newe 36

heuristics
problem-solving
natural language

THE SEARCH FOR GENERALITY*

ALLEN NEWELL**
GEORGE ERNST

Carnegie Institute of Technology
Pittsburgh, Pennsylvania

INTRODUCTION

The task of this paper is to assess the current state of research into problem solving programs. We have in mind a collection of programs, mostly known as *heuristic programs*, constructed as deliberate efforts in artificial intelligence. That these should be singled out as appropriate representatives of *research into problem solving* is neither entirely arbitrary nor entirely justified. We shall return to this question at the end.

We may consider a problem solver to be a process that takes a problem as input and provides (when successful) the solution as output. The problem consists of the problem statement, or what is immediately given; and auxiliary information, which is potentially relevant to the problem but available only as the result of processing. The problem solver has available certain methods for attempting to solve the problem. These are to be applied to an internal representation of the problem. For the problem solver to be able to work on a problem it must first transform the problem statement from its external form into the internal representation. Thus (roughly), the class of problems the problem solver can convert into its internal representation determines how broad or general it is; and its success in obtaining solutions to problems in internal form determines its power. Whether or not universal, such a decomposition fits well the structure of present problem solving programs.

A method requires a formulation of a problem specific enough so that something definite can be said about how to attempt a solution. The method need not attempt the solution directly; most methods transform the problem into a different form so that other methods can be applied. In general, methods which require highly detailed and specific problem formulations are not of wide applicability. Thus there is a premium on highly general formulations for which methods are still possible.

The early problem solving programs were concerned primarily with problems that are difficult by human standards. To play chess and checkers, to prove theorems, and to solve puzzles, all demonstrate intellectual prowess. In compensation, these programs are exceedingly specific. Selecting a move in chess is certainly no more general a task than, say, inverting an 8×8 matrix. However, despite their task specificity, a common problem formulation and basic method, that of heuristic search, underlies them all.

Our concern in this paper is to characterize recent work, relate it to these earlier efforts, and indicate what new elements have been introduced. Overall, we find the following:

1. There has been a shift away from a concern with difficulty and toward a concern with generality. This means both a concern that the problem solver accept a general language for the problem statement, and that the internal representation be very general. In compensation, there is less emphasis on the difficulty of the problem.

2. The basic formulation used in earlier problem

*This research was supported by Contract SD-146 from the Advanced Research Projects Agency of the Department of Defense. We are indebted to H. A. Simon for many discussions on the topic of this paper.

**The analysis on which this paper is based is a joint effort of George Ernst and myself, and it is appropriate that he be co-author. For infelicities of style and errors of fact, I alone remain responsible.

solvers, heuristic search, continues to underlie much of the recent work.

3. A basic technique, matching two forms to determine appropriate substitutions for variables, which accounts for much of the success in the early heuristic programs, continues to be an important technique in recent programs.

We proceed, first, by laying out the essentials of heuristic search, asking what modifications and extensions have occurred. Then we analyze a few programs, attending both to their increasing concern with generality and to the problem formulations they use.

HEURISTIC SEARCH

The formulation given below underlies all the early problem solvers. For clarity we give it in a simplified prototype form, followed by a sequence of generalizations. These are intended to cover actual applications, and to indicate roughly the limits to which the formulation has been so far extended. The references are illustrative, not exhaustive. (We use *list* whenever we wish to indicate that a set is enumerated.)

> There is a set of objects {X} and a list of operators {Q}, such that, if X and Q are given, then Q applied to X gives either some new object or a signal that Q is inapplicable. Given an initial situation X_0 and a desired situation X_d, find a sequence of operators, S: $Q_1, \ldots Q_k$, that will transform X_0 into X_d.

Generalizations:

{X}	given by schemas, from which spespecific objects can be obtained by substitution of constants for variables.[33,41]
{Q}	given by a list of schemas from which specific operators can be obtained by substitution of constants for variables.[11,25]
{Q}	may contain operators with several inputs or several outputs.[27,38]
X_0	given by a list of objects,[27] or by a schema.[41]
X_d	given by a list of objects,[25] by a schema,[25] by a more complex description,[10] or by a testing procedure.[34]
S	may be more complex than a simple path. For example, all paths meeting certain conditions,[1,20] or all paths from the outputs of an operator having multiple outputs.[3,11] These make S a tree.

Despite the variation in formulation, problems of this type have all been tackled with the same basic method of trying possible sequences of operators. That is, the tree of objects generated by application of operators is searched for X_d. Since this tree grows exponentially, either systematic or random search suffices only for small problems. Success generally depends on whether sufficient heuristic rules can narrow the search so that X_d is discovered in reasonable time.

The devices used to limit and guide search can be cataloged. They are of relatively few types, being dictated by the basic information available in the problem formulation. The various types occur in problem solvers relatively independently of the specific task they are doing. Consequently, the rudiments of a design technology exist that allow one to put together a problem solver, using this formulation, for a new task.

Table 1 gives the various types of devices so far used (again, references are illustrative, not exhaustive). Almost all these devices have been used in the early problem solvers. Discussions of them in the context of particular problems can be found in a recent compendium[9] which reprints many of the original papers. This work also includes the excellent theoretical survey by Minsky[23] and an extensive bibliography.

A few entries appear in Table 1 that would have been missing at the last IFIPS Congress: One, a program[37] for finding mating combinations in chess, uses the amount of branching from a position (for the opponent) as a criteria for rejecting the position. The rationale is that a mate exists only if the opponent is highly constrained. However, this rule also limits the search to that part of the tree which does not branch much. Consequently, the number of positions increases only linearly with depth instead of exponentially.

A second new entry, on planning, comes from a program[40] that must find a path between two points on a two dimensional grid with only certain elementary jumps legal, e.g., a Knight's jump, and with arbitrary domains prohibited. Prior solutions to problems of the form *get from* X_1 *to* X_2 were generalized to become plans for getting from any X to (symbolically) $X + (X_2 - X_1)$ by ignoring the order of application of the operators. This latter hopefully permitted the adjustment of the plan to take into account prohibited domains. Although the hope turned out to be vain, the study still repre-

Table 1—Types of Devices Used

- Termination of search at X
  - Evaluation functions:
    - Features defined by task area, e.g., mobility in chess[26-34]
    - Models, e.g., diagrams in geometry[11]
  - Depth of search[3,26]
  - Processing limits, e.g., effort, space[38]
  - Rate of growth, e.g., terminate if too many continuations[37]
- Restrictions of operators applied at X
  - Limit total number[18,26]
  - Determinants of selection:
    - Function of the operator, e.g., attack in chess[26,34]
    - Reduce the difference between X and X_d[27,35]
  - If operators given by schema:
    - Selection of feasible operators (those that won't fail) by matching schema input to X[1,38,41]
    - Selection of desirable operators by matching schema output to X_d[27,35]
- Creation of subproblems (new X_d to be obtained)
  - If operator inapplicable, obtain an X_d to which it will apply[27]
- Search strategies (which X of those already generated to use next?)
  - Evaluation functions: simplest;[25] estimated success;[1] estimated cost[38]
  - Fixed order: depth first;[26] breadth first[25]
- Order of application of operators
  - By features of task area, e.g., attack before defend in chess[26]
  - Randomly[15]
  - By success in past problems[15]
  - By success in other parts of present problem[22]
- Avoidance of duplication (of generating an X already obtained)[11]
- Planning (using solutions to associated problems as guides)
  - Simplified problem[2,27]
  - Recursive simplification until problem solvable[39]
  - Solutions to past problems[40]

sents one of the few attempts to focus on a specific problem solving device rather than a task.

An obvious, but powerful mechanism, lies behind several entries in Table 1. Given a desired object X_d and a process P which produces a sequence of candidate solutions $X_1, X_2, \ldots X_n$, then the problem of producing X_d can be solved by roughly n/2 productions and n/2 identity tests if the order of generation is essentially random with respect to X_d. However, if there exists a schema X(y), such that the different X are obtainable by substitution for the variable y, then, by matching X(y) to X_d, the value of y that makes X(y) identical with X_d can be determined directly. The cost is only that of the comparison (slightly more complicated than the identity test) and the substitution; more important, the cost no longer depends on n, the total number of candidates.

We emphasize this matching process, not because it is new—it has been used in every heuristic theorem prover—but because it continues to play an important role in current programs.

TOWARDS A GENERAL INTERNAL REPRESENTATION

It would be desirable if there existed a formal system so general that all problems, however represented externally, could be translated into the form of proving a theorem in this system; yet so specific that general proof methods could be developed for it. The first order predicate calculus is an impressive candidate for such a system. This logical calculus consists of all expressions that can be formed from a set of primitive constants combined according to the Boolean connectives (*and*, *or*, *implies*, *not*) and the quantifiers—*For all x P is true* or *There exists an x such that P is true*—where *P* is any proposition in the calculus, and *x* may take as values the primitive constants. Extensive progress in the mathematization of science and the reduction of mathematics to logic has led to the well founded faith that this calculus is adequate to the expression of all problems in science and technology that can be expressed with precision. These results do not say that translation is easy, nor that it is convenient when obtained, but in principle we may look on the predicate calculus as a general internal representation.

It is known that no effective procedure exists for solving the general proof problem in the predicate calculus. Yet the proof problem has undergone successive reductions and purifications, so that by now it has a simple and definite structure. It is not our purpose to review the continuing efforts to construct problem solving programs for the predicate calculus. Two recent papers[33,41] show where matters stand and give many examples. There is good progress. However, we would like to relate the current formulation of this proof problem to heuristic search.

The objects consist of lists of expressions, each expression being a (possible) composite function. For example:

$$X_1\text{: } [P(f(x),c,g(f(c),b)),\bar{R}(y,g(z,b))]$$

$$X_2\text{: } [\bar{P}(w,z,v)]$$

$$X_3\text{: } [R(b,x)]$$

The ultimate operands can either be constants (b,c) or variables (v,w,x,y,z). The main functions (P,R) are predicates and can appear either asserted (P) or negated ($\overline{P}$). Each object represents the disjunction of its predicates; universal quantification is assumed over all variables; and the functions and constants occur because of existential quantification, e.g., f(x) is an object which exists for each x.

There is a single operator, Q, which takes two inputs (other operators are possible, but this one will do):

Q(X,X′): Take the union of X and X′, cancelling predicates that are the negation of each other and have identical operands.

For example:

$$Q(X_1,X_2) = [\overline{R}(y,g(c,b))] \qquad X_4$$

where the P and $\overline{P}$ expressions cancel after appropriate substitutions for w, z and v. The original situation consists of a set of objects, say X_1, X_2 and X_3 (in interpretation, their conjunction). The final desired object is the empty set, i.e., the one obtained by finally cancelling everything out. This corresponds to showing that a contradiction obtains; namely, the assertion of a predicate and its negation. Proving a theorem can always be put into the form of assuming its negation and showing a contradiction. In our example, applying Q to X_1 and X_2 as indicated brings us part way toward the empty set; applying Q again to X_4 and X_3 finishes the job.

It is known in this formulation that if the original theorem is provable, then there is a path in the tree that leads to the empty set. Of course, there is no telling how far out it is, so there is no way of deciding when a theorem is not provable. Thus, the problem looks identical to that, say, of finding the checkmate position in chess. More important, the early theorem provers (when interpreted in this formulation) proceeded by making substitutions systematically and testing afterwards whether the null set had been obtained. In contrast, the current programs match expressions in order to produce combinations that reduce the size of the resulting object, i.e., bring it nearer to the empty set. As already noted, this is exactly the technique that provided much of the power for the early heuristic theorem provers. It is possible to demonstrate this substantial identity of formulation and procedure by giving GPS[27] problems in the above formulation. The encodings are somewhat clumsy, so that only small problems can be used, but GPS is able to accept the task and find proofs.[7]

Most of the work in theorem proving in the predicate calculus has focussed on the proof problem, ignoring the initial translation. Simple problems in well formalized mathematics, such as group theory, provide examples of appropriate difficulty. Although, as mentioned, there is well founded faith that precisely formulated problems can be expressed in the predicate calculus, there is (naturally enough) less known about the expressibility of problems in not yet formalized domains. There has been a continuing effort, identified mostly with work on the Advice Taker,[21] to formalize problems of everyday reasoning using the first order predicate calculus as the internal representation. A typical toy problem is the *banana problem* in which a monkey can see a banana, cannot reach it, but can reach a box upon which to stand. The problem (after formalization) is to prove that the monkey can get the banana.

Several programs working on problems of this sort seem to exist, although only one appears to be documented.[3] This program has succeeded in solving the several toy problems proposed from time to time by McCarthy. It is an example of a heuristic search program, objects being logic expressions, and the operators the various rules of inference. As in other theorem provers, the match routine plays an important role in selecting out the inferences that will work.

An important issue about the nature of the internal representation is contained in these efforts. Shall the internal representation be a description of the problem or shall it be a more direct representation, i.e., a model? The distinction can be made clear by the banana problem. Using the predicate calculus, the objects are linguistic expressions asserting some fact about the situation. Problem solving proceeds in the space of expressions by means of general rules of inference. An alternative (realized in the chess and checker programs) is to have a data structure that represents the physical situation: the monkey, the banana, the box, their positions in the cage, etc. The operators effect transformations on the data structure that move the monkey about the cage, move the box, etc. There are many similarities in these two formulations, as there must be since they represent the same simple situation. But there are differences too. With descriptions it is much easier to represent imperfect information: how does one represent in the model that the monkey is in one place *or* another? With descriptions it is possible to have a uniform problem solving method for all problems—this is the appeal of the predicate calculus generally. But models can carry implicitly information that must be expressed if one describes the situation. That the monkey can only be in one place at one time follows from the structure of the model; it may have to be stated and used repeatedly in inferences if one is operating descriptively.

We do not wish to convert this distinction to a dichotomy: both representations could be used. The barriers to doing so are primarily the organizational problems of constructing more complex programs (not to be underestimated) and the difficulties of interrelation when both are changing as the result of processing.

TOWARDS A GENERAL PROBLEM STATEMENT

A general problem solver requires a general language for its problem statement, and the appearance of several programs that use some approximation to natural language as input constitutes the strongest indication of the current concern with generality.* These programs follow upon an intensive period of development of mechanical syntax analysis,[4] and earlier question-answering programs whose action part was more complex retrieval than problem solving.[12,19]

As an example of the tasks, one program attempts to solve simple story problems in algebra.[5] A typical problem and solution is:

MARY IS TWICE AS OLD AS ANN WAS WHEN MARY WAS AS OLD AS ANN IS NOW. IF MARY IS 24 YEARS OLD, HOW OLD IS ANN?

ANN'S AGE IS 18.

Another program,[17] although still incomplete, is intended to work on simple physical problems, such as occur in the elementary calculation of physics and chemistry. Another[16] deals with an external situation involving geometrical line drawings and statements about these drawings in a simplified natural language; the problem is to determine whether the assertions are true. Notice in all these programs that the problem solving capabilities are modest in human terms. However, the programs themselves are impressive, since we are not yet used to programs with this degree of flexibility. In all these programs there is a clear separation between a translation of the external problem statement into an internal representation, and the application of problem solving methods to this representation.

*See Refs. 5, 6, 16, 17 and 29.

As a more detailed example, consider SIR,[29] which answers questions about a data base that it has accumulated in interaction with a human. A typical protocol is shown below.

MAN: EVERY FIREMAN OWNS A PAIR-OF-RED-SUSPENDERS.
SIR: I UNDERSTAND.
MAN: DOES A PAIR-OF-RED-SUSPENDERS OWN A PAIR-OF-RED-SUSPENDERS Q
SIR: NO, THEY ARE THE SAME.
MAN: DOES A DOCTOR OWN A PAIR-OF-RED-SUSPENDERS Q
SIR: INSUFFICIENT INFORMATION.
MAN: A FIRECHIEF IS A FIREMAN.
SIR: I UNDERSTAND.
MAN: DOES A FIRECHIEF OWN A PAIR-OF-RED-SUSPENDERS Q
SIR: YES.

SIR works from the data presented to it, but it knows about the meaning of words such as OWN and EVERY. It is expected to handle insufficient an ambiguous information in a reasonable way, and to make elementary inferences.

The internal representation used by SIR is a network of nodes, each one representing one of the objects SIR can discuss, e.g., FIREMAN. Relationships are represented by links from one object to another, labeled according to the relationship, e.g., OWN. Given the node for FIREMAN, to find the nodes corresponding to objects that FIREMAN OWNS is an elementary operation. Thus, the total collection of information available to SIR is welded into a large, interlinked network.

The transormation into the internal representation uses formats which are matched against the input expressions. These formats both recognize what action is required and identify the appropriate operands. If the input is a statement, then the network is augmented; if a question, then an attempt is made to determine the answer.

Of most interest to us is how SIR's problem solving relates to heuristic search. In the running program each form has its own special program to carry out the search for the answer recognized by the form. These programs take into account the various properties of the relations, e.g., transitivity and symmetry, and organize a search of the net accordingly. This network forms a model of the situation, not a description of it. In fact, one could imagine a different SIR in which the internal representation was simply a translation of the external expressions into the predicate calculus and the problem solver would be a theorem prover of the types already discussed. Indeed, such a scheme is considered in discussing how to generalize SIR,[30] using the model as an efficient retrieval scheme from which useful statements for the theorem prover are constructed.

But the model is also powerful as a space in which to problem solve. To see this, consider the transitive law (which typifies the inferences made by SIR) in a form used for question-answering:

> X INCLUDES Z if there exists a Y such that X INCLUDES Y and Y INCLUDES Z.

A theorem prover working with statements has this law as a general rule of inference. To be applied, the available objects (expressions such as P INCLUDES S, S INCLUDES T, etc.) must be searched until a pair is found that can be matched against the two forms. In SIR's model, if we are at P (say) we can immediately find those nodes that are related by INCLUDES, say S; and once at S we can again immediately find those related by INCLUDES, e.g., T. There is no matching to be done; the identifications were made as the net was constructed. The model makes at *acquisition time* all the immediate inferences from an object that can be made, leaving to *problem solving time* all the searching of the sequences. If a problem solver asks many questions involving the same elementary steps; then this model has some of the same virtues over the descriptive theorem prover that a compiler has over an interpreter. On the other hand, this model is not possible if new objects are created in the process of reasoning (as they are in the usual theorem provers, or in the chess and checker players).

CONCLUSION

By selecting certain current problem solving programs for discussion we have attempted to emphasize, first, the increased concern with problems of generality; and second, the continued use of tools which proved so powerful for the initial efforts—heuristic search and matching. We would like to press the case further, and pose the question whether the heuristic search problem formulation is indeed as fundamental as it seems, or whether it is just one of many. Perhaps the same devices show up over and over because we know each other's programs so well.

But this question is already well beyond the scope of the present paper. There have been several successful programs that induct relationships[8,28,36] and discover concepts.[14] These programs do not use in any obvious way a heuristic search formulation, although matching is important. But we have not had space to discuss them. There are other problem solving programs that do involve search, but whose structure we have not examined.[2,13,32] Furthermore, other problem formulations of great generality exist; most notably for pattern recognition and for constrained maximization. We have not made clear the relationship of these to the present formulation of heuristic search. Perhaps most important, it is still possible to see the present collection of tasks that have been subject to investigation by problem solving programs as a small sample, highly biassed toward the formal and *well defined*. In contrast, the full range of human intellectual activity—which remains the major guidepost to what problem solving programs ought to be able to do—seems mostly ill defined.[24,31]

In short, the types of programs we have chosen as representative of research into problem solving constitute a hazard on the future as well as a report on the recent past.

REFERENCES

1. S. Amarel, "On the Automatic Formation of a Computer Program Which Represents a Theory," *in* M. C. Yovits, G. Jacobi and G. Goldstein (eds.), *Self-Organizing Systems*, Spartan, Washington, D. C., 1962.
2. E. Berlekamp, "Program for Double Dummy Bridge Problems," *J. ACM*, 10, 1963. pp. 357–364.
3. F. Black, *A Deductive Question Answering System*, Unpublished Ph.D. thesis, Harvard University, 1964.
4. D. G. Bobrow, "Syntactic Analysis of English by Computer—A Survey," *Proc. Fall Joint Computer Conf.*, 24, 1963. pp. 365–387.
5. D. G. Bobrow, "A Question-Answering System for High School Algebra Word Problem," *Proc. Fall Joint Computer Conf.*, 26, 1964. pp. 591–614.
6. J. Darlington, *Translating Ordinary Language into Symbolic Logic*, Memo MAC-M-149, Project MAC, MIT, 1964.
7. G. Ernst and A. Newell, unpublished research, 1965.

8. T. G. Evans, "A Heuristic Program to Solve Geometry-Analogy Problems," *Proc. Spring Joint Computer Conf.*, 25, 1964. pp. 327–338.
9. E. Feigenbaum and J. Feldman (eds.), *Computers and Thought*, McGraw-Hill, New York, 1963.
10. J. Feldman, F. M. Tonge and H. Kantor, "Empirical Explorations of a Hypotheses-Testing Model of Binary Choice Behavior," *in* A. C. Hoggatt and F. E. Balderston (eds.), *Symposium on Simulation Models*, South-Western, Cincinnati, Ohio, 1963.
11. H. Gelernter, J. R. Hansen and D. W. Loveland, "Empirical Explorations of the Geometry-Theorem Proving Machine," *Proc. Western Joint Computer Conf.*, 17, 1960. pp. 143–147.
12. B. G. Green, A. K. Wolf, C. Chomsky and K. Laughery, "Baseball: An Automatic Question Answerer," *Proc. Western Joint Computer Conf.*, 19, 1961. pp. 219–224. (Reprinted in Feigenbaum and Feldman, above.)
13. A. Horman, "Programs for Machine Learning," *Information and Control* (Part I), 5, 4, 1962. pp. 347–367; (Part II), 7, 1, 1964. pp. 55–77.
14. E. Johnson, *An Information-Processing Model of One Kind of Problem Solving*, Psychological Monographs, Whole No. 581, 1964.
15. R. L. Karg and G. L. Thompson, "A Heuristic Approach to Solving Traveling Salesman Problems," *Management Science*, 10, 2, 1964. pp. 225–248.
16. R. A. Kirsch, "Computer Interpretation of English Language Test and Picture Patterns," *IEEE Trans. Electronic Computers*, EC-13, 1964. pp. 363–376.
17. D. J. Kuck and G. K. Krulee, "A Problem Solver with Formal Descriptive Inputs," *in* J. T. Tou and R. H. Wilcox (eds.), *Computer and Information Sciences*, Spartan, Washington, D.C., 1964. pp. 344–374.
18. A. A. Kuehn and M. J. Hamburger, "A Heuristic Program for Locating Warehouses," *Management Sciences*, 10, 1964. pp. 225–248.
19. R. K. Lindsay, "Inferential Memory as the Basis of Machines Which Understand Natural Languages," *in* E. Feigenbaum and J. Feldman (eds.), *Computers and Thought*, McGraw-Hill, New York, 1963. pp. 217–233.
20. R. London, "A Computer Program for Discovering and Proving Recognition Rules for Backus Normal Form Grammars," *Proc. 19th Annual Conf., ACM*, 1964. pp. A1.3-1–A1.3-7.
21. J. McCarthy, "Programs with Common Sense," *in* D. V. Blake and A. M. Uttley (eds.), *Symposium on the Mechanization of Thought Processes*, H.M.S.O., London, 1959. pp. 75–84.
22. J. McCarthy, Private communication about a mate-in-two chess program, 1959.
23. M. Minsky, "Steps toward Artificial Intelligence," *Proc. of the IRE*, 49, 1961. pp. 8–30. (Reprinted in Feigenbaum and Feldman, above.)
24. U. Neisser, "The Imitation of Man by Machine," *Science*, 139, 1963. pp. 193–197.
25. A. Newell, J. C. Shaw and H. A. Simon, "Empirical Explorations with the Logic Theory Machine," *Proc. Western Joint Computer Conf.*, 11, 1957. pp. 218–239. (Reprinted in Feigenbaum and Feldman, above.)
26. A. Newell, J. C. Shaw and H. A. Simon, "Chess Playing Programs and the Problem of Complexity," *IBM J. of Research and Development*, 2, 4, 1958. pp. 320–335. For other chess programs see references. (Reprinted in Feigenbaum and Feldman, above.)
27. A. Newell, J. C. Shaw and H. A. Simon, "Report on a General Problem Solver," *Information Processing*, UNESCO, Paris, 1959. (See also Feigenbaum and Feldman, above.)
28. M. Pivar and M. Finkelstein, "Automation, Using LISP, of Inductive Inference on Sequences," *in* E. C. Berkeley and D. G. Bobrow (eds.), *The Programming Language LISP: Its Operation and Applications*, Information International, Cambridge, Mass., 1964.
29. B. Raphael, "A Computer Program Which Understands," *Proc. Fall Joint Computer Conf.*, 26, 1964.
30. B. Raphael, *SIR: A Computer Program for Semantic Information Retrieval*, Project MAC, MIT, 1964.
31. W. R. Reitman, "Heuristic Decision Procedures, Open Constraints, and the Structure of Ill-Defined Problems," *in* M. W. Shelley and G. L. Bryan (eds.), *Human Judgments and Optimality*, John Wiley and Sons, New York, 1964.
32. W. R. Reitman, R. Shoup and R. Graves, "Argus: An Information Processing Model of Human Thinking," *Behavioral Science*, 9, 3, 1964. pp. 270–280.
33. J. A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," *J. ACM*, 12, 1, 1965. pp. 23–41.

34. A. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM J. Research and Development*, 3, 1959. pp. 211–229. (Reprinted in Feigenbaum and Feldman, above.)

35. H. A. Simon, "Experiments with a Heuristic Compiler," *J. ACM*, 10, 4, 1963. pp. 493–506.

36. H. A. Simon and K. Kotovsky, "Human Acquisition of Concepts for Sequential Patterns," *Psychological Review*, 70, 6, 1963. pp. 534–546.

37. H. A. Simon and P. Simon, "Trial and Error Search in Solving Difficult Problems: Evidence from the Game of Chess," *Behavioral Science*, 7, 4, 1962. pp. 425–429.

38. J. R. Slagle, "A Heuristic Program That Solves Symbolic Integration Problems in Freshman Calculus," *J. ACM*, 10, 4, 1963. pp. 507–520. (Reprinted in Feigenbaum and Feldman, above.)

39. F. M. Tonge, *A Heuristic Program for Assembly Line-Balancing*, Prentice-Hall, Englewood Cliffs, N. J., 1961. (See also Feigenbaum and Feldman, above.)

40. L. Travis, "Experiments with a Theorem-Utilizing Program," *Proc. Spring Joint Computer Conf.*, 25, 1964. pp. 339–358.

41. L. Wos, D. Carson and G. Robinson, "The Unit Preference Strategy in Theorem Proving," *Proc. Fall Joint Computer Conf.*, 26, 1964. pp. 615–622.