Psychological Review
Vol. 65, No. 3, 1958

ELEMENTS OF A THEORY OF HUMAN PROBLEM SOLVING

ALLEN NEWELL, J. C. SHAW
The RAND Corporation
AND HERBERT A. SIMON
Carnegie Institute of Technology

In this paper we shall set forth the elements of a theory of human problem solving, together with some evidence for its validity drawn from the currently accepted facts about the nature of problem solving. What questions should a theory of problem solving answer? First, it should predict the performance of a problem solver handling specified tasks. It should explain how human problem solving takes place: what processes are used, and what mechanisms perform these processes. It should predict the incidental phenomena that accompany problem solving, and the relation of these to the problem-solving process. For example, it should account for "set" and for the apparent discontinuities that are sometimes called "insight." It should show how changes in the attendant conditions—both changes "inside" the problem solver and changes in the task confronting him—alter problem-solving behavior. It should explain how specific and general problem-solving skills are learned, and what it is that the problem solver "has" when he has learned them.

Information Processing Systems

Questions about problem-solving behavior can be answered at various levels and in varying degrees of detail. The theory to be described here explains problem-solving behavior in terms of what we shall call *information processes.* If one considers the organism to consist of effectors, receptors, and a control system for joining these, then this theory is mostly a theory of the control system. It avoids most questions of sensory and motor activities. The theory postulates:

1. A control system consisting of a number of *memories,* which contain symbolized information and are interconnected by various ordering relations. The theory is not at all concerned with the physical structures that allow this symbolization, nor with any properties of the memories and symbols other than those it explicitly states.

2. A number of *primitive information processes,* which operate on the information in the memories. Each primitive process is a perfectly definite operation for which known physical mechanisms exist. (The mechanisms are not necessarily known to exist in the human brain, however—we are only concerned that the processes be described without ambiguity.)

3. A perfectly definite set of rules for combining these processes into whole *programs* of processing. From a program it is possible to deduce unequivocally what externally observable behaviors will be generated.

At this level of theorizing, *an explanation of an observed behavior of the organism is provided by a program of primitive information processes that generates this behavior.*

A program viewed as a theory of behavior is highly specific: it describes one organism in a particular class of situations. When either the situation or the organism is changed, the program must be modified. The program can be used as a theory—that is, as a predictor of behavior—in two distinct ways. First, it makes many precise predictions that can be tested in detail regarding the area of behavior it is designed to handle. For example, the theory considered in this paper predicts exactly how much difficulty an organism with the specified program will encounter in solving each of a series of mathematical problems: which of the problems it will solve, how much time (up to a proportionality constant) will be spent on each, and so on.

Second, there will be important qualitative similarities among the programs that an organism uses in various situations, and among the programs used by different organisms in a given situation. The program that a human subject uses to solve mathematical problems will be similar in many respects to the program he uses to choose a move in chess; the program one subject uses for any such task will resemble the programs used by other subjects possessing similar training and abilities. If there were no such similarities, if each subject and each task were completely idiosyncratic, there could be no theory of human problem solving. Moreover, there is some positive evidence, as we shall see, that such similarities and general characteristics of problem-solving processes do exist.

In this paper we shall limit ourselves to this second kind of validation of our theory of problem solving. We shall predict qualitative characteristics of human problem-solving behavior and compare them with those that have already been observed and described. Since all of the available data on the psychology of human problem solving are of this qualitative kind, no more detailed test of a program is possible at present. The more precise validation must wait upon new experimental work.[1]

In succeeding sections we shall describe an information-processing program for discovering proofs for theorems in logic. We shall compare its behavior qualitatively with that of human problem solvers. In general, the processes that compose the program are familiar from everyday experience and from research on human problem solving: searching for possible solutions, generating these possibilities out of other elements, and evaluating partial solutions and cues. From this standpoint there is nothing particularly novel about the theory. It rests its claims on other considerations:

1. It shows specifically and in detail how the processes that occur in human problem solving can be compounded out of elementary information processes, and hence how they can be carried out by mechanisms.

2. It shows that a program incorporating such processes, with appropriate organization, can in fact solve problems. This aspect of problem solving has been thought to be "mysterious" and unexplained because it was not understood how sequences of simple processes could account for the successful solution of complex problems. The theory dissolves the mystery by showing that nothing more need be added to the constitution of a successful problem solver.

Relation to Digital Computers

The ability to specify programs precisely, and to infer accurately the behavior they will produce, derives from the use of high-speed digital computers. Each specific theory—each program of

[1] Several studies of individual and group problem-solving behavior with logic problems have been carried out by O. K. Moore and Scarvia Anderson (5). The problems Moore and Anderson gave their subjects are somewhat different from those handled by our program, and hence a detailed comparison of behavior is not yet possible. We are now engaged, with Peter Houts, in replicating and extending the experiments of Moore and Anderson with human subjects and at the same time modifying our program to predict the human laboratory behavior in detail.

information processes that purports to describe some human behavior—is coded for a computer. That is, each primitive information process is coded to be a separate computer routine, and a "master" routine is written that allows these primitive processes to be assembled into any system we wish to specify. Once this has been done, we can find out exactly what behavior the purported theory predicts by having the computer "simulate" the system.

We wish to emphasize that we are not using the computer as a crude analogy to human behavior—we are not comparing computer structures with brains, nor electrical relays with synapses. Our position is that the appropriate way to describe a piece of problem-solving behavior is in terms of a program: a specification of what the organism will do under varying environmental circumstances in terms of certain elementary information processes it is capable of performing. This assertion has nothing to do—directly—with computers. Such programs could be written (now that we have discovered how to do it) if computers had never existed.[2] A program is no more, and no less, an analogy to the behavior of an organism than is a differential equation to the behavior of the electrical circuit it describes. Digital computers come into the picture only because they can, by appropriate programming, be induced to execute the same sequences of information processes that humans execute when they are solving problems. Hence, as we shall see, these programs describe both human and machine problem solving at the level of information processes.[3]

With this discussion of the relation of programs to machines and humans behind us, we can afford to relax into convenient, and even metaphoric, uses of language without much danger of misunderstanding. It is often convenient to talk about the behavior implied by a program as that of an existing physical mechanism doing things. This mode of expression is legitimate, for if we take the trouble to put any particular program in a computer, we have in fact a machine that behaves in the way prescribed by the program. Similarly, for concreteness, we will often talk as if our theory of problem solving consisted of statements about the ability of a computer to do certain things.

THE LOGIC THEORIST

We can now turn to an example of the theory. This is a program capable of solving problems in a particular domain—capable, specifically, of discovering proofs for theorems in elementary symbolic logic. We shall call this program the Logic Theorist (LT).[4] We assert that the behavior of this program, when the stimulus consists of the instruction that it prove a particular theorem, can be used to predict the be-

[2] We can, in fact, find a number of attempts in the psychological literature to explain behavior in terms of programs—or the prototypes thereof. One of the most interesting, because it comes relatively close to the modern conception of a computer program, is Adrian de Groot's analysis of problem solving by chess players (2). The theory of de Groot is based on the thought-psychology of Selz, a somewhat neglected successor to the Wurzburg school. Quite recently, and apparently independently, we find the same idea applied by Jerome S. Bruner and his associates to the theory of concept formation (1). Bruner uses the term "strategy," derived from economics and game theory, for what we have called a program.

[3] For a fuller discussion of this point see (9).

[4] In fact, matters are a little more complicated, for in the body of this paper we will consider both the basic program of LT and a number of variants on this program. We will refer to all of these variants, interchangeably, as "LT." This will not be confusing, since the exact content of the program we are considering at any particular point will always be clear from the context.

havior of (certain) humans when they are faced with the same problem in symbolic logic.

The program of LT was not fashioned directly as a theory of human behavior; it was constructed in order to get a program that would prove theorems in logic. To be sure, in constructing it the authors were guided by a firm belief that a practicable program could be constructed only if it used many of the processes that humans use. The fact remains that the program was not devised by fitting it directly to human data. As a result, there are many details of LT that we would not expect to correspond to human behavior. For example, no particular care was exercised in choosing the primitive information processes to correspond, point by point, with elementary human processes. All that was required in writing the program was that the primitive processes constitute a sufficient set and a convenient set for the type of program under study.

Since LT has been described in detail elsewhere (6, 8), the description will not be repeated here. It will also be unnecessary to describe in detail the system of symbolic logic that is used by LT. For those readers who are not familiar with symbolic logic, we may remark that problems in the sentential calculus are at about the same level of difficulty and have somewhat the same "flavor" as problems in high school geometry.[5]

Design of the Experiments

First we will describe the overt behavior of LT when it is presented with problems in elementary symbolic logic. In order to be concrete, we will refer to an experiment conducted on a digital computer. We take an ordinary general-purpose digital computer,[6] and store in its memory a program for interpreting the specifications of LT. Then we load the program that specifies LT. The reader may think of this program as a collection of techniques that LT has acquired for discovering proofs. These techniques range from the ability to read and write expressions in symbolic logic to general schemes for how a proof might be found.

Once we have loaded this program and pushed the start button, the computer, to all intents and purposes, *is* LT. It already knows how to do symbolic logic, in the sense that the basic rules of operation of the mathematics are already in the program (analogously to a human's knowing that "equals added to equals give equals" in elementary algebra).

We are now ready to give LT a task. We give it a list of the expressions (axioms and previously proved theorems) that it may take as "given" for the task at hand. These are stored in LT's memory. Finally, we present LT with another expression and instruct it to discover a proof for this expression.

From this point, the computer is on its own. The program plus the task uniquely determines its behavior. It attempts to find a proof—that is, it tries various techniques, and if they don't

[5] LT employs the sentential calculus as set forth in Chapters 1 and 2 of A. N. Whitehead and Bertrand Russell, *Principia Mathematica* (10)—the "classic" of modern symbolic logic. A simple introduction to the system of *Principia* will be found in (3).

[6] The experiments described here were carried out with the RAND JOHNNIAC computer. The JOHNNIAC is an automatic digital computer of the Princeton type. It has a word length of 40 bits, with two instructions in each word. Its fast storage consists of 4,096 words of magnetic cores, and its secondary storage consists of 9,216 words on magnetic drums. Its speed is about 15,000 operations per second. The programming techniques used are described more fully in (6). The experiments are reported in more detail in (7).

work, it tries other techniques. If LT finds a legitimate proof, it prints this out on a long strip of paper. There is, of course, no guarantee that it will find a proof; after working for some time, the machine will give up—that is, it will stop looking for a proof.

Now the experimenters know exactly what is in the memory of LT when it starts—indeed, they created the program. This, however, is quite different from saying that the experimenters can predict everything LT will do. In principle this is possible; but in fact the program is so complex that the only way to make detailed predictions is to employ a human to simulate the program by hand. (A human can do anything a digital computer can do, although it may take him considerably longer.)

1. As the initial experiment, we stored the axioms of *Principia Mathematica*, together with the program, in the memory of LT, and then presented to LT the first 52 theorems in Chapter 2 of *Principia* in the sequence in which they appear there. LT's program specified that as a theorem was proved it was stored in memory and was available, along with the axioms, as material for the construction of proofs of subsequent theorems. With this program and this order of presentation of problems, LT succeeded in proving 38 (73%) of the 52 theorems. About half of the proofs were accomplished in less than a minute each; most of the remainder took from one to five minutes. A few theorems were proved in times ranging from 15 minutes to 45 minutes. There was a strong relation between the times and the lengths of the proofs—the time increasing sharply (perhaps exponentially) with each additional proof step.

2. The initial conditions were now restored by removing from LT's memory the theorems it had proved. (Translate: "A new subject was obtained who knew how to solve problems in logic but was unfamiliar with the particular problems to be used in the experiment.") When one of the later theorems of Chapter 2 (Theorem 2.12) was presented to LT, it was not able to find a proof, although when it had held the prior theorems in memory, it had found one in about ten seconds.

3. Next, an experiment was performed intermediate between the first two. The axioms and Theorem 2.03 were stored in memory, but not the other theorems prior to Theorem 2.12, and LT was again given the task of proving the latter. Now, using Theorem 2.03 as one of its resources, LT succeeded—in fifteen minutes—where it had failed in the second experiment. The proof required three steps. In the first experiment, with all prior theorems available, the proof required only one step.

Outcome of the Experiments

From these three series of experiments we obtain several important pieces of evidence that the program of LT is qualitatively like that of a human faced with the same task. The first, and most important, evidence is that LT does in fact succeed in finding proofs for a large number of theorems.

Let us make this point quite clear. Since LT can actually discover proofs for theorems, its program incorporates a *sufficient* set of elementary processes arranged in a sufficiently effective strategy to produce this result. Since no other program has ever been specified for handling successfully these kinds of problem-solving tasks, no definite alternative hypothesis is available. We are well aware of the standard argument that "similarity of function does not imply similarity of process." However useful a caution this may be, it should not blind us to the fact that specification of a set of mechanisms sufficient to produce observed behavior is strong confirmatory evidence for the theory embodying these mechanisms, especially when it is contrasted with theories that cannot establish their sufficiency.

The only alternative problem-solving mechanisms that have been completely specified for these kinds of tasks are simple algorithms that carry out exhaustive searches of all possibilities, substituting "brute force" for the selective search of LT. Even with the speeds available to digital computers, the principal algorithm we have devised as an alternative to LT would require times of the order of hundreds or even thousands of years to prove theorems that LT proves in a few minutes. LT's success does not depend on the "brute force" use of a computer's speed, but on the use of heuristic processes like those employed by humans.[7] This can be seen directly from examination of the program, but it also shows up repeatedly in all the other behavior exhibited by LT.

The second important fact that emerges from the experiments is that LT's success depends in a very sensitive way upon the order in which problems are presented to it. When the sequence is arranged so that before any particular problem is reached some potentially helpful intermediate results have already been obtained, then the task is easy. It can be made progressively harder by skipping more and more of these intermediate stepping-stones. Moreover, by providing a single "hint," as in the third experiment (that is, "Here is a theorem that might help"), we can induce LT to solve a problem it had previously found insoluble. All of these results are easily reproduced in the laboratory with humans. To compare LT's behavior with that of a human subject, we would first have to train the latter in symbolic logic (this is equivalent to reading the program into LT), but without using the specific theorems of Chapter 2 of *Principia Mathematica* that are to serve as problem material. We would then present problems to the human subject in the same sequence as to LT. For each new sequence we would need naive subjects, since it is difficult to induce a human subject to forget completely theorems he has once learned.

[7] A quantitative analysis of the power of the heuristics incorporated in LT will be found in (7).

Performance Processes in the Logic Theorist

We can learn more about LT's approximation to human problem solving by instructing it to print out some of its intermediate results—to work its problems on paper, so to speak. The data thus obtained can be compared with data obtained from a human subject who is asked to use scratch paper as he works a problem, or to think aloud.[8] Specifically, the computer can be instructed to print out a record of the subproblems it works on and the methods it applies, successfully and unsuccessfully, while seeking a solution. We can obtain this information at any level of detail we wish, and make a correspondingly detailed study of LT's processes.

To understand the additional information provided by this "thinking aloud" procedure, we need to describe a little

[8] Evidence obtained from a subject who thinks aloud is sometimes compared with evidence obtained by asking the subject to theorize introspectively about his own thought processes. This is misleading. Thinking aloud is just as truly behavior as is circling the correct answer on a paper-and-pencil test. What we infer from it about other *processes* going on inside the subject (or the machine) is, of course, another question. In the case of the machine, the problem is simpler than in the case of the human, for we can determine exactly the correspondence between the internal processes and what the machine prints out.

more fully how LT goes about solving problems. This description has two parts: (*a*) specifying what constitutes a proof in symbolic logic; (*b*) describing the methods that LT uses in finding proofs.

Nature of a Proof

A proof in symbolic logic (and in other branches of logic and mathematics) is a sequence of statements such that each statement: (*a*) follows from one or more of the others that precede it in the sequence, or (*b*) is an axiom or previously proved theorem.[9] Here "follows" means "follows by the rules of logic."

LT is given four rules of inference:

Substitution. In a true expression (for example, "[p or p] implies p") there may be substituted for any variable a new variable or expression, provided that the substitution is made throughout the original expression. Thus, by substituting p or q for p in the expression "(p or p) implies p," we get: "([p or q] or [p or q]) implies (p or q)" but *not:* "([p or q] or p) implies p."

[9] The axioms of symbolic logic and the theories that follow from them are all tautologies, true by virtue of the definitions of their terms. It is their tautological character that gives laws of logic their validity, independent of empirical evidence, as rules of inductive inference. Hence the very simple axioms that we shall use as examples here will have an appearance of redundancy, if not triviality. For example, the first axiom of *Principia* states, in effect, that "if any particular sentence (call it p) is true, or if that same sentence (p) is true, then that sentence (p) is, indeed, true"—for example, "if frogs are fish, or if frogs are fish, then frogs are fish." The "if—then" is trivially and tautologically true irrespective of whether p is true, for in truth frogs are not fish. Since our interest here is in problem solving, not in logic, the reader can regard LT's task as one of manipulating symbols to produce desired expressions, and he can ignore the material interpretations of these symbols.

Replacement. In a true expression a connective ("implies," etc.) may be replaced by its definition in terms of other connectives. Thus "A implies B" is defined to be "not-A or B"; hence the two forms can be used interchangeably.

Detachment. If "A" is a true expression and "A implies B" is a true expression, then B may be written down as a true expression.

Syllogism (Chaining). It is possible to show by two successive applications of detachment that the following is also legitimate: If "a implies b" is a true expression and "b implies c" is a true expression, then "a implies c" is also a true expression.

Proof Methods

The task of LT is to construct a proof sequence deriving a problem expression from the axioms and the previously proved theorems by the rules of inference listed above. But the rules of inference, like the rules of any mathematical system or any game, are permissive, not mandatory. That is, they state what sequences *may* legitimately be constructed, not what particular sequence should be constructed in order to achieve a particular result (i.e., to prove a particular problem expression). The set of "legal" sequences is exceeding large, and to try to find a suitable sequence by trial and error alone would almost always use up the available time or memory before it would exhaust the set of legal sequences.[10]

To discover proofs, LT uses *methods* which are particular combinations of information processes that result in coordinated activity aimed at progress in a particular direction. LT has four methods (it could have more): *substi-*

[10] See (7). The situation here is like that in chess or checkers where the player knows what moves are legal but has to find in a reasonable time a move that is also "suitable"—that is, conducive to winning the game.

tution, detachment, forward chaining, and *backward chaining*. Each method focuses on a single possibility for achieving a link in a proof.

The substitution method attempts to prove an expression by generating it from a known theorem employing substitutions of variables and replacements of connectives.

The detachment method tries to work backward, utilizing the rule of detachment to obtain a new expression whose proof implies the proof of the desired expression. This possibility arises from the fact that if B is to be proved, and we already know a theorem of the form "A implies B," then proof of A is tantamount to proof of B.

Both chaining methods try to work backward to new problems, using the rule of syllogism, analogously to the detachment method. Forward chaining uses the fact that if "a implies c" is desired and "a implies b" is already known, then it is sufficient to prove "b implies c." Backward chaining runs the argument the other way: desiring "a implies c" and knowing "b implies c" yields "a implies b" as a new problem.

The methods are the major organizations of processes in LT, but they are not all of it. There is an executive process that coordinates the use of the methods, and selects the subproblems and theorems upon which the methods operate. The executive process also applies any learning processes that are to be applied. Also, all the methods utilize common subprocesses in carrying out their activity. The two most important subprocesses are the *matching* process, which endeavors to make two given subexpressions identical, and the *similarity test,* which determines (on the basis of certain computed descriptions) whether two expressions are "similar" in a certain sense (for details, cf. 8).

LT can be instructed to list its attempts, successful and unsuccessful, to use these methods, and can list the new subproblems generated at each stage by these attempts. We can make this concrete by an example:

Suppose that the problem is to prove "p implies p." The statement "(p or p) implies p" is an axiom; and "p implies (p or p)" is a theorem that has already been proved and stored in the theorem memory. Following its program, LT first tries to prove "p implies p" by the substitution method, but fails because it can find no similar theorem in which to make substitutions.

Next, it tries the detachment method. Letting B stand for "p implies p," several theorems are found of the form "A implies B." For example, by substitution of not-p for q, "p implies (q or p)" becomes "p implies (not-p or p)"; this becomes, in turn, by replacement of "or" by "implies": "p implies (p implies p)." Discovery of this theorem creates a new subproblem: "Prove A"—that is, "prove p." This subproblem, of course, leads nowhere, since p is not a universally true theorem, hence cannot be proved.

At a later stage in its search LT tries the chaining method. Chaining forward, it finds the theorem "p implies (p or p)" and is then faced with the new problem of proving that "(p or p) implies p." This it is able to do by the substitution method, when it discovers the corresponding axiom.

All of these steps, successful and unsuccessful, in its proof—and the ones we have omitted from our exposition, as well—can be printed out to provide us with a complete record of how LT executed its program in solving this particular problem.

SOME CHARACTERISTICS OF THE PROBLEM-SOLVING PROCESS

Using as our data the information provided by LT as to the methods it tries, the sequence of these methods, and the theorems employed, we can ask whether its procedure shows any resemblance to the human problem-solving process as it has been described in psychological literature. We find that there are, indeed, many such resemblances, which we summarize under the following headings: set, insight, concept formation, and structure of the problem-subproblem hierachy.

Set

The term "set," sometimes defined as "a readiness to make a specified response to a specified stimulus" (4, p. 65), covers a variety of psychological phenomena. We should not be surprised to find that more than one aspect of LT's behavior exhibits "set," nor that these several evidences of set correspond to quite different underlying processes.

1. Suppose that after the program has been loaded in LT, the axioms and a sequence of problem expressions are placed in its memory. Before LT undertakes to prove the first problem expression, it goes through the list of axioms and computes a description of each for subsequent use in the "similarity" tests. For this reason, the proof of the first theorem takes an extra interval of time amounting, in fact, to about twenty seconds. Functionally and phenomenologically, this computation process and interval represent a *preparatory set* in the sense in which that term is used in reaction-time experiments. It turns out in LT that this preparatory set saves about one third of the computing time that would otherwise be required in later stages of the program.

2. *Directional set* is also evident in LT's behavior. When it is attempting a particular subproblem, LT tries first to solve it by the substitution method. If this proves fruitless, and only then, it tries the detachment method, then chaining forward, then chaining backward. Now when it searches for theorems suitable for the substitution method, it will not notice theorems that might later be suitable for detachment (different similarity tests being applied in the two cases). It attends single-mindedly to possible candidates for substitution until the theorem list has been exhausted; then it turns to the detachment method.

3. Hints and the change in behavior they induce have been mentioned earlier. Variants of LT exist in which the order of methods attempted by LT, and the choice of units in describing expressions, depend upon appropriate hints from the experimenter.

4. Effects from directional set occur in certain learning situations—as illustrated, for example, by the classical experiments of Luchins. Although LT at the present time has only a few learning mechanisms, these will produce strong effects of directional set if problems are presented to LT in appropriate sequences. For example, it required about 45 minutes to prove Theorem 2.48 in the first experiment because LT, provided with all the prior theorems, explored so many blind alleys. Given only the axioms and Theorem 2.16, LT proved Theorem 2.48 in about 15 minutes because it now considered a quite different set of possibilities.

The instances of set observable in the present program of LT are natural and unintended by-products of a program constructed to solve problems in an efficient way. In fact, it is difficult to see how we could have avoided such effects. In its simplest aspect, the problem-solving process is a search for a solution in a very large space of possible solutions. The possible solutions must be examined in *some* particular sequence, and if they are, then certain possible solutions will be examined before others. The particular rule that induces the order of search induces thereby a definite set in the ordinary psychological meaning of that term.

Preparatory set also arises from the need for processing efficiency. If certain information is needed each time a possible solution or group of solutions is to be examined, it may be useful to compute this information, once and for all, at the beginning of the problem-solving process, and to store it instead of recomputing it each time.

The examples cited show that set can arise in almost every aspect of the problem-solving process. It can govern the sequence in which alternatives are examined (the "method" set), it can select the concepts that are used in classifying perceptions (the "viewing" set), and it can consist in preparatory processes (the description of axioms).

None of the examples of set in LT relate to the way in which information is stored in memory. However, one would certainly expect such set to exist, and certain psychological phenomena bear this out—the set in association experiments, and so-called "incubation" processes. LT as it now stands is inadequate in this respect.

Insight

In the psychological literature, "insight" has two principal connotations: (*a*) "suddenness" of discovery, and (*b*) grasp of the "structure" of the problem, as evidenced by absence of trial and error. It has often been pointed out that there is no necessary connection between the absence of overt trial-and-error behavior and grasp of the problem structure, for trial and error may be perceptual or ideational, and no obvious cues may be present in behavior to show that it is going on.

In LT an observer's assessment of how much trial and error there is will depend on how much of the record of its problem-solving processes the computer prints out. Moreover, the amount of trial and error going on "inside" varies within very wide limits, depending on small changes in the program.

The performance of LT throws some light on the classical debate between proponents of trial-and-error learning and proponents of "insight," and shows that this controversy, as it is usually phrased, rests on ambiguity and confusion. LT searches for solutions to the problems that are presented it. This search must be carried out in some sequence, and LT's success in actually finding solutions for rather difficult problems rests on the fact that the sequences it uses are not chosen casually but do, in fact, depend on problem "structure."

To keep matters simple, let us consider just one of the methods LT uses —proof by substitution. The number of valid proofs (of *some* theorem) that the machine can construct by substitution of new expressions for the variables in the axioms is limited only by its patience in generating expressions. Suppose now that LT is presented with a problem expression to be proved by substitution. The crudest trial-and-error procedure we can imagine is for the machine to generate substitutions in a predetermined sequence that is independent of the expression to be proved, and to compare each of the resulting expressions with the problem expression, stopping when a pair are identical (cf. 7).

Suppose, now, that the generator of substitutions is constructed so that it is *not* independent of the problem expression—so that it tries substitutions in different sequences depending on the nature of the latter. Then, if the dependence is an appropriate one, the amount of search required on the average can be reduced. A simple strategy of this sort would be to try in the axioms only substitutions involving variables that actually appear in the problem expression.

The actual generator employed by LT is more efficient (and hence more "insightful" by the usual criteria) than this. In fact, it works backward from the problem expression, and takes into account necessary conditions that a substitution must satisfy if it is to work. For example, suppose we are substituting in the axiom "p implies (q or p)," and are seeking to prove "r implies (r or r)." Working backward, it is clear that *if* the latter expression can be obtained from the former by substitution at all, then the variable that must be substituted for p is r. This can be seen by examining the first variable in each expression, without considering the rest of the expression at all (cf. 7).

Trial and error is reduced to still smaller proportions by the method for searching the list of theorems. Only those theorems are extracted from the list for attempted substitution which are "similar" in a defined sense to the problem expression. This means, in practice, that substitution is attempted in only about ten per cent of the theorems. Thus a trial-and-error search of the theorem list to find theorems similar to the problem expression is substituted for a trial-and-error series of attempted substitutions in each of the theorems.

In these examples, the concept of proceeding in a "meaningful" fashion is entirely clear and explicit. Trial-and-error attempts take place in some "space" of possible solutions. To approach a problem "meaningfully" is to have a strategy that either permits the search to be limited to a smaller subspace, or generates elements of the space in an order that makes probable the discovery of one of the solutions early in the process.

We have already listed some of the most important elements in the program of LT for reducing search to tolerable proportions. These are: (*a*) the description programs to select theorems that are "likely" candidates for substitution attempts; (*b*) the process of working backwards, which uses information about the goal to rule out large numbers of attempts without actually trying them. In addition to these, the executive routine may select the sequence of subproblems to be worked on in an order that takes up "simple" subproblems first.

Concepts

Most of the psychological research on concepts has focused on the processes of their formation. The current version of LT is mainly a performance program, and hence shows no concept formation. There is in the program, however, a clearcut example of the use of concepts in problem solving. This is the routine for describing theorems and searching for theorems "similar" to the problem expression or some part of it in order to attempt substitutions, detachments, or chainings. All theorems having the same description exemplify a common concept. We have, for example, the concept of an expression that has a single variable, one argument place on its left side, and two argument places on its right side: "p implies (p or p)" is an expression exemplifying this concept; so is "q implies (q implies q)."

The basis for these concepts is purely pragmatic. Two expressions having the same description "look alike" in some undefined sense; hence, if we are seeking to prove one of them as a theorem, while the other is an axiom or theorem already proved, the latter is likely construction material for the proof of the former.

Hierarchies of Processes

Another characteristic of the behavior of LT that resembles human problem-solving behavior is the hierarchical structure of its processes. Two kinds of hierarchies exist, and these will be described in the next two paragraphs.

In solving a problem, LT breaks it down into component problems. First of all, it makes three successive attempts: a proof by substitution, a proof by detachment, or a proof by chaining. In attempting to prove a theorem by any of these methods, it divides its task into two parts: first, finding likely raw materials in the form of axioms or theorems previously proved; second, using these materials in matching. To find theorems similar to the problem expression, the first step is to compute a description of the problem expression; the second step is to search the list of theorems for expressions with the same description. The description-computing program divides, in turn, into a program for computing the number of levels in the expression, a program for computing the number of distinct variables, and a program for computing the number of argument places.

LT has a second kind of hierarchy in the generation of new expressions to be proved. Both the detachment and chaining methods do not give proofs directly but, instead, provide new alternative expressions to prove. LT keeps a list of these subproblems, and, since they are of the same type as the original problem, it can apply all its problem-solving methods to them. These methods, of course, yield yet other subproblems, and in this way a large network of problems is developed during the course of proving a given logic expression. The importance of this type of hierarchy is that it is not fixed in advance, but grows in response to the problem-solving process itself, and shows some of the flexibility and transferability that seem to characterize human higher mental processes.

The problem-subproblem hierarchy in LT's program is quite comparable with the hierarchies that have been discovered by students of human problem-solving processes, and particularly by de Groot in his detailed studies of the thought methods of chess players (2, pp. 78–83, 105–111). Our earlier discussion of insight shows how the program structure permits an efficient combination of trial-and-error search with systematic use of experience and cues in the total problem-solving process.

SUMMARY OF THE EVIDENCE

We have now reviewed the principal evidence that LT solves problems in a manner closely resembling that exhibited by humans in dealing with the same problems. First, and perhaps most important, it is in fact capable of finding proofs for theorems—hence incorporates a system of processes that is sufficient for a problem-solving mechanism. Second, its ability to solve a particular problem depends on the sequence in which problems are presented to it in much the same way that a human subject's behavior depends on this sequence. Third, its behavior exhibits both preparatory and directional set. Fourth, it exhibits insight both in the sense of vicarious trial and error leading to "sudden" problem solution, and in the sense of employing heuristics to keep the total amount of trial and error within reasonable bounds. Fifth, it employs simple concepts to classify the expressions with which it deals. Sixth, its program exhibits a complex organized hierarchy of problems and subproblems.

COMPARISON WITH OTHER THEORIES

We have proposed a theory of the higher mental processes, and have shown how LT, which is a particular exemplar of the theory, provides an explanation for the processes used by humans to solve problems in symbolic logic. What is the relation of this explanation to others that have been advanced?

Associationism

The broad class of theories usually labelled "associationist" share a generally behaviorist viewpoint and a commitment to reducing mental functions to elementary, mechanistic neural events. We agree with the associationists that the higher mental processes can be performed by mechanisms—indeed, we have exhibited a specific set of mechanisms capable of performing some of them.

We have avoided, however, specifying these mechanisms in neurological or pseudo-neurological terms. Problem solving—at the information-processing level at which we have described it—has nothing specifically "neural" about it, but can be performed by a wide class of mechanisms, including both human brains and digital computers. We do not believe that this functional equivalence between brains and computers implies any structural equivalence at a more minute anatomical level (e.g., equivalence of neurons with circuits). Discovering what neural mechanisms realize these information-processing functions in the human brain is a task for another level of theory construction. Our theory is a theory of the information processes involved in problem solving, and not a theory of neural or electronic mechanisms for information processing.

The picture of the central nervous system to which our theory leads is a picture of a more complex and active system than that contemplated by most associationists. The notions of "trace," "fixation," "excitation," and "inhibition" suggest a relatively passive electrochemical system (or, alternatively, a passive "switchboard"), acted upon by stimuli, altered by that action, and subsequently behaving in a modified manner when later stimuli impinge on it.

In contrast, we postulate an information-processing system with large storage capacity that holds, among other things, complex strategies (programs) that may be evoked by stimuli. The stimulus determines what strategy or strategies will be evoked; the content of these strategies is already largely determined by the previous experience of the system. The ability of the system to respond in complex and highly selective ways to relatively simple stimuli is a consequence of this storage of programs and this "active" response to stimuli. The phenomena of set and insight that we have already described and the hierarchical structure of the response system are all consequences of this "active" organization of the central processes.

The historical preference of behaviorists for a theory of the brain that pictured it as a passive photographic plate or switchboard, rather than as an active computer, is no doubt connected with the struggle against vitalism. The invention of the digital computer has acquainted the world with a device—obviously a mechanism—whose response to stimuli is clearly more complex and "active" than the response of more traditional switching networks. It has provided us with operational and unobjectionable interpretations of terms like "purpose," "set," and "insight." The real importance of the digital computer for the theory of higher mental processes lies not merely in allowing us to realize such processes "in the metal" and outside the brain, but in providing us with a much profounder idea than we have hitherto had of the characteristics a mechanism must possess if it is to carry out complex information-processing tasks.

Gestalt Theories

The theory we have presented resembles the associationist theories largely in its acceptance of the premise of mechanism, and in few other respects. It resembles much more closely some of the Gestalt theories of problem solving, and perhaps most closely the theories of "directed thinking" of Selz and de Groot. A brief overview of Selz's conceptions of problem solving, as expounded by de Groot, will make its relation to our theory clear.

1. Selz and his followers describe problem solving in terms of processes or "operations" (2, p. 42). These are clearly the counterparts of the basic processes in terms of which LT is specified.

2. These operations are organized in a strategy, in which the outcome of each step determines the next (2, p. 44). The strategy is the counterpart of the program of LT.

3. A problem takes the form of a "schematic anticipation." That is, it is posed in some such form as: Find an X that stands in the specified relation R to the given element E (2, pp. 44–46). The counterpart of this in LT is the problem: Find a *sequence of sentences* (X) that stands in the relation of *proof* (R) to the given *problem expression* (E). Similarly, the subproblems posed by LT can be described in terms of schematic anticipations: for example, "Find an expression that is 'similar' to the expression to be proved." Many other examples can be supplied of "schematic anticipations" in LT.

4. The method that is applied toward solving the problem is fully specified by the schematic anticipation. The counterpart in LT is that, upon receipt of the problem, the executive program for solving logic problems specifies the next processing step. Similarly, when a subproblem is posed—like "prove the theorem by substitution"—the response to this subproblem is the initiation of a corresponding program (here, the method of substitution).

5. Problem solving is said to involve (*a*) finding means of solution, and (*b*) applying them (2, pp. 47–53). A counterpart in LT is the division between the similarity routines, which find "likely" materials for a proof, and the matching routines, which try to use these materials. In applying means, there are needed both *ordering* processes (to assign priorities when more than one method is available) and *control* processes (to evaluate the application) (2, p. 50).

6. Long sequences of solution methods are coupled together. This coupling may be *cumulative* (the following step builds on the result of the preceding) or *subsidiary* (the previous step was unsuccessful, and a new attempt is now made) (2, p. 51). In LT the former is illustrated by a successful similarity comparison followed by an attempt at matching; the latter by the failure of the method of substitution, which is then followed by an attempt at detachment.

7. In cumulative coupling, we can distinguish *complementary* methods from *subordinated* methods (2, p. 52). The former are illustrated by successive substitutions and replacements in successive elements of a pair of logic expressions. The latter are illustrated by the role of matching as a subordinate process in the detachment method.

We could continue this list a good deal further. Our purpose is not to suggest that the theory of LT can or should be translated into the language of "directed thinking." On the contrary, the specification of the program for LT clarifies to a considerable extent notions whose meanings are only vague in the earlier literature. What the list illustrates is that the processes that we observe in LT are basically the same as the processes that have been observed in human problem solving in other contexts.

PERFORMANCE AND LEARNING

LT is primarily a performance machine. That is to say, it solves problems rather than learning how to solve problems. However, although LT does not learn in all the ways that a human problem solver learns, there are a number of important learning processes in the program of LT. These serve to illustrate some, but not all, of the forms of human learning.

Learning in LT

By learning, we mean any more or less lasting change in the response of the system to successive presentations of the same stimulus. By this definition—which is the customary one—LT does learn.

1. When LT has proved a theorem, it stores this theorem in its memory. Henceforth, the theorem is available as material for the proof of subsequent theorems. Therefore, whether LT is able to prove a particular theorem depends, in general, on what theorems it has previously been asked to prove.

2. LT remembers, during the course of its attempt to prove a theorem, what subproblems it has already tried to solve. If the same subproblem is obtained twice in the course of the attempt at a proof, LT will remember and will not try to solve it a second time if it has failed a first.

3. In one variant, LT remembers what theorems have proved useful in the past in conjunction with particular methods and tries these theorems first when applying the method in question. Hence, although its total repertory of methods remains constant, it learns to apply particular methods in particular ways.

These are types of learning that would certainly be found also in human problem solvers. There are other kinds of human learning that are not yet represented in LT. We have already mentioned one—acquiring new methods for attacking problems. Another is modifying the descriptions used in searches for similar theorems, to increase the efficiency of those searches. The latter learning process may also be regarded as a process for concept formation. We have under way a number of activities directed toward incorporating new forms of learning into LT, but we will postpone a more detailed discussion of these until we can report concrete results.

What is Learned

The several kinds of learning now found in LT begin to cast light on the pedagogical problems of "what is learned?" including the problems of transfer of training. For example, if LT simply stored proofs of theorems as it found these, it would be able to prove a theorem a second time very rapidly, but its learning would not transfer at all to new theorems. The storage of *theorems* has much broader transfer value than the storage of *proofs*, since, as already noted, the proved theorems may be used as stepping stones to the proofs of new theorems. There is no mystery here in the fact that the transferability of what is learned is dependent in a very sensitive way upon the form in which it is learned and remembered. We hope to draw out the implications, psychological and pedagogical, of this finding in our subsequent research on learning.

CONCLUSION

We should like, in conclusion, only to draw attention to the broader implications of this approach to the study of information-processing systems. The heart of the approach is describing the behavior of a system by a well specified program, defined in terms of elementary information processes. In this approach, a specific program plays the role that is played in classical systems of applied mathematics by a specific system of differential equations.

Once the program has been specified, we proceed exactly as we do with traditional mathematical systems. We attempt to deduce general properties of the system from the program (the equations); we compare the behavior predicted from the program (from the equations) with actual behavior observed in experimental or field settings; we modify the program (the equations) when modification is required to fit the facts.

The promise of this approach is several-fold. First, the digital computer provides us with a device capable of realizing programs, and hence, of actually determining what behavior is implied by a program under various environmental conditions. Second, a program is a very concrete specification of the processes, and permits us to see whether the processes we postulate are realizable, and whether they are sufficient to produce the phenomena. The vaguenesses that have plagued the theory of higher mental processes and other parts of psychology disappear when the phenomena are described as programs.

In the present paper we have illustrated this approach by beginning the construction of a thoroughly operational theory of human problem solving. There is every reason to believe that it will prove equally fruitful in application to the theories of learning, of perception, and of concept formation.

REFERENCES

1. BRUNER, J. S., GOODNOW, J., & AUSTIN, G. *A study of thinking*. New York: Wiley, 1956.
2. DE GROOT, A. *Het Denken van den Schaker*. Amsterdam: Noord-Hollandsche Uitgevers Maatschappij, 1946.
3. HILBERT, D., & ACKERMANN, W. *Principles of mathematical logic*. New York: Chelsea, 1950.
4. JOHNSON, D. M. *The psychology of thought and judgment*. New York: Harper, 1955.
5. MOORE, O. K., & ANDERSON, S. B. Search behavior in individual and group problem solving. *Amer. sociol. Rev.*, 1955, 19, 702–714.
6. NEWELL, A., & SHAW, J. C. Programming the logic theory machine. *Proceedings Western Joint Computer Conference* (Institute of Radio Engineers), 1957, 230–240.
7. NEWELL, A., SHAW, J. C., & SIMON, H. A. Empirical explorations with the logic theory machine. *Proceedings Western Joint Computer Conference* (Institute of Radio Engineers), 1957, 218–230.
8. NEWELL, A., & SIMON, H. A. The logic theory machine: A complex information processing system. *Transactions on information theory* (Institute of Radio Engineers), 1956, Vol. IT-2, No. 3, 61–79.
9. SIMON, H. A., & NEWELL, A. Models, their uses and limitations. In L. D. White (Ed.), *The state of the social sciences*. Chicago: Univer. Chicago Press, 1956. Pp. 66–83.
10. WHITEHEAD, A. N., & RUSSELL, B. *Principia mathematica*. Vol. I. (2nd ed.) Cambridge: Cambridge Univer. Press, 1925.

(Received July 10, 1957)