution of an action normally leaves a mark on the database, either by updating it directly or by affecting the environment which the program is sampling.

Thus, for the task depicted in Fig. 1 the 'database' is just a one-dimensional array of four numbers depicting the state last sensed by the program. We can imagine a rule like that shown in Fig. 2(a). Let us suppose that its condition part matches the current state. The effect of rule-execution will be to create a new state corresponding to one of the 16 successor-states constructible from the possible values listed in Fig. 2(b). Thereupon the new successor-state is sensed and the database updated. The matching process restarts, and the cycle iterates until one or more of the state variables passes outside permitted bounds and a failure is signalled.

(a) **if** cart is far left
  and cart is hardly moving
  and pole is hardly leaning
  and pole is hardly swinging
then do
  rightwards
end

(b) cart position: far left, somewhat left;
cart velocity: hardly moving, moving right;
pole angle: hardly leaning, somewhat left;
pole swing: hardly swinging, swinging left

Figure 2.(a) A specimen rule for the pole-and-cart problem, for which the successful strategy consisted of 225 rules of this form. **(b)** On application to a situation matching the rule's condition part, a new situation results which must be one of these 2 × 2 × 2 × 2 derivable possibilities.

At this point the learning routine has the opportunity to perform a post-mortem, and it is likely that the rule displayed (Fig. 2(a)) will be implicated. The advice it gives (in effect to move the cart to safety away from the edge) looks sensible; but, because of the effects upon the behaviour of the pole, it is short-sighted. As a result of the rightwards acceleration imparted by the prescribed action, the pole will soon present a dire problem: how to prevent it from toppling over leftwards without first having to run the cart back and possibly beyond the left end of the track.

In the experiments from which this example is taken, a cumulative tally of statistical evidence on the results of actions was maintained for the complete set of 225 rules. The tally was automatically updated for every rule used during a given run. When, as in the case used for illustration, the rule is faulty, then sooner or later the balance of evidence tips against it and the rule's action part is edited by substitution of 'leftwards' for 'rightwards', or *vice versa* as the case may be.

The BOXES program also exemplified McCarthy's dictum: 'In order for a program to be capable of learning something it must first be capable of being told it.'[4] By dint of much practice, R. A. Chambers was able to acquire moderate expertise in performing the control task himself. In this mode the human decisions were signalled to the machine by light pen. In each cycle the light pen was first interrogated for a decision. If none was found, then the program's own decision was implemented instead. The remainder of the cycle, including the interpreting and gathering of statistical evidence, proceeded just the same. BOXES thus harvested advice from an expert tutor hand in hand with trial-and-error accretion. It was able to do this only because the production-rule representation which it introduced was *appropriate for the reception of advice*. This program was the earliest working rule-based system of the modern kind.

Meanwhile Eastwood[5] had independently constructed a solution to the pole-and-cart problem by use of the Newtonian dynamical equations together with classical control theory. Thanks to this, the BOXES work was able to provide a vantage point for looking at orthodox control engineering from an artificial intelligence perspective. The conclusion was that the executable strategy arrived at by tuning a collection of situation-action rules is more economical in operation than the complex fabric of partial differential equations in which the classical adaptive control solution is expressed. Crunching numbers is more costly than situation-action look-up. For problems more complex than pole-balancing, such as controlling a bicycle, the mathematical modelling approach can be expected to become intractable if real-time operation is desired. This may be related to the fact that children are not sent on courses in mechanics, calculus, and control theory in order to help them learn to ride bicycles. Tutorial preparation for the task of designing, rather than being, a bicycle-rider would be another matter. The point at issue, not widely appreciated, concerns the adequacy of machine representations to this or that purpose, and in particular the tradeoff which exists between *heuristic adequacy* and *epistemological adequacy* (see Ref. 6 for definitions of these terms). The BOXES representation is a purely heuristic model; that is to say, it models a skill. The differential equation representation seeks primarily to model the causal and predictive structure of the domain in which the skill is to be exercised. The contrast can be expressed as follows:

Heuristic model: situations → actions
Causal model: situations × actions → situations.

To construct a sophisticated predictive model when all that is required is *performance* of a skill (as opposed to its explication) is to build a cannon to swat a fly. Not only may the cost of firing it be prohibitive, it may not even win on the performance criterion.

If, on the other hand, the primary requirement of the user is that the system should act as a source of scientific understanding rather than of engineering performance, then of course the matter stands differently. In this connection the possibility of automatically generating heuristic models from causal models deserves investigation. Some preliminary studies have been made.[7]

TWO LEVELS OF MACHINE LEARNING

In the BOXES algorithm the action parts of rules are modified. Biologically, we would speak of the *reinforcement of responses* to a fixed set of *stimuli*. The equivalent in ordinary computation would be the construction of a tabulation of some mathematical function (either by calculation or by 'being told') in which individual entries in the look-up table are tested in the field and corrected in the light of their track-records. In the POP-2 programming system, facilities for doing this sort of thing are provided by a library facility known as 'memo functions'.[10]

The more interesting forms of learning belong to the complementary type, in which the condition parts of rules are changed. Biologically, we would speak of *classification of the stimulus*. In computation, where we might speak of 'situation learning' in contrast to 'action learning', a counterpart would be the restructuring of a look-up table without necessarily changing its entries. A version of such a procedure will be familiar to anyone who has done a tabulation in 'critical table' style as commended by Comrie in his explanatory introduction to *Chambers' Mathematical Tables*[9] (see Table 2). A program which minimally resets where necessary the bounds on argument-intervals to accommodate newly computed entries represents such machine learning. The machine dynamically revises its remembered *situations*, i.e. the argument-sets, found to map onto given result-values. Van Emden's POP-2 implementation of memo-functions allows this to be done.[10]

Table 2. (a). Comrie's explanation of 'critical tables'. (b) Comrie's account of the construction of 'critical tables'.

(a)

x	$\sin x$
0	
	0.0
2	
	0.1
8	
	0.2
14	
	0.3
20	
	0.4
26	
	0.5
33	
	0.6
40	
	0.7
48	
	0.8
58	
	0.9
71	
	1.0

A function is normally tabulated at equal intervals of the independent variable or argument. If, however, the function is changing slowly in its last figure, it is possible to write down *all* the values that the function may assume, and then to show the limiting values of the independent variable that lead to each function value. Suppose we wanted sines to one decimal from angles in the first quadrant known to the nearest degree. The necessary critical table is shown alongside.

In this form the argument (here x) is on the left and the respondent (here $\sin x$) on the right as usual, but on lines half way between those of the argument. Thus any angle *between* 20° and 26° has a sine of 0.4.

(b)

$\sin x$	x
	0
0.05	2.9
0.15	8.6
0.25	14.5
0.35	20.5
0.45	26.7
0.55	33.4
0.65	40.5
0.75	48.6
0.85	58.2
0.95	71.8

The method of preparing such a table is to compute the values of x corresponding to half-way values of $\sin x$, as shown in the table alongside. The argument in critical tables is never rounded *off*. It is rounded *down* if it is increasing numerically, and *up* if it is decreasing numerically, regardless of whether the respondent is increasing or decreasing. Thus the table shows 2° rather than 3° for its first significant entry, and use of the table gives 0.1 for sin 3° to one decimal, which is correct.

The application to machine learning is brought out by considering the case where no inverse procedure is to hand for obtaining the x-values corresponding to the bounds on each interval into which it is proposed to subdivide the y-variable. The machine must then perforce compute $f(x)$ *ad hoc* for each new value of x as it comes along. However, if the complete repertoire of possible y-values is first listed, as in Comrie's construction, then each newly computed $f(x)$ can be used either to confirm or to extend the interval embracing those values of x

which map onto this particular y after rounding, or more generally canonicalization, of the result. If the x–y values correspond to situation-action pairs, then the process described amounts to the inductive learning of a strategy expressed in production rule form. Marsh has elaborated a worked example around this theme.[11]

PRACTICAL NEED

Some three years ago enquiry was made through Butler Cox & Partners on the subject of programmer productivity. The state of affairs then revealed is summarized in Table 3. The author's published comment at the time was: If the markets of the booming microelectronics industries are not to collapse from program starvation, radical innovation, not just improvement here and there, is needed in automatic programming. . .[8] Various schools and branches of computer science claim special relevance to this problem. Conspicuous among these is artificial intelligence.

Table 3. Growth of various indices of the computing industry since 1955

Indicator	1955	1965	1975	1985
Industry size	1	20	80	320
Hardware performance for fixed cost	1	100	10 000	1 000 000
Programmer productivity	1	2.0	2.7	3.6

In what follows, studies of learning with immediate application to programmer productivity will be discussed. The theme is that help is at hand for one particular chore which analysts and programmers of complex systems have to face, namely the need to encode rules of thumb adopted intuitively by human practitioners of skilled procedures. One does not design a program to do audits, for example, without first talking to an auditor. Unfortunately, human practitioners tend to describe their own rules of operation in terms which do not subsequently stand the test of practice. The story is told of a large cheese factory whose camemberts were a by-word. Crucial to their renown was the company's procedure for quality control, by which every hundredth cheese was sampled to ensure that the production process was still on the narrow path separating the marginally unripe from the marginally over-ripe. Success rested on the uncanny powers developed by one very old man, whose procedure was to thrust his index finger into the cheese, close his eyes, and utter an opinion.

If only because of the expert's age and frailty, automation seemed to be required, and an ambitious R & D project was launched. After much escalation of cost and elaboration of method, which included lowering into the cheeses various steel probes wired to strain gauges and other sensors, no progress had been registered. Substantial inducements were offered to the sage for a precise account of how he did the trick. He could offer little, beyond the advice: 'It's got to *feel* right!' In the end it turned out that feel had nothing to do with it. After breaking the crust of the cheese with his finger, the expert was interpreting subliminal signals from his sense of smell.

As Feigenbaum has made evident in his contributions to this topic,[12] the world of human expertise is not always such a black hole of inscrutability; but as he also attests, it is often quite black enough. The expert practitioner does indeed thwart the analyst's efforts to obtain clear, complete, and reliable accounts of how key descriptive concepts of his craft should be operationally defined—the concept of 'ripe' in the foregoing example. This is where current laboratory results with inductive learning holds out promise of a remedy. He may not be able to tell you what to do with them, but the expert (although not the cheese expert of the anecdote) can usually supply a list of primitive features which at least contains all those which are relevant, even though it may be padded out with additional features which the expert thinks are relevant but which are not. The expert typically possesses the further gift of being able to induce a grasp of the given concept in a trainee, by selecting and administering a well-contrived sequence of examples.

We can sketch the domain expert's profile as in Table 4. Failure under heading (2) can be extreme, as in checkers masters. A. L. Samuel (personal communication) failed, after intensive study to find any significant connection between the verbal and written prescriptions of the masters on the one hand and their actual play in tournaments on the other. For effective machine play to be evolved in this domain, it would seem that it is machine learning or nothing, allowing under 'learning' the induction-based training of the machine by a master, described in the text.

Table 4. Two views of an expert's repertoire of skills

	Power to recognize examples of key concepts (1)	Power to describe these (2)	Power to identify key primitives (3)	Power to generate good tutorial examples (4)
As seen by the expert	Good	Excellent	Excellent	Good
As seen by the analyst	Excellent	Very poor	Fairly good	Excellent

So far, research with expert systems has been concentrated on attempting to exploit heading (2) of this tabulation. However, as the scope of systems increases, so does the impracticality of extracting all the needed descriptive patterns by this route. Eventually we must stop harrassing the expert for manifestations of (2), which he is unable and untrained to deliver, and instead seek from him that which he *can* supply, namely (1), (3) and (4). What do we then do with these? Given a learning program analogous to the faculty possessed by a human trainee, we say to our domain expert: 'Forget about explaining this concept. Give us a list of primitive features for our programmers to code and load. Then sit down with the machine and *train it*!'

New programming tools are making this practicable. Rule acquisition from experts can be made quicker and cheaper by an order of magnitude. Moreover we get better rules, as demonstrated in a recent study by

Michalski and Chilausky,[13] who compared the accuracy of machine-derived with human-derived rules. Advantages have been noted even in respect of sheer computational efficiency (i.e. the execution-costs of rules), amounting in a recent study by Quinlan to a five-fold superiority over the best that could be achieved by hand coding.[14] This super-fast decision structure, however, proved to be opaque to the eye of the domain expert. Inscrutability to man of the executable products of machine learning raises serious issues, to which we must now turn.

COMPUTER INDUCTION— FRIENDLY AND UNFRIENDLY

Learning through stimulus reclassification belongs under computer induction. The biological counterpart is concept formation. Because computer users are human, rather than, say, Martian, it is of interest to enquire whether or not the descriptive structures formed by computer induction are intelligible as concepts. The products of inductive learning can be termed 'friendly' or 'unfriendly' according to whether or not the machine-generated descriptions make sense to the human practitioner.

Inductive learning procedures can thus be classified not only according to the production cost, execution cost, scope, and accuracy of their products, but also according to user-friendliness. Knowledge-based systems are coming into play in a widening range of industrial applications, some of them in socially critical areas such as the control and troubleshooting of complex machinery. As rising programmer costs exert increasing pressure towards automatic construction of rule bases for such systems, so the importance will grow of disciplining the methods employed to do this. It will not be desirable for control rooms in nuclear power stations, air traffic control centres, and the like to become polluted with uncomprehended descriptions generated by their associated computing systems. As a safeguard we recommend a man/machine approach, the basic idea of which will be illustrated with a test-tube size example. Our name for this disciplined method is 'structured induction'.

THE KING-PAWN-KING END-GAME

The play of King and Pawn against King (KPK) is one of the elementary end-games, yet contains a surprising degree of complexity. It is very hard to program using a conventional tree-search plus evaluation function, since the maximum depth of win (Pawn promotion) is 37. The burden of decision is thus shifted from search onto the evaluation function, which tends to be too simple and weak a vehicle for coping adequately with all the exceptional positions which crop up. For KPK it takes months to write a good program, and even longer to produce a correct one. Zuidema gives an account of the programming problems arising from an even more elementary ending, namely King-Rook-King.[15] Success left Zuidema, a chess-master and an accomplished computer scientist, so exhausted that he resolved to abstain from further work in the field.

The KPK task was defined as the classification by machine of legal KPK positions into the decision classes DRAWN and LOST (as seen from the lone King's viewpoint). The task was convenient for studying the extent to which computer induction can be made to replace programming as a source of expert rules. The chess expert or master can perform the classification 'at a glance'. At the same time he can normally say a great deal about how he does it, and much of this is systematized in the chess books, including definitions and examples of the key subconcepts, or primitive attributes, from which the main concept is built. None the less it remains extremely costly in time and brain-power fully to solve this problem by any straightforward programming approach.

For assessing the accuracy of machine-synthesized and programmer-synthesized representations, M. R. B. Clarke's precomputed database for KPK was available,[16] comprising a look-up table for the complete problem space. A diagram from Clarke's paper has been reproduced in the upper part of Fig. 3. It depicts the fact that for every descriptive concept there exists a spectrum of equally complete and correct machine representations, differing according to the balance struck between calculation and memory. The optimal compromise is determined by the relative costs of these two commodities. If memory is arbitrarily cheap, then Clarke's table look-up is optimal. If, on the other hand, processing is cheap than exhaustive search, not depicted in the diagram, would be best. Processing in this case must of course be fast enough for computations to be completed within the client's waiting time.

For the 'elementary' chess ending King and Pawn *versus* King, four different machine representations have been compared, strung out along a spectrum from all look-up to mostly calculation. Only one of these (an advice-oriented program by Bramer) is both human-intelligible and human-executable. Beal's, although only exemplifying the small step from 20 descriptive patterns to 50,[17] lies outside the 'human window' and is totally meaningless to the eye of a chess expert. A representation synthesized by computer induction using Quinlan's ID3 was equally inscrutable, although also highly efficient. When, however, the problem was done using ID3 in

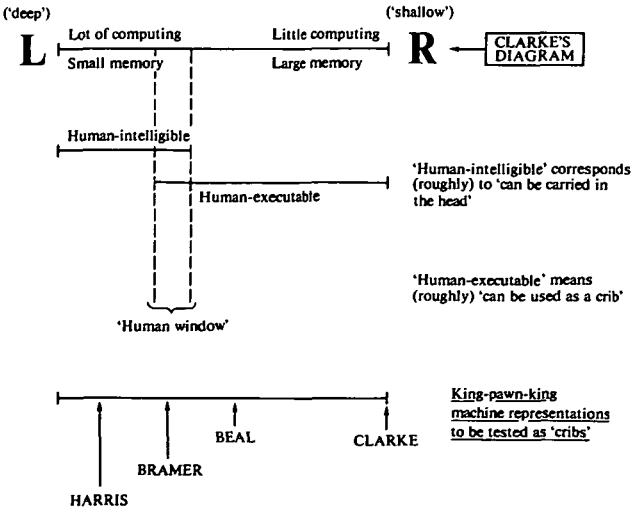

Figure 3. A comparison of four machine representations for the 'elementary' chess ending: King and Pawn versus King.

Table 5. A tabular expression of Figure 3

Program	Amount of search	No of concepts ('patterns')	Humanly intelligible?	Humanly executable?
Harris	A lot	About 10	Yes	No
Bramer	Little	About 20	Yes	Yes
Beal	Very little	About 50	No	Yes
Clarke	None	About 100 000	No	Yes

'structured induction' style, a representation was obtained expressible in half a dozen easily comprehended constituent patterns, superior on all-round assessment to any of the programs of human authorship (Shapiro and Niblett, 1981, see later).[18]

A window has been superimposed on the diagram of Fig. 3. The location and extent of this 'human window' are determined by the brain's own limitations on calculation and memory—approximately 20 binary discriminations per second, and 10^{10} bits respectively. By this last figure we denote only that part of memory used to store information acquired during a person's lifetime: the limit is set by the maximal acquisition-rate, not by the extent of potential storage. If a representation falls to the left of the window, then it is too condensed for the brain's feeble calculation powers to apply and get an answer in acceptable time. Exhaustive minimax search would be an extreme example: however *understandable* this procedure is, there would be no prospect of an unaided human being able to check it out in a case where the machine's classification had been disputed. If a representation falls to the right of the window, then it is too expanded to be held in ready-access form by a device (the brain) which only has room in short-term memory for about seven pointers at one time, these being used to address for purposes of access the subconcepts ('patterns') held in long-term memory from which the total concept is constructed. The result is that although the unaided human can perfectly well check out such a representation in a disputed case (in the extreme, by look-up in a printout of Clarke's table) it makes no sense to him. The situation is as depicted in Table 5. The programs listed were all of human authorship. It shows the two major determinants of executability and intelligibility by the human subject. The first is inversely related to the amount of search, the second is inversely related to the number of patterns to be processed. A third determinant, of a structural nature, comes into the picture, as brought to light in the Shapiro-Niblett experiments described later. It is plainly of great interest, now that we are in a position to use computer induction partially to automate the synthesis of such programs, to discover the conditions under which these machine products do or do not fall within the human window.

QUINLAN'S ID3 PROGRAM FOR INDUCTIVE LEARNING

For the Edinburgh experiments the induction engine used by Shapiro and Niblett was Quinlan's PASCAL program ID3, developed from Hunt's Concept Learning System (CLS). Quinlan's extensions to Hunt's algorithm allow the program iteratively to grow a small example set from an exhaustive set of situations stored on a database, such as, for example, the Clarke look-up table in the KPK case under consideration. It is thus intended to be used to accomplish *database compression*, rather than training by example from the virtual database implicitly contained in the expert's knowledge of the given domain. We shall start, therefore, with the use made of Quinlan's unmodified algorithm to replace Clarke's bulky database with a relatively compact, and equally complete and correct, representation in the form of a decision tree.

The small example set dynamically maintained by the program during its operation on a given induction problem is referred to by Quinlan as 'the window'. Because of the inconvenient clash between this term and the term 'human window' employed here for a quite different notion, this set of examples will be referred to as the 'working set'. The first step in database compression is to use the repertoire of primitive attributes supplied by the expert to replace the exhaustive set of individual entries with a smaller set of 'instances'. Each instance is a vector of attribute-values, and if the attribute-repertoire is adequate, all members of a given instance will be associated with the same decision-class in the database: otherwise a 'contradiction' exists. The 'working set' of instances is used to induce a rule at each iteration of the induction process, as follows:

1. Initialize the working set, of user-determined size, with a random sample of instances.
2. Induce a rule from the working set.
3. Deal with contradictions.
4. Mark exceptions to current rule found by sampling outside the working set.
5. If no exceptions found, go to 8.
6. Merge a user-determined number of exceptions into the working set (in one mode, discardable instances are dropped to make room).
7. Print statistics for this iteration and go to 2.
8. Print rule and exit.

Step 2 consists of the following:

1. If all members of the set belong to the same decision class then label with the corresponding class name.
2. Otherwise, split the set into subsets by applying the next attribute.
3. Repeat the above for each non-empty subset so formed, continuing until all subsets are labelled.

The choice of 'next attribute' is critical to performance. ID3 uses an information-theoretic criterion for this.

Step 3 involves marking with a keyword 'search' the point(s) in the incomplete rule where contradictions arise. This informs the user that the current set of attributes is not adequate to classify the working set, and another technique must be used at this point when the rule is executed; alternatively the attribute set must be changed.

When a complete and correct rule has been generated, the working set contains example situations that embody all the relevant features for a classification database for the domain, often leading to spectacular compressions. In passing it may be noted that for a given domain and attribute set, the size of the minimal exemplary set is a useful index of the domain's complexity. For more information on ID3, Quinlan's accounts may be consulted.[14,19]

Using a set of 31 attributes, of which 29 were incorporated in the synthesized rule, Shapiro and Niblett obtained the result summarized in Table 6. The induced decision-tree rule looked good by the criteria of synthesis-cost, compactness, and execution efficiency; but it made no sense to the chess expert. Not only was it conceptually opaque, but difficult in the extreme to execute by hand, for example for checking purposes.

Table 6. Results obtained by Niblett and Shapiro using Quinlan's ID3/PASCAL on a DEC-11/34 using 31 attributes for the KPK domain

Number of examples in final working set	306
Number of nodes in final tree	79
Time taken to generate rule	113 CPU sec

It had been demonstrated that an equally efficient representation could be generated which was extremely clear and understandable to the expert, and also executable by him without difficulty. This 'human window' representation was achieved by first subdividing the main top-level concept (DRAWN versus LOST) into a few intermediate-level concepts by the kind of top-down analysis recommended in structured programming. These intermediate-level patterns were then induced as decision-tree rules, to be used as attribute-recognizers in a final ID3 pass to induce the top-level concept.

Top-down analysis had already been performed by Niblett in the course of writing an 'advice table' for the KPK domain in the form of a PROLOG program (see Ref. 20 for a worked example of the advice table concept). In this way a decomposition into five suitable subprograms had already been identified. The remainder of the procedure is described below (from a technical memorandum by Shapiro and Niblett).

We decided to extend the use of ID3, which had been used to solve the NLOST2P problem in the KRKN domain to the more complex problem of determining whether any KPK position is lost or drawn. Due to the difficulty of the problem we split it into five subproblems, corresponding to the pieces of advice in the KPK advice table.

For each subproblem a decision tree was generated which determined whether the relevant piece of advice was satisfiable from any given position. Each subproblem was tackled using a different though sometimes overlapping set of attributes.

The following stages were necessary for the generation of the ID3 decision trees.

1. A database was generated for the piece of advice in question. This stated for every position whether the piece of advice was satisfiable from that position. The generation used techniques described by Niblett.[20] Each database required about 20 minutes CPU time to generate on the Edinburgh/ICF DEC 10.
2. A set of attributes was chosen by the 'domain specialist' [TN]. These attributes were specific to the advice in question.
3. The database was used to determine whether these attributes created any 'clashes'. A clash occurs when two positions have identical values over all the user-defined attributes, but in one position the advice is satisfiable while in the other it is not. There are at least two ways of tackling this problem. In one the user supplies very many attributes to ensure there are no clashes, letting ID3 itself choose an adequate subset of these. The method we chose was to select a small set of attributes and examine those positions where clashes arose, using ID3 to validate the chosen attribute set.
4. The attribute set is refined in the light of the clashes that occur. The domain expert will decide either to refine the definition of a current attribute, or to add one or more attributes to the set. This decision is made (at present) on intuitive grounds based on the expert's classification of positions.
5. Steps 3 and 4 above are iterated until a clash-free attribute set is found. A decision rule for the advice is then induced by ID3.

Results for the subproblems are shown in Table 7. Note that very large economies have been obtained in machine time, reflecting the great reduction in the total number of training examples needed to induce a complete and correct rule. The rule itself together with its five subrules, is *in toto* of comparable compactness to that of Table 6, but it offers a striking contrast in transparency and checkability by a human expert. The final form taken

Table 7. Quantitative data on the results of tackling the same problem as that of Table 6, by structured induction

Subproblem 1: pawn-can-run	
no of examples in final working set	16
no of nodes in final tree	17
time taken to generate rule	6 CPU sec
Subproblem 2: rookpawn	
no of examples in final working set	20
no of nodes in final tree	21
time taken to generate rule	9 CPU sec
Subproblem 3: get-to-mainpatt	
no of examples in final working set	7
no of nodes in final tree	11
time taken to generate rule	2 CPU sec
Subproblem 4: rank 56	
no of examples in final working set	7
no of nodes in final tree	11
time taken to generate rule	2 CPU sec
Subproblem 5: rank 7	
no of examples in final working set	4
no of nodes in final tree	7
time taken to generate rule	1 CPU sec
Special attribute: interfere (necessary for certain BTM positions)	
Combining all of the above gave the following:	
Main problem: position-is-DRAWN	
no of examples in final working set	9
no of nodes in final tree	13
time taken to generate rule	2 CPU sec

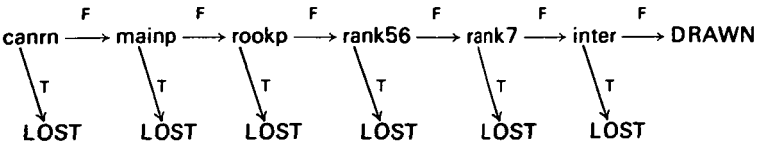

Figure 4. The top-level decision rule generated by structured induction is shown here as a decision tree.

by the top-level rule is given in Fig. 4, which also characterizes all five subproblem rules—for 'canrn', 'mainp', 'rookp', etc. and the 'interfere' attribute. Each of the subpatterns is represented as an ID3-induced decision tree exhibiting the same fully-ordered check-list form. Such a form puts the least possible burden on the limited short-term memory of the human user of the rule.

Apart from the fact that all six rules make perfectly good sense to the expert (some of this sense was, after all, introduced by the initial top-down analysis) a feature emerged which all the trees have in common, and which gives a possible clue to their user-intelligibility. Each tree has a depth equal to the number of its non-terminal nodes. In other words each decision tree has the form of an ordered list of tests, each test being a potential exit point. To process such a list makes minimal demands upon working memory. It is therefore not too surprising that human beings, whose short-term memory has only about seven locations (see Ref. 21), should find such decision-structures easy to handle mentally. An equivalent way of characterizing this form is to say that each decision tree is a rule-based system in miniature.

Comparison of Tables 6 and 7 reveals a further gain of structured over unstructured induction using ID3. The total number of examples in the final working set in the case of unstructured induction is over 300, as compared with less than one quarter of that number for the structured method.

Further simplification of some of the decision rules occurred when induction was done using 'interactive ID3', a locally developed variant (by A Shapiro) which uses the expert as a 'virtual database', i.e. as its source of training examples. This corresponds to the anticipated mode of operation for domains of economic relevance, and eliminates the costly preliminary preparation of exhaustive tabulations which in the typical application will in any case not be feasible. For this reason we draw particular encouragement from these first results with the interactive version.

The right way to handle computer induction so as to generate humanly transparent decision-structures is at present an open question. The Shapiro-Niblett experiment illustrates the use of a fast, ultra-simple algorithm, throwing the burden onto strict user-discipline to ensure a humanly acceptable product. An alternative philosophy is that adopted by Ryszard Michalski at the University of Illinois, namely so to structure the induction algorithm that its products are guaranteed in advance to possess the desired user-friendly properties. Both approaches are currently under test in a variety of application domains. It will not be possible to pronounce on their respective pros and cons until controlled comparisons have been completed using common bench-mark problems.

Acknowledgments

This work was assisted by grants from the Science Research Council in support of software and micro-electronic aids to expert systems research. Acknowledgment is also made of facilities provided by the Department of Computer Science, University of Illinois, where the greater part of this paper was written.

REFERENCES

1. D. Michie, Experiments on the mechanisation of game-learning. 1. characterisation of the model and its parameters. *The Computer Journal* **6**, 232–236 (1963).
2. R. A. Chambers and D. Michie, Man-machine cooperation on a learning task, in *Computer Graphics Techniques and Applications*, ed. by R. Parslow, R. Prowse and R. Elliott Green, pp. 79–86. Plenum, London (1969).
3. E. B. Hunt, J. Marin and P. Stone, *Experiments in Induction*, Academic Press, New York (1966).
4. J. McCarthy, Programs with common sense, in *Mechanisation of Thought Processes, Proceedings of Symposium, NPL, November 1958*, pp. 77–84. HMSO, London (1959).
5. E. Eastwood, Control theory and the engineer. *Proceedings of the Institute of Electrical and Electronic Engineers* **115**. (No 1), 203–211 (1968).
6. J. McCarthy and P. J. Hayes, Some philosophical problems from the standpoint of artificial intelligence, in *Machine Intelligence*, Vol. 4, ed. by B. Meltzer and D. Michie, pp. 463–502. Edinburgh University Press, Edinburgh (1969).
7. R. M. Hunt, *A New Approach to Heuristic Model Creation*, CS397.DM Term Project Report, University of Illinois, Urbana.
8. D. Michie, Competing with the costs of human skills. *Computer Weekly* (12 October 1978).
9. L. J. Comrie, *Chamber's Six-Figure Mathematical Tables*, D. van Nostrand, New York.
10. M. H. van Emden, *LIB Memo Functions*, in POP-2 Program Library Documentation, School of Artificial Intelligence, Edinburgh (1974).
11. D. Marsh, Memo functions, The graph traverser and a simple control situation, in *Machine Intelligence*, Vol. 5, ed. by D. Michie and B. Melzer, pp. 281–300. Edinburgh University Press, Edinburgh and American Elsevier, New York (1970).
12. E. A. Feigenbaum, Expert Systems in the 1980s. *Infotech State of the Art Review, 1980* Infotech Ltd, Maidenhead (1980).
13. R. S. Michalski and R. L. Chilausky, Learning by being told and learning from examples, in *Policy Analysis and Information Systems* (special issue on knowledge acquisition and induction) (1980).
14. J. R. Quinlan, Semi-autonomous acquisition of pattern-based knowledge, *Proceedings of the 69th Infotech State of the Art Conference on Expert Systems*, pp. 11/3–11/15. Infotech Ltd, Maidenhead (1980).
15. C. Zuidema, Chess, how to program the exceptions? *Afdeling Informatica*, IW 21/74, Mathematisch Centrum, Amsterdam (1974).
16. M. R. B. Clarke, A quantitative study of king and pawn against king, in *Advances in Computer Chess*, Vol. 1, ed. by M. R. B. Clarke, pp. 108–118. Edinburgh University Press, Edinburgh (1977).

17. D. Beal and M. R. B. Clarke, The construction of economic and correct algorithms for king and pawn against king, in *Advances in Computer Chess*, Vol. 2, ed. by M. R. B. Clarke, pp. 1–30. Edinburgh University Press, Edinburgh (1980).
18. T. Niblett and A. Shapiro, Automatic induction of classification rules in a chess end game, in *Advances in Computer Chess*, Vol. 3, ed. by M. R. B. Clarke, forthcoming.
19. J. R. Quinlan, Discovering rules by induction from large collections of examples, in *Expert Systems in the Microelectronic Age*, ed. by D. Michie, pp. 168–201. Edinburgh University Press, Edinburgh (1979).
20. T. B. Niblett, A provably correct advice strategy for the end-game of king and pawn against king. To appear in *Machine Intelligence*, Vol. 10, Ellis Horwood Chichester and Halsted Press (Wiley), New York (1982).
21. G. A. Miller, The magical number seven, plus or minus two: some limits on our capacity for processing information, *Psychological Review* **63**, pp. 81–97 (1956).

OTHER RELEVANT PAPERS

A. P. Ambler, H. G. Barrow, C. M. Brown, R. M. Burstall and R. J. Popplestone, A versatile system for computer-controlled assembly. *Artificial Intelligence* **6**, 129–156 (1975).
B. G. Buchanan, D. H. Smith, W. C. White, R. Gritter, E. Feigenbaum, J. Lederberg and C. Djerassi, Applications of Artificial Intelligence for Chemical Inference. XXII. Automatic Rule Formation in Mass Spectrometry by Means of the Meta-DENDRAL Program. *Journal of the American Chemical Society* **98**, 6168–6178 (1976).
M. Bramer, An optimal pattern-based algorithm for king and pawn against king. *Advances in Computer Chess*, vol. 2, ed. by M. R. B. Clarke, Edinburgh University Press, Edinburgh (1980).
I. Bratko, Proving correctness of strategies in the AL1 assertional language, *Inf. Proc. Letters* **7**, 223–230 (1978).
T. A. Dolatta, *Data Processing in 1980–85*, SEN 0–471–21786–7.

Received January 1981

Note added in proof: Since the foregoing was written, a new and radical approach to inductive learning within the framework of logic programming has been described by E. Y. Shapiro in 'An algorithm that infers theories from facts' in the Proceedings of the 7th International Joint Conference on Artificial Intelligence, August 1981. Ehud Shapiro's PROLOG program MIS (Model Inference System) will be included in the bench-mark tests referred to in the closing passage of this article.