27

Planning and Robots

James Doran*
Department of Machine Intelligence and Perception
University of Edinburgh

Introduction

This paper is a survey and discussion of research work relevant to the task of constructing some kind of reasoning robot. The emphasis is entirely on the organization of the reasoning processes, in particular planning, rather than on hardware. In practice the reasoning would most probably be carried out within a digital computer.

My objective is to clarify the relationship between some superficially rather disparate approaches to this task, and simultaneously to indicate what seem to be the key problem areas.

No new experimental results are presented, but the approach to the subject which I have adopted is a consequence of earlier experimentation with a simple computer simulation of a robot (Doran 1968a, 1969).

Heuristic problem-solving programs

Our starting point is heuristic problem-solving by computers. The following outline of the present state of the subject is very brief, but sufficient for our purposes. For more detailed information the reader may refer to the valuable survey by Sandewall (1969).

Heuristic problem-solving has been dominated over the last decade by what may be called the General Problem Solver (GPS) tradition (Newell, Shaw and Simon 1959, 1960, Hormann 1965, Ernst and Newell 1969, Quinlan and Hunt 1968). The various versions of the GPS program, its descendants, and the associated proposals and speculations, form an impressive body of research. However, although a great deal has been learnt from the GPS work about how to write a single computer program to solve a variety of problems, actual performance has always been somewhat disappointing, and the work has sometimes been marred by a certain vagueness of presentation. Typically the GPS can, with a substantial amount of human aid, solve quite a wide range of puzzles of the 'party-game' variety, for example the 'Tower of Hanoi'

* Present address: S.R.C. Atlas Computer Laboratory, Didcot, Berkshire.

and 'Missionaries and Cannibals' problems, together with very simple problems in potentially much more complex task areas such as the integral calculus.

Another substantial body of work on general problem-solving is that associated with the Graph Traverser program (Doran and Michie 1966, Doran 1967, Michie 1967, Doran 1968, Michie, Fleming and Oldfield 1968, Michie and Ross 1970). Work on the Graph Traverser has tended to be rather more concrete, and less ambitious, than that on the GPS, and has not claimed to have psychological significance. The performance of the program in practice is rather better. In the next secion I shall outline the basic Graph Traverser algorithm and indicate where the GPS differs from it.

Important projects devoted to much more specific applications of heuristic problem-solving are those of Evans (1964) and Moses (1967). The task areas are geometric-analogy problems and symbolic integration respectively, and the performance level attained is quite high.

At the present time several new trends are appearing:

(a) heuristic problem-solving of the GPS or Graph Traverser type is sufficiently familiar to be embedded in more complex programs or programming languages (for example, Burstall 1968, Popplestone 1970),

(b) the theory, rather than the practice, of heuristic problem-solving is being explored (for example, Ernst 1968, Nilsson 1969, Banerji 1969, Sandewall 1969a, Pohl 1970),

(c) the problem of automating the process of finding a good problem representation is being explored (for example, Amarel 1968, 1969),

(d) automatic theorem-proving procedures are being applied to problem-solving (for example, Green 1969).

All these trends have some relevance to the remainder of this paper.

Problem-solving and planning by robots

In this section and the next we shall consider the transition from heuristic problem-solving as exemplified by the Graph Traverser, to planning by a robot as exemplified by my own work and that of Marsh (Doran 1967, 1967a, 1968a, 1969; Marsh 1970; Michie 1967, 1968a; Popplestone 1967). I do not suggest that the Graph Traverser is the only starting point for such work. I use this example because I am particularly well acquainted with it.

The primary components of the Graph Traverser 'schema' are as follows. The problem to be solved must be capable of representation as a graph, in the mathematical sense, where the nodes of the graph correspond in some sense to possible *states* of the problem and the arcs to possible operations or *operators* which transform one state into another. This problem graph is defined to the program by specifying the set of possible states and the set of possible operators and their effects. It must then be possible to interpret the original problem as either the task of finding a path across the graph from one particular node (the starting state) to another (the goal state), or the

task of finding a node with some desired property. In this paper we shall always be concerned with the former task.

The solution procedure of the Graph Traverser is actually to 'grow', in the computer store, a larger and larger portion of the problem graph in the form of a *search tree* rooted at the starting node. Growth continues until the node or path sought is discovered. In order to do this efficiently the program uses, in general, a heuristic state evaluation function and heuristic operator selection techniques to grow the search tree in the most promising direction. Full details of this process can be obtained from the references quoted.

The GPS diverges substantially from the Graph Traverser by laying great stress on the difficulty that may arise in applying a selected operator to a particular state, and the way in which a recursive call of the program can be used to deal with this difficulty. The converse of this is that the GPS pays less attention to holding in store a ramified search tree and to the use of a state evaluation function.

Obviously, there is more to the Graph Traverser than I have described, and the interested reader is again referred to the source papers. We shall, however, need here one further concept, that of a *partial solution path.* When the program fills the available computer store with its search tree without finding a solution path, then it commits itself to all or part of a particular branch of the tree as the first part of the required path. This enables it to discard part of the search tree and to re-use the store thus freed.

A typical application of the Graph Traverser is to some kind of sliding-block puzzle, where the states correspond to possible configurations of the pieces of the puzzle, and the operators to possible movement of the pieces.

The transition from a problem-solving context to that of planning by a robot is a simple one, at least at first sight. By planning we mean the robot's need to consider possible courses of action in the light of some desired goal, and to select and carry out the most promising. Hence all one need do is to identify the concept of a state, in Graph Traverser terminology, with some state of the robot and its environment, and to identify the concept of an operator with some possible act on the part of the robot. A solution path, once discovered, is then a plan of action which the robot must execute rather than simply exhibit to the outside world as in the problem-solving case.

My own work, which developed out of some of the earlier Graph Traverser work, simulated a robot living in a very simple 'cage' environment. A state, better called a *perceived state* in this case, corresponded to the little that the robot could perceive of its surroundings at any time, and an operator to one of a set of simple movements (stepping, turning to the left or right). A simple motivational system provided goal states.

The *planning tree*, corresponding to the Graph Traverser search tree, was grown by a simple depth first procedure with heuristic cut-off, partly because a state evaluation function is much less useful as a means of directing the growth of the tree in a robot situation. However, a simple evaluation function

was used, with other heuristic procedures, to select partial solution paths, that is to say partial plans, when a full plan could not be found.

Rather more important was the incorporation into the planning tree of a form of 'expectimaxing' (*see* Michie and Chambers 1968). The need for this arose because of ambiguity in the robot's perceptual input. Specifically, any particular perceived state might well be the outcome of a variety of possible relationships between the robot and its environment. This led to a degree of uncertainty on the part of the robot as to what the outcome of any particular act might be. The robot tried to cope with this uncertainty by evaluating plans in terms of their predicted average benefit, using an expectimaxing procedure.

Planning and learning

The simulated robot described in the previous section could have had built into it detailed information about the effects of its possible movements upon its perceptions, in the same way that the Graph Traverser program is told what is the immediate effect of any operation on any configuration of pieces of a sliding-block puzzle. In fact, it was expected to collect such information from experience. Thus, in this case, planning was very closely associated with learning. The following types of learning occurred in the system:

(a) learning of the relationship between acts and perceptions by noting the effects of individual acts, by making generalizations about the effects of acts, and by noting that certain complicated transitions from one perceived state to another can always be achieved,

(b) learning which acts to employ in particular situations and the benefits to be expected – a kind of habit formation.

These learning capabilities can be regarded as a rather complex form of the rote learning employed by Samuel in his checkers programs (Samuel 1959, 1967). They by no means exhaust the possibilities for learning in such a system. Much more powerful forms of generalizations and abstraction are needed, and some degree of self-optimization of the variety of parameter settings controlling the planning would be possible. Any work on learning in heuristic problem-solving systems is potentially relevant (Quinlan 1969, Michie and Ross 1970). Nor can strategies for discarding information, for 'forgetting', be ignored. They may be crucial to success.

The whole question of learning in such a Graph Traverser based robot control system has been tackled from a slightly different angle by Marsh (1970). Here the learning aspects of the system have been clarified and isolated by the use of 'memo-functions' (Michie 1967a, 1968, Popplestone 1967). Marsh has also obtained experimental results quantifying the benefit to be obtained by using memo-functions at different points in the planning process.

Before leaving this section we should note two other areas in which such

a system will ultimately have to show learning ability. These are time and attention.

A robot, if it is to exist in a dynamic environment, must be able to estimate the passage of time, to allow for the time it takes to form and execute plans (*see* Toda 1962), and to estimate how a complex dynamic situation (for example, traffic at a road junction) will develop in time.

A robot must also be able to decide which sensors to read when. The point here is that a system with any degree of serial reasoning cannot be attending to the input from all of its sensors all of the time. It follows that the robot's attention strategy is a highly important part of its planning process. Too little attention to the outside world could lead to sudden disaster, confusion, or, more subtly, to a growing misconception by the robot of just what its situation actually is! Too much attention could waste time and hinder action.

Parallel processing

When a planning system of the type which we have been discussing is required to learn from experience the effects of the acts available to it, then it will build up a network of state–act–state transitions in its memory. This network is the equivalent of the problem graph in the problem-solving situation, but unlike the problem graph, it is actually kept as a network of data-structures and pointers in the computer store, at least until specific compression mechanisms go to work.

Planning is then not a question of reconstructing a fragment of this memory network, but of tracing out the planning tree on the already constructed network. This means that any planning algorithm which uses state evaluation followed by 'backing-up' of values, for example, expectimaxing, can plausibly be reinterpreted as a network flow process of the following general type:

(1) 'excitation' is inserted into the network at the goal state, and
(2) continuously transmitted by the arcs,
(3) continuously redistributed by the nodes, and
(4) detected by a receiver at the node corresponding to the robot's current state, which
(5) implements the act corresponding to the incoming arc carrying the greatest excitation.

A system of this type, the STELLA learning machine, has been explored in detail by Andreae (*see* Andreae and Cashin 1969) though in practical simulation work the parallel aspects of the machine had to be simulated on a serial computer. The model of animal learning behaviour put forward by Deutsch (1964) also has this general form, but is less precisely specified than the STELLA automaton.

In each of these two examples the networks through which excitation

flows are formed by receptor–motor units which automatically link up in accordance with the behaviour of the system's perceptual environment.

Network flow systems are attractive partly because they seem 'natural' in some sense, and partly because once one postulates a parallel processing capability of this type, the problems associated with the computational load and organizational complexity of major tree searches promise to disappear. However, it is far from clear how such network systems can be persuaded to yield really complex behaviour. The results of the work on perceptrons and other such self-organizing systems do not encourage optimism (Minsky and Papert 1969).

We should also note two important research projects which involve parallel processing and robot control, though not planning. These are the simulation studies on instinctive behaviour by Friedman (1969) and on the function of the reticular formation in the vertebrate by Kilmer and others (Kilmer, McCulloch and Blum 1969).

Complex planning

So far I have used the word 'planning' to refer to a robot's ability to consider alternative sequences of acts and to select the most promising among them. A close parallel has been drawn with the operation of a particular heuristic problem-solving program, the Graph Traverser.

However, in the context of heuristic problem-solving the word 'planning' has almost invariably been used for some more complex procedure *added* to the basic heuristic search in order to make it more effective. I shall employ the term *complex planning* for such additional procedures.

One form of complex planning is the use of *macro-operators*. What happens is that the original problem graph has added to it new arcs corresponding to concatenations of the original operators. These macro-operators are then used in the usual way to help grow the search tree. Clearly one needs some way of automatically constructing such operators, and equally some way of decomposing them when they appear in a solution path or plan.

The use of macro-operators in heuristic problem-solving has been tried out by, among others, Travis (1964), Hormann (1965) and Michie (1967). The concept is closely related to the mathematical concepts of a lemma and a theorem. Examples of the use of macro-operators in robot simulation work are provided by Nilsson and Raphael (1967) and Doran (1969). In each of these two latter examples the situation is complicated by the fact that the definitions of the macro-operators, or better macro-acts, are not available at plan execution time, so that when such an act is to be carried out a complex sub-process is set in motion.

The use of macro-operators is only one rather simple form of complex planning. The planning procedure proposed for the GPS (Newell, Shaw and Simon 1959) envisaged the abstraction both of the states and of the operators of the original problem formulation, in order to create a new simplified

version of it. The solution to this simple problem would then guide the solution of the original problem. Minsky(1961) has discussed complex planning of this 'homomorphic model' type, and has stressed the potential reduction in total search effort to be won. In the same paper he has also considered the use of 'semantic models' as a form of complex planning in a mathematical context. The successful geometry theorem-proving program of Gelernter (1959), which used a 'diagram' to test the validity of propositions, is a well-known example of this form of planning.

Recently Sandewall (1969) has defined a Planning Problem Solver (PPS). This is an attempt to explore in detail complex planning of the 'homomorphic model' type as applied to the GPS. His lengthy discussion covers the use of what I have called 'expectimaxing' at the complex planning level (compare Minsky's remarks, 1961), and indicates how concepts drawn from board game-playing programs such as α-β pruning can be carried over to this situation. He also proves certain optimality results for the PPS.

Sandewall's optimality results take us some way towards answering the very general question: 'When is complex planning beneficial, in the sense that it reduces the total amount of computation needed to solve the problem (or find a plan), and when is it more trouble than it is worth?' The answer to this question is clearly bound up with the problem of characterizing and actually finding good complex planning models for particular problems. And this problem itself is very closely related to the problem of problem representation as treated by Amarel (1968, 1969) and others.

Planning and formal systems

The present Stanford Research Institute (SRI) hardware robot project (Nilsson 1969a), not to be confused with the preliminary simulation work mentioned earlier, has experimented with more than one *ad hoc* heuristic search method for planning. These have made use of the 'map' or 'grid model' with which the robot is provided.

Of greater interest is the use of a first-order predicate calculus resolution theorem-prover to control the robot, as described by Green (1969). In barest outline, the procedure followed in order to have the robot carry out some task is to require the theorem-prover to prove the theorem that a situation can exist in which the task has been completed. During the proof procedure the theorem-prover constructs the sequence of acts, that is the plan, which the robot must execute to perform the task. Everything which the robot has to know in order to perform the task is formulated by the experimenter as a set of axioms (the 'axiom model') within the formal language of predicate calculus. Achieving this formulation may itself be far from straightforward, and at the present time the system can cope only with very simple tasks. In his paper Green discusses such complications as acts with more than one possible outcome, and tasks such that the robot's plan must include making some observation and using its result.

Green's work falls broadly within the Advice Taker tradition. The Advice Taker was proposed by McCarthy (1959) as a program capable of 'common-sense' reasoning, and capable of accepting and using advice whilst solving a problem. The program was to reason by manipulating sentences in formal languages, and therefore would have a very powerful means of storing and using its knowledge of the world. As a means to this end, McCarthy advocated the formalization of such everyday concepts as situation, causality, ability, and knowledge.

From this initial impetus a great deal of valuable work has resulted. However some of the more philosophical problems at the heart of the Advice Taker project still seem to defy any very coherent solution (McCarthy and Hayes 1969).

Since the Advice Taker work is much concerned with reasoning about actions and what they can achieve, it is not surprising that the gulf between this work and, say, the GPS work is less wide than at first appears (Hubermann 1965). For example, the concepts of situation, fluent, action, and strategy discussed by McCarthy and Hayes (1969) correspond roughly to the concepts of a state, a property of a state, an act, and a plan as used in this paper. It remains to be seen whether the attempt to formalize such concepts within a coherent logical system, rather than merely to incorporate them within some actual computer program or hardware device, is the only or even the best way to make progress.

Plans and programs

McCarthy and Hayes (1969) explore in some detail the parallel between a plan or course of action and a computer program. Their aim is partly to have procedures which prove results about computer programs do the same for plans.

Almost all programs involve loops, where a certain sequence of operations is repeated until some test is satisfied. One is led to consider plans with a similar property. The 'San Diego' problem, attributed to McCarthy in this context, is illuminating. The problem is to formulate a plan by which a motorist can get from San Francisco, say, to San Diego given the following information:

(a) there is no map obtainable with both San Francisco and San Diego marked on it,

(b) any filling station will provide a map of its local area, which will both indicate the direction of San Diego and show the location of at least one other filling station in that direction,

(c) the motorist is now at a filling station in San Francisco.

The solution to the problem is the following plan:

'Collect a map from the filling station you are at and drive to the filling station shown on the map which is furthest in the direction of San Diego. *Repeat* this procedure *until* you arrive in San Diego.'

Clearly the plan is one big loop and test. Equally clearly no planning procedure of the Graph Traverser type is going to generate such a plan (Michie and Popplestone 1969).

A different approach to this general topic is that of the psychologists, Miller, Galanter, and Pribram (1960), in their well-known book *Plans and the Structure of Behaviour*. They also point to the parallel between a computer program and a plan (*loc. cit.*, p. 16). Their TOTE unit (Test–Operate–Test–Exit unit), which they propose as the fundamental building block of plans, is just the 'loop and test' combination we are considering. For example, they exhibit a plan for knocking a nail into a plank which, simplifying somewhat, is '*repeat* the operation of hitting the nail with the hammer *until* its head is flush with the surface of the plank'.

Unfortunately, the discussion of Miller and his colleagues stops well short of an automatic procedure for generating such plans. It does imply, however, that plan generation should involve loops from the outset. Does this mean that we should abandon informal problem-solving methods for planning, and instead turn to the work concerned with the automatic generation of computer programs (*see*, for example, Green 1969) ? The prospect is not too inviting for those with an interest in human as well as machine intelligence.

What might be a 'natural' way to solve the San Diego problem – that is, a way which common sense and introspection suggest might be the way in which a person might solve it? By trying to answer this question we may obtain some productive new ideas. Consider the following procedure:

(1) visualize a very crude outline map of California with San Diego and San Francisco marked, and also with the motorist's location marked (initially at San Francisco),

(2) consider possible sequences of 'relevant' acts for the motorist – driving to filling stations, buying maps, and so on – manipulating his location on the image map as seems appropriate,

(3) observe that one possible sequence of acts has two interesting properties – that the acts form a repeating pattern and that the sequence corresponds to a roughly linear motion of the motorist's location on the image map,

(4) by extrapolating the motion on the map observe that if the repetition of acts is continued, then the motorist will reach San Diego,

(5) convert this observation into the required plan.

The reader may or may not feel that this solution procedure is plausible or in any way illuminating. He should compare Amarel's analysis of the 'mutilated chequer-board' problem (Amarel 1969). It does suggest that the key steps in the solution of such a problem may be abstraction followed by a rather simple form of extrapolation, in this case extrapolation of motion in a visual image.

Models and percepts

A robot must acquire or be given knowledge about its environment and itself.

This knowledge is often called its 'world model'. Examples of different kinds of world model which we have encountered are:

(a) the SRI robot's *axiom model* for use by the resolution theorem-prover (Nilsson 1969a),

(b) the SRI robot's *grid model* which can loosely be described as a 'map' of its surroundings (Nilsson 1969a),

(c) *perceptual cause and effect models* which store perceived states and the way in which they are modified by acts (Deutsch, 1964; Doran, 1968, 1969; Andreae and Cashin, 1969).

We can reasonably add to this list:

(d) *semantic memory systems* of one kind or another (for example, Quillian 1969, Becker 1969),

(e) *belief systems* such as those developed by Colby and his co-workers (Colby and Smith 1969).

The precise relationships between these superficially very different ways of storing and using knowledge about the world have not been much explored. I have two minor comments:

(1) Colby's term 'belief system' should perhaps be rather more widely used. It has advantages compared with 'world model' in that it avoids any association with physical 'scale' models, and correctly implies that a robot's 'knowledge' is bound in practice to include approximations to the truth as well as outright errors. It also encourages consideration of the 'conviction' with which a robot should hold a belief.

(2) A key heuristic problem for the robot is that of selecting the beliefs *relevant* to the prediction of the outcome of any particular course of action. This problem has been discussed by, for example, Nilsson and Raphael (1967, p. 243) and McCarthy and Hayes (1969 – the 'frame problem').

I shall end with some speculative remarks to be associated with entry (c) in the foregoing list – perceptual cause and effect models. These remarks concern reasoning processes which operate in terms of perceptual images or 'percepts'. We are all familiar with subjective perceptual images and, for example, some people seem to 'think' in terms of visual images far more than others. What does reasoning in terms of perceptual images mean, if anything, when we are talking about a robot?

Suppose that a robot is observing, through a TV camera, two cylinders standing upright on a flat surface. We may suppose that, as the result of a complex piece of picture processing (possibly involving the robot's expectation of what is to be seen, as well as what is actually there) a data structure is generated, representing in some sense the important information in the observation. This abstracted picture we may reasonably call a visual percept. If the robot now stores away this percept as it is, then it has also implicitly stored away such beliefs as: 'the tall object is wider than the short object'. However the robot may continue the processing and formulate and store

derived beliefs in some quite different coding, for example, a predicate calculus axiom model. Since this coding does not arise naturally from the picture processor, it is reasonable that it should not be called perceptual.

In human begins we can point to the distinction between, on the one hand. a fuzzy abstracted *visual* image of a tall wide object beside a short narrow object and, on the other hand, a fuzzy abstracted *aural* image of the words: 'the tall object is wider than the short object'. This is a rather special example because the coding into which the visual percept is translated itself involves perceptual (aural) images.

What form might reasoning in terms of visual percepts take? One possibility has already been indicated in connection with the San Diego problem. The key process might be as follows. Given two such percepts, and constraints which effectively specify what constitutes a valid visual percept (compare Clowes 1969), it will often be possible to merge them to form a third. The information implicit in the new percept will then have been deduced, in some sense, from the information implicit in the originals. Experimental psychologists have studied examples of such reasoning (Huttenlocher 1968, Handel, London and DeSoto 1968).

I am not suggesting that what I have called perceptual reasoning could not be encompassed, in theory, within some suitable formal system (compare Hayes 1970). What I am suggesting is that it might have practical advantages for robot systems operating in the real world.

Acknowledgements

This paper has been written during my tenure of a Science Research Council Fellowship. I have benefited from many discussions with Professor Donald Michie and Dr R. Burstall, both of the Department of Machine Intelligence and Perception, University of Edinburgh

REFERENCES

Amarel, S. (1968) On representations of problems of reasoning about actions. *Machine Intelligence 3*, pp. 131–72 (ed. Michie, D.). Edinburgh: Edinburgh University Press.

Amarel, S. (1969) Problem solving and decision making by computer: an overview. Paper distributed at the Symposium on Cognitive Studies and Artificial Intelligence Research, 2–8 March 1969. University of Chicago Center for Continuing Education.

Andreae, J.H. & Cashin, P.M. (1969) A learning machine with monologue. *Int. Journal of Man–Machine Studies*, **1**, 1–20.

Banerji, R. B. (1969) *Theory of problem solving*. New York: Elsevier.

Becker, J. D. (1969) The modelling of simple analogic and inductive processes in a semantic memory system. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 655–68 (eds Walker D. E. & Norton, L.M.). New York: Association for Computing Machinery.

Burstall, R.M. (1968) Writing search algorithms in functional form. *Machine Intelligence 3*, pp. 373–86 (ed. Michie D.). Edinburgh: Edinburgh University Press.

Clowes, M.B. (1969) Pictorial relationships – a syntactic approach. *Machine Intelligence 4*, pp. 361–84 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Colby, K.M. & Smith, D.C. (1969) Dialogues between humans and an artificial belief system. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 319–24 (eds Walker, D.E. & Norton, L.M.). New York: Association for Computing Machinery.

Deutsch, J.A. (1964) *The Structural Basis of Behavior*. Cambridge: University Press.

Doran, J.E. (1967) An approach to automatic problem-solving. *Machine Intelligence 1*, pp. 105–23 (eds Collins, N.L. & Michie, D.). Edinburgh: Oliver and Boyd.

Doran, J.E. (1967a) Designing a pleasure-seeking automaton. *Research Memorandum MIP–R–28*. Department of Machine Intelligence and Perception, University of Edinburgh.

Doran, J.E. (1968) New developments of the Graph Traverser. *Machine Intelligence 2*, pp. 119–35 (eds Dale, E. & Michie, D.). Edinburgh: Oliver and Boyd.

Doran, J.E. (1968a) Experiments with a pleasure-seeking automaton. *Machine Intelligence 3*, pp. 195–215 (ed. Michie, D.). Edinburgh: Edinburgh University Press.

Doran, J.E. (1969) Planning and generalization in an automaton/environment system. *Machine Intelligence 4*, pp. 433–54 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Doran, J.E. & Michie, D. (1966) Experiments with the Graph Traverser program. *Proc. R. Soc. A*, **294**, 235–59.

Ernst, G.W. (1968) Sufficient conditions for the success of GPS. *Memorandum SRC–68–17*. Systems Research Center, Case Western Reserve University.

Ernst, G.W. & Newell, A. (1969) *GPS: a Case Study in Generality and Problem Solving*. ACM Monograph Series. New York: Academic Press.

Evans, T.G. (1964) A heuristic program to solve geometric-analogy problems. *AFIPS*, **25**, 327–38. SJCC, Baltimore: Spartan Books.

Feigenbaum, E.A. & Feldman, J. (eds) (1963) *Computers and Thought*. New York: McGraw-Hill.

Friedman, L. (1969) Robot control strategy. *Proceedings of the International Joint Conference on Artificial Intelligence* (eds Walker, D.E. & Norton, L.M.) pp. 527–40. New York: Association for Computing Machinery.

Gelernter, H. (1959) Realization of a geometry-theorem proving machine. *Proceedings of an International Conference on Information Processing*, pp. 273–82. Paris: UNESCO House. Reprinted in Feigenbaum and Feldman (1963).

Green, C. (1969) Application of theorem proving to problem solving. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 219–39 (eds Walker, D.E. & Norton, L.M.). New York: Association for Computing Machinery.

Handel, S., London, M. & DeSoto, C. (1968) Reasoning and spatial representations. *J. of Verbal Learning and Verbal Behavior*, **2**.

Hayes, P.J. (1970) Robotologic. *Machine Intelligence 5*, pp. 533–54 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Hormann, A. (1965) Gaku: an artificial student. *Behav. Sci.*, **10**, 88–107.

Hubermann, B. (1965) The Advice Taker and GPS. *Research Memorandum No. 33*. Stanford Artificial Intelligence Project, Stanford University.

Huttenlocher, J. (1969) Constructing spatial images: a strategy in reasoning. *Paper distributed to the Annual Conference of the British Psychological Society*.

Kilmer, W.L., McCulloch, W.S. & Blum, J. (1969) A model of the vertebrate central command system. *Int. J. Man–Machine Studies*, **1**, 279–309.

Marsh D. (1970) Memo functions, the Graph Traverser and a simple control situation. *Machine Intelligence 5*, pp. 281–300 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

McCarthy, J. (1959) Programs with common sense. *Proceedings of a Symposium on the the Mechanization of Thought Processes*, pp. 75–91. London: HMSO. Reprinted in Minsky (1968).

McCarthy, J. & Hayes, P.J. (1969) Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence 4*, pp. 463–502 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Michie, D. (1967) Strategy-building with the Graph Traverser. *Machine Intelligence 1*, pp. 137–54 (eds Collins, N.L. & Michie D.). Edinburgh: Oliver and Boyd.

Michie, D. (1967a) Memo functions: a language feature with 'rote-learning' properties. *Research Memorandum MIP–R–29*, Department of Machine Intelligence and Perception, University of Edinburgh.

Michie, D. (1968) 'Memo' functions and machine learning. *Nature*, **218**, 19–22.

Michie, D. (1968a) Information and behaviour: a commentary on a note by R.L. Gregory. *Research Memorandum MIP–R–37*, Department of Machine Intelligence and Perception, University of Edinburgh.

Michie, D. & Chambers, R.A. (1968) BOXES: an experiment in adaptive control. *Machine Intelligence 2*, pp. 137–52 (eds Dale, E. & Michie, D.). Edinburgh: Oliver and Boyd.

Michie, D., Fleming, J.G. & Oldfield, J.V. (1968) Comparison of heuristic, interactive and unaided methods of solving a shortest route problem. *Machine Intelligence 3*, pp. 245–55 (ed. Michie, D.). Edinburgh: Edinburgh University Press.

Michie, D. & Popplestone, R.J. (1969) Freddy's first three years. *Experimental Programming 1968–1969*. Department of Machine Intelligence and Perception, University of Edinburgh.

Michie, D. & Ross, R. (1970) Experiments with the adaptive Graph Traverser. *Machine Intelligence 5*, pp. 301–18 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Miller, G.A., Galanter, E. & Pribram, K.H. (1960) *Plans and the Structure of Behavior*. New York: Holt, Rinehart and Winston Inc.

Minsky, M. (1961) Steps towards artificial intelligence. *Proc. Inst. Radio Engineers*, **49**, 8–30. Reprinted in Feigenbaum and Feldman (1963).

Minsky, M. (ed.) (1968) *Semantic Information Processing*. Cambridge Mass. and London: MIT Press.

Minsky, M. & Papert, S. (1969) *Perceptrons*. Cambridge Mass. and London: MIT Press.

Moses, J. (1967) *Symbolic Integration*. Ph.D. dissertation, MIT Mathematics Department.

Newell, A., Shaw, J.C. & Simon, H.A. (1959) Report on a general problem-solving program. *Proceedings of an International Conference on Information Processing*, pp. 256–64. Paris: UNESCO.

Newell, A., Shaw, J.C. & Simon, H.A. (1960) A variety of intelligent learning in a general problem solver. *Self-organizing Systems*, pp. 153–89 (eds Yovits, M.C. & Cameron, S.). London: Pergamon Press.

Nilsson, N.J. (1969) Searching problem-solving and game-playing trees for minimal cost solutions. *Proceedings of the IFIP Congress* 1968. Amsterdam: North Holland.

Nilsson, N.J. (1969a) A mobile automaton: an application of artificial intelligence techniques. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 509–20 (eds Walker, D.E. & Norton, L.M.). New York: Association for Computing Machinery.

Nilsson, N.J. & Raphael, B. (1967) Preliminary design of an intelligent robot. *Computer and Information Sciences II*, pp. 235–59 (ed. Tou, J.). New York: Academic Press.

Pohl, I. (1970) First results on the effect of error in heuristic search. *Machine Intelligence 5*, pp. 219–36 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Popplestone, R.J. (1967) Memo functions and the POP-2 language. *Research Memorandum MIP–R–30*. Department of Machine Intelligence and Perception, University of Edinburgh.

Popplestone, R.J. (1970) Experiments with automatic induction. *Machine Intelligence 5*, pp. 203–16 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Quillian, M.R. (1969) The teachable language comprehender: a simulation program and theory of language. *Comm. Ass. comput. Mach.*, **12**, 459–76.

Quinlan, J.R. (1969) A task-independent experience-gathering scheme for a problem-solver. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 193–7 (eds Walker, D.E. & Norton, L.M.). New York: Association for Computing Machinery.

Quinlan, J.R. & Hunt, E.B. (1968) A formal deductive problem-solving system. *J. Ass. comput. Mach.*, **15**, 625–46.

Samuel, A.L. (1959) Some studies in machine learning using the game of checkers. *IBM J. of Res. and Dev.*, **3**, 211–29. Reprinted in Feigenbaum and Feldman (1963).

Samuel, A.L. (1967) Some studies in machine learning using the game of checkers, 2 – recent progress. *IBM J. of Res. and Dev.*, **11**, 601–17.

Sandewall, E.J. (1969) Concepts and methods for heuristic search. *Proceedings of the International Joint Conference on Artificial Intelligence* (eds Walker, D.E. & Norton, L.M.). New York: Association for Computing Machinery.

Sandewall, E.J. (1969a) A planning problem solver based on look-ahead in stochastic game trees. *J. Ass. comput. Mach.*, **16**, 364–82.

Toda, M. (1962) The design of a fungus-eater: a model of human behavior in an unsophisticated environment. *Behav. Sci.*, **7**, 164–83.

Travis, L.G. (1964) Experiments with a theorem utilizing program. *AFIPS*, **25**, 339–58. SJCC. Baltimore: Spartan Books.