The Soar Papers

1969–1982

CHAPTER 1

Heuristic Programming: Ill-Structured Problems

A. Newell, Carnegie Mellon University

We observe that on occasion expressions in some language are put forward that purport to state "a problem." In response a method (or algorithm) is advanced that claims to solve the problem. That is, if input data are given that meet all the specifications of the problem statement, the method produces another expression in the language that is the solution to the problem. If there is a challenge as to whether the method actually provides a general solution to the problem (i.e., for all admissible inputs), a proof may be forthcoming that it does. If there is a challenge to whether the problem statement is well defined, additional formalization of the problem statement may occur. In the extreme this can reach back to formalization of the language used to state the problem, until a formal logical calculus is used.

We also observe that for other problems that people have and solve there seems to be no such formalized statement and formalized method. Although usually definite in some respects problems of this type seem incurably fuzzy in others. That there should be ill-defined problems around is not very surprising. That is, that there should be expressions that have some characteristics of problem statements but are fragmentary seems not surprising. However, that there should exist systems (i.e., men) that can solve these problems without the eventual intervention of formal statements and formal methods does pose an issue. Perhaps there are two domains of problems, the well structured and the ill structured. Formalization always implies the first. Men can deal with both kinds. By virtue of their capacity for working with ill-structured problems, they can transmute some of these into well-structured (or formalized) problems. But the study of formalized problems has nothing to say about the domain of ill-structured problems. In particular, there can never be a formalization of ill-structured problems, hence never a theory (in a strict sense) about them. All that is possible is the conversion of particular problems from ill structured to well structured via the one transducer that exists, namely, man.

Perhaps an analog is useful: well-structured problems are to ill-structured problems as linear systems are to nonlinear systems, or as

I wish to acknowledge the help of J. Moore in clarifying the nature of the methods of artificial intelligence and also my discussions with my colleague H. A. Simon. The research was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-146).

stable systems are to unstable systems, or as rational behavior is to non-rational behavior. In each case, it is not that the world has been neatly divided into two parts, each with a theory proper to it. Rather, one member of the pair is a very special case about which much can be said, whereas the other member consists of all the rest of the world—uncharted, lacking uniform approach, inchoate, and incoherent.

This is not the only view, of course. Alternatively, one can assert that all problems can be formulated in the same way. The formalization exists because there is some symbolic system, whether on paper or in the head, that holds the specific, quite definite information in the problem statement. The fragmentation of problem statement that occurs when an attempt is made to explicate the problem only shows that there are serious (perhaps even profound) communication problems. But it does not say that ill-structured problems are somehow different in nature.

Thus we have an issue—somewhat ill structured, to be sure—but still an issue. What are ill-structured problems and are they a breed apart from well-structured ones? This chapter is essentially an essay devoted to exploring the issue, as it stands in 1968.

We have an issue defined. What gives life to it are two concerns, one broad, one narrow. At root, there is the long-standing concern over the rationalization of human life and action. More sharply stated, this is the challenge of art by science. In terms of encounters long resolved, it is whether photography will displace painting, or whether biology and physiology can contribute to the practice of medicine. In terms of encounters now in twilight, it is whether new products come from applied science or from lone inventors. In terms of encounters still active, it is whether the holistic diagnostic judgment of the clinical psychologist is better than the judgments of a regression equation [12]. For the purpose of this essay, of course, it is to what extent management science can extend into the domain of business judgment.

When put in this context, the issue has a charge. The concern flows partly from economics, where it is now usually labeled the problem of automation. Concern also flows from a problem of identity, in which some are compelled to ask what attributes man can uniquely call his own. As has been pointed out, probably most thoroughly by Ellul [3], it makes no sense to separate hardware and software, that is, to separate machines from procedures, programs, and formalized rules. They are all expressions of the rationalization of life, in which human beings become simply the agents or carriers of a universalistic system of orderly relations of means to ends.

Thus, viewed broadly, the issue is emotionally toned. However, this fact neither eliminates nor affects the scientific questions involved, although it provides reasons for attending to them. Our aim in this essay is essentially scientific, a fact which leads to the second, more narrow context.

Within management science the nature of rationalization has varied somewhat over time. In the early days, those of Frederick Taylor, it was expressed in explicit work procedures, but since World War II it has been expressed in the development of mathematical models and quantitative methods. In 1958 we put it as follows:

> A problem is well structured to the extent that it satisfies the following criteria:
>
> 1. It can be described in terms of numerical variables, scalar and vector quantities.
> 2. The goals to be attained can be specified in terms of a well-defined objective function—for example, the maximization of profit or the minimization of cost.
> 3. There exist computational routines (*algorithms*) that permit the solution to be found and stated in actual numerical terms. Common examples of such algorithms, which have played an important role in operations research, are maximization procedures in the calculus and calculus of variations, linear-programming algorithms like the stepping-stone and simplex methods, Monte Carlo techniques, and so on [21, pp. 4–5].

Ill-structured problems were defined, as in the introduction of this essay, in the negative: all problems that are not well structured. Now the point of the 1958 paper, and the reason that it contrasted well- and ill-structured problems, was to introduce heuristic programming as relevant to the issue:

> With recent developments in our understanding of heuristic processes and their simulation by digital computers, the way is open to deal scientifically with ill-structured problems—to make the computer co-extensive with the human mind [21, p. 9].

That is, before the development of heuristic programming (more generally, artificial intelligence) the domain of ill-structured problems had been the exclusive preserve of human problem solvers. Now we had other systems that also could work on ill-structured problems and that could be studied and used.

This 1958 paper is a convenient marker for the narrow concern of the present essay. It can symbolize the radical transformation brought by the computer to the larger, almost philosophical concern over the nature and possibilities for rationalization. The issue has become almost technical, although now it involves three terms, where before it involved only two:

- What is the nature of problems solved by formal algorithms?
- What is the nature of problems solved by computers?
- What is the nature of problems solved by men?

We have called the first well-structured problems; the last remains the residual keeper of ill-structured problems; and the middle term offers the opportunity for clarification.

Our course will be to review the 1958 paper a little more carefully, leading to a discussion of the nature of problem solving. From this will emerge an hypothesis about the nature of generality in problem solving, which will generate a corresponding hypothesis about the nature of ill-structured problems. With theses in hand, we first consider some implications of the hypotheses, proceed to explore these a little, and finally bring out some deficiencies.

The 1958 paper asserted that computers (more precisely, computers appropriately programmed) could deal with ill-structured problems, where the latter was defined negatively. The basis of this assertion was twofold. First, there had just come into existence the first successful heuristic programs, that is to say, programs that performed tasks requiring intelligence when performed by human beings. They included a theorem-proving program in logic [15], a checker-playing program [19], and a pattern recognition program [20]. These were tasks for which algorithms either did not exist or were so immensely expensive as to preclude their use. Thus, there existed some instances of programs successfully solving interesting ill-structured problems. The second basis was the connection between these programs and the nature of human problem solving [16]. Insofar as these programs reflected the same problem-solving processes as human beings used, there was additional reason to believe that the programs dealt with ill-structured problems. The data base for the assertion was fairly small, but there followed in the next few years additional heuristic programs that provided support. There was one that proved theorems in plane geometry, one that did symbolic indefinite integration, a couple of chess programs, a program for balancing assembly lines, and several pattern recognition programs [5].

The 1958 paper provided no positive characterization of ill-structured problems. Although it could be said that some ill-structured problems were being handled, these might constitute a small and particularly "well-formed" subset. This was essentially the position taken by Reitman, in one of the few existing direct contributions to the question of ill-formed problems [17, 18]. He observed, as have others, that all of the heuristic programs, although lacking well-specified algorithms, were otherwise quite

precisely defined. In particular, the test whereby one determined whether the problem was solved was well specified, as was the initial data base from which the problem started. Thus, he asserted that all existing heuristic programs were in a special class by virtue of certain aspects being well defined, and thus shed little light on the more general case.

Stating this another way, it is not enough for a problem to become ill structured only with respect to the methods of solution. It is required also to become ill structured with respect to both the initial data and the criteria for solution. To the complaint that one would not then really know what the problem was, the rejoinder is that almost all problems dealt with by human beings are ill structured in these respects. To use an example discussed by Reitman, in the problem of making a silk purse from a sow's ear, neither "silk purse" nor "sow's ear" is defined beyond cavil. To attempt really to solve such a problem, for instance, would be to search for some ways to stretch the implicit definitions to force acceptance of the criteria, for example, chemical decomposition and resynthesis.

Reitman attempted a positive characterization of problems by setting out the possible forms of uncertainty in the specification of a problem: the ways in which the givens, the sought-for transformation, or the goal could be ill defined. This course has the virtue, if successful, of defining "ill structured" independently of problem solving and thus providing a firm base on which to consider how such problems might be tackled. I will not follow him in this approach, however. It seems more fruitful here to start with the activity of problem solving.

1. THE NATURE OF PROBLEM SOLVING

A rather general diagram, shown in Fig. 10.1, will serve to convey a view of problem solving that captures a good deal of what is known, both casually and scientifically. A problem solver exists in a task environment, some small part of which is the immediate stimulus for evoking the problem and which thus serves as the initial problem statement.[1] This external representation is translated into some internal representation (a condition, if you please, for assimilation and acceptance of the problem by the problem solver). There is located within the memory of the problem solver a collection of methods. A method is some organ-

[1] Its statement form is clear when given linguistically, as in "Where do we locate the new warehouse?" Otherwise, "statement" is to be taken metaphorically as comprising those clues in the environment attended to by the problem solver that indicate to him the existence of the problem.

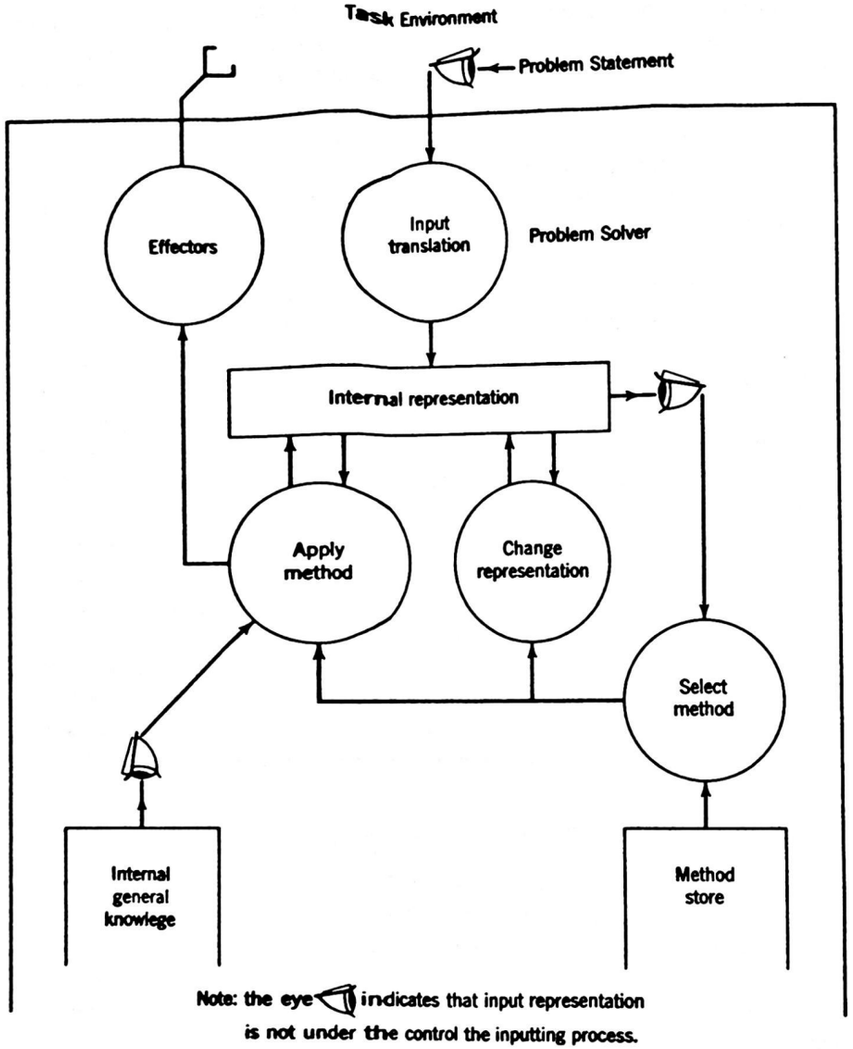


Figure 10.1. General schema of a problem solver.

ized program or plan for behavior that manipulates the internal representation in an attempt to solve the problem. For the type of problem solvers we have in mind—business men, analysts, etc.—there exist many relatively independent methods, so that the total behavior of the problem

solver is made up as an iterative cycle in which methods are selected on the basis of current information (in the internal representation) and tried with consequent modification of the internal representation, and a new method is selected.

Let us stay with this view of problem solving for a while, even though it de-emphasizes some important aspects, such as the initial determination of an internal representation, its possible change, and the search for or construction of new methods (by other methods) in the course of problem solving. What Fig. 10.1 does emphasize is the method—the discrete package of information that guides behavior in an attempt at problem solution. It prompts us to inquire about its anatomy.

1.1. The Anatomy of a Method

Let us examine some method in management science. The simplex method of linear programming will serve admirably. It is well known, important, and undoubtedly a method. It also works on well-structured problems, but we will take due account of this later. The basic linear programming problem is as follows.

Given: a set of positive real variables

$$x_j \geqslant 0, \qquad j = 1, \ldots, n$$

and real constants

$$a_{ij}, b_i, c_j, \qquad i = 1, \ldots, m; j = 1, \ldots, n$$

maximize

$$z = \sum_j c_j x_j$$

subject to

$$\sum_j a_{ij} x_j \leqslant b_i, \qquad i = 1, \ldots, m$$

Figure 10.2 shows the standard data organization used in the simplex method and gives the procedure to be followed. We have left out the procedure for getting an initial solution, that is, we assume that the tableau of Fig. 10.2 is initially filled in. Likewise, we have ignored the detection of degeneracy and the procedure for handling it.

There are three parts to the simplex method. The first is the *statement of the problem.* This determines the entities you must be able to identify in a situation and what you must know about their properties and mutual interrelationships. Unless you can find a set of nonnegative numerical quantities, subject to linear constraints and having a linear objective function to be maximized, you do not have a LP problem, and

the method cannot help you. The second part of the method is the *procedure* that delivers the solution to the problem. It makes use only of information available in the problem statement. Indeed, these two are coupled together as night and day. Any information in the problem statement which is known not to enter into the procedure can be discarded, thereby making the problem just that much more general.

Simplex tableau

Basis	b_i	c_1 x_1	c_2 x_2	...	c_n x_n	0 x_{n+1}	...	0 x_{n+m}
x_{j_1}								
x_{j_2}								
.				t_{ij}				
.								
x_{j_m}								
z_j								
$z_j - c_j$								

Procedure

1. Let j_0 = column with min $\{z_j - c_j | z_j - c_j < 0\}$; if no $z_j - c_j < 0$, then at maximum z, end.
2. Let i_0 = row with min $\{b_i/t_{ij_0} | b_i/t_{ij_0} > 0\}$; if no $b_i/t_{ij_0} > 0$, then z unbounded, end.
3. For row i_0, $t_{i_0 j} \leftarrow t_{i_0 j}/t_{i_0 j_0}$.
4. For row $i \neq i_0$, $t_{ij} \leftarrow t_{ij} - t_{i_0 j} t_{ij_0}$ ($t_{i_0 j}$ is the value from step 3).

Define

$$x_{n+i} = b_i - \sum_j a_{ij} x_j, \qquad i = 1, \ldots, m$$

$$c_{n+i} = 0$$

$$a_{i,n+k} = 1 \text{ if } i = k, 0 \text{ otherwise}$$

Figure 10.2. Simplex method.

The third part of the method is the proof or *justification* that the procedure in fact delivers the solution to the problem (or delivers it within certain specified limits). The existence of this justification has several consequences. One, already noted, is the complete adaptation of means to ends—of the shaping of the problem statement so that it is as general as possible with respect to the procedure. Another consequence is a toleration of apparent meaninglessness in the procedure. It makes no difference that there seems to be neither rhyme nor reason to the steps of the method in Fig. 10.2. Careful analysis reveals that they are in fact just those steps necessary to the attainment of the solution. This feature is characteristic of mathematical and computational methods generally and sometimes is even viewed as a hallmark.

An additional part of the simplex method is a *rationale* that can be used to make the method understandable. The one usually used for the simplex is geometrical, with each constraint being a (potential) boundary plane of the space of feasible solutions. Then the simplex procedure is akin to climbing from vertex to vertex until the maximal one is reached. This fourth part is less essential than the other three.

The first three parts seem to be characteristic of all methods. Certainly, examples can be multiplied endlessly. The quadratic formula provides another clear one:

Problem statement:	Find x such that $ax^2 + bx + c = 0$.
Procedure:	compute $x = b/2a \pm \frac{1}{2}a\sqrt{b^2 - 4ac}$.
Justification:	(substitute formula in $ax^2 + bx + c$ and show by algebraic manipulation that 0 results).

In each case a justification is required (and forthcoming) that establishes the relation of method to problem statement. As we move toward more empirical methods, the precision of both the problem statement and the procedure declines, and concurrently the precision of the justification; in fact, justification and plausible rationale merge into one.

1.2. Generality and Power

We need to distinguish the generality of a method from its power. A method lays claim via its problem statement to being applicable to a certain set of problems, namely, to all those for which the problem statement applies. The generality of a method is determined by how large the set of problems is. Even without a well-defined domain of all problems, or any suitable measure on the sets of problems, it is still often possible to compare two problem statements and judge one to be more inclusive than another, hence one method more general than the other.

A method that is applicable only to locating warehouses is less general than one that is applicable to problems involving the location of all physical resources. But nothing interesting can be said about the relative generality of a specific method for inventory decisions versus one for production scheduling.

Within the claimed domain of a method we can inquire after its ability to deliver solutions: the higher this is, the more powerful the method. At least three somewhat independent dimensions exist along which this ability can be measured. First, the method may or may not solve every problem in the domain; and we may loosely summarize this by talking of the probability of solution. Second, there may exist a dimension of quality in the solution, such as how close an optimizing method gets to the peak. Then methods can differ on the quality of their solutions. (To obtain a simple characteristic for this requires some summarization over the applicable domain, but this feature need not concern us here.) Third, the method may be able to use varying amounts of resources. Then, judgments of probability of solution and of quality are relative to the amount of resources used. Usually the resource will be time, but it can also be amount of computation, amount of memory space, number of dollars to acquire information, and so on. For example, most iterative methods for solving systems of equations do not terminate in a finite number of iterations, but produce better solutions if run longer; the rate of convergence becomes a significant aspect of the power of such methods.

In these terms the simplex method would seem to rank as one of limited generality but high power. The restrictions to linearity, both in the constraints and the objective function, and to a situation describable by a set of real numbers, all constrict generality. But the simplex method is an algorithm within its domain and guarantees delivery of the complete solution. It is not the least general method, as is indicated by the transportation problem with its more specialized assumptions; nor is it the most powerful method for its domain, since it can be augmented with additional schemes that obtain solutions more expeditiously.

Evidently there is an inverse relationship between the generality of a method and its power. Each added condition in the problem statement is one more item that can be exploited in finding the solution, hence in increasing the power. If one takes a method, such as the simplex method, and generalizes the problem statement, the procedure no longer solves every problem in the wider domain, but only a subset of these. Thus the power diminishes. The relationship is not one-one, but more like a limiting relationship in which the amount of information in the problem statement puts bounds on how powerful the method can be. This relationship is important enough to the argument of this essay that we indicate it symbolically in Fig. 10.3. The abcissa represents increasing information in the problem statement, that is, decreasing generality. The ordinate represents increasing power. For each degree of generality an upper bound exists on the possible power of a method, though there are clearly numerous methods which do not fully exploit the information in the problem statement.

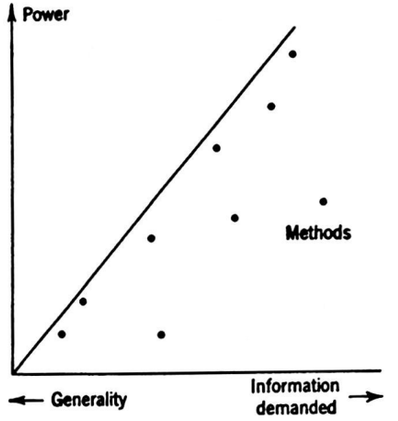


Figure 10.3. Generality versus power.

2. TWO HYPOTHESES: ON GENERALITY AND ON ILL-STRUCTURED PROBLEMS

With this view of method and problem solver we can move back toward the nature of ill-structured problems. However, we need to address one intermediate issue: the nature of a general problem solver. The first heuristic programs that were built laid claims to power, not to generality. A chess or checker program was an example of artificial intelligence because it solved a problem difficult by human standards; there was never a pretense of its being general. Today's chess programs cannot even play checkers, and vice versa.

Now this narrowness is completely consistent with our general experience with computer programs as highly special methods for restricted tasks. Consider a typical subroutine library, with its specific routines for inverting matrices, computing the sine, carrying out the simplex method, and so on. The only general "programs" are the higher-level

programming languages, and these are not problem solvers in the usual sense, but only provide means to express particular methods.[2] Thus the view has arisen that, although it may be possible to construct an artificial intelligence for any highly specific task domain, it will not prove possible to provide a general intelligence. In other words, it is the ability to be a general problem solver that marks the dividing line between human intelligence and machine intelligence.

The formulation of method and problem solver given earlier leads rather directly to a simple hypothesis about this question:

Generality Hypothesis. A general problem solver is one that has a collection of successively weaker methods that demand successively less of the environment in order to be applied. Thus a good general problem solver is simply one that has the best of the weak methods.

This hypothesis, although itself general, is not without content. (To put it the way that philosophers of science prefer, it is falsifiable.) It says that there are no special mechanisms of generality—nothing beyond the willingness to carry around specific methods that make very weak demands on the environment for information. By the relationship expressed in Fig. 10.3 magic is unlikely, so that these methods of weak demands will also be methods of low power. Having a few of them down at the very low tip in the figure gives the problem solver the ability to tackle almost any kind of problem, even if only with marginal success.

There are some ways in which this generality hypothesis is almost surely incorrect or at least incomplete, and we will come to these later; but let us remain with the main argument. There is at least one close association between generality and ill-structured problems: it is man that can cope with both. It is also true that ill-structured problems, whatever else may be the case, do not lend themselves to sharp solutions. Indeed, their lack of specificity would seem to be instrumental in prohibiting the use of precisely defined methods. Since every problem does present some array of available information—something that could meet the conditions of a problem statement of some method—the suspicion arises that lack of structure in a problem is simply another indication that there are not methods of high power for the particular array of information available. Clearly this situation does not prevail absolutely, but only with respect to a given problem solver and his collection of methods (or, equally, a population of highly similar problem solvers). We can phrase this suspicion in sharper form:

[2] The relationship of programming languages to problem solvers, especially as the languages become more problem-oriented, is unexplored territory. Although relevant to the main question of this essay, it cannot be investigated further here.

Ill-structured Problem Hypothesis. A problem solver finds a problem ill structured if the power of his methods that are applicable to the problem lies below a certain threshold.

The lack of any uniform measure of power, with the consequent lack of precision about a threshold on this power, is not of real concern: the notion of ill-structuredness is similarly vague. The hypothesis says that the problem of locating a new warehouse will look well structured to a firm that has, either by experience, analysis, or purchase, acquired a programmed procedure for locating warehouses, providing it has decided that the probability of obtaining an answer of suitable quality is high enough simply to evoke the program in the face of the new location problem. The problem will look ill structured to a firm that has only its general problem-solving abilities to fall back on. It can only have the most general faith that these procedures will discover appropriate information and use it in appropriate ways in making the decision.

My intent is not to argue either of these two hypotheses directly, but rather to examine some of their implications. First, the weak methods must be describable in more concrete terms. This we will do in some detail, since it has been the gradual evolution of such methods in artificial intelligence that suggested the hypotheses in the first place. Second, the picture of Fig. 10.3 suggests not only that there are weak methods and strong ones, but that there is continuity between them in some sense. Phrased another way, at some level the methods of artificial intelligence and those of operations research should look like members of the same family. We will also look at this implication, although somewhat more sketchily, since little work has been done in this direction. Third, we can revisit human decision makers in ill-structured situations. This we do in an even more sketchy manner, since the main thrust of this essay stems from the more formal concerns. Finally, after these (essentially positive) explications of the hypotheses, we will turn to discussion of some difficulties.

3. THE METHODS OF HEURISTIC PROGRAMMING

There has been continuous work in artificial intelligence ever since the article quoted at the beginning of this chapter [21] took note of the initial efforts. The field has had two main branches. We will concentrate on the one called heuristic programming. It is most closely identified with the programmed digital computer and with problem solving. Also, almost all the artificial intelligence efforts that touch management science are included within it. The other branch, identified with pattern

recognition, self-organizing systems, and learning systems, although not exempt from the observations to be made here, is sufficiently different to preclude its treatment.

A rather substantial number of heuristic programs have been constructed or designed and have gone far enough to get into the literature. They cover a wide range of tasks: game playing, mostly chess, checkers, and bridge; theorem proving, mostly logic, synthetic geometry, and various elementary algebraic systems; all kinds of puzzles; a range of management science tasks, including line balancing, production scheduling, and warehouse location; question-answering systems that accept quasi-English of various degrees of sophistication; and induction problems of the kind that appear on intelligence tests. The main line of progress has constituted a meandering tour through new task areas which seemed to demand new analyses. For example, there is considerable current work on coordinated effector-receptor activity (e.g., hand-eye) in the real world—a domain of problems requiring intelligence that has not been touched until this time.

Examination of this collection of programs reveals that only a few ideas seem to be involved, despite the diversity of tasks. These ideas, if properly expressed, can become a collection of methods in the sense used earlier. Examination of these methods shows them to be extraordinarily weak compared with the methods, say, of linear programming. In compensation, they have a generality that lets them be applied to tasks such as discovering proofs of theorems, where strong methods are unknown.[3]

It thus appears that the work in heuristic programming may provide a first formalization of the kind of weak methods called for by our two hypotheses. (To be sure, as already noted, psychological invention runs the other way: the discovery that there seems to be a small set of methods underlying the diversity of heuristic programs suggested the two hypotheses.)

It might be claimed that the small set of methods shows, not parsimony, but the primitive state of development of the field and that investigators read each other's papers. Although there is clearly some force to this argument, to an important degree each new task attempted in heuristic programming represents an encounter with an unknown set of

[3] Strong methods of proof discovery do get developed in the course of mathematical progress, but their effect is to reduce whole areas to a calculus. The development of the operational calculus—later the Laplace and Fourier transforms—is a case in point. But the present theorem-proving programs and the methods that lie behind them do not involve mathematical advances; rather they appear to capture methods available for proof discovery within the existing state of ignorance.

demands that have to be met on their own terms. Certainly the people who have created heuristic programs have often felt this way. In fact, the complaint is more often the opposite to the above caveat—that artificial intelligence is a field full of isolated cases with no underlying coherency.

In fact, the view expressed in this chapter is not widely held. There is some agreement that all heuristic theorem provers and game players make use of a single scheme, called heuristic search. But there is little acknowledgment that the remainder of the methods listed below constitute some kind of basic set.

With this prelude, let us describe briefly some methods. An adequate job cannot be done in a single chapter; it is more an undertaking for a textbook. Hopefully, however, some feeling for the essential characteristics of generality and power can be obtained from what is given. The first three, generate-and-test, match, and hill climbing, rarely occur as complete methods in themselves (although they can), but are rather the building blocks out of which more complex methods are composed.

3.1. *Generate-and-Test*

This is the weak method par excellence. All that must be given is a way to generate possible candidates for solution plus a way to test whether they are indeed solutions. Figure 10.4 provides a picture of generate-and-test that permits us to view it as a method with a problem statement and a procedure. The flow diagram in the figure adopts some conventions that will be used throughout. They allow expresson of the central idea of a method without unnecessary detail. The lines in the diagram show the flow of data, rather than the flow of control—more in the style of an analog computer flow diagram than a digital computer flow diagram. Thus the nodes represent processes that receive inputs and deliver outputs. If a node is an item of data, as in the predicate P or the set $\{x\}$ (braces are used to indicate sets), it is a memory process that makes the data item available. A process executes (or fires) when it receives an input; if there are several inputs, it waits until all appropriate ones have arrived before firing.

A generator is a process that takes information specifying a set and produces elements of that set one by one. It should be viewed as autonomously "pushing" elements through the system. Hence there is a flow of elements from generate to the process called test. Test is a process that determines whether some condition or predicate is true of its input and behaves differentially as a result. Two different outputs are possible: satisfied (+) and unsatisfied (−). The exact output behavior depends

Problem statement

Given: a generator of the set $\{x\}$;
a test of the predicate P defined on elements of $\{x\}$;
Find: an element of $\{x\}$ that satisfies $P(x)$.

Procedure

$$\{x\} \dashrightarrow \text{generate} \xrightarrow{\text{element input}} \text{test}\ (\overset{P}{\downarrow}) \xrightarrow{+\ \text{solution}}$$

Justification

To show y is a solution if and only if $y \in \{x\}$ and $P(y)$.
Notation: $\pi \rightarrow \alpha$ means that process π produces α;
α/β means that α is on line labeled β.
Test has associated with it a predicate P on one variable, such that:

test $\rightarrow \alpha/+$ if and only if $P(\alpha)$ and α/input;
test $\rightarrow \alpha/-$ if and only if $\neg P(\alpha)$ and α/input.

Generate has associated with it a set $\{x\}$ such that:
generate $\rightarrow \alpha$/element only if $\alpha \in \{x\}$;
$\alpha \in \{x\}$ implies there exists a time when generate $\rightarrow \alpha$/element.
Working backward from the flow line labeled solution, we get:
1. y/solution if and only if test $\rightarrow y/+$.
2. test $\rightarrow y/+$ if and only if $P(y)$ and y/input.

Now we need only show that y/input if and only if $y \in \{x\}$.
3. y/input if and only if generate $\rightarrow y$/element.
4. generate $\rightarrow y$/element only if $y \in \{x\}$.

Now we need only show that $y \in \{x\}$ implies generate $\rightarrow y$/element; however, the best we can do is:
5. $y \in \{x\}$ implies there exists a time when generate $\rightarrow y$/element.

Figure 10.4. Generate-and-test method.

on the needs of the rest of the processing. The input can be passed through on one condition and nothing done on the other, in which case test acts as a filter or gate. The input can be passed through in both cases, but on different output lines, in which case test acts as a binary switch.

The set associated with a generator and the predicate associated with a test are not inputs. Rather, they are constructs in terms of which the behavior of the process can be described. This is done in Fig. 10.4 by listing a set of propositions for each process. The single arrow (→) indicates production of an output, and the slash (/) indicates that a data item is on the line with the given label. Thus the first proposition under test says that test produces an item on its + output line if and only if that item was input and also satisfies the associated predicate. For any particular generator the associated set must be fully specified, but clearly that specification can be shared in particular ways between the structure of the generator and some actual inputs; for example, a generator could take an integer as input and produce the integers greater than the input, or it could have no input at all and simply generate the positive integers. The same situation holds for test or any other process: its associated constructs must be fully specified, but that specification can be shared in different ways between the structure of the process and some of its inputs. Sometimes we will put the associated construct on the flow diagram, as we have in Fig. 10.4, to show the connection between the processes in the flow diagram and the constructs used in the statement of the problem. We use dotted lines to show that these are not really inputs, although inputs could exist that partially specify them.

We have provided a sketch of a justification that the procedure of the method actually solves the problem. In substance the proof is trivial. To carry it through in detail requires formalization of both the procedure and the language for giving the problem statements and the properties known to hold if a process is executed [6]. The handling of time is a bit fussy and requires more formal apparatus than is worthwhile to present here. Note, for instance, that if the generator were not allowed to go to conclusion, generate-and-test would not necessarily produce a solution. Similar issues arise with infinite sets. Justifications will not be presented for the other methods. The purpose of doing it for this (simplest) one is to show that all the components of a method—problem statement, procedure, and justification—exist for these methods of artificial intelligence. However, no separate rationale is needed for generate-and-test, partly because of its simplicity and partly because of the use of a highly descriptive procedural language. If we had used a machine code, for instance, we might have drawn the procedure of Fig. 10.4 as an informal picture of what was going on.

Generate-and-test is used as a complete method, for instance, in opening a combination lock (when in desperation). Its low power is demonstrated by the assertion that a file with a combination lock is a "safe." Still, the method will serve to open the safe eventually. Generate-and-test is often used by human beings as a second method for finding lost items, such as a sock or a tiepin. The first method relies on recollections about where the item was left or last seen. After this has failed,

generate-and-test is evoked, generating the physical locations in the room one by one, and looking in each.

The poor record of generate-and-test as a complete method should not blind one to its ubiquitous use when other information is absent. It is used to scan the want ads for neighborhood rentals after the proper column is discovered (to the retort "What else?", the answer is, "Right! That's why the method is so general"). In problem-solving programs it is used to go down lists of theorems or of subproblems. It serves to detect squares of interest on chessboards, words of interest in expressions, and figures of interest in geometrical displays.

3.2. *Match*

We are given the following expression in symbolic logic:

$$e\text{: } (p \vee q) \supset ((p \vee q) \vee (r \supset p))$$

A variety of problems arise from asking whether e is a member of various specified sets of logic expressions. Such problems can usually be thrown into the form of a generate-and-test, at which point the difficulty of finding the solution is directly proportional to the size of the set.

If we know more about the structure of the set, better methods are available. For instance, consider the following two definitions of sets:

S_1: $x \supset (x \vee y)$, where x and y are any logic expressions.

Examples: $p \supset (p \vee q)$, $q \supset (q \vee q)$, $(p \vee p) \supset ((p \vee p) \vee p)$, ...

S_2: α, where α may be replaced (independently at each occurrence) according to the following schemes:

$\alpha \leftarrow q$, $\alpha \leftarrow (p \vee \alpha)$, $\alpha \leftarrow \alpha \supset \alpha$.

Examples: q, $p \vee q$, $q \supset q$, $p \vee (p \vee q)$, $(p \vee q) \supset (p \vee q)$, ...

In S_1, x and y are variables in the standard fashion, where each occurrence of the variable is to be replaced by its value. In S_2 we have defined a replacement system, where each separate occurrence of the symbol α may be replaced by any of the given expressions. These may include α, hence lead to further replacements. A legal logic expression exists only when no α's occur.

It is trivial to determine that e is a member of the set of expressions defined by S_1, and not so trivial to determine that it is not a member of the set defined by S_2. The difference is that for S_1 we could simply match the expressions against the form and determine directly the values of the variables required to do the job. In the case of S_2 we had essentially to generate-and-test. (Actually, the structure of the replacement system

permits the generation to be shaped somewhat to the needs of the task, so it is not pure generate-and-test, which assumes no knowledge of the internal structure of the generator.)

Figure 10.5 shows the structure of the match method, using the same symbolism as in Fig. 10.4 for the generate-and-test. A key assumption, implicit in calling X and F expressions, is that it is possible to generate the subparts of X and F, and that X and F are equal if and only if corresponding subparts are equal. Thus there are two generators, which produce corresponding subparts of the two expressions as elements. These are compared: if equal, the generation continues; if not equal, a test is made if the element from the form is a variable. If it is, a substitution of the corresponding part of X for the variable is possible, thus making the two expressions identical at that point, and permitting generation to continue. The generators must also produce some special end signal, whose co-occurrence is detected by the compare routine to determine that a solution has been found.

The match procedure sketched in Fig. 10.5 is not the most general one possible. Operations other than substitution can be used to modify the form (more generally, the kernel structure) so that it is equal to X. There can be several such operations with the type of difference between the two elements selecting out the appropriate action. This action can

Problem statement

Given: expressions made up of parts from a set S;
a set of variables $\{v\}$ with values in S;
a form F, which is an expression containing variables;
an expression X.
Find: if X is in the set defined by F; that is,
Find: values for $\{v\}$ such that $X = F$ (with values substituted).

Procedure

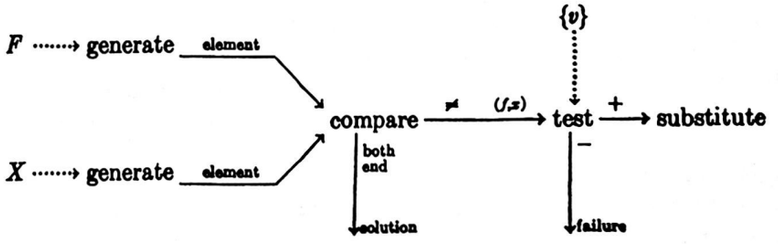


Figure 10.5. Match method.

result in modification of X as well as F. It is possible to write a single procedure that expresses these more general possibilities, but the detail does not warrant it. The essential point is that generation occurs on the parts of the expressions, and when parts fail to correspond it is possible to make a local decision on what modifying operation is necessary (though perhaps not sufficient) for the two expressions to become equal.

Matching is used so pervasively in mathematical manipulation, from algebraic forms to the conditions of a theorem, that our mathematical sophistication leads us not to notice how powerful it is. Whenever a set of possible solutions can be packaged as a form with variables, the search for a solution is no longer proportional to the size of the set of all possible solutions, but only to the size of the form itself. Notice that the generate process in generate-and-test (Fig. 10.4) operates on quite a different set from the generate of the match (Fig. 10.5).

Beside the obvious uses in proving theorems and doing other mathematics, matching shows up in tasks that seem remote from this discipline. One of them, as shown below, is inducing a pattern from a part. Another use is in answering questions in quasi-natural language. In the latter, information is extracted from the raw text by means of forms, with the variables taking subexpressions in the language as values.

3.3. *Hill Climbing*

The most elementary procedure for finding an optimum is akin to generate-and-test, with the addition that the candidate element is compared against a stored element—the best so far—and replaces it if higher. The element often involves other information in addition to the position in the space being searched, for example, a function value. With just a little stronger assumptions in the problem statement, the problem can be converted into an analog of climbing a hill. There must be available a set of operators that find new elements on the hill, given an existing element. That is, new candidate elements are generated by taking a step from the present position (one is tempted to say a "nearby" step, but it is the operators themselves that define the concept of nearness). Thus the highest element so far plays a dual role, both as the base for generation of new elements and as the criterion for whether they should be kept.

Figure 10.6 provides the capsule formulation of hill climbing. Generation is over the set of operators, which are then applied to the best x so far, until a better one is found. This method differs from the various forms of steepest ascent in not finding the best step from the current position before making the next step.

Problem statement

Given: a comparison of two elements of a set $\{x\}$ to determine which is greater;
a set of operators $\{q\}$ whose range and domain is $\{x\}$
[i.e., $q(x) = x'$, another element of $\{x\}$].

Find: the greatest $x \in \{x\}$.

Procedure

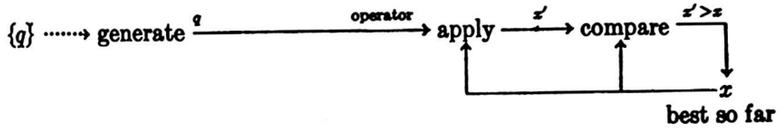


Figure 10.6. Hill climbing.

A great deal has been written about hill climbing, and the interested reader will find a thorough discussion within the context of more elaborate methods for finding optima in Chapter 9 on structured heuristic programming. Here we note only the familiar fact that the method does not guarantee to find the best element in the space. An additional condition, unimodality, is required; otherwise the procedure may end up on a local peak, which is lower than the highest peak. Actually, unimodality is not easy to define in the most general situation to which hill climbing applies, since the underlying space (which is "seen" only through the operators) need not have neighborhoods in the sense required to define a local peak.

Hill climbing shows up in a subordinate way in many heuristic programs, especially in the adaptive setting of parametric values. For example, in one program that attempted to construct programs satisfying certain criteria [9], the various basic instructions were selected at random and used to extend the program built so far. The entire program was an elementary form of heuristic search, discussed in Section 3.4. But superimposed on it was a hill-climbing program that gradually modified the probabilities of selecting the basic instructions so as to maximize the yield of programs over the long haul. The operators randomly jiggled the selection probabilities around (always maintaining their sum equal to one). The comparison was made on a statistical basis after observing the performance with the new probabilities for, say, 100 randomly selected problems.

Management science is much concerned with seeking optima, although, as mentioned above, the methods used are more elaborate. This can be

illustrated by a heuristic program developed by Kuehn and Hamburger [11] for locating warehouses so as to balance the costs of distribution (which decrease with more warehouses) and the costs of operation (which increase with more warehouses). The program consists of three separate optimizers: a cascade of a generate-and-test optimizer and a steepest ascent optimizer, followed by a simple hill climber, followed by a set of simple generate-and-test optimizers. Figure 10.7 gives the problem statement (leaving out details on the cost functions) and an indication of how the problem is mapped into the problem space[4] for optimization. Three operators are defined, corresponding to the three separate stages already mentioned. The procedure is given for the first stage (called the Main Routine), but not for the other two (called the Bump and Shift Routine).

The elements of the problem space consist of all subsets of warehouses, taken from a list of possible sites (which is a subset of the total set of sites with customer demand). The program builds up the set of warehouses by the operation of adding one warehouse at a time. The actual data structure corresponding to the element consists not only of the list of warehouses but also the assignment of each customer to a warehouse, the partial cost borne by that warehouse, and the total cost of operating (TC). (That incremental cost calculations are easier to make than calculations starting from scratch is an important aspect of the efficiency of programs such as this one.) The main part of the program simply considers adding new warehouses (i.e., taking steps in the problem space) and comparing these against the current position on total cost. It is a steepest ascent scheme, since it goes through the whole set and then picks the best one. The additional wrinkle is to eliminate from the set of unused warehouses any whose costs are less than the current position, thus depleting the set to be considered. In fact, the first stage terminates when this set becomes empty.

The operator generator delivers only a fixed subset of all possible warehouse sites. It does this by a simple hill-climbing scheme whereby the best n sites are chosen from the total set on the basis of local cost (LC), which is the cost savings to be made from handling the local demand at the same site as the warehouse (n is a parameter of the program). This cascading of two optimizers keeps the second one from becoming excessively costly.

The next two stages (the Bump and Shift Routine) make minor

[4] We often use the term problem space to refer to the set of potential solutions as defined by the problem statement of a method. It includes the associated operations for moving around in the space.

Problem statement

Given: a set of customers, $\{c\}$, with locations and sales volume;
a set of factories, $\{f\}$, with locations;
a set of warehouse sites, $\{w\}$, with transportation costs to customers and to factories, and operating costs.

Find: a set of warehouses that minimizes total costs.

Problem space for hill climbing

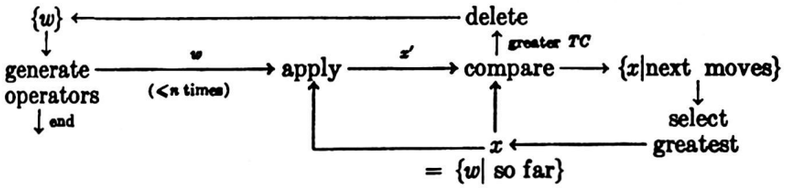
Elements: $\{x|x$ is a subset of $\{w\}\}$,
for each x can compute $TC(x)$.

Initial element: the null set.

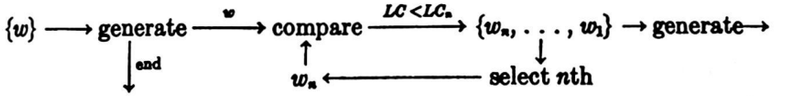
Desired element: the element with lowest TC.

Operators: 1. Add w to x;
2. delete w from x;
3. move $w \in x$ to location of customer of w.
Note: all these permit incremental calculation of TC, since only paired comparisons with existing w for each customer affected are required.

Procedure for stage 1 (operator 1)



Generate operators:



$TC(x)$ = total cost of x to supply all customers.
$LC(w)$ = local cost of w to supply customers at location of w.

Figure 10.7. Warehouse location heuristic program.

adjustments in the element (the set of warehouses) that results from the first phase. Warehouses are successively eliminated if they fail to pay for themselves. This is hill climbing if the order of elimination affects subsequent costs; otherwise it is simply generating through the parts of the system, making local changes. Note that no new warehouses are

added to the solution element after deletion. Finally, as the last stage, each warehouse is moved around locally in its own territory to find the most advantageous location. This stage constitutes a series of independent generate-and-test optimizations, since no interaction between warehouses is involved.

3.4. *Heuristic Search*

The best-known method in heuristic programming is the one whereby the problem is cast as a search through an exponentially expanding space of possibilities—as a search which must be controlled and focused by the application of heuristics. All of the game-playing and theorem-proving programs make use of this method, as well as many of the management science applications [13].[5]

Figure 10.8 gives the most elementary variant of the method. It assumes a space of elements, the problem space, which contains one element representing the initial position, and another representing the final or desired position. Also available is a fixed set of operators, which when applied to elements in space produce new elements. (Operators need not always be applicable.) The problem is to produce the final desired position, starting at the initial one.

With only this information available, the method involves a search that expands in a tree-like fashion. The initial element x_0 is the initial current position; operators are selected and applied to it; each new element is compared with x_d to see whether the problem is solved; if not, it is added to a list of obtained positions (also called the "try list" or the "subproblem list"); and one of these positions is selected from which to continue the search. If about B of the operators applied are applicable to an obtained position, about B^D elements will have been reached after D steps.

The search is guided (i.e., the tree pruned) by appropriate selection and rejection of operators and elements. The flow diagram provides a scheme upon which the various possibilities can be localized. The most elementary ones are unconditional: a rule for operator selection or for element selection. The latter is often accomplished by keeping an ordered list of elements and simply selecting the first one on the list; hence the order is dictated by the insertion process. The simplest rules have been given names. Thus, if the list is last-in-first-out (so that insertion is always at the front), the resulting search is *depth first*. This scheme was used by almost all early game-playing programs. If the list is first-in-first-out (so that insertion is always at the end), the resulting search is *breadth first*. This scheme has been used by some theorem provers.

An element may be completely rejected at the time of insertion. One of the most frequent heuristics is to reject if the element is already in the obtained list, that is, to avoid duplication. (This is a heuristic, since, though always beneficial, it requires extra effort and memory; thus it may not pay compared to, say, additional search.) Information already available in the scheme can be used; for instance, a rather important

[5] A few game players are exceptions. They use a recognition method that learns to associate to each game position (or class of positions) a good move. Although theoretically capable of handling complex games through the development of an appropriate classification, this method has not been used in any but simple games.

Problem statement

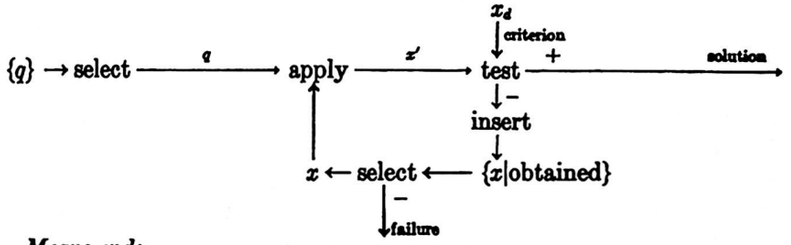
Given: a set $\{x\}$, the problem space;
a set of operators $\{q\}$ with range and domain in $\{x\}$;
an initial element, x_0;
a desired element, x_d.

Find: a sequence of operators, $q_1, q_2, \ldots, q_n$, such that they transform x_0 into x_d:

$$q_n[q_{n-1} \ldots q_1(x_0) \ldots] = x_d$$

Procedure

Basic:



Means-end:

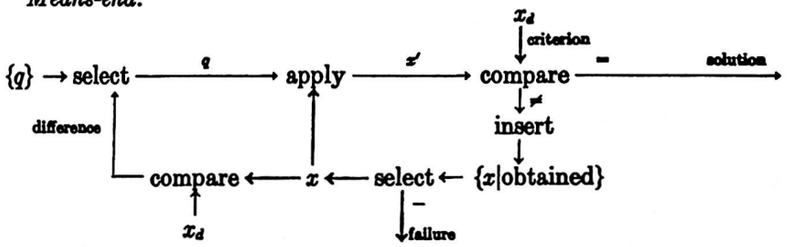


Figure 10.8. Heuristic search method.

variant is called means-ends analysis. The current element x is compared with the desired element x_d, and the resulting difference is used to select the operator, that is, the operator (means) is selected with a view toward the end. The flow diagram for this variant is given in the figure below the basic heuristic search.

The situation described in Fig. 10.8 is overly simple in several respects. Initially, a set of elements may be given, rather than just one, with the search able to start from any of them. Likewise, finding any one of a set of elements can be desired, rather than just finding a single one. This final element (or set) can be given by a test instead of by an element, although this has consequences for variants such as means-ends analysis. More important, the operators need not involve only a single input and a single output. In logic, for instance, important rules of inference, such as *modus ponens*, take two inputs and deliver a single output: from a and $a \supset b$ infer b. Thus we can have multiple inputs for an operator. Similarly, if one is working backwards in logic (that is, from the conclusion to premises that make this conclusion follow), *modus ponens* shows up as an operator that has a single input but a set of outputs: to get b, prove a and $a \supset b$. Furthermore, *all* of the outputs must be obtained; thus independent subproblems must radiate from each element of the output set in order to solve the problem.

Figure 10.9 shows one of the early theorem provers, LT (the Logic Theorist), which worked on elementary symbolic logic [15]. It is a heuristic search, using a breadth first strategy. However, it also uses generate-and-test and match, so that, like the warehouse location program, it has a composite structure. Comparison of LT with the basic heuristic search method will show that there are two unexpected twists to formulating the task of theorem proving for heuristic search. First, the problem has to be turned around, so that LT works backward from the original goal toward the given theorems. Thus the rules of inference must be expressed in an inverse sense. Second, the assumed theorems enter into the task both in the generation of operations and in the test. This actually reflects a restriction to the generality of LT, since it insists that one of the two expressions in the backward rules of inference be tied immediately to a theorem, rather than being a subproblem which need only make contact eventually.

The only elaborations of LT from the basic heuristic search procedure are the insertion of a test for similarity between t and x before trying to apply the operator, and the rejection of duplicate elements, which requires keeping a list of the elements already tried. The test for solution is elaborated in the minimal generate-and-test way to take into account

Problem statement

Given: the rules of inference of propositional logic;
a set of theorems, $\{t\}$, assumed valid;
a theorem, x_0.

Find: a proof of x_0 from the assumed theorems.

Problem space for heuristic search

Elements: logic expressions, $\{x\}$.
Initial element: theorem to be proved, x_0.
Desired elements: any of the assumed theorems, $\{t\}$.
Operators: pairs (m, x), where t is any assumed theorem and m is an expression of the rules of inference, working backwards:

MDt (Detachment): $(t{:}a \supset b, x{:}b) \rightarrow a$
MChF (Chaining forward): $(t{:}a \supset b, x{:}a \supset c) \rightarrow b \supset c$
MChB (Chaining backward): $(t{:}a \supset b, x{:}d \supset b) \rightarrow d \supset a$

Procedure

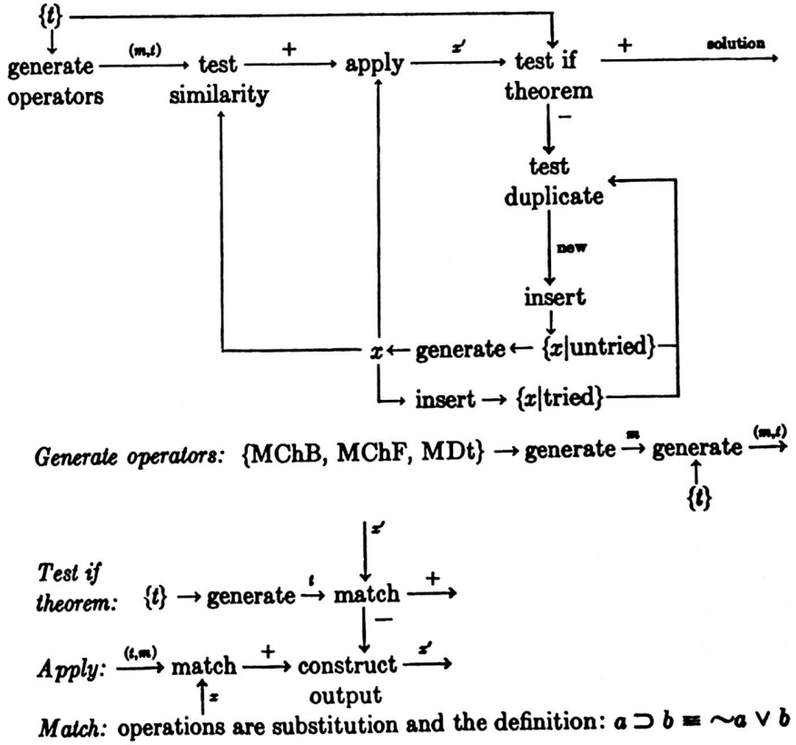


Figure 10.9. LT: Logic Theorist.

that any of the set of theorems will do. Although we have not shown the internal structure of the match, it does use definitions as well as substitutions to make a theorem and an expression the same.

3.5. *Induction*

Figure 10.10 shows a task that clearly involves inductive reasoning: you are to determine which of the figures 1–5 bears the same relationship to figure C as figure B does to figure A [4]. Similar problems exist in extrapolating series [22]: for example, what is the blank in *abcbcdcd*_?

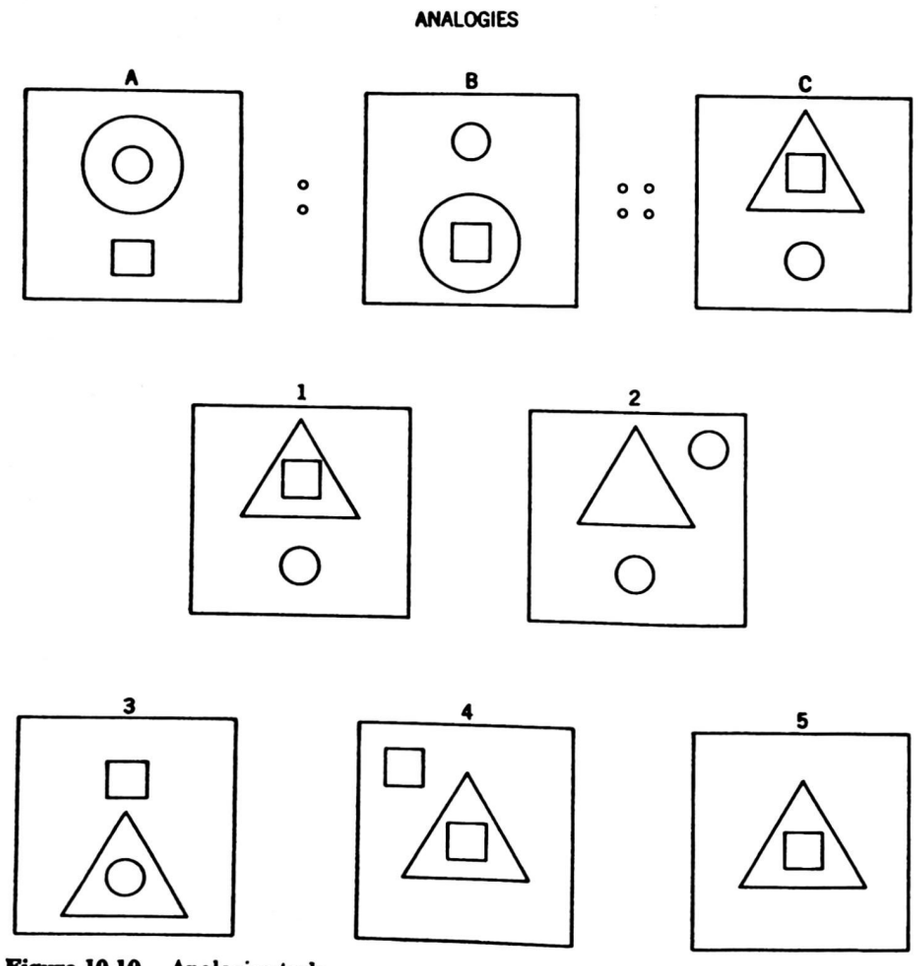


Figure 10.10. Analogies task.

Another similar task is to discover a concept, given a sequence of items, some of which exemplify the concept whereas others do not [8]: for example, if xoxox, xoxxo, oxoxo are positive instances and xooxo, xxoox, xoooo are negative instances, what is oxxxo?

Computer programs have been constructed for these tasks. They show a certain diversity due to the gross shape of the task; that is, the task of Fig. 10.10 gives one instance of the concept (A:B) and five additional possible ones {C:1, C:2, . . . , C:5}, whereas the series provides a long sequence of exemplars if one assumes that each letter can be predicted from its predecessors. However, most of the programs use a single method, adapted to the particular top-level task structure.[6] Figure 10.11 gives the method, although somewhat more sketchily than for the others.

The first essential feature of the method is revealed in the problem statement, which requires the problem to be cast as one of finding a function or mapping of the given data into the associated (or predicted) data. The space of functions is never provided by the problem poser—certainly not in the three examples just presented. Often it is not even clear what the range and domain of the function should be. For the series extrapolation task, to view $\{x{:}y\}$ as $\{a{:}b,\ ab{:}c,\ abc{:}b, \ldots,$ *abcbcdcd*:_$\}$ is already problem solving. Thus the key inductive step is the assumption of some space of functions. Once this is done the problem reduces to finding in this space one function (or perhaps the simplest one) that fits the exemplars.

The second essential feature of the method is the use of a form or kernel for the function. This can be matched (in the sense of the match method) against the exemplars. Evidence in the items then operates directly to specify the actual function from the kernel. Implicit in the procedure in Fig. 10.11 is that, inside the match, generation on the kernel (refer back to Fig. 10.5) produces, not the parts of the kernel itself, but the predictions of the y associated with the presented x. However, parts of the kernel expression must show through in these predictions, so that the modification operations of the match can specify or modify them in the light of differences. When the kernel expression actually has variables in it, the prediction from the kernel is sometimes a variable. Its value can be made equal to what it should be from the given $x{:}y$, and thus the kernel expression itself specified. Often the modification operations are like linguistic replacement rules, and then the matter is somewhat more complex to describe.

[6] Most, but not all. Several adapt the paradigm used for pattern recognition programs. In addition, a method called the method of successive differences is applicable to series extrapolation where the terms of the series are expressed as numbers.

Problem statement

Given: a domain $\{x\}$;
a range $\{y\}$;
a generator of associated pairs $\{x,y\}$.

Find: a function f with domain $\{x\}$ and range $\{y\}$ such that $f(x) = y$ for all $\{x:y\}$.

Additional assumption (almost never given with the problem statement, and therefore constituting the actual inductive step):

Given: a set of functions $\{f\}$ constructable from a set of kernel forms $\{k\}$.

Procedure

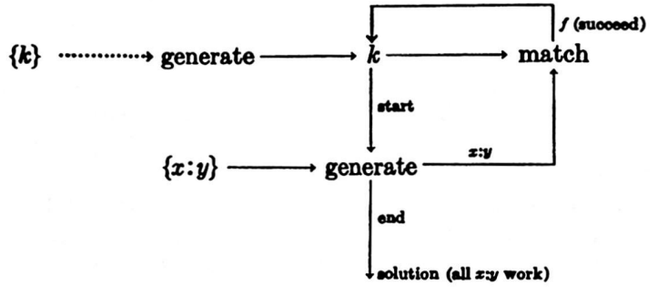


Figure 10.11. Induction method.

It is not often possible to express the entire space of functions as a single form (whence a single match would do the job). Consequently a sequential generation of the kernels feeds the match process. Sometimes clues in the exemplars are used to order the generation; more often, generation is simply from the simplest functions to the more complex.

This method is clearly a version of "hypothesis-and-test." However the latter term is used much more generally than to designate the class of induction problems handled by this method. Furthermore, there is nothing in hypothesis-and-test which implies the use of match; it may be only generate-and-test. Consequently, we choose to call the method simply the induction method, after the type of task it is used for.

3.6. Summary

The set of methods just sketched—generate-and-test, hill climbing, match, heuristic search, and induction—constitutes a substantial fraction of all methods used in heuristic programming. To be sure, this is only a judgment. No detailed demonstration yet exists. Also, one or two important methods are missing; for example, an almost universally used paradigm for pattern recognition.

Two characteristics of the set of methods stand out. First, explicit references to processes occur in the problem statement, whereas this is not true of mathematical methods, such as the simplex method. Thus generate-and-test specifies that you must have a generator and you must have a test; then the procedure tells how to organize these. This feature seems to be related to the strength of the method. Methods with stronger assumptions make use of known processes whose existence is implied by the assumptions. In the simplex method generation on a set of variables is done over the index, and the tests used are equality and inequality on real numbers. Hence there is no need to posit directly, say, the generator of the set.

The second characteristic is the strong similarity of the methods to each other. They give the impression of ringing the various changes on a small set of structural features. Thus there appear to be only two differences between heuristic search and hill climbing. First, it is necessary to compare for the greater element in hill climbing; heuristic search needs only a test for solution (although it can use the stronger comparison, as in means-ends analysis). Second, hill climbing keeps only the best element found so far, that is, it searches the problem space from where it is. Heuristic search, on the other hand, keeps around a set of obtained elements and selects from it where next to continue the search. In consequence, it permits a more global view of the space than hill climbing—and pays for it, not only by extra memory and processing, but also by the threat of exponential expansion.

Similarly, the difference between match and heuristic search is primarily one of memory for past actions and positions. Our diagram for match does not reveal this clearly, since it shows only the case of a single modification operation, substitution; but with a set of modification operations (corresponding to the set of operators in heuristic search) the match looks very much like a means-ends analysis that never has to back up.

Finally, the more complex processes, such as LT and the warehouse program, seem to make use of the more elementary ones in recognizable combinations. Such combination does not always take the form of distinct units tied output to input (i.e., of closed subroutines), but a flavor still exists of structures composed of building blocks.

In reviewing these methods instruction in the details of artificial intelligence has not been intended. Hopefully, however, enough information has been given to convince the reader of two main points: (1) there is in

heuristic programming a set of methods, as this term was used in the beginning of the paper; and (2) these methods make much weaker demands for information on the task environment than do methods such as the simplex, and hence they provide entries toward the lower, more general end of the graph in Fig. 10.3.

4. THE CONTINUNITY OF METHODS

If the two hypotheses that we have stated are correct, we should certainly expect there to be methods all along the range exhibited in Fig. 10.3. In particular, the mathematical methods of management science should not be a species apart from the methods of artificial intelligence, but should be distinguished more by having additional constraints in the problem statement. Specific mathematical content should arise as statements strong enough to permit reasonable mathematical analysis are introduced.

Evidence for this continuity comes from several sources. One is the variety of optimization techniques, ranging from hill climbing to the calculation methods of the differential calculus, each with increasing amounts of specification. Another is the existence of several methods, such as so-called branch and bound techniques, that seem equally akin to mathematical and heuristic programming. Again, dynamic programming, when applied to tasks with little mathematical structure, leads to procedures which seem not to differ from some of the methods in heuristic programming, for example, the minimax search techniques for playing games such as chess and checkers.

What we should like most of all is that each (or at least a satisfactory number) of the mathematical methods of management science would lie along a line of methods that extends back to some very weak but general ancestors. Then, hopefully, the effect on the procedure of the increasing information in the problem statement would be visible and we could see the continuity directly.

As a simple example, consider inverting a matrix. The normal algorithms for this are highly polished procedures. Yet one can look at inversion as a problem—as it is to anyone who does not know the available theory and the algorithms based on it. Parts of the problem space are clear: the elements are matrices (say of order n), hence include both the given matrix, **A**, the identity matrix, **I**, and the desired inverse, **X**. The problem statement is to find **X** such that $\mathbf{AX} = \mathbf{I}$. Simply generating and testing is not a promising way to proceed, nor is expressing **X** as a form, multiplying out, and getting n^2 equations to solve. Not only are these poor approaches, but also they clearly are not the remote ancestors of the existing algorithms.

If the inverse is seen as a transformation on **A**, carrying it into **I**, a more interesting specification develops. The initial object is **A**, the desired object is **I**, and the operators consist of premultiplication (say) by some basic set of matrices. Then, if operators $\mathbf{E}_1, \mathbf{E}_2, \ldots, \mathbf{E}_k$ transform **A** into **I**, we have $\mathbf{E}_k \cdots \mathbf{E}_2\mathbf{E}_1\mathbf{A} = \mathbf{I}$, hence $\mathbf{E}_k \cdots \mathbf{E}_2\mathbf{E}_1\mathbf{I} = \mathbf{A}^{-1}$. If the basic operators are the elementary row operations (permute two rows, add one row to another, multiply a row by a constant), we have the basic ingredients of several of the existing direct algorithms (those that use elimination rather than successive approximation). These algorithms prescribe the exact transformations to be applied at each stage, but if we view this knowledge as being degraded we can envision a problem solver doing a heuristic search (or perhaps hill climbing if the approach were monotone). Better information about the nature of the space should lead to better selection of the transformation until existing algorithms are approached.

4.1. An Example: the Simplex Method

The simplex method clearly involves both optimization and search, hence should eventually show kinship with the methods that we have been describing. We should be able to construct a sequence of methods, each with somewhat less stringent conditions and therefore with more general applicablilty but less power. Power can be measured here by applying the method to the original linear programming (LP) problem, where the true nature of the problem is known.

Figure 10.12 reformulates the LP problem and the simplex algorithm in the present notation. Indices have been suppressed as much as possible, partly by using the scalar product, in the interests of keeping the figures uncluttered. The problem statement for the simplex method, which we call SM from now on, is a special case of the LP problem, having equalities for constraints rather than inequalitities, but involving $n + m$ variables rather than just n. The transformation between problems is straightforward (although the original selection of this specialization is not necessarily so).

The elements of the problem space (called bases) are all the subsets of m out of the $n + m$ variables. An element consists of much more information than just the subset of variables, of course, namely, of the entire tableau shown in Fig. 10.2. The operators theoretically could be any rules that replace one subset of m variables with another. In fact, they involve adding just a single variable, hence removing one. This

Problem statement for LP problem

Given: a set of n variables, $\{x\}$, where each $x \geqslant 0$:
let $\bar{x}$ be the n-tuple $(x_1, x_2, \ldots, x_n)$;
a set of m constraints, $\{g = b - \bar{a}\bar{x}\}$:
let the feasible region be $\{\bar{x} | g \geqslant 0\}$;
an objective function, $z = \bar{c}\bar{x}$.

Find: $\bar{x}$ in the feasible region such that z is maximum.

Problem statement for SM, the simplex method

Given: a set of $n + m$ variables, $\{x\}$, where each $x \geqslant 0$:
let $\bar{x}$ be the $(n + m)$-tuple $(x_1, x_2, \ldots, x_{n+m})$;
a set of m constraints, $\{g = b - \bar{a}\bar{x}\}$:
let the feasible region be $\{\bar{x} | g = 0\}$;
an objective function, $z = \bar{c}\bar{x}$.

Find: $\bar{x}$ in the feasible region such that z is maximum.

Note: any LP problem can be solved if this one can. It is a separate problem to provide the translation between them (define $x_{n+i} = g_i$ and determine $\bar{c}$ and the $\bar{a}$ accordingly).

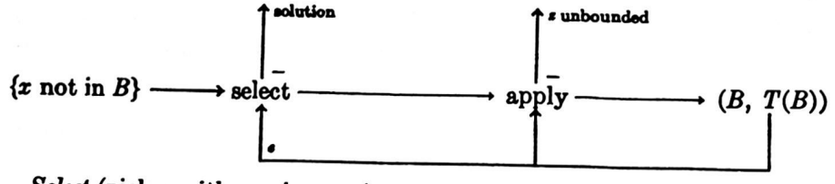
Problem space for SM

Elements: $\{B$ (bases), the $\binom{n+m}{m}$ subsets of m variables from $\{x\}\}$;
with B is associated $T(B)$ (the tableau) containing:
a feasible x such that $x \in B$ implies $x > 0$; otherwise $x = 0$;
the current value of z for $\bar{x}$;
the exchange rate (e) for each x [$-(z - c)$ in tableau];
auxiliary data to permit application of any operator.

Initial element: B_0, a feasible basis (not obtained by SM).

Operators: $\{x$ not in $B\}$.

Procedure



Select (pick x with maximum e):

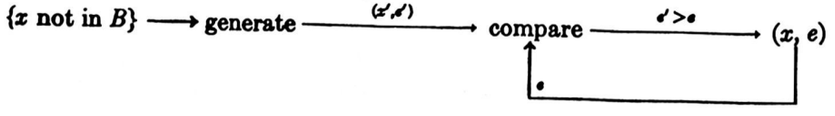


Figure 10.12. SM: reformulation of simplex method.

would still leave $(n - m)m$ operators (any of $n - m$ in, any of m out), except that no choice exists on the one to be removed. Hence, there are just $n - m$ operators, specified by the $n - m$ variables not in the current basis. Applying these operators to the current element consists of almost the entire calculation of the simplex procedure specified in Fig. 10.2 (actually steps 2, 3, and 4), which amounts to roughly $m(n + m)$ multiplications (to use a time-honored unit of computational effort).

The procedure in Fig. 10.12 looks like a hill climber with the comparison missing: as each operator is selected, the new (B,T) is immediately calculated (i.e., the tableau updated) and a new operator selected. No comparison is needed because the selection produces only a single operator, and this is known to advance the current position. The procedure for selecting the operator reveals that the process generates over all potential operators—over all x not in the current basis—and selects the best one with respect to a quantity called the exchange rate (e). Thus the select is a simple optimizer, with one exception (not indicated in the figure): it is given an initial bias of zero, so that only operators with $e > 0$ have a chance.

The exchange rate is the rate of change of the objective function (z) with a change in x, given movement along a boundary of the constraints, where the only variables changing are those already in the basis. Given this kind of exploration in the larger space of the $n + m$ variables, e measures how fast z will increase or decrease. Thus, $e > 0$ guarantees that the compare routine is not needed in the main procedure.

The selection of an x with the maximum exchange rate does not guarantee either the maximum increase from the current position or the minimum number of steps to the optimum. The former could be achieved by inserting a compare routine and trying all operators from the current position; but this would require many times as much effort for (probably) small additional gain. However, since the space is unimodal in the feasible region, the procedure does guarantee that eventually an optimum will be reached.

We need a method at the general end of the scale against which the SM can be compared. Figure 10.13 gives the obvious one, which we call M1. It retains the shape of the problem but without any specific content. The problem is still to optimize a function, f, of n positive variables, subject to m inequality constraints, $\{g\}$. But the only thing known about f and the g is that f is unimodal in the feasible set (which accounts for its descriptive title). This justifies using hill climbing as the procedure. The operators must be any increments to the current position, either positive or negative. Many will produce a location outside the feasible

Problem statement for M1, the unimodal objective method

Given: a set of n variables, $\{x\}$, where each $x \geq 0$:
let $\bar{x}$ be the n-tuple $(x_1, x_2, \ldots x_n)$;
a set of m constraints, $\{g(\bar{x}) \geq 0\}$:
let the feasible region be $\{\bar{x} | g \geq 0\}$;
an objective function $z = f(\bar{x})$;
f is unimodal in the feasible region.

Find: $\bar{x}$ in the feasible region such that z is maximum.

Problem space PS1, for hillclimbing

Elements: $\{\bar{x}\}$.
Initial element: x_0, a feasible solution (not obtained by M1).
Operators: $\{\Delta\bar{x}$, where each Δx is any real number$\}$,
and $\bar{x}' = \bar{x} + \Delta\bar{x}$.

Procedure

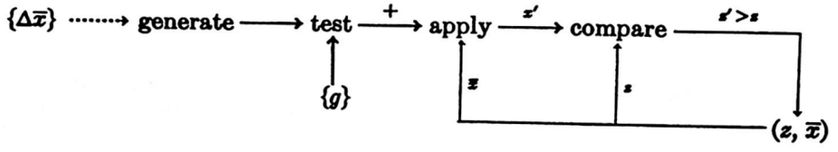


Figure 10.13. M1: unimodal objective method.

region, but these can be rejected by testing against the g. The procedure in the figure does not provide any information on how to generate the operators.

If M1 is compared to SM, several differences are apparent. First, and most striking, the problem space for M1 is n-dimensional Euclidean space, whereas for SM it is a finite set of $\binom{n+m}{m}$ points in $(n+m)$-dimensional space. Thus the search space has been drastically reduced, independently of what techniques are used to search it. Second, and almost as striking, M1 has all of $\{\Delta\bar{x}\}$ as operators (i.e., all of n-dimensional space again), whereas SM has only $n - m$ operators. Third, a unique operator is selected for application on the basis of partial information; it always both applies and improves the position. In M1 there is no reason to expect an operator either to produce a feasible solution or, if it does, to obtain an improvement; thus, extensive testing and comparing must be done. Finally, we observe that the cost per step in M1 (when applied to the same LP problem as SM) is mk, where k is the number of variables in $\Delta\bar{x}$ and would normally be rather small. Compared to $m(m+n)$ for SM, this yields the one aspect favoring M1. One can take about $(m+n)/k$ steps in the space of M1 for each step in the space of SM. However, the thrashing around necessary at each point to obtain a positive step will largely nullify this advantage.

This list of differences suggests constructing a sequence of methods that extend from M1 to SM, with decreasing spaces and operators and increasing cost per step (to pay for the additional sophistication). Much of the gain, of course, will come without changing problem spaces, but from acquiring better operator selection. There may be relatively few substantial changes of problem space. Figures 10.14 and 10.15 provide

Problem statement for M2, the monotone objective method

Given: the conditions of M1, plus
f is monotone in the feasible region.

Find: $\bar{x}$ in the feasible region such that z is maximum.

Problem space PS2, main hill climbing

Elements: $\{\bar{x}$ on boundary of feasible region (at least one $x = 0$ or $g = 0)\}$.
Initial element: x_0 in feasible region (not obtained by M2).
Operators: $\{x\}$, where $\bar{x}'$ = the point on boundary given by M2*.

Problem statement for M2°, M2-operator method

Given: the conditions of M2, plus
$\bar{x}$ is on the boundary;
$x \in \{x\}$.

Find: $\bar{x}$ on the boundary such that
Δx to increase z not feasible;
all other x unchanged;
z increased.

Additional assumption for efficiency:
$g(\bar{x}) = 0$ can be solved for any x with all other x fixed.

Problem space for M2°, hill climbing

Elements: $\{\bar{x}\}$.
Initial element: x, given by M2 operator.
Operators: $\{\Delta x$, with appropriate sign$\}$, where $\bar{x}' = \bar{x} + \Delta x$.

Problem space PS1 for M1

Used as backup when PS2 terminates without optimum.

Figure 10.14. M2: monotone objective method.

Problem statement for M3, the consistent exchange problem

Given: the conditions of M2, plus
if an x is exchanged for other variables by moving along a maximal boundary, then Δz has a consistent sign.

Find: $\bar{x}$ in the feasible region such that z is a maximum.

Problem space PS3, main hill climbing

Elements: {$\bar{x}$ on the maximal boundary of feasible set; i.e., no x can be changed to increase z, holding other x fixed}.

Initial element: x_0, a feasible solution on the maximal boundary (not obtained by M3).

Operators: $\{x\}$, where $\bar{x}'$ = the point on the maximal boundary given by M3*.

Problem statement for M3*, M3-operator method

Given: the condition of M3, plus
$\bar{x}$ is on a maximal boundary;
$x \in \{x\}$.

Find: $\bar{x}$ on the maximal boundary, such that exchange for x to increase z is not feasible;
z increased.

Additional assumption for efficiency:
any system of k equations, $\{g(\bar{x}) = 0\}$, can be solved for any set of k variables with the others fixed.

Problem space for M3*

Elements: $\{x\}$.

Initial element: x, given by M3 operator.

Operators: {Δx, with appropriate sign} and {$\Delta\bar{x}$, subsets of $\{x\}$}, where $\bar{x}' = \bar{x} + \Delta\bar{x}$.

Figure 10.15. M3: consistent exchange method.

two that seem to reflect some of the major boundaries to be crossed in getting from M1 to SM.

Figure 10.14 shows method M2, which adds the assumption that the objective function, f, is monotone in the feasible region. In the LP problem $\partial f/\partial x$ is constant, but this is not required to justify the main conclusion; namely, that if a given change in a varible is good, more change in the same direction is better. The effect of this is to create new operators and, through them, a new space. The basic decision is always to drive a variable to a boundary (in the direction of increasing z, of course). Thus the space for M2 becomes the boundary set of the original space for M1 (those points where at least one of the constraints, including the $x \geqslant 0$, attains zero). The operators in the space of M2 are full steps to a boundary (what are sometimes called macromoves in artificial intelligence). Now, finding the boundary is still a problem, although a more manageable one. Thus M2 has a second problem method, M2*, for this purpose. As described in Fig. 10.14 it can be a simple hill climber.

An additional strong assumption has been made in the procedure of M2, namely, that only changes in a single variable, x, will be considered. This reduces the number of operators, as well as making the operator submethod M2* simpler. It is not justified by the assumptions of the problem statement, however, and consequently M2 will terminate at suboptimal positions where no single variable can be changed to increase z without decreasing some other variables. (This is often called the maximal or the Pareto optimum set.) Rather than relax the operators to a wider class, the original method, M1, is held in reserve to move off the maximal set. (However, if done with small steps, this is extremely inefficient for the LP problem, since the system just jitters its way slowly up a bounding plane.)

The description of M2* gives an additional assumption: each equation $g(\bar{x})$ can be solved directly for any x, given that the values of the other x's are determined. This permits direct calculation of the extreme value of x on a boundary that is maximal. Slightly stronger conditions on the g's allow determination of the first constraint to become binding, without multiple evaluations.

Figure 10.15 shows method M3, which adds the assumption that the exchange rate (e) always has a consistent sign as one moves along the feasible region, in response to introducing a variable x (what we have called exchanging). Again, in the LP problem e is constant, but this is not required to justify the main conclusion: that in a maximal situation, if adjustments are made in other variables to allow a particlular variable to increase, and the gain from the exchange is positive, it will always be positive; hence the new variable should be exchanged for as much as possible, namely, until another boundary is reached. This assumption not only allows a better way of dealing with the maximal cul-de-sac than does M2, with its regression to M1, but also permits the problem space to be changed radically a second time.

The elements of the space now become the set of maximal points, thus a small subset of the space of M2. The operators remain the same; the individual variables. The application of an operator again cor-

responds to the solving of a subproblem, hence is accomplished by a submethod, M2*. The problem is as follows: given x (with a positive exchange rate), to advance it as far as possible. This means solving the constraints simultaneously for the variables, so as to remain on a boundary. As a change in the selected x is made, the current x moves off the maximal boundary by violating either the constraints or maximality. Adjustments must be made in the other x's to restore these conditions. What the new clause in the problem statement provides is not a way of making the adjustments, but a guarantee that if a change is once found that does increase z (after adjustment) it should be pushed to the limit.

We have not described M3*, other than to indicate the available operators. At its most general (i.e., assuming no other information), it requires a two-stage process, one to discover a good direction and the other to push it. The latter is again a two-stage process, one to change the selected $\bar{x}$ and the other to make the adjustments. We have included an additional assumption, similar to the one for M2*, that a direct way exists of solving systems of contraints for some variables in terms of others. This clearly can make an immense difference in the total efficiency of problem solving but does not alter the basic structuring of the task.

M3 is already a recognizable facsimile of SM. The space has been cut to all subsets of the variables, although the final contraction to subsets of m variables has not occurred. (It is implicit in the problem statement of M3, with some mild conditions on the g's, but has not been brought out.) The operators of M3 and SM are the same. More precisely, they are isomorphic—the process of applying an operator is quite different in the two methods. There are still some steps to go. The kinds of methods that are possible for the operator need explication. They are represented in M3* only by the assumption that systems of equations can be solved. But the schemes in SM use special properties of linear systems. Similarly, we have not explored direct calculation of the exchange rates, with the subsequent replacement of comparison in the main method by comparison in the operator, to avoid expensive computation.

We have not carried this example through in complete detail, nor have we established very many points on the path from a crude hill climber to SM. The two points determined are clearly appropriate ones and capture some of the important features of the method. They are not unexpected points, of course, since linear programming is well understood. The viewpoint underlying the analysis is essentially combinatorial, and such aspects have been thoroughly explored (e.g., see [23]). If these intermediate problems have any peculiar flavor, it is that they become established where the search spaces change, and these need not always correspond to nice mathematical properties, abstractly considered. Thus convexity is not posited and its implications explored; rather a change of search space is posited and the problem statement that admits it sought.

A single ancestral lineage should not be expected. Just as theorems can have many proofs, so methods can have many decompositions of their information. In fact, in one respect at least the line represented by M2 and M3 does violence to SM. It never recognizes the shift of problem into a set of equality constraints with the consequent change in dimensionality. Thus, the g's and the x's are handled separately, whereas it is a very distinct feature of the simplex procedure that it handles them uniformly. One could easily construct another line starting from SM, which would preserve this feature. (It would face a problem of making the transition to M1.)

The examples selected—linear programming and matrix inversion—are certainly ones that seem most amenable to the kind of analysis we have proposed. If we considered methods, say, for determining inventory levels, the story might be different. Nevertheless, perhaps the case for continuity between weak and strong methods has been made plausible.

5. HUMAN BEHAVIOR IN ILL-STRUCTURED PROBLEMS

In the two issues discussed so far—the existence of weak methods, and the continuity between weak and strong methods—we have not seemed to be dealing directly with ill-structured problems. To re-evoke the concern of Reitman, the problem statements that we have exhibited seem quite precise. (Indeed, we took pains to make them so and in a more technical exposition would have completely formalized them.) According to our hypotheses the world is always formalized, seen from the viewpoint of the methods available, which require quite definite properties to operate. A human problem solver, however, would not feel that a problem was well structured just because he was using a method on it. Our second hypothesis identifies this feeling with the low power of the applicable methods.

The concern just expressed is still well taken. If we examine some problem solvers who are working on "really ill-structured" problems, what will we find? They will necessarily be human, since as noted earlier, men are the gatekeepers of this residual class of problems. Thus we cannot observe their problem-solving processes directly but must infer them from their behavior.

To have something definite in mind consider the following problem solvers and tasks:

A financial adviser: what investments should be made in a new account?
A foreman: is a given subordinate well adjusted to his work?
A marketing executive: which of two competitors will dominate a given market to which his firm is considering entry?

None of these problems is as ill structured as the proverbial injunctions to "know thyself" (asked of every man) and to "publish or perish" (asked of the academician). Still they are perhaps more typical of management problems than these two almost completely open-ended problems. They do have the feature of most concern to Reitman; namely, neither the criteria for whether a solution is acceptable nor the data base upon which to feed are particularly well defined.

The framework we have been using says that below the surface we should discover a set of methods operating. Our two hypotheses assert, first, that we should find general but weak methods; and, second, that we should not find methods that deal with the unstructured aspects (however they come to be defined) through any mechanism other than being general enough to apply to a situation with little definite information.

Our first implication would seem to be upset if we discover that the human being has available very strong methods that are applicable to these ill-structured problems. This is a rather difficult proposition to test, since, without the methods themselves to scrutinize, we have very little basis for judging the nature of problem solving. Powerful methods imply good solutions, but if only men solve the problem, comparative quality is hard to judge.

The three tasks in our list have the virture that comparisons have been made between the solutions obtained by human effort and those obtained by some mechanical procedure. For the second two the actual tasks are close nonmanagement analogs of the tasks listed. However, they all have the property (implicit in our list) that the problem solver is a man who by profession is concerned with solving the stated type of problem. This condition is important in discussing real management problems, since the capabilities of a novice (e.g., a college student used as a subject in an experiment) may differ considerably from those of the professional. In particular, the novice may differ in the direction of using only very general reasoning abilities (since he is inexperienced), whereas the professional may have special methods.

In all of the cases the result is the same. Rather simple mechanical procedures seem to do as well as the professional problem solver or even better; certainly they do not do worse.

The first task was investigated by Clarkson [1] in one of the early simulation studies. He actually constructed a program to simulate a trust investment officer in a bank. Thus the program and the human being attain the same level of solution. The program itself consists of a series of elementary evaluations as a data base, plus a recognition structure (called a discrimination net) to make contact between the specific situation and the evaluation; there are also several generate-and-tests. Thus the program does not have any special mechanisms for dealing with ill-structuredness. Indeed it deals with the task in a highly structured way, though with a rather large data base of information. The key point is that the human being, who can still be hypothesized to have special methods for ill-structured situations (since his internal structure is unknown), does not show evidence of this capability through superior performance.

The second task in its nonmanagement form is that of clinical judgment. It has been an active, substantial—and somewhat heated—concern in clinical psychology ever since the forties. In its original form, as reviewed by Meehl [12], it concerned the use of statistical techniques versus the judgments of clinicians. With the development of the computer it has broadened to any programmed procedure. Many studies have been done to confront the two types of judgment in an environment sufficiently controlled and understood to reveal whether one or the other was better. The results are almost uniformly that the programmed procedures perform no worse (and often better) than the human judgment of the professional clinician, even when the clinician is allowed access to a larger "data base" in the form of his direct impressions of the patient. Needless to say, specific objections, both methodological and substantive, have been raised about various studies, so the conclusion is not quite as clear-cut as stated. Nevertheless, it is a fair assertion that no positive evidence of the existence of strong methods of unknown nature has emerged.[7]

The third task is really an analog of an analog. Harris [7], in order to investigate the clinical versus statistical prediction problem just discussed, made use of an analog situation, which is an equally good analog to the marketing problem in our list. He tested whether formulas for predicting the outcome of college football games are better than human judgment. To get the best human judgments (i.e., professional) he made use of coaches of rival teams. Although there is a problem of bias, these coaches clearly have a wealth of information of as professional a nature

[7] A recent volume [10] contains some recent papers in this area, which provide access to the literature.

as the marketing manager has about the market for his goods. On the program side, Harris used some formulas whose predictions are published each week in the newspapers during the football season. An unfortunate aspect of the study is that these formulas are proprietary, although enough information is given about them to make the study meaningful. The result is the same: the coaches do slightly worse than the formulas.

Having found no evidence for strong methods that deal with unstructured problems, we might feel that our two hypotheses, are somewhat more strongly confirmed. However, unless the weak methods used by human beings bear some relationship to the ones we have enumerated, we should take little comfort. For our hypotheses take on substantial meaning only when the weak methods become explicit. There is less solid evidence on what methods people use than on the general absence of strong methods. Most studies simply compare performance, and do not attempt to characterize the methods used by the human problem solver. Likewise, many of the psychological studies on problems solving, although positive to our argument [14], employ artificial tasks that are not sufficiently ill structured to aid us here. The study by Clarkson just reviewed is an exception, since he did investigate closely the behavior of his investment officer. The evidence that this study provides is positive.

Numerous studies in the management science literature might be winnowed either to support or refute assertions about the methods used. Work in the behavioral theory of the firm [2], for instance, provides a picture of the processes used in organizational decision making that is highly compatible with our list of weak methods—searching for alternatives, changing levels of aspiration, etc. However, the characterizations are sufficiently abstract that a substantial issue remains whether they can be converted into methods that really do the decision making. Such descriptions abstract from task content. Now the methods that we have described also abstract from task content. But we know that these can be specialized to solve the problems they claim to solve. In empirical studies we do not know what other methods might have to be added to handle the actual detail of the management decision.

Only rarely are studies performed, such as Clarkson's, in which the problem is ill structured, but the analysis is carried out in detail. Reitman has studied the composition of a fugue by a professional composer [18], which is certainly ill structured enough. Some of the methods we have described, such as means-ends analysis, do show up there. Reitman's characterization is still sufficiently incomplete, however, that no real evidence is provided on our question.

6. DIFFICULTIES

We have explored three areas in which some positive evidence can be adduced for the two hypotheses. We have an explicit set of weak methods; there is some chance that continuity can be established between the weak and the strong methods; and there is some evidence that human beings do not have strong methods of unknown nature for dealing with ill-structured problems. Now it is time to consider some difficulties with our hypotheses. There are several.

6.1. The Many Parts of Problem Solving

At the beginning of this essay we noted that methods were only a part of problem solving, but nevertheless persisted in ignoring all the other parts. Let us now list some of them:

Recognition	Information acquisition
Evaluation	Executive construction
Representation	Method construction
Method identification	Representation construction

A single concern applies to all of these items. Do the aspects of problem solving that permit a problem solver to deal with ill-structured problems reside in one (or more) of these parts, rather than in the methods? If so, the discussions of this essay are largely beside the point.

This shift could be due simply to the power (or generality) of a problem solver not being localized in the methods rather than to anything specific to ill-structuredness. The first two items on the list illustrate this possibility. Some problems are solved directly by recognition; for example, who is it that has just appeared before my eyes? In many problems we seem to get nowhere until we suddenly "just recognize" the essential connection or form of the solution. Gestalt psychology has made this phenomenon of sudden restructuring central to its theory of problem solving. If it were true, our two hypotheses would certainly not be valid. Likewise for the second item, our methods say more about the organization of tests than about the tests themselves. Perhaps most of the power resides in sophisticated evaluations. This would work strongly against our hypotheses. In both examples it is possible, of course, that hypotheses of similar nature to ours apply. In the case of evaluations, for example, it might be that ill-structured problems could be handled only because the problem solver always had available some dis-

tinctions that applied to every situation, even though with less and less relevance.

The third item on the list, representation of problems, also raises a question of the locus of power (rather than of specific mechanisms related to ill-structured problems). At a global level we talk of the representation of a problem in a mathematical model, presumably a translation from its representation in some other global form, such as natural language. These changes of the basic representational system are clearly of great importance to problem solving. It seems, however, that most problems, both well structured and ill structured, are solved without such shifts. Thus the discovery of particularly apt or powerful global representations does not lie at the heart of the handling of ill-structured problems.

More to the point might be the possibility that only special representations can hold ill-structured problems. Natural language or visual imagery might be candidates. To handle ill-structured problems is to be able to work in such a representation. There is no direct evidence to support this, except the general observations that human beings have (all) such representations, and that we do not have good descriptions of them.

More narrowly, we often talk about a change in representation of a problem, even when both representations are expressed in the same language or imagery. Thus we said that Fig. 10.12 contained two representations of the LP problem, the original and the one for the simplex method. Such transformations of a problem occur frequently. For example, to discuss the application of heuristic search to inverting matrices we had to recast the problem as one of getting from the matrix A to I, rather than of getting from the initial data $(A, I, AX = I)$ to X. Only after this step was the application of the method possible. A suspicion arises that changes of representation at this level—symbolic manipulation into equivalent but more useful form—might constitute a substantial part of problem solving. Whether such manipulations play any special role in handling ill-structured problems is harder to see. In any event, current research in artificial intelligence attempts to incorporate this type of problem solving simply as manipulations in another, more symbolic problem space. The spaces used by theorem provers, such as LT, are relevant to handling such changes.

Method identification, the next item, concerns how a problem statement of a method comes to be identified with a new problem, so that each of the terms in the problem statement has its appropriate referent in the problem as originally given. Clearly, some process performs this identification, and we know from casual experience that it often requires an exercise of intellect. How difficult it is for the LP novice to "see" a new problem as an LP problem, and how easy for an old hand!

Conceivably this identification process could play a critical role in dealing with ill-structured problems. Much of the structuring of a problem takes place in creating the identification. Now it might be that methods still play the role assigned to them by our hypotheses, but even so it is not possible to instruct a computer to handle ill-structured problems, because it cannot handle the identification properly. Faced with an appropriate environment, given the method and told that it was the applicable one, the computer still could not proceed to solve the problem. Thus, though our hypotheses would be correct, the attempt to give them substance by describing methods would be misplaced and futile.

Little information exists about the processes of identification in situations relevant to this issue. When the situation is already formalized, matching is clearly appropriate. But we are concerned precisely with identification from a unformalized environment to the problem statement of a method. No substantial programs exist that perform such a task. Pattern recognition programs, although clearly designed to work in "natural" environments, have never been explored in an appropriately integrated situation. Perhaps the first significant clues will come out of the work, mentioned at the beginning of this chapter and still in its early stages, on how a machine can use a hand and eye in coordination. Although the problems that such a device faces seem far removed from management science problems, all the essentials of method identification are there in embryo. (Given that one has a method for picking up blocks, how does one identify how to apply this to the real world, seen through a moving television eye?)

An additional speculation is possible. The problem of identification is to find a mapping of the elements in the original representation (say, external) into the new representation (dictated by the problem statement of the method to be applied). Hence there are methods for the solution to this, just as for any other problem. These methods will be like those we have exhibited. (Note, however, that pattern recognition methods would be included.) The construction of functions in the induction method may provide some clues about how this mapping might be found. As long as the ultimate set of contacts with the external representation (represented in these identification methods as generates and tests) were rather elementary, such a reduction would indeed answer the issue raised and leave our hypotheses relevant.

An important aspect of problem solving is the acquisition of new information, the next item on the list. This occurs at almost every step,

of course, but most of the time it is directed at a highly specific goal; for instance, in method identification, which is a major occasion for assimilating information, acquisition is directed by the problem statement. In contrast, we are concerned here with the acquisition of information to be used at some later time in unforeseen ways. The process of education provides numerous examples of such accumulation.

For an ill-structured problem one general strategy is to gather additional information, without asking much about its relevance until obtained and examined. Clearly, in the viewpoint adopted here, a problem may change from ill structured to well structured under such a strategy, if information is picked up that makes a strong method applicable.

The difficulty posed for our hypotheses by information acquisition is not in assimilating it to our picture of methods. It is plausible to assume that there are methods for acquisition and even that some of them might be familiar; for example, browsing through a scientific journal as generate-and-test. The difficulty is that information acquisition could easily play a central role in handling ill-structured problems but that this depends on the specific content of its methods. If so, then without an explicit description of these methods our hypotheses cannot claim to be relevant. These methods might not formalize easily, so that ill-structured problems would remain solely the domain of human problem solvers. The schemes whereby information is stored away yet seems available almost instantly —as in the recognition of faces or odd relevant facts—are possibly aspects of acquisition methods that may be hard to explicate.

The last three items on the list name things that can be constructed by a problem solver and that affect his subsequent problem-solving behavior. Executive construction occurs because the gross shape of a particular task may have to be reflected in the top-level structure of the procedure that solves it. The induction method, with the three separate induction tasks mentioned, provides an example. Each requires a separate executive structure, and we could not give a single unified procedure to handle them all. Yet each uses the same fundamental method. Relative to our hypotheses, construction seems only to provide additional loci for problem-solving power. This item could become important if it were shown that solutions are not obtained to ill-structured problems without some construction activity.

The extended discussion of the parts of the problem-solving process other than methods, and the ways in which they might either refute or nullify our two hypotheses, stems from a conviction that the major weakness of these hypotheses is the substantial incompleteness of our knowledge about problem solving. They have been created in response to partial evidence, and it seems unlikely that they will emerge unscathed as some of these other parts become better known.

6.2. *Measures of Informational Demands*

Throughout the chapter we have talked as if adding information to a problem statement leads to a decrease in generality and an increase in power. Figure 10.3 is the baldest form of this assertion. At the most general level it seems plausible enough. Here one crudely identifies the number of conditions in the problem statement with the size of the space being searched: as it gets smaller, so the problem solver must grow more powerful. At a finer level of analysis, however, this assertion seems often violated, and in significant ways; for example, a linear programming problem is changed into an integer programming problem by the addition of the constraint that the variables $\{x\}$ range over the positive integers rather than the positive reals. But this makes the problem harder, not easier. Of course, it may be that existing methods of integer programming are simply inefficient compared to what they could be. This position seems tenuous, at best. It is preferable, I think, to take as a major difficulty with these hypotheses that they are built on foundations of sand.

6.3. *Vague Information*

It is a major deficiency of these hypotheses (and of this chapter) that they do not come to grips directly with the nature of vague information. Typically, an ill-structured problem is full of vague information. This might almost be taken as a definition of such a problem, except that the term vague is itself vague.

All extant ideas for dealing with vagueness have one concept in common: they locate the vagueness in the referent of a quite definite (hence un-vague) expression. To have a probability is to have an indefinite event, but a quite definite probability. To have a subset is to have a quite definite expression (the name or description of the subset) which is used to refer to an indefinite, or vague, element. Finally, the constructs of this chapter are similarly definite. The problem solver has a definite problem statement, and all the vagueness exists in the indefinite set of problems that can be identified with the problem statement.[8]

[8] Reitman's proposals, although we have not described them here, have the same definite character [17]. So also does the proposal by Zadeh for "fuzzy" sets [24].

The difficulty with this picture is that, when a human problem solver has a problem he calls ill structured, he does not seem to have definite expressions which refer to his vague information. Rather he has nothing definite at all. As an external observer we might form a definite expression describing the range (or probability distribution) of information that the subject has, but this "meta" expression is not what the subject has that is this information.

It seems to me that the notion of vague information is at the core of the feeling that ill-structured problems are essentially different from well-structured ones. Definite processes must deal with definite things, say, definite expressions. Vague information is not definite in any way. This chapter implies a position on vague information; namely, that there are quite definite expressions in the problem solver (his problem statement). This is a far cry from a theory that explains the different varieties of vague information that a problem solver has. Without such explanations the question of what is an ill-structured problem will remain only half answered.

7. CONCLUSION

The items just discussed—other aspects of problem solving, the measurement of power and generality, and the concept of vagueness—do not exhaust the difficulties or deficiencies of the proposed hypotheses. But they are enough to indicate their highly tentative nature. Almost surely the two hypotheses will be substantially modified and qualified (probably even compromised) with additional knowledge. Even so, there are excellent reasons for putting them forth in bold form.

The general nature of problems and of methods is no longer a quasi-philosophic enterprise, carried on in the relaxed interstices between the development of particular mathematical models and theorems. The development of the computer has initiated the study of information processing, and these highly general schema that we call methods and problem-solving strategies are part of its proper object of study. The nature of generality in problem solving and of ill-structuredness in problems is also part of computer science, and little is known about either. The assertion of some definite hypotheses in crystallized form has the virtue of focusing on these topics as worthy of serious, technical concern.

These two hypotheses (to the extent that they hold true) also have some general implications for the proper study of management science. They say that the field need not be viewed as a collection of isolated mathematical gems, whose application is an art and which is largely excluded from the domain of "nonquantifiable aspects" of management.[9] Proper to management science is the creation of methods general enough to apply to the ill-structured problems of management—taking them on their own terms and dealing with them in all their vagueness—and not demanding more in the way of data than the situations provide. To be sure, these methods will also be weak but not necessarily weaker than is inherent in the ill-structuring of the task.

That management science should deal with the full range of management problems is by no means a new conclusion. In this respect these two hypotheses only reinforce some existing strands of research and application. They do, however, put special emphasis on the extent to which the hard mathematical core of management science should be involved in ill-structured problems. They say such involvement is possible.

BIBLIOGRAPHY

1. G. P. E. Clarkson, *Portfolio Selection: a Simulation of Trust Investment*, Prentice-Hall, Englewood Cliffs, N.J., 1962.
2. R. M. Cyert and J. G. March, *A Behavioral Theory of the Firm*, Prentice-Hall, Englewood Cliffs, N.J., 1963.
3. J. Ellul, *The Technological Society*, Knopf, New York, 1964.
4. T. G. Evans, "A Heuristic Program to Solve Geometric Analogy Problems," *Proc. Spring Joint Computer Conference*, Vol. 25, 1964, pp. 327–338.
5. E. A. Feigenbaum and J. Feldman (eds.), *Computers and Thought*, McGraw-Hill, New York, 1963. (Reprints many of the basic papers.)
6. R. W. Floyd, "Assigning Meanings to Programs," *Proc. Am. Math. Soc., Symposium on Applied Mathematics*, Vol. 19, 1967, pp. 19–32.
7. J. Harris, "Judgmental versus Mathematical Prediction: an Investigation by Analogy of the Clinical versus Statistical Controversy," *Behav. Sci.*, 8, No. 4, 324–335 (Oct. 1963).
8. E. S. Johnson, "An Information-Processing Model of One Kind of Problem Solving," *Psychol. Monog.*, Whole No. 581, 1964.
9. T. Kilburn, R. L. Grimsdale, and F. H. Summer, "Experiments in Machine Learning and Thinking," *Proc. International Conference on Information Processing*, UNESCO, Paris, 1959.
10. B. Kleinmuntz (ed.), *Formal Representation of Human Judgment*, Wiley, New York, 1968.
11. A. A. Kuehn and M. J. Hamburger, "A Heuristic Program for Locating Warehouses," *Mgmt. Sci.*, 9, No. 4, 643–666 (July 1963).

[9] One need only note the extensive calls to arms issued to the industrial operations researcher to consider the total decision context, and not merely that which he can quantify and put into his model, to realize the firm grip of this image.

12. P. E. Meehl, *Clinical vs. Statistical Prediction,* University of Minnesota Press, Minneapolis, Minn., 1954.
13. A. Newell and G. Ernst, "The Search for Generality," in *Proc. IFIP Congress 65* (E. W. Kalenich, ed.), Spartan Books, New York, 1965, pp. 17–24.
14. A. Newell and H. A. Simon, "Programs as Theories of Higher Mental Processes," in *Computers in Biomedical Research* (R. W. Stacey and B. Waxman, eds.), Vol. 2, Academic Press, New York, 1965, pp. 141–172.
15. A. Newell, J. C. Shaw, and H. A. Simon, "Empirical Explorations of the Logic Theory Machine: a Case Study in Heuristic," *Proc. Western Joint Computer Conference,* Feb. 1957, pp. 218–230. Reprinted in Ref. [5].
16. A. Newell, J. C. Shaw, and H. A. Simon, "Elements of a Theory of Human Problem Solving," *Psychol. Rev.,* 65, No. 3, 151–166 (May 1958).
17. W. R. Reitman, "Heuristic Decision Procedures, Open Constraints, and the Structure of Ill-Defined Problems," in *Human Judgments and Optimability* (M. W. Shelly and G. L. Bryan, eds.), Wiley, New York, 1964, pp. 282–315.
18. W. R. Reitman, *Cognition and Thought,* Wiley, New York, 1965.
19. A. L. Samuel, "Some Studies in Machine Learning, Using the Game of Checkers," *IBM J. Res. and Devel.* 3, 221–229 (July 1959). Reprinted in Ref. [5].
20. O. Selfridge, "Pattern Recognition and Modern Computers," and G. P. Dinneen, "Programming Pattern Recognition," *Proc. Western Joint Computer Conference,* 1955, pp. 91–93, 94–100.
21. H. A. Simon and A. Newell, "Heuristic Problem Solving: the Next Advance in Operations Research," *Opns. Res.,* 6, No. 1, 1–10 (Jan.–Feb. 1958).
22. H. A. Simon and K. Kotovsky, "Human Acquisition of Concepts for Sequential Patterns," *Psychol. Rev.,* 70, 534–546 (1963).
23. A. W. Tucker, "Combinatorial Algebra of Matrix Games and Linear Programs," in *Applied Combinatorial Mathematics* (E. F. Beckenbach, ed.), Wiley, New York, 1964, pp. 320–347.
24. L. A. Zadeh, "Fuzzy Sets," *Inform. and Control,* 8, 338–353 (1965).

CHAPTER 2

Reasoning, Problem Solving, and Decision Processes: The Problem Space as a Fundamental Category

A. Newell, Carnegie Mellon University

ABSTRACT

The notion of a problem space is well known in the area of problem solving research, both in cognitive psychology and artificial intelligence. The *Problem Space Hypothesis* is enunciated that the scope of problem spaces is to be extended to all symbolic cognitive activity. The chapter is devoted to explaining the nature of this hypothesis and describing some of its potential implications, with no attempt at a critical marshalling of the evidence pro and con. Two examples are used, one a typical problem solving activity (the Tower of Hanoi) and the other syllogistic reasoning. The latter is an example where the search behavior typical of problem spaces is not clearly in evidence, so it provides a useful area to explore the extension of the concept. A focal issue used in the chapter is the origin of the numerous flow diagrams that serve as theories of how subjects behave in tasks in the psychological laboratory. On the Problem Space Hypothesis these flow diagrams derive from the interaction of the task environment and the problem space.

INTRODUCTION

I am concerned with human goal-oriented cognition: what humans do when they bring to bear what they know to attain some end. I take my title from the session in which I have been invited to give an opening presentation, because it exhibits a particular feature of cognitive psychology's current state that I can use as a starting point.

Substantial areas of psychological study exist in *reasoning* (Falmagne, 1975, Revlin & Mayer, 1978, Wason & Johnson-Laird, 1972); *problem solving*