STEPS TOWARD ARTIFICIAL INTELLIGENCE

by Marvin Minsky

Introduction

A visitor to our planet might be puzzled about the role of computers in our technology. On the one hand, he would read and hear all about wonderful "mechanical brains" baffling their creators with prodigious intellectual performance. And he (or it) would be warned that these machines must be restrained, lest they overwhelm us by might, persuasion, or even by the revelation of truths too terrible to be borne. On the other hand, our visitor would find the machines being denounced, on all sides, for their slavish obedience, unimaginative literal interpretations, and incapacity for innovation or initiative; in short, for their inhuman dullness.

Our visitor might remain puzzled if he set out to find, and judge for himself, these monsters. For he would find only a few machines (mostly "general-purpose" computers, programmed for the moment to behave according to some specification) doing things that might claim any real intellectual status. Some would be proving mathematical theorems of rather undistinguished character. A few machines might be playing certain games, occasionally defeating their designers. Some might be distinguishing between hand-printed letters. Is this enough to justify so much interest, let alone deep concern? I believe that it is; that we are on the threshold of an era that will be strongly influenced, and quite possibly dominated, by intelligent problem-solving machines. But our purpose is not to guess about what the future may bring; it is only to try to describe and explain what seem now to be our first steps toward the construction of "artificial intelligence."

Along with the development of general-purpose computers, the past few years have seen an increase in effort toward the discovery and mechanization of problem-solving processes. Quite a number of papers have appeared describing theories or actual computer programs concerned with game playing, theorem proving, pattern recognition, and other domains which would seem to require some intelligence. The literature does not include any general discussion of the outstanding problems of this field.

In this article, an attempt will be made to separate out, analyze, and find the relations between some of these problems. Analysis will be supported with enough examples from the literature to serve the introductory function of a review article, but there remains much relevant work not described here. This report is highly compressed, and therefore, cannot begin to discuss all these matters in the available space.

There is, of course, no generally accepted theory of "intelligence"; the analysis is our own and may be controversial. We regret that we cannot give full personal acknowledgments here—suffice it to say that we have discussed these matters with almost every one of the cited authors.

It is convenient to divide the problems into five main areas: Search, Pattern Recognition, Learning, Planning, and Induction; these comprise the main divisions of the report. Let us summarize, the entire argument very briefly:

A computer can do, in a sense, only what it is told to do. But even when we do not know exactly how to solve a certain problem, we may program a machine to *Search* through some large space of solution attempts. Unfortunately, when we write a straightforward program for such a search, we usually find the resulting process to be enormously inefficient. With *Pattern Recognition* techniques, efficiency can be greatly improved by restricting the machine to use its methods only on the kind of attempts for which they are appropriate. And with *Learning*, efficiency is further improved by directing Search in accord with earlier experiences. By actually analyzing the situation, using what we call *Planning* methods, the machine may obtain a really fundamental improvement by replacing the originally given Search by a much smaller, more appropriate exploration. Finally, in the section on *Induction*, we consider some rather more global concepts of how one might obtain intelligent machine behavior.

I. The Problem of Search[1]

If, for a given problem, we have a means for checking a proposed solution, then we can solve the problem by testing all possible answers. But this

[1] The adjective "heuristic," as used here and widely in the literature, means *related to improving problem-solving performance;* as a noun it is also used in regard to any method or trick used to improve the efficiency of a problem-solving system. A

always takes much too long to be of practical interest. Any device that can reduce this search may be of value. If we can detect relative improvement, then "hill-climbing" (Sec. I-B) may be feasible, but its use requires some structural knowledge of the search space. And unless this structure meets certain conditions, hill-climbing may do more harm than good.

When we talk of problem-solving in what follows we will usually suppose that all the problems to be solved are initially *well defined* (McCarthy, 1956). By this we mean that with each problem we are given some systematic way to decide when a proposed solution is acceptable. Most of the experimental work discussed here is concerned with such well-defined problems as are met in theorem-proving, or in games with precise rules for play and scoring.

In one sense all such problems are trivial. For if there exists a solution to such a problem, that solution can be found eventually by any blind exhaustive process which searches through all possibilities. And it is usually not difficult to mechanize or program such a search.

But for any problem worthy of the name, the search through all possibilities will be too inefficient for practical use. And on the other hand, systems like chess, or nontrivial parts of mathematics, are too complicated for complete analysis. Without complete analysis, there must always remain some core of search, or "trial and error." So we need to find techniques through which the results of *incomplete analysis* can be used to make the search more efficient. The necessity for this is simply overwhelming: a search of all the paths through the game of checkers involves some 10^{40} move choices (Samuel, 1959*a*), in chess, some 10^{120} (Shannon, in Newman, 1956). If we organized all the particles in our galaxy into some kind of parallel computer operating at the frequency of hard cosmic rays, the latter computation would still take impossibly long; we cannot expect improvements in "hardware" alone to solve all our problems! Certainly we must use whatever we know in advance to guide the trial generator. And we must also be able to make use of results obtained along the way.[2,3]

"heuristic program," to be considered successful, must work well on a variety of problems, and may often be excused if it fails on some. We often find it worthwhile to introduce a heuristic method which happens to cause occasional failures, if there is an over-all improvement in performance. But imperfect methods are not necessarily heuristic, nor vice versa. Hence "heuristic" should not be regarded as opposite to "foolproof"; this has caused some confusion in the literature.

[2] McCarthy (1956) has discussed the enumeration problem from a recursive-function-theory point of view. This incomplete but suggestive paper proposes, among other things, that "the enumeration of partial recursive functions should give an early place to compositions of functions that have already appeared."

I regard this as an important notion, especially in the light of Shannon's results (1949) on two-terminal switching circuits—that the "average" *n*-variable switching

A. Relative Improvement, Hill-climbing, and Heuristic Connections

A problem can hardly come to interest us if we have no background of information about it. We usually have some basis, however flimsy, for detecting *improvement;* some trials will be judged more successful than others. Suppose, for example, that we have a *comparator* which selects as the better, one from any pair of trial outcomes. Now the comparator cannot, alone, serve to make a problem well defined. No goal is defined. But if the comparator-defined relation between trials is "transitive" (*i.e,* if *A dominates B* and *B dominates C* implies that *A dominates C*), then we can at least define "progress," and ask our machine, given a time limit, to do the best it can.

But it is essential to observe that a comparator by itself, however shrewd, cannot alone give any improvement over exhaustive search. The comparator gives us information about partial success, to be sure. But we need also some way of using this information to direct the pattern of search in promising directions; to select new trial points which are in some sense "like," or "similar to," or "in the same direction as" those which have given the best previous results. To do this we need some additional structure on the search space. This structure need not bear much resemblance to the ordinary spatial notion of direction, or that of distance, but it must somehow tie together points which are heuristically related.

We will call such a structure a *heuristic connection.* We introduce this term for informal use only—that is why our definition is itself so informal. But we need it. Many publications have been marred by the misuse, for this purpose, of precise mathematical terms, *e.g., metric* and *topological.* The term "connection," with its variety of dictionary meanings, seems just the word to designate a relation without commitment as to the exact nature of the relation.

An important and simple kind of heuristic connection is that defined when a space has coordinates (or parameters) and there is also defined a numerical "success function" *E* which is a reasonably smooth function of the coordinates. Here we can use local optimization or *hill-climbing* methods.

function requires about $2^n/n$ contacts. This disaster does not usually strike when we construct "interesting" large machines, presumably because they are based on composition of functions already found useful. One should not overlook the pioneering paper of Newell (1955), and Samuel's discussion of the minimaxing process in (1959*a*).

[3] In 1952 and especially in 1956, Ashby has an excellent discussion of the search problem. (However, I am not convinced of the usefulness of his notion of "ultrastability," which seems to be little more than the property of a machine to search until something stops it.)

B. Hill-climbing

Suppose that we are given a black-box machine with inputs $\lambda_1, \ldots, \lambda_n$ and an output $E(\lambda_1, \ldots, \lambda_n)$. We wish to maximize E by adjusting the input values. But we are not given any mathematical description of the function E; hence we cannot use differentiation or related methods. The obvious approach is to explore locally about a point, finding the direction of steepest ascent. One moves a certain distance in that direction and repeats the process until improvement ceases. If the hill is smooth this may be done, approximately, by estimating the gradient component $\partial E/\partial\lambda_i$ separately for each coordinate λ_i. There are more sophisticated approaches (one may use noise added to each variable, and correlate the output with each input, see Fig. 1), but this is the general idea. It is a fundamental technique, and we see it always in the background of far more complex systems. Heuristically, its great virtue is this: the sampling effort (for determining the direction of the gradient) grows, in a sense, only linearly with the number of parameters. So if we can solve, by such a method, a certain kind of problem involving many parameters, then the addition of more parameters of the same kind ought not cause an inordinate increase in difficulty. We are particularly interested in problem-solving methods which can be so extended to more difficult problems. Alas, most interesting systems which involve combinational operations usually grow exponentially more difficult as we add variables.

A great variety of hill-climbing systems have been studied under the names of "adaptive" or "self-optimizing" servomechanisms.

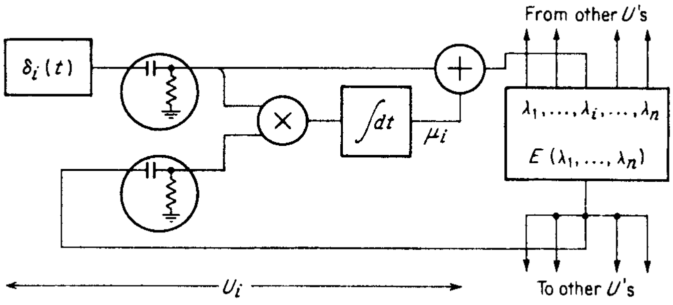
From other U's
$\delta_i(t)$
$\int dt$
μ_i
$\lambda_1, \ldots, \lambda_i, \ldots, \lambda_n$
$E(\lambda_1, \ldots, \lambda_n)$
U_i
To other U's

Figure 1. "Multiple simultaneous optimizers" search for a (local) maximum value of some function $E(\lambda_1, \ldots, \lambda_n)$ of several parameters. Each unit U_i independently "jitters" its parameter λ_i, perhaps randomly, by adding a variation $\delta_i(t)$ to a current mean value μ_i. The changes in the quantities δ_i and E are correlated, and the result is used to (slowly) change μ_i. The filters are to move d-c components. This simultaneous technique, really a form of coherent detection, usually has an advantage over methods dealing separately and sequentially with each parameter. [Cf. the discussion of "informative feedback" in Wiener (1948, pp. 133ff.).]

C. Troubles with Hill-climbing

Obviously, the gradient-following hill-climber would be trapped if it should reach a *local peak* which is not a true or satisfactory optimum. It must then be forced to try larger steps or changes.

It is often supposed that this false-peak problem is the chief obstacle to machine learning by this method. This certainly can be troublesome. But for really difficult problems, it seems to us that usually the more fundamental problem lies in finding any significant peak at all. Unfortunately the known E functions for difficult problems often exhibit what we have called (Minsky and Selfridge, 1960) the *"Mesa Phenomenon"* in which a small change in a parameter usually leads to either no change in performance or to a large change in performance. The space is thus composed primarily of flat regions or "mesas." Any tendency of the trial generator to make small steps then results in much aimless wandering without compensating information gains. A profitable search in such a space requires steps so large that hill-climbing is essentially ruled out. The problem-solver must find other methods; hill-climbing might still be feasible with a different heuristic connection.

Certainly, in our own intellectual behavior we rarely solve a tricky problem by a steady climb toward success. I doubt that in any one simple mechanism, *e.g.*, hill-climbing, will we find the means to build an efficient and general problem-solving machine. Probably, an intelligent machine will require a variety of different mechanisms. These will be arranged in hierarchies, and in even more complex, perhaps recursive, structures. And perhaps what amounts to straightforward hill-climbing on one level may sometimes appear (on a lower level) as the sudden jumps of "insight."

II. The Problem of Pattern Recognition

In order not to try all possibilities, a resourceful machine must classify problem situations into categories associated with the domains of effectiveness of the machine's different methods. These pattern-recognition methods must extract the heuristically significant features of the objects in question. The simplest methods simply match the objects against standards or prototypes. More powerful "property-list" methods subject each object to a sequence of tests, each detecting some *property* of heuristic importance. These properties have to be invariant under commonly encountered forms of distortion. Two important problems arise here—inventing new useful properties, and combining many properties to form a recognition system. For complex problems, such methods will have to be augmented by facilities for subdividing complex objects and describing the complex relations between their parts.

Any powerful heuristic program is bound to contain a variety of different methods and techniques. At each step of the problem-solving process the machine will have to decide what aspect of the problem to work on, and then which method to use. A choice must be made, for we usually cannot afford to try all the possibilities. In order to deal with a goal or a problem, that is, to choose an appropriate method, we have to recognize what kind of thing it is. Thus the need to choose among actions compels us to provide the machine with classification techniques, or means of evolving them. It is of overwhelming importance that the machine have classification techniques which are realistic. But "realistic" can be defined only with respect to the environments to be encountered by the machine, and with respect to the methods available to it. Distinctions which cannot be exploited are not worth recognizing. And methods are usually worthless without classification schemes which can help decide when they are applicable.

A. Teleological Requirements of Classification

The useful classifications are those which match the goals and methods of the machine. The objects grouped together in the classifications should have something of heuristic value in common; they should be "similar" in a useful sense; they should depend on relevant or essential features. We should not be surprised, then, to find ourselves using inverse or teleological expressions to define the classes. We really do want to have a grip on "the class of objects which can be transformed into a result of form Y," that is, the class of objects which will satisfy some goal. One should be wary of the familiar injunction against using teleological language in science. While it is true that talking of goals in some contexts may dispose us towards certain kinds of animistic explanations, this need not be a bad thing in the field of problem-solving; it is hard to see how one can solve problems without thoughts of purposes. The real difficulty with teleological definitions is technical, not philosophical, and arises when they have to be used and not just mentioned. One obviously cannot afford to use for classification a method which actually requires waiting for some remote outcome, if one needs the classification precisely for deciding whether to try out that method. So, in practice, the ideal teleological definitions often have to be replaced by practical approximations, usually with some risk of error; that is, the definitions have to be made *heuristically effective,* or economically usable. This is of great importance. (We can think of "heuristic effectiveness" as contrasted to the ordinary mathematical notion of "effectiveness" which distinguishes those definitions which can be realized at all by machine, regardless of efficiency.)

B. Patterns and Descriptions

It is usually necessary to have ways of assigning *names*—symbolic expressions—to the defined classes. The structure of the names will have a crucial influence on the mental world of the machine, for it determines what kinds of things can be conveniently thought about. There are a variety of ways to assign names. The simplest schemes use what we will call *conventional* (or *proper*) names; here, arbitrary symbols are assigned to classes. But we will also want to use complex *descriptions* or *computed names;* these are constructed for classes by processes which *depend on the class definitions.* To be useful, these should reflect some of the structure of the things they designate, abstracted in a manner relevant to the problem area. The notion of description merges smoothly into the more complex notion of *model;* as we think of it, a model is a sort of active description. It is a thing whose form reflects some of the structure of the thing represented, but which also has some of the character of a working machine.

In Sec. III we will consider "learning" systems. The behavior of those systems can be made to change in reasonable ways depending on what happened to them in the past. But by themselves, the simple learning systems are useful only in recurrent situations; they cannot cope with any significant novelty. Nontrivial performance is obtained only when learning systems are supplemented with classification or pattern-recognition methods of some inductive ability. For the variety of objects encountered in a nontrivial search is so enormous that we cannot depend on recurrence, and the mere accumulation of records of past experience can have only limited value. Pattern Recognition, by providing a heuristic connection which links the old to the new, can make learning broadly useful.

What is a "pattern"? We often use the term teleologically to mean a set of objects which can in some (useful) way be treated alike. For each problem area we must ask, "What patterns would be useful for a machine working on such problems?"

The problems of *visual* pattern recognition have received much attention in recent years and most of our examples are from this area.

C. Prototype-derived Patterns

The problem of reading *printed* characters is a clearcut instance of a situation in which the classification is based ultimately on a fixed set of "prototypes"—*e.g.,* the dies from which the type font was made. The individual marks on the printed page may show the results of many distortions. Some distortions are rather systematic: change in size, position, orientation. Some are of the nature of noise: blurring, grain, low contrast, etc.

If the noise is not too severe, we may be able to manage the identification by what we call a *normalization and template-matching* process. We first remove the differences related to size and position—that is, we *normalize* the input figure. One may do this, for example, by constructing a similar figure inscribed in a certain fixed triangle (see Fig. 2); or one may transform the figure to obtain a certain fixed center of gravity and a unit second central moment. [There is an additional problem with rotational

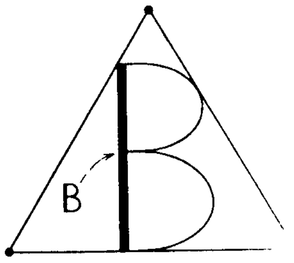

Figure 2. A simple normalization technique. If an object is expanded uniformly, without rotation, until it touches all three sides of a triangle, the resulting figure will be unique, and pattern recognition can proceed without concern about relative size and position.

equivalence where it is not easy to avoid all ambiguities. One does not want to equate "6" and "9." For that matter, one does not want to equate (0,*o*), or (*X*,x) or the *o*'s in x_o and x^o, so that there may be context dependency involved.] Once normalized, the unknown figure can be compared with *templates* for the prototypes and, by means of some measure of *matching,* choose the best fitting template. Each "matching criterion" will be sensitive to particular forms of noise and distortion, and so will each normalization procedure. The inscribing or boxing method may be sensitive to small specks, while the moment method will be especially sensitive to smearing, at least for thin-line figures, etc. The choice of a matching criterion must depend on the kinds of noise and transformations commonly encountered. Still, for many problems we may get acceptable results by using straightforward correlation methods.

When the class of equivalence transformations is very large, *e.g.*, when local stretching and distortion are present, there will be difficulty in finding a uniform normalization method. Instead, one may have to consider a process of adjusting locally for best fit to the template. (While measuring the matching, one could "jitter" the figure locally; if an improvement were found the process could be repeated using a slightly different change, etc.) There is usually no practical possibility of applying to the figure *all* of the admissible transformations. And to recognize the *topological* equivalence of pairs such as those in Fig. 3 is likely beyond any practical kind of iterative local-improvement or hill-climbing matching procedure. (Such recognitions can be mechanized, though, by methods which follow lines, detect vertices, and build up a *description* in the form, say, of a vertex-connection table.)

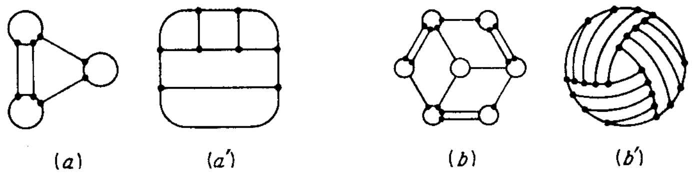

Figure 3. The figures *A, A'* and *B, B'* are topologically equivalent pairs. Lengths have been distorted in an arbitrary manner, but the connectivity relations between corresponding points have been preserved. In Sherman (1959) and Haller (1959) we find computer programs which can deal with such equivalences.

The template-matching scheme, with its normalization and direct comparison and matching criterion, is just too limited in conception to be of much use in more difficult problems. If the transformation set is large, normalization, or "fitting," may be impractical, especially if there is no adequate heuristic connection on the space of transformations. Furthermore, for each defined pattern, the system has to be presented with a prototype. But if one has in mind a fairly abstract class, one may simply be unable to represent its essential features with one or a very few concrete examples. How could one represent with a single prototype the class of figures which have an even number of disconnected parts? Clearly, the template system has negligible descriptive power. The property-list system frees us from some of these limitations.

D. Property Lists and "Characters"

We define a *property* to be a two-valued function which divides figures into two classes; a figure is said to have or not have the property according to whether the function's value is 1 or 0. Given a number N of distinction properties, we could define as many as 2^n subclasses by their set intersections and, hence, as many as 2^{2^n} *patterns* by combining the properties with AND's and OR's. Thus, if we have three properties, *rectilinear, connected,* and *cyclic,* there are eight subclasses (and 256 patterns) defined by their intersections (see Fig. 4).

If the given properties are placed in a fixed order then we can represent any of these elementary regions by a vector, or string of digits. The vector so assigned to each figure will be called the *Character* of that figure (with respect to the sequence of properties in question). [In "Some Aspects of Heuristic Programming and Artificial Intelligence" (1959*a*), we use the

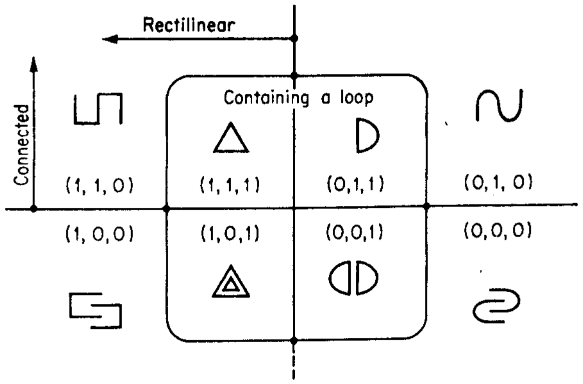

Figure 4. The eight regions represent all the possible configurations of values of the three properties "rectilinear," "connected," "containing a loop." Each region contains a representative figure, and its associated binary "Character" sequence.

term *characteristic* for a property without restriction to 2 values.] Thus a square has the Character (1,1,1) and a circle the Character (0,1,1) for the given sequence of properties.

For many problems one can use such Characters as names for categories and as primitive elements with which to define an adequate set of patterns. Characters are more than conventional names. They are instead very rudimentary forms of description (having the form of the simplest symbolic expression—the *list*) whose structure provides some information about the designated classes. This is a step, albeit a small one, beyond the template method; the Characters are not simple instances of the patterns, and the properties may themselves be very abstract. Finding a good set of properties is the major concern of many heuristic programs.

E. Invariant Properties

One of the prime requirements of a good property is that it be invariant under the commonly encountered equivalence transformations. Thus for visual Pattern Recognition we would usually want the object identification to be independent of uniform changes in size and position. In their pioneering paper Pitts and McCulloch (1947) describe a general technique for forming invariant properties from noninvariant ones, assuming that the transformation space has a certain (group) structure. The idea behind their mathematical argument is this: suppose that we have a function P of figures, and suppose that for a given figure F we define $[F] = \{F_1, F_2, \ldots\}$ to be the set of all figures equivalent to F under the given set of transformations; further, define $P[F]$ to be the set $\{P(F_1), P(F_2), \ldots\}$ of values of P on those figures. Finally, define $P^*[F]$ to be AVERAGE ($P[F]$). Then we have a new property P^* whose values are independent of the selection of F from an equivalence class defined by the transformations. We have to be sure that when different representatives are chosen from a class the collection $[F]$ will always be the same in each case. In the case of continuous transformation spaces, there will have to be a *measure* or the equivalent associated with the set $[F]$ with respect to which the operation AVERAGE is defined, say, as an integration.[4]

This method is proposed (Pitts and McCulloch, 1947) as a neurophysiological model for pitch-invariant hearing and size-invariant visual

[4] In the case studied in Pitts and McCulloch (1947) the transformation space is a *group* with a uniquely defined measure: the set $[F]$ can be computed without repetitions by scanning through the application of all the transforms T_α to the given figure so that the invariant property can be defined by

$$P^*(F) = \int_{\alpha \in G} P(T_\alpha(F))\, d\mu$$

where G is the group and μ the measure. By substituting $T_\beta(F)$ for F in this, one can see that the result is independent of choice of β since we obtain the same integral over $G\beta^{-1} = G$.

recognition (supplemented with visual centering mechanisms). This model is discussed also by Wiener.[5] Practical application is probably limited to one-dimensional groups and analog scanning devices.

In much recent work this problem is avoided by using properties already invariant under these transformations. Thus a property might count the number of connected components in a picture—this is invariant under size and position. Or a property may count the number of vertical lines in a picture—this is invariant under size and position (but not rotation).

F. Generating Properties

The problem of generating useful properties has been discussed by Selfridge (1955); we shall summarize his approach. The machine is given, at the start, a few basic transformations $A_1, \ldots, A_n$, each of which transforms, in some significant way, each figure into another figure. A_1 might, for example, remove all points *not on a boundary* of a solid region; A_2 might leave only *vertex* points; A_3 might *fill up hollow regions*, etc. (see Fig. 5). Each sequence $A_{i1}A_{i2} \ldots A_{ik}$ of these forms a new transformation, so that there is available an infinite variety. We provide the machine also with one or more "terminal" operations which convert a picture into a number, so that any sequence of the elementary transformations, followed by a terminal operation, defines a property. [Dineen (1955) describes how these processes were programmed in a digital computer.] We can start with a few short sequences, perhaps chosen randomly. Selfridge describes how the machine might learn new useful properties.

> *We now feed the machine A's and O's telling the machine each time which letter it is. Beside each sequence under the two letters, the machine builds up distribution functions from the results of applying the sequences to the image. Now, since the sequences were chosen completely randomly, it may well be that most of the sequences have very flat distribution functions; that is, they* [*provide*] *no information,*

[5] See pp. 160ff. of Wiener (1948).

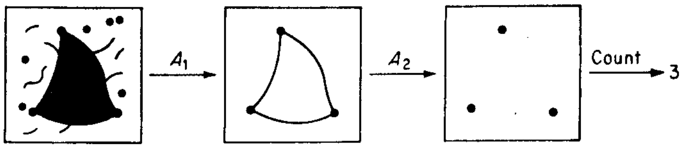

Figure 5. An arbitrary sequence of picture transformations, followed by a numerical-valued function, can be used as a *property* function for pictures. A_1 removes all points which are not at the edge of a solid region. A_2 leaves only vertex points—at which an arc suddenly changes direction. The function C simply counts the number of points remaining in the picture. All remarks in the text could be generalized to apply to properties like A_1A_2C, which can have more than two values.

and the sequences are therefore [by definition] not significant. Let it discard these and pick some others. Sooner or later, however, some sequences will prove significant; that is, their distribution functions will peak up somewhere. What the machine does now is to build up new sequences like the significant ones. This is the important point. If it merely chose sequences at random it might take a very long while indeed to find the best sequences. But with some successful sequences, or partly successful ones, to guide it, we hope that the process will be much quicker. The crucial question remains: how do we build up sequences "like" other sequences, but not identical? As of now we think we shall merely build sequences from the transition frequencies of the significant sequences. We shall build up a matrix of transition frequencies from the significant ones, and use those as transition probabilities with which to choose new sequences.

We do not claim that this method is necessarily a very good way of choosing sequences—only that it should do better than not using at all the knowledge of what kind of sequences has worked. It has seemed to us that this is the crucial point of learning.[6]

It would indeed be remarkable if this failed to yield properties more useful than would be obtained from completely random sequence selection. The generating problem is discussed further in Minsky (1956*a*). Newell, Shaw, and Simon (1960*b*) describe more deliberate, less statistical, techniques that might be used to discover sets of properties appropriate to a given problem area. One may think of the Selfridge proposal as a system which uses a finite-state language to describe its properties. Solomonoff (1957, 1960) proposes some techniques for discovering common features of a set of expressions, *e.g.*, of the descriptions of those properties of already established utility; the methods can then be applied to generate new properties with the same common features. I consider the lines of attack in Selfridge (1955), Newell, Shaw and Simon (1960*a*), and Solomonoff (1960, 1958), although still incomplete, to be of the greatest importance.

G. Combining Properties

One cannot expect easily to find a *small* set of properties which will be just right for a problem area. It is usually much easier to find a large set of properties each of which provides a little useful information. Then one is faced with the problem of finding a way to combine them to make the desired distinctions. The simplest method is to choose, for each class, a typical character (a particular sequence of property values) and then to use some matching procedure, *e.g.*, counting the numbers of agreements and disagreements, to compare an unknown with these chosen "Character

[6] See p. 93 of Selfridge (1955).

prototypes." The linear weighting scheme described just below is a slight generalization on this. Such methods treat the properties as more or less independent evidence for and against propositions; more general procedures (about which we have yet little practical information) must account also for nonlinear relations between properties, *i.e.*, must contain weighting terms for joint subsets of property values.

1. "BAYES NETS" FOR COMBINING INDEPENDENT PROPERTIES

We consider a single experiment in which an object is placed in front of a property-list machine. Each property E_i will have a value, 0 or 1. Suppose that there has been defined some set of "object classes" F_j, and that we want to use the outcome of this experiment to decide in which of these classes the object belongs.

Assume that the situation is basically probabilistic, and that we know the probability p_{ij} that, if the object is in class F_j then the ith property E_i will have value 1. Assume further that these properties are independent; that is, even given F_j, knowledge of the value of E_i tells us nothing more about the value of a different E_k in the same experiment. (This is a strong condition—see below.) Let ϕ_j be the absolute probability that an object is in class F_j. Finally, for this experiment define V to be the particular set of i's for which the E_i's are 1. Then this V represents the Character of the object. From the definition of conditional probability, we have

$$\Pr(F_j,V) = \Pr(V) \cdot \Pr(F_j|V) = \Pr(F_j) \cdot \Pr(V|F_j)$$

Given the Character V, we want to guess which F_j has occurred (with the least chance of being wrong—the so-called *maximum likelihood* estimate); that is, for which j is $\Pr(F_j|V)$ the largest? Since in the above $\Pr(V)$ does not depend on j, we have only to calcuate for which j is

$$\Pr(F_j) \cdot \Pr(V|F_j) = \phi_j \Pr(V|F_j)$$

the largest. Hence, by our independence hypothesis, we have to maximize

$$\phi_j \cdot \prod_{i\in V} p_{ij} \cdot \prod_{i\in \bar{V}} q_{ij} = \phi_j \prod_{i\in V} \frac{p_{ij}}{q_{ij}} \cdot \prod_{\text{all } i} q_{ij} \tag{1}$$

These "maximum-likelihood" decisions can be made (Fig. 6) by a simple network device.[7]

[7] At the cost of an additional network layer, we may also account for the possible cost g_{jk} that would be incurred if we were to assign to F_k a figure really in class F_j; in this case the minimum cost decision is given by the k for which

$$\sum_j g_{jk}\phi_j \prod_{i\in V} p_{ij} \prod_{i\in \bar{V}} q_{ij}$$

is the least. $\bar{V}$ is the complement set to V. q_{ij} is $(1 - p_{ij})$.

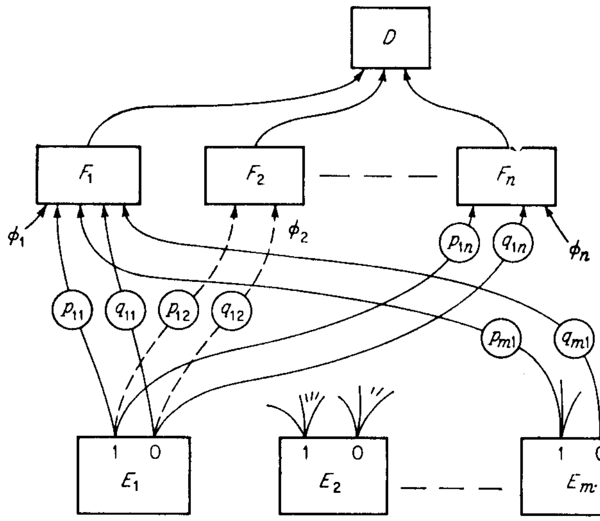

Figure 6. "Net" model for maximum-likelihood decisions based on linear weightings of property values. The input data are examined by each "property filter" E_i. Each E_i has "0" and "1" output channels, one of which is excited by each input. These outputs are weighted by the corresponding p_{ij}'s, as shown in the text. The resulting signals are multiplied in the F_j units, each of which "collects evidence" for a particular figure class. [We could have used here log (p_{ij}), and *added* at the F_j units.] The final decision is made by the topmost unit D, who merely chooses that F_j with the largest score. Note that the logarithm of the coefficient p_{ij}/q_{ij} in the second expression of (1) can be construed as the "weight of the evidence" of E_i in favor of F_j. [See also Papert (1961) and Rosenblatt (1958).]

These nets resemble the general schematic diagrams proposed in the "Pandemonium" model of Selfridge (1959) (see his fig. 3). It is proposed there that some intellectual processes might be carried out by a hierarchy of simultaneously functioning submachines suggestively called "demons." Each unit is set to detect certain patterns in the activity of others and the output of each unit announces the degree of confidence of that unit that it sees what it is looking for. Our E_i units are Selfridge's "data demons." Our units F_j are his "cognitive demons"; each collects from the abstracted data evidence for a specific proposition. The topmost "decision demon" D responds to that one in the multitude below it whose shriek is the loudest.[8]

It is quite easy to add to this "Bayes network model" a mechanism which will enable it to *learn* the optimal connection weightings. Imagine that, after each event, the machine is told which F_j has occurred; we could implement this by sending back a signal along the connections leading to that F_j unit. Suppose that the connection for p_{ij} (or q_{ij}) contains a two-terminal device (or "synapse") which stores a number w_{ij}. Whenever the joint event $(F_j, E_i = 1)$ occurs, we modify w_{ij} by replacing it by

[8] See also the report in Selfridge and Neisser (1960).

$(w_{ij} + 1)\theta$, where θ is a factor slightly less than unity. And when the joint event $(F_j, E_i = 0)$ occurs, we decrement w_{ij} by replacing it with $(w_{ij})\theta$. It is not difficult to show that the expected values of the w_{ij}'s will become proportional to the p_{ij}'s [and, in fact, approach $p_{ij}[\theta/(1 - \theta)]$. Hence, the machine tends to learn the optimal weighting on the basis of experience. (One must put in a similar mechanism for estimating the ϕ_j's.) The variance of the normalized weight $w_{ij}[(1 - \theta)/\theta]$ approaches $[(1 - \theta)/(1 + \theta]p_{ij}q_{ij}$. Thus a small value for θ means rapid learning but is associated with a large variance, hence, with low reliability. Choosing θ close to unity means slow, but reliable, learning. θ is really a sort of memory decay constant, and its choice must be determined by the noise and stability of the environment—much noise requires long averaging times, while a changing environment requires fast adaptation. The two requirements are, of course, incompatible and the decision has to be based on an economic compromise.[9]

2. POSSIBILITIES OF USING RANDOM NETS FOR BAYES DECISIONS

The nets of Fig. 6 are very orderly in structure. Is all this structure necessary? Certainly if there were a great many properties, *each of which provided very little marginal information,* some of them would not be missed. Then one might expect good results with a mere sampling of all the possible connection paths w_{ij}. And one might thus, *in this special situation,* use a random connection net.

The two-layer nets here resemble those of the "Perceptron" proposal of Rosenblatt (1958). In the latter, there is an additional level of connections coming directly from randomly selected points of a "retina." Here the properties, the devices which abstract the visual input data, are simple functions which add some inputs, subtract others, and detect whether the result exceeds a threshold. Equation (1), we think, illustrates what is of value in this scheme. It does seem clear that a maximum-likelihood type of analysis of the output of the property functions can be handled by such nets. But these nets, with their simple, randomly generated, connections can probably never achieve recognition of such patterns as "the class of figures having two separated parts," and they cannot even achieve the effect of template recognition without size and position normalization (unless sample figures have been presented previously in essentially all sizes and positions). For the chances are extremely small of finding, by random methods, enough properties usefully correlated with patterns appreciably more abstract than those of the prototype-derived kind. And these networks can really only separate out (by weighting) information in the individual input properties; they cannot extract further information present in nonadditive form. The "Perceptron" class of machines have facilities neither for obtaining better-than-chance properties nor for assembling

[9] See also Minsky and Selfridge (1960), and Papert (1961).

better-than-additive combinations of those it gets from random construction.[10]

For recognizing *normalized* printed or hand-printed characters, single-point properties do surprisingly well (Highleyman and Kamentsky, 1960); this amounts to just "averaging" many samples. Bledsoe and Browning (1959) claim good results with point-pair properties. Roberts (1960) describes a series of experiments in this general area. Doyle (1959) without normalization but with quite sophisticated properties obtains excellent results; his properties are already substantially size- and position-invariant. A general review of Doyle's work and other pattern-recognition experiments will be found in Selfridge and Neisser (1960).

For the complex discrimination, *e.g.*, between one and two connected objects, the property problem is very serious, especially for long wiggly objects such as are handled by Kirsch (1957). Here some kind of recursive processing is required and combinations of simple properties would almost certainly fail even with large nets and long training.

We should not leave the discussion of some decision net models without noting their important limitations. The hypothesis that, for given j, the p_{ij} represent independent events, is a very strong condition indeed. Without this hypothesis we could still construct maximum-likelihood nets, but we would need an additional layer of cells to represent all of the joint events V; that is, we would need to know all the $\Pr(F_j|V)$. This gives a general (but trivial) solution, but requires 2^n cells for n properties, which is completely impractical for large systems. What is required is a system which computes some sampling of all the joint conditional probabilities, and uses these to estimate others when needed. The work of Uttley (1956, 1959) bears on this problem, but his proposed and experimental devices do not yet clearly show how to avoid exponential growth.[11]

H. Articulation and Attention—Limitations of the Property-list Method

Because of its fixed size, the property-list scheme is limited (for any given set of properties) in the detail of the distinctions it can make. Its ability to deal with a compound scene containing several objects is critically weak, and its direct extensions are unwieldy and unnatural. If a machine can recognize a chair and a table, it surely should be able to tell us that "there is a chair and a table." To an extent, we can invent properties which allow some capacity for superposition of object Characters.[12] But there is no way to escape the information limit.

[10] See also Roberts (1960), Papert (1961), and Hawkins (1958). We can find nothing resembling an analysis [see (1) above] in Rosenblatt (1958) or his subsequent publications.

[11] See also Papert (1961).

[12] Cf. Mooers' technique of Zatocoding (1956*a*, 1956*b*).

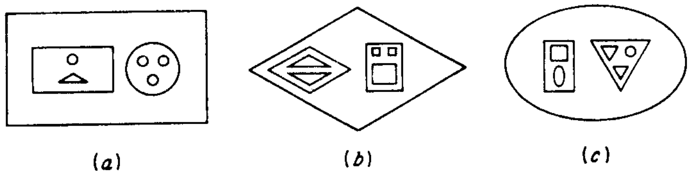

Figure 7. The picture (*a*) is first described verbally in the text. Then, by introducing notation for the relations "inside of," "to the left of" and "above," we construct a symbolic description. Such descriptions can be formed and manipulated by machines. By abstracting out of the complex relation between the parts of the figure we can use the same formula to describe the related pictures (*b*) and (*c*), changing only the list of primitive parts. It is up to the programmer to decide at just what level of complexity a part of a picture should be considered "primitive"; this will depend on what the description is to be used for. We could further divide the drawings into vertices, lines, and arcs. Obviously, for some applications the relations would need more metrical information, *e.g.*, specification of lengths or angles.

What is required is clearly (1) a *list* (*of whatever length is necessary*) of the primitive objects in the scene and (2) a statement about the relations among them. Thus we say of Fig. 7*a*, "A rectangle (1) contains two subfigures disposed horizontally. The part on the left is a rectangle (2) which contains two subfigures disposed vertically; the upper a circle (3) and the lower a triangle (4). The part on the right . . . etc." Such a description entails an ability to separate or "articulate" the scene into parts. (Note that in this example the articulation is essentially *recursive;* the figure is first divided into two parts; then each part is described using the same machinery.) We can formalize this kind of description in an expression language whose fundamental grammatical form is a pair (R,L) whose first member R names a *relation* and whose second member L is an *ordered list* $(x_1, x_2, \ldots, x_n)$ of the objects or subfigures which bear that relation to one another. We obtain the required flexibility by allowing the members of the list L to contain not only the names of "elementary" figures but also "subexpressions" of the form (R,L) designating complex subfigures. Then our scene above may be described by the expression

$$[\odot, (\square, (\rightarrow, \{(\odot, (\square, (\downarrow, (\bigcirc, \triangle)))), (\odot, (\bigcirc, (\nabla, (\bigcirc, \quad \bigcirc, \bigcirc))))\})))]$$

where $(\odot, (x,y))$ means that y is contained in x; $(\rightarrow,(x,y))$ means that y is to the right of x; $(\downarrow,(x,y))$ means that y is below x, and $(\triangle,(x,y,z))$ means that y is to the right of x and z is underneath and between them. The symbols $\square$, $\bigcirc$, and $\triangle$ represent the indicated kinds of primitive geometric objects. This expression-pair description language may be regarded as a simple kind of "list-structure" language. Powerful computer techniques have been developed, originally by Newell, Shaw and Simon, for manipulating symbolic expressions in such languages for purposes of heuristic programming. (See the remarks at the end of Sec. IV. If some of the members of a list are themselves lists, they must be surrounded by exterior parentheses, and this accounts for the accumulation of parentheses.)

It may be desirable to construct descriptions in which the complex relation is extracted, *e.g.*, so that we have an expression of the form FG where F is an expression which at once denotes the composite relation between all the primitive parts listed in G. A complication arises in connection with the "binding" of variables, *i.e.*, in specifying the manner in which the elements of G participate in the relation F. This can be handled in general by the "λ" notation (McCarthy, 1960) but here we can just use integers to order the variables.

For the given example, we could describe the relational part F by an expression

$$\odot(1,\rightarrow(\odot(2,\downarrow(3,4)),\odot(5,\nabla(6,7,8))))$$

in which we now use a "functional notation"; "$(\odot, (x,y))$" is replaced by "$\odot(x,y)$," etc., making for better readability. To obtain the desired description, this expression has to be applied to an ordered list of primitive objects, which in this case is $(\square,\square,\bigcirc,\triangle,\bigcirc,\bigcirc.\bigcirc,\bigcirc)$. This composite functional form allows us to abstract the composite relation. By changing only the object list we can obtain descriptions also of the objects in Fig. 7*b* and *c*.

The important thing about such "articular" descriptions is that they can be obtained by *repeated application of a fixed set of pattern-recognition techniques*. Thus we can obtain *arbitrarily complex* descriptions from a fixed complexity classification mechanism. The new element required in the mechanism (beside the capacity to manipulate the list structures) is the ability to articulate—to "attend fully" to a selected part of the picture and bring all one's resources to bear on that part. In efficient problem-solving programs, we will not usually complete such a description in a single operation. Instead, the depth or detail of description will be under the control of other processes. These will reach deeper, or look more carefully, only when they have to, *e.g.*, when the presently available description is inadequate for a current goal. The author, together with L. Hodes, is working on pattern-recognition schemes using articular descriptions. By manipulating the formal descriptions we can deal with overlapping and incomplete figures, and several other problems of the "Gestalt" type.

It seems likely that as machines are turned toward more difficult problem areas, *passive* classification systems will become less adequate, and we may have to turn toward schemes which are based more on internally generated hypotheses, perhaps "error-controlled" along the lines proposed by MacKay (1956).

Space requires us to terminate this discussion of pattern-recognition and description. Among the important works not reviewed here should be mentioned those of Bomba (1959) and Grimsdale *et al.* (1959), which involve elements of description, Unger (1959) and Holland (1960) for parallel processing schemes, Hebb (1949) who is concerned with physiological description models, and the work of the Gestalt psychologists, notably Kohler (1947), who have certainly raised, if not solved, a number of important questions. Sherman (1959), Haller (1959) and others have completed programs using line-tracing operations for topological classification. The papers of Selfridge (1955, 1956) have been a major influence on work in this general area.

See also Kirsch *et al.* (1957) for discussion of a number of interesting computer image processing techniques, and see Minot (1959) and Stevens (1957) for reviews of the reading machine and related problems. One should also examine some biological work, *e.g.*, Tinbergen (1951) to see instances in which some discriminations which seem, at first glance very complicated are explained on the basis of a few apparently simple properties arranged in simple decision trees.

III. Learning Systems

In order to solve a new problem, one should first try using methods similar to those that have worked on similar problems. To implement this "basic learning heuristic" one must generalize on past experience, and one way to do this is to use success-reinforced decision models. These learning systems are shown to be averaging devices. Using devices which learn also which events are associated with reinforcement, *i.e.*, reward, we can build more autonomous "secondary reinforcement" systems. In applying such methods to complex problems, one encounters a serious difficulty—in distributing credit for success of a complex strategy among the many decisions that were involved. This problem can be managed by arranging for local reinforcement of partial goals within a hierarchy, and by grading the training sequence of problems to parallel a process of maturation of the machine's resources.

In order to solve a new problem one uses what might be called the basic learning heuristic—first try using methods similar to those which have worked, in the past, on similar problems. We want our machines, too, to benefit from their past experience. Since we cannot expect new situations to be precisely the same as old ones, any useful learning will have to involve generalization techniques. There are too many notions associated with "learning" to justify defining the term precisely. But we may be sure that any useful learning system will have to use records of the past as *evidence for more general propositions;* it must thus entail some commitment or other about "inductive inference." (See Sec. V*B*.) Perhaps the simplest way of generalizing about a set of entities is through constructing a new one which is an "ideal," or rather, a typical member of that set; the usual way to do this is to smooth away variation by some sort of averaging technique. And indeed we find that most of the *simple* learning devices do incorporate some averaging technique—often that of averaging some sort of product, thus obtaining a sort of correlation. We shall discuss this family of devices here, and some more abstract schemes in Sec. V.

A. Reinforcement

A reinforcement process is one in which some aspects of the behavior of a system are caused to become more (or less) prominent in the future as a consequence of the application of a "reinforcement operator" Z. This operator is required to affect only those aspects of behavior for which instances have actually occurred recently.

The analogy is with "reward" or "extinction" (not punishment) in animal behavior. The important thing about this kind of process is that it is "operant" [a term of Skinner (1953)]; the reinforcement operator does not initiate behavior, but merely selects that which the Trainer likes from that which has occurred. Such a system must then contain a device M which generates a variety of behavior (say, in interacting with some environment) and a Trainer who makes critical judgments in applying the available reinforcement operators. (See Fig. 8.)

Let us consider a very simple reinforcement model. Suppose that on each presentation of a stimulus S an animal has to make a choice, *e.g.*, to

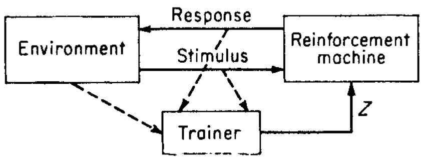

Figure 8. Parts of an "operant reinforcement" learning system. In response to a stimulus from the environment, the machine makes one of several possible responses. It remembers what decisions were made in choosing this response. Shortly thereafter, the Trainer sends to the machine positive or negative reinforcement (reward) signal; this increases or decreases the tendency to make the same decisions in the future. Note that the Trainer need not know how to solve problems, but only how to detect success or failure, or relative improvement; his function is selective. The Trainer might be connected to observe the actual stimulus-response activity, or, in a more interesting kind of system, just some function of the state of the environment.

turn left or right, and that its probability of turning right, at the *n*th trial, is p_n. Suppose that *we* want it to turn right. Whenever it does this we might "reward" it by applying the operator Z_+;

$$p_{n+1} = Z_+(p_n) = \theta p_n + (1 - \theta) \qquad 0 < \theta < 1$$

which moves p a fraction $(1 - \theta)$ of the way toward unity.[13] If we dislike what it does we apply negative reinforcement,

$$p_{n+1} = Z_-(p_n) = \theta p_n$$

moving p the same fraction of the way toward 0. Some theory of such "linear" learning operators, generalized to several stimuli and responses, will be found in Bush and Mosteller (1955). We can show that the learning result is an average weighted by an exponentially-decaying time factor: Let Z_n be ± 1 according to whether the *n*th event is rewarded or extinguished and replace p_n by $c_n = 2p_n - 1$ so that $-1 \leq c_n \leq 1$, as for a correlation coefficient. Then (with $c_0 = 0$) we obtain by induction

$$c_{n+1} = (1 - \theta) \sum_{i=0}^{n} \theta^{n-i} Z_i$$

and since

$$\frac{1}{1 - \theta} \approx \sum_{0}^{n} \theta^{n-i}$$

we can write this as

$$c_{n+1} \approx \frac{\Sigma \theta^{n-i} Z_i}{\Sigma \theta^{n-i}} \qquad (1)$$

If the term Z_i is regarded as a product of (i) how the creature responded and (ii) which kind of reinforcement was given, then c_n is a kind of correlation function (with the decay weighting) of the joint behavior of these quantities. The ordinary, uniformly weighted average has the same general form but with time-dependent θ:

$$c_{n+1} = \left(1 - \frac{1}{N}\right) c_n + \frac{1}{N} Z_n \qquad (2)$$

In (1) we have again the situation described in Sec. II*G*1; a small value of θ gives fast learning, and the possibility of quick adaptation to a changing environment. A near-unity value of θ gives slow learning, but also smooths away uncertainties due to noise. As noted in Sec. II*G*1, the response distribution comes to approximate the probabilities of rewards of the alternative responses. (The importance of this phenomenon has, I think, been overrated; it is certainly not an especially rational strategy. One reasonable alternative is that of computing the numbers p_{ij} as indi-

[13] Properly, the reinforcement functions should depend both on the *p*'s and on the previous reaction—reward should decrease *p* if our animal has just turned to the left. The notation in the literature is also somewhat confusing in this regard.

cated, but actually playing at each trial the "most likely" choice. Except in the presence of a hostile opponent, there is usually no reason to play a "mixed" strategy.[14])

In Samuel's coefficient-optimizing program (1959*b*) [see Sec. III*C*1], there is a most ingenious compromise between the exponential and the uniform averaging methods: the value of N in (2) above begins at 16 and so remains until $n = 16$, then N is 32 until $n = 32$, and so on until $n = 256$. Thereafter N remains fixed at 256. This nicely prevents violent fluctuations in c_n at the start, approaches the uniform weighting for a while, and finally approaches the exponentially weighted correlation, all in a manner that requires very little computation effort! Samuel's program is at present the outstanding example of a game-playing program which matches average human ability, and its success (in real time) is attributed to a wealth of such elegancies, both in heuristics and in programming.

The problem of extinction or "unlearning" is especially critical for complex, hierarchical, learning. For, once a generalization about the past has been made, one is likely to build upon it. Thus, one may come to select certain properties as important and begin to use them in the characterization of experience, perhaps storing one's memories in terms of them. If later it is discovered that some other properties would serve better, then one must face the problem of translating, or abandoning, the records based on the older system. This may be a very high price to pay. One does not easily give up an old way of looking at things, if the better one demands much effort and experience to be useful. Thus the *training sequences* on which our machines will spend their infancies, so to speak, must be chosen very shrewdly to insure that early abstractions will provide a good foundation for later difficult problems.

Incidentally, in spite of the space given here for their exposition, I am not convinced that such "incremental" or "statistical" learning schemes should play a central role in our models. They will certainly continue to appear as components of our programs but, I think, mainly by default. The more intelligent one is, the more often he should be able to learn from an experience something rather definite; *e.g.,* to reject or accept a hypothesis, or to change a goal. (The obvious exception is that of a truly statistical environment in which averaging is inescapable. But the heart of problem-solving is always, we think, the combinatorial part that gives rise to searches, and we should usually be able to regard the complexities caused by "noise" as mere annoyances, however irritating they may be.) In this connection we can refer to the discussion of memory in Miller, Galanter and Pribram (1960).[15] This seems to be the first major work

[14] The question of just how often one should play a strategy different from the estimated optimum, in order to gain information, is an underlying problem in many fields. See, *e.g.*, Shubik (1960).

[15] See especially chap. 10.

in psychology to show the influence of work in the artificial intelligence area, and its programme is generally quite sophisticated.

B. Secondary Reinforcement and Expectation Models

The simple reinforcement system is limited by its dependence on the Trainer. If the Trainer can detect only the *solution* of a problem, then we may encounter "mesa" phenomena which will limit performance on difficult problems. (See Sec. I*C*.) One way to escape this is to have the machine learn to generalize on what the Trainer does. Then, in difficult problems, it may be able to give itself partial reinforcements along the way, *e.g.*, upon the solution of relevant subproblems. The machine in Fig. 9 has some such ability. The new unit *U* is a device that learns which external stimuli are strongly correlated with the various reinforcement signals, and responds to such stimuli by reproducing the corresponding reinforcement signals. (The device *U* is *not* itself a reinforcement learning device; it is more like a "Pavlovian" conditioning device, treating the *Z* signals as "unconditioned" stimuli and the *S* signals as conditioned stimuli.) The heuristic idea is that any signal from the environment which in the past has been well correlated with (say) positive reinforcement is likely to be an indication that something good has just happened. If the training on early problems was such that this is realistic, then the system eventually should be able to detach itself from the Trainer, and become autonomous. If we further permit "chaining" of the "secondary reinforcers," *e.g.*, by admitting the connection shown as a dotted line in Fig. 9, the scheme becomes quite powerful, in principle. There are obvious pitfalls in admitting

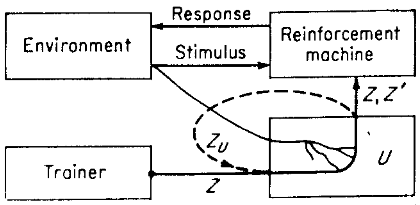

Figure 9. An additional device *U* gives the machine of Fig. 8 the ability to learn which signals from the environment have been associated with reinforcement. The primary reinforcement signals *Z* are routed through *U*. By a Pavlovian conditioning process (not described here), external signals come to produce reinforcement signals like those that have frequently succeeded them in the past. Such signals might be abstract, *e.g.*, verbal encouragement. If the "secondary reinforcement" signals are allowed, in turn, to acquire further external associations (through, *e.g.*, a channel Z_U as shown) the machine might come to be able to handle chains of subproblems. But something must be done to stabilize the system against the positive symbolic feedback loop formed by the path Z_U. The profound difficulty presented by this stabilization problem may be reflected in the fact that, in lower animals, it is very difficult to demonstrate such chaining effects.

such a degree of autonomy; the values of the system may drift to a "non-adaptive" condition.

C. Prediction and Expectation

The evaluation unit U is supposed to acquire an ability to tell whether a situation is good or bad. This evaluation could be applied to *imaginary* situations as well as to real ones. If we could estimate the consequences of a proposed action (without its actual execution), we could use U to evaluate the (estimated) resulting situation. This could help in reducing the effort in search, and we would have in effect a machine with some ability to look ahead, or *plan*. In order to do this we need an additional device P which, given the description of a situation and an action, will predict a description of the likely result. (We will discuss schemes for doing this in Sec. IV*C*.) The device P might be constructed along the lines of a reinforcement learning device. In such a system the required reinforcement signals would have a very attractive character. For the machine must reinforce P positively when the *actual outcome resembles that which was predicted*—accurate expectations are rewarded. If we could further add a premium to reinforcement of those predictions which have a novel aspect, we might expect to discern behavior motivated by a sort of curiosity. In the reinforcement of mechanisms for confirmed novel expectations (or new explanations) we may find the key to simulation of intellectual motivation.[16]

SAMUEL'S PROGRAM FOR CHECKERS

In Samuel's "generalization learning" program for the game of checkers (1959*a*) we find a novel heuristic technique which could be regarded as a simple example of the "expectation reinforcement" notion. Let us review very briefly the situation in playing two-person board games of this kind. As noted by Shannon (1956) such games are in principle finite, and a best strategy can be found by following out all possible continuations—if he goes there I can go there, or there, etc.—and then "backing up" or "minimaxing" from the terminal positions, won, lost, or drawn. But in practice the full exploration of the resulting colossal "move tree" is out of the question. No doubt, some exploration will always be necessary for such games. But the tree must be pruned. We might simply put a limit on depth of exploration—the number of moves and replies. We might also limit the number of alternatives explored from each position—this requires some heuristics for selection of "plausible moves."[17] Now, if the backing-up technique is still to be used (with the incomplete move tree) one has to

[16] See also chap. 6 of Minsky (1954).

[17] See the discussion of Bernstein (1958) and the more extensive review and discussion in the very suggestive paper of Newell, Shaw and Simon (1958*b*).

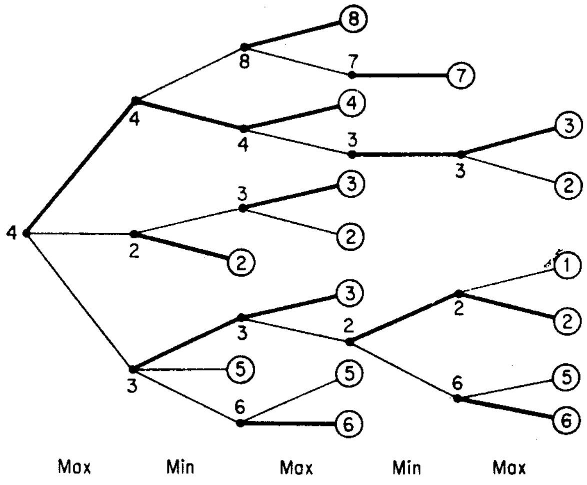

Figure 10. "Backing up" the static evaluations of proposed moves in a game tree. From the vertex at the left, representing the present position in a board game, radiate three branches, representing the *player's* proposed moves. Each of these might be countered by a variety of *opponent* moves, and so on. According to some program, a finite tree is generated. Then the worth to the player of each terminal board position is estimated (see text). If the opponent has the same values, he will choose to minimize the score, while the player will always try to maximize. The heavy lines show how this minimaxing process backs up until a choice is determined for the present position.

The full tree for chess has the order of 10^{120} branches—beyond the reach of any man or computer. There is a fundamental heuristic exchange between the effectiveness of the evaluation function and the extent of the tree. A very weak evaluation (*e.g.*, one which just compares the players' values of pieces) would yield a devastating game if the machine could explore all continuations out to, say, 20 levels. But only 6 levels, roughly within the range of our presently largest computers, would probably not give a brilliant game; less exhaustive strategies, perhaps along the lines of Newell, Shaw, and Simon (1958*b*), would be more profitable.

substitute for the absolute "win, lose, or draw" criterion some other "static" way of evaluating nonterminal positions.[18] (See Fig. 10.) Perhaps the simplest scheme is to use a weighted sum of some selected set of "property" functions of the positions—mobility, advancement, center control, and the like. This is done in Samuel's program, and in most of its predecessors. Associated with this is a multiple-simultaneous-optimizer method for discovering a good coefficient assignment (using the correlation technique noted in Sec. III*A*). But the source of reinforcement signals in

[18] In some problems the backing-up process can be handled in closed analytic form so that one may be able to use such methods as Bellman's "Dynamic Programming" (1957). Freimer (1960) gives some examples for which limited "look-ahead" doesn't work.

Samuel (1959*a*) is novel. One cannot afford to play out one or more entire games for each single learning step. Samuel measures instead *for each move* the difference between what the evaluation function yields *directly* of a position and what it *predicts* on the basis of an extensive continuation exploration, *i.e.*, backing up. The sign of this error, "Delta," is used for reinforcement; thus the system may learn something at *each move.*[19]

D. The Basic Credit-assignment Problem for Complex Reinforcement Learning Systems

In playing a complex game such as chess or checkers, or in writing a computer program, one has a definite success criterion—the game is won or lost. But in the course of play, each ultimate success (or failure) is associated with a vast number of internal decisions. If the run is successful, how can we assign credit for the success among the multitude of decisions? As Newell noted,

> *It is extremely doubtful whether there is enough information in "win, lose, or draw" when referred to the whole play of the game to permit any learning at all over available time scales. . . . For learning to take place, each play of the game must yield much more information. This is . . . achieved by breaking the problem into components. The unit of success is the goal. If a goal is achieved, its subgoals are reinforced; if not they are inhibited. (Actually, what is reinforced is the transformation rule that provided the subgoal.) . . . This also is true of the other kinds of structure: every tactic that is created provides information about the success or failure of tactic search rules; every opponent's action provides information about success or failure of likelihood inferences; and so on. The amount of information relevant to learning increases directly with the number of mechanisms in the chess-playing machine.*[20]

We are in complete agreement with Newell on this approach to the problem.[21]

It is my impression that many workers in the area of "self-organizing" systems and "random neural nets" do not feel the urgency of this prob-

[19] It should be noted that Samuel (1959*a*) describes also a rather successful checker-playing program based on recording and retrieving information about positions encountered in the past, a less abstract way of exploiting past experience. Samuel's work is notable in the variety of experiments that were performed, with and without various heuristics. This gives an unusual opportunity to really find out how different heuristic methods compare. More workers should choose (other things being equal) problems for which such variations are practicable.

[20] See p. 108 of Newell (1955).

[21] See also the discussion in Samuel (p. 22, 1959*a*) on assigning credit for a change in "Delta."

lem. Suppose that one million decisions are involved in a complex task (such as winning a chess game). Could we assign to each decision element one-millionth of the credit for the completed task? In certain special situations we can do just this—*e.g.*, in the machines of Rosenblatt (1958), Roberts (1960), and Farley and Clark (1954), etc., where the connections being reinforced are to a sufficient degree independent. But the problem-solving ability is correspondingly weak.

For more complex problems, with decisions in hierarchies (rather than summed on the same level) and with increments small enough to assure probable convergence, the running times would become fantastic. For complex problems we will have to define "success" in some rich local sense. Some of the difficulty may be evaded by using carefully graded "training sequences" as described in the following section.

FRIEDBERG'S PROGRAM-WRITING PROGRAM

An important example of comparative failure in this credit-assignment matter is provided by the program of Friedberg (1958, 1959) to solve program-writing problems. The problem here is to write programs for a (simulated) very simple digital computer. A simple problem is assigned, *e.g.*, "compute the AND of two bits in storage and put the result in an assigned location." A generating device produces a random (64-instruction) program. The program is run and its success or failure is noted. The success information is used to reinforce *individual instructions* (in fixed locations) so that each success tends to increase the chance that the instructions of successful programs will appear in later trials. (We lack space for details of how this is done.) Thus the program tries to find "good" instructions, more or less independently, for each location in program memory. The machine did learn to solve some extremely simple problems. But it took of the order of 1000 times longer than pure chance would expect. In part II of Friedberg *et al.* (1959), this failure is discussed, and attributed in part to what we called (Sec. I*C*) the "Mesa phenomena." In changing just one instruction at a time the machine had not taken large enough steps in its search through program space.

The second paper goes on to discuss a sequence of modifications in the program generator and its reinforcement operators. With these, and with some "priming" (starting the machine off on the right track with some useful instructions), the system came to be only a little worse than chance. Friedberg *et al.* (1959) conclude that with these improvements "the generally superior performance of those machines with a success-number reinforcement mechanism over those without does serve to indicate that such a mechanism can provide a basis for constructing a learning machine." I disagree with this conclusion. It seems to me that each of the "improvements" can be interpreted as serving only to increase the step size of the search, that is, the randomness of the mechanism; this helps to avoid the Mesa phenomenon and thus approach chance behavior. But it certainly does not show that the "learning mechanism" is working—one would want at least to see some better-than-chance results before arguing this point. The trouble, it seems, is with credit-assignment. The credit for a working program can only be assigned to functional groups of instructions, *e.g.,* subroutines, and as these operate in hierarchies we should not expect individual instruction reinforcement to work well.[22] It seems surprising that it was not recognized in Friedberg *et al.* (1959) that the doubts raised earlier were probably justified! In the last section of Friedberg *et al.* (1959) we see some real success obtained by breaking the problem into parts and solving them sequentially. (This successful demonstration using division into subproblems does not use any reinforcement mechanism at all.) Some experiments of similar nature are reported in Kilburn, Grimsdale and Sumner (1959).

It is my conviction that no scheme for learning, or for pattern recognition, can have very general utility unless there are provisions for recursive, or at least hierarchical, use of previous results. We cannot expect a learning system to come to handle very hard problems without preparing it with a reasonably graded sequence of problems of growing difficulty. The first problem must be one which can be solved in reasonable time with the initial resources. The next must be capable of solution in reasonable time by using reasonably simple and accessible combinations of methods developed in the first, and so on. The only alternatives to this use of an adequate "training sequence" are (1) advanced resources, given initially, or (2) the fantastic exploratory processes found perhaps only in the history of organic evolution.[23] And even there, if we accept the general view of Darlington (1958) who emphasizes the heuristic aspects of genetic systems, we must have developed early (in, *e.g.,* the phenomena of meiosis and crossing-over) quite highly specialized mechanisms providing for the segregation of groupings related to solutions of subproblems. Recently, much effort has been devoted to the construction of training sequences in connection with programming "teaching machines." Naturally, the psychological literature abounds with theories of how complex behavior is built

[22] See the introduction to Friedberg (1958) for a thoughtful discussion of the plausibility of the scheme.

[23] It should, however, be possible to construct learning mechanisms which can select for themselves reasonably good training sequences (from an always complex environment) by prearranging a relatively slow development (or "maturation") of the system's facilities. This might be done by prearranging that sequence of goals attempted by the primary Trainer match reasonably well, at each stage, the complexity of performance mechanically available to the pattern-recognition and other parts of the system. One might be able to do much of this by simply limiting the depth of hierarchical activitv, perhaps only later permitting limited recursive activity.

up from simpler. In our own area, perhaps the work of Solomonoff (1957), while overly cryptic, shows the most thorough consideration of this dependency on *training sequences.*

IV. Problem-solving and Planning

The solution, by machine, of really complex problems will require a variety of administration facilities. During the course of solving a problem, one becomes involved with a large assembly of interrelated subproblems. From these, at each stage, a very few must be chosen for investigation. This decision must be based on (1) estimates of relative difficulties and (2) estimates of centrality of the different candidates for attention. Following subproblem selection (for which several heuristic methods are proposed), one must choose methods appropriate to the selected problems. But for really difficult problems, even these step-by-step heuristics for reducing search will fail, and the machine must have resources for analyzing the problem structure in the large—in short, for "planning." A number of schemes for planning are discussed, among them the use of models —analogous, semantic, and abstract. Certain abstract models, "Character-Algebras," can be constructed by the machine itself, on the basis of experience or analysis. For concreteness, the discussion begins with a description of a simple but significant system (LT) which encounters some of these problems.

A. The "Logic Theory" Program of Newell, Shaw and Simon

It is not surprising that the testing grounds for early work on mechanical problem-solving have usually been areas of mathematics, or games, in which the rules are defined with absolute clarity. The "Logic Theory" machine of Newell and Simon (1956*a*, 1957*a*), called "LT" below, was a first attempt to prove theorems in logic, by frankly heuristic methods. Although the program was not by human standards a brilliant success (and did not surpass its designers), it stands as a landmark both in heuristic programming and also in the development of modern automatic programming.

The problem domain here is that of discovering proofs in the Russell-Whitehead system for the propositional calculus. That system is given as a set of (five) axioms and (three) rules of inference; the latter specify how certain transformations can be applied to produce new theorems from old theorems and axioms.

The LT program is centered around the idea of "working backward" to find a proof. Given a theorem *T* to be proved, LT searches among the axioms and previously established theorems for one from which *T* can be deduced by a single application of one of three simple "Methods" (which embody the given rules of inference). If one is found, the problem is solved. Or the search might fail completely. But finally, the search may yield one or more "problems" which are usually propositions from which *T* may be deduced directly. If one of these can, in turn, be proved a theorem the main problem will be solved. (The situation is actually slightly more complex.) Each such subproblem is adjoined to the "subproblem list" (after a limited preliminary attempt) and LT works around to it later. The full power of LT, such as it is, can be applied to each subproblem, for LT can use itself as a subroutine in a recursive fashion.

The heuristic technique of working backward yields something of a teleological process, and LT is a forerunner of more complex systems which construct hierarchies of goals and subgoals. Even so, the basic administrative structure of the program is no more than a nested set of searches through lists in memory. We shall first outline this structure and then mention a few heuristics that were used in attempts to improve performance.

```
1. Take the next problem from problem list.
   (If there are no more problems, EXIT with total failure.)
2. Choose the next of the three basic Methods.
   (If no more methods, go to 1.)
3. Choose the next member of the list of axioms and previous theorems.
   (If no more, go to 2.)
   Then apply the Method to the problem, using the chosen theorem
      or axiom.
   If problem is solved, EXIT with complete proof.
   If no result, go to 3.
   If new subproblem arises, go to 4.
4. Try the special (substitution) Method on the subproblem.
   If problem is solved, EXIT with complete proof.
   If no result, put the subproblem at the end of the problem list and
      go to 3.
```

Among the heuristics that were studied were (1) a *similarity test* to reduce the work in step 4 (which includes another search through the theorem list), (2) a *simplicity test* to select apparently easier problems from the problem list, and (3) a *strong nonprovability test* to remove from the problem list expressions which are probably false and hence not provable. In a series of experiments "learning" was used to find which earlier theorems had been most useful and should be given priority in step 3. We cannot review the effects of these changes in detail. Of interest was the balance between the extra cost for administration of certain heuristics and the resultant search reduction; this balance was quite delicate in some cases when computer memory became saturated. The system seemed to be quite sensitive to the training sequence—the order in which problems were given. And some heuristics which gave no significant over-all improvement did nevertheless affect the class of solvable problems. Curiously enough, the general efficiency of LT was not greatly improved by any or all of these devices. But all this practical experience is reflected in the design of the much more sophisticated "GPS" system described briefly in Sec. IV*D*2.

Wang (1960) has criticized the LT project on the grounds that there exist, as he and others have shown, mechanized proof methods which, for the particular run of problems considered, use far less machine effort than does LT and which have the advantage that they will ultimately find a proof for any provable proposition. (LT does not have this exhaustive "decision procedure" character and can fail ever to find proofs for some theorems.) The authors of "Empirical Explorations of the Logic Theory Machine," perhaps unaware of the existence of even moderately efficient exhaustive methods, supported their arguments by comparison with a particularly inefficient exhaustive procedure. Nevertheless, I feel that some of Wang's criticisms are misdirected. He does not seem to recognize that the authors of LT are not so much interested in proving these theorems as they are in the general problem of solving difficult problems. The combinatorial system of Russell and Whitehead (with which LT deals) is far less simple and elegant than the system used by Wang.[24] [Note, *e.g.*, the emphasis in Newell, Shaw and Simon (1958*a*, 1958*b*).] Wang's problems, while *logically* equivalent, are *formally* much simpler. His methods do not include any facilities for using previous results (hence they are sure to degrade rapidly at a certain level of problem complexity), while LT is fundamentally oriented around this problem. Finally, because of the very effectiveness of Wang's method on the *particular* set of theorems in question, he simply did not have to face the fundamental heuristic problem of *when to decide to give up on a line of attack.* Thus the formidable performance of his program (1960) perhaps diverted his attention from heuristic problems that must again spring up when real mathematics is ultimately encountered.

This is not meant as a rejection of the importance of Wang's work and discussion. He and others working on "mechanical mathematics" have discovered that there are proof procedures which are much more efficient than has been suspected. Such work will unquestionably help in constructing intelligent machines, and these procedures will certainly be preferred, when available, to "unreliable heuristic methods." Wang, Davis and

[24] Wang's procedure (1960*a*), too, works backward, and can be regarded as a generalization of the method of "falsification" for deciding truth-functional tautology. In Wang (1960*b*) and its unpublished sequel, he introduces more powerful methods (for much more difficult problems).

Putnam, and several others are now pushing these new techniques into the far more challenging domain of theorem proving in the predicate calculus (for which exhaustive decision procedures are no longer available). We have no space to discuss this area,[25] but it seems clear that a program to solve real mathematical problems will have to combine the mathematical sophistication of Wang with the heuristic sophistication of Newell, Shaw and Simon.[26]

B. Heuristics for Subproblem Selection

In designing a problem-solving system, the programmer often comes equipped with a set of more or less distinct "Methods"—his real task is to find an efficient way for the program to decide where and when the different methods are to be used.

Methods which do not dispose of a problem may still transform it to create new problems or subproblems. Hence, during the course of solving one problem we may become involved with a large assembly of interrelated subproblems. A "parallel" computer, yet to be conceived, might work on many at a time. But even the parallel machine must have procedures to allocate its resources because it cannot simultaneously apply all its methods to all the problems. We shall divide this administrative problem into two parts: the selection of those subproblem(s) which seem most critical, attractive, or otherwise immediate, and, in the next section, the choice of which method to apply to the selected problem.

In the basic program for LT (Sec. IV*A*), subproblem selection is very simple. New problems are examined briefly and (if not solved at once) are placed at the end of the (linear) problem list. The main program proceeds along this list (step 1), attacking the problems in the order of their generation. More powerful systems will have to be more judicious (both in generation and selection of problems) for only thus can excessive branching be restrained.[27] In more complex systems we can expect to consider for each subproblem, at least these two aspects: (1) its apparent "centrality"—how will its solution promote the main goal, and (2) its apparent "difficulty"—how much effort is it liable to consume. We need heuristic methods to estimate each of these quantities and, further, to

[25] See Davis and Putnam (1960), and Wang (1960*b*).

[26] All these efforts are directed toward the reduction of search effort. In that sense they are all heuristic programs. Since practically no one still uses "heuristic" in a sense opposed to "algorithmic," serious workers might do well to avoid pointless argument on this score. The real problem is to find methods which significantly delay the apparently inevitable exponential growth of search trees.

[27] Note that the simple scheme of LT has the property that each generated problem will eventually get attention, even if several are created in a step 3. If one were to turn *full* attention to each problem, as generated, one might never return to alternate branches.

select accordingly one of the problems and allocate to it some reasonable quantity of effort.[28] Little enough is known about these matters, and so it is not entirely for lack of space that the following remarks are somewhat cryptic.

Imagine that the problems and their relations are arranged to form some kind of directed-graph structure (Minsky, 1956*b*; Newell and Simon, 1956*b*; Gelernter and Rochester, 1958). The main problem is to establish a "valid" path between two initially distinguished nodes. Generation of new problems is represented by the addition of new, not-yet-valid paths, or by the insertion of new nodes in old paths. Then problems are represented by not-yet-valid paths, and "centrality" by location in the structure. Associate with each connection, quantities describing its current validity state (solved, plausible, doubtful, etc.) and its current estimated difficulty.

1. GLOBAL METHODS

The most general problem-selection methods are "global"—at each step they look over the entire structure. There is one such simple scheme which works well on at least one rather degenerate interpretation of our problem graph. This is based on an electrical analogy suggested to us by a machine designed by Shannon [related to one described in Shannon (1955) which describes quite a variety of interesting game-playing and learning machines] to play a variant of the game marketed as "Hex" (and known among mathematicians as "Nash"). The initial board position can be represented as a certain network of resistors. (See Fig. 11.) One player's goal is to construct a *short-circuit* path between two given boundaries; the opponent tries to open the circuit between them. Each move consists of shorting (or opening), irreversibly, one of the remaining resistors. Shannon's machine applies a potential between the boundaries and selects that resistor which carries the largest current. Very roughly speaking, this resistor is likely to be most critical because changing it will have the largest effect on the resistance of the net and, hence, in the goal direction of shorting (or opening) the circuit. And although this argument is not perfect, nor is this a perfect model of the real combinatorial situation, the machine does play extremely well. (It can make unsound moves in certain artificial situations, but no one seems to have been able to force this during a game.)

The use of such a global method for problem selection requires that the available "difficulty estimates" for related subproblems be arranged to

[28] One will want to see if the considered problem is the same as one already considered, or very similar. See the discussion in Gelernter and Rochester (1958). This problem might be handled more generally by simply *remembering* the (Characters of) problems that have been attacked, and checking new ones against this memory, *e.g.*, by methods of Mooers (1956), looking more closely if there seems to be a match.

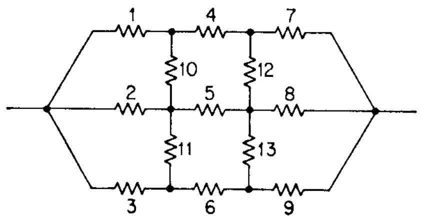

Figure 11. This board game (due to C. E. Shannon) is played on a network of equal resistors. The first player's goal is to open the circuit between the end points; the second player's goal is to short the circuit. A move consists of opening or shortening a resistor. If the first player begins by opening resistor 1, the second player might counter by shorting resistor 4, following the strategy described in the text. The remaining move pairs (if *both* players use that strategy) would be (5, 8) (9, 13) (12, 10 or 2) (2 or 10 *win*). In this game the first player should be able to force a win, and the maximum-current strategy seems always to do so, even on larger networks.

combine in roughly the manner of resistance values. Also, we could regard this machine as using an "analog model" for "planning." (See Sec. IV*D*.)[29]

2. LOCAL, AND "HEREDITARY," METHODS

The prospect of having to study at each step the whole problem structure is discouraging, especially since the structure usually changes only slightly after each attempt. One naturally looks for methods which merely *update* or modify a small fragment of the stored record. Between the extremes of the "first-come-first-served" problem-list method and the full global-survey methods, lie a variety of compromise techniques. Perhaps the most attractive of these are what we will call the *Inheritance* methods—essentially recursive devices.

In an Inheritance method, the effort assigned to a subproblem is determined only by its immediate ancestry; at the time each problem is created it is assigned a certain total quantity Q of time or effort. When a problem is later split into subproblems, such quantities are assigned to them by some local process which *depends only on their relative merits and on what remains of* Q. Thus the centrality problem is managed implicitly. Such schemes are quite easy to program, especially with the new programming systems such as IPL (Newell and Tonge, 1960*c*) and LISP (McCarthy, 1960) (which are themselves based on certain hereditary or recursive operations). Special cases of the Inheritance method arise when one can get along with a simple all-or-none Q, *e.g.*, a "stop condition"—this yields the

[29] A variety of combinatorial methods will be matched against the network-analogy opponent in a program being completed by R. Silver, Lincoln Laboratory, MIT, Lexington, Mass.

exploratory method called "backtracking" by Golumb (1961). The decoding procedure of Wozencraft (1961) is another important variety of Inheritance method.

In the complex exploration process proposed for chess by Newell, Shaw, and Simon (1958*b*) we have a form of Inheritance method with a *non-numerical stop condition.* Here, the subproblems inherit *sets of goals to be achieved.* This teleological control has to be administered by an additional goal-selection system and is further complicated by a global (but reasonably simple) stop rule of the back-up variety (Sec. III*C*). (*Note:* we are identifying here the move-tree-limitation problem with that of problem selection.) Even though extensive experimental results are not yet available, we feel that the scheme of Newell, Shaw, and Simon (1958*b*) deserves careful study by anyone planning serious work in this area. It shows only the beginning of the complexity sure to come in our development of intelligent machines.[30]

C. "Character-Method" Machines

Once a problem is selected, we must decide which method to try first. This depends on our ability to classify or characterize problems. We first compute the Character of our problem (by using some pattern recognition technique) and then consult a "Character-Method" table or other device which is supposed to tell us which method(s) are most effective on problems of that Character. This information might be built up from experience, given initially by the programmer, deduced from "advice" (McCarthy, 1959), or obtained as the solution to some other problem, as suggested in the GPS proposal (Newell, Shaw and Simon, 1959*a*). In any case, this part of the machine's behavior, regarded from the outside, can be treated as a sort of stimulus-response, or "table look-up," activity.

If the Characters (or descriptions) have too wide a variety of values, there will be a serious problem of filling a Character-Method table. One might then have to reduce the detail of information, *e.g.*, by using only a few important properties. Thus the *Differences* of GPS (see Sec. IV*D*2) describe no more than is necessary to define a single goal, and a priority scheme selects just one of these to characterize the situation. Gelernter and Rochester (1958) suggest using a property-weighting scheme, a special case of the "Bayes net" described in Sec. II*G*.

D. Planning

Ordinarily one can solve a complicated problem only by dividing it into a number of parts, each of which can be attacked by a smaller search (or be further divided). Generally speaking, a successful division will reduce

[30] Some further discussion of this question may be found in Slagle (1961).

the search time not by a mere fraction, but by a *fractional exponent.* In a graph with 10 branches descending from each node, a 20-step search might involve 10^{20} trials, which is out of the question, while the insertion of just four *lemmas* or *sequential subgoals* might reduce the search to only 5×10^4 trials, which is within reason for machine exploration. Thus it will be worth a relatively enormous effort to find such "islands" in the solution of complex problems.[31] Note that even if one encountered, say, 10^6 failures of such procedures before success, one would still have gained a factor of perhaps 10^{10} in over-all trial reduction! *Thus practically any ability at all to "plan," or "analyze," a problem will be profitable,* if the problem is difficult. It is safe to say that all simple, unitary, notions of how to build an intelligent machine will fail, rather sharply, for some modest level of problem difficulty. Only schemes which actively pursue an analysis toward obtaining a set of *sequential goals* can be expected to extend smoothly into increasingly complex problem domains.

Perhaps the most straightforward concept of planning is that of using a *simplified model* of the problem situation. Suppose that there is available, for a given problem, some other problem of "essentially the same character" but with less detail and complexity. Then we could proceed first to solve the simpler problem. Suppose, also, that this is done using a second set of methods, which are also simpler, but in some correspondence with those for the original. *The solution to the simpler problem can then be used as a "plan" for the harder one.* Perhaps each step will have to be expanded in detail. But the multiple searches will *add, not multiply,* in the total search time. The situation would be ideal if the model were, mathematically, a *homomorphism* of the original. But even without such perfection the model solution should be a valuable guide. In mathematics one's proof procedures usually run along these lines: one first assumes, *e.g.,* that integrals and limits always converge, in the planning stage. Once the outline is completed, in this simpleminded model of mathematics, then one goes back to try to "make rigorous" the steps of the proof, *i.e.,* to replace them by chains of argument using genuine rules of inference. And even if the plan fails, it may be possible to patch it by replacing just a few of its steps.

Another aid to planning is the *semantic,* as opposed to the homomorphic, model (Minsky, 1956*a*, 1959*a*). Here we may have an *interpretation* of the current problem within another system, not necessarily simpler, but with which we are more familiar and have already more powerful methods. Thus, in connection with a plan for the proof of a theorem, we will want to know whether the proposed lemmas, or islands in the proof, are actually *true;* if not, the plan will surely fail. We can often easily tell if a proposition is true by looking at an interpretation. Thus the truth of a

[31] See sec. 10 of Ashby (1956).

proposition from plane geometry can be supposed, at least with great reliability, by actual measurement of a few constructed drawings (or the analytic geometry equivalent). The geometry machine of Gelernter and Rochester (1958, 1959) uses such a semantic model with excellent results; it follows closely the lines proposed in Minsky (1956*a*).

1. THE "CHARACTER-ALGEBRA" MODEL

Planning with the aid of a model is of the greatest value in reducing search. Can we construct machines which find their own models? I believe the following will provide a general, straightforward way to construct certain kinds of useful, abstract models. The critical requirement is that we be able to compile a "Character-Method Matrix" (in addition to the simple Character-Method table in Sec. IV*C*). *The CM matrix is an array of entries which predict with some reliability what will happen when methods are applied to problems.* Both of the matrix dimensions are indexed by problem Characters; if there is a method *which usually transforms problems of character C_i into problems of character C_j* then let the matrix entry C_{ij} be the name of that method (or a list of such methods). If there is no such method the corresponding entry is null.

Now suppose that there is no entry for C_{ij}—meaning that we have no *direct* way to transform a problem of type C_i into one of type C_j. Multiply the matrix by itself. If the new matrix has a non-null (i,j) entry then there must be a sequence of *two* methods which effects the desired transformation. If that fails, we may try higher powers. Note that [if we put unity for the (i,i) terms] we can reach the 2^n matrix power with just n multiplications. We don't need to define the symbolic multiplication operation; one may instead use arithmetic entries—putting *unity for any non-null entry* and zero for any null entry in the original matrix. This yields a simple connection, or flow diagram, matrix, and its nth power tells us something about its set of paths of length 2^n.[32] [Once a non-null entry is discovered, there exist efficient ways to find the corresponding sequences of methods. The problem is really just that of finding paths through a maze, and the method of Moore (1959) would be quite efficient. Almost any problem can be converted into a problem of finding a chain between two terminal expressions in some formal system.] If the Characters are taken to be abstract representations of the problem expressions, this "Character-Algebra" model can be as abstract as are the available pattern-recognition facilities. See Minsky (1956*a*, 1959*a*).

The critical problem in using the Character-Algebra model for planning is, of course, the *prediction reliability of the matrix entries*. One cannot expect the Character of a result to be strictly determined by the Character of the original and the method used. And the reliability of the pre-

[32] See, *e.g.*, Hohn, Seshu, and Aufenkamp (1957).

dictions will, in any case, deteriorate rapidly as the matrix power is raised. But, as we have noted, any plan at all is so much better than none that the system should do very much better than exhaustive search, even with quite poor prediction quality.

This matrix formulation is obviously only a special case of the character planning idea. More generally, one will have descriptions, rather than fixed characters, and one must then have more general methods to calculate from a description what is likely to happen when a method is applied.

2. CHARACTERS AND DIFFERENCES

In the GPS (General Problem Solver) proposal of Newell, Shaw, and Simon (1959*a*, 1960*a*) we find a slightly different framework: they use a notion of Difference between two problems (or expressions) where we speak of the Character of a single problem. These views are equivalent if we take our problems to be links or connections between expressions. But this notion of Difference (as the Character of a pair) does lend itself more smoothly to teleological reasoning. For what is the goal defined by a problem but to *reduce the "difference" between the present state and the desired state?* The underlying structure of GPS is precisely what we have called a "Character-Method Machine" in which each kind of Difference is associated in a table with one or more methods which are known to "reduce" that Difference. Since the characterization here depends always on (1) the current problem expression and (2) the desired end result, it is reasonable to think, as its authors suggest, of GPS as using "means-end" analysis.

To illustrate the use of Differences, we shall review an example (Newell, Shaw, and Simon, 1960*a*). The problem, in elementary propositional calculus, is to prove that from $S \wedge (-P \supset Q)$ we can deduce $(Q \vee P) \wedge S$. The program looks at both of these expressions with a recursive *matching* process which branches out from the main connectives. The first Difference it encounters is that S occurs on different sides of the main connective "$\wedge$." It therefore looks in the Difference-Method table under the heading "change position." It discovers there a method which uses the theorem $(A \wedge B) \equiv (B \wedge A)$ which is obviously useful for removing, or "reducing," differences of position. GPS applies this method, obtaining $(-P \supset Q) \wedge S$. GPS now asks what is the Difference between this new expression and the goal. This time the matching procedure gets down into the connectives inside the left-hand members and finds a Difference between the connectives "$\supset$" and "$\vee$." It now looks in the CM table under the heading "Change Connective" and discovers the appropriate method using $(-A \supset B) \equiv (A \vee B)$. It applies this method, obtaining $(P \vee Q) \wedge S$. In the final cycle, the difference-evaluating procedure discovers the need for a "change position" inside the left member, and applies a method using $(A \vee B) \equiv (B \vee A)$. This completes the solution of the problem.[33]

Evidently, the success of this "means-end" analysis in reducing general search will depend on the degree of specificity that can be written into the Difference-Method table—basically the same requirement for an effective Character-Algebra.

It may be possible to *plan* using Differences, as well. One might imagine a "Difference-Algebra" in which the predictions have the form $D = D' D''$. One must construct accordingly a difference-factorization algebra for discovering longer chains $D = D_1 \cdots D_n$ and corresponding method plans. We should note that one *cannot* expect to use such planning methods with such primitive Differences as are discussed in Newell, Shaw, and Simon (1960*a*); for these cannot form an adequate Difference-Algebra (or Character-Algebra). Unless the characterizing expressions have many levels of descriptive detail, the matrix powers will too swiftly become degenerate. This degeneracy will ultimately limit the capacity of any formal planning device.

One may think of the general planning heuristic as embodied in a recursive process of the following form. Suppose we have a problem P:

1. Form a plan for problem P.
2. Select first (next) step of the plan.
   (If no more steps, exit with "success.")
3. Try the suggested method(s):
   Success: return to (*b*), *i.e.*, try next step in the plan.
   Failure: return to (*a*), *i.e.*, form new plan, or perhaps change current plan to avoid this step.
   Problem judged too difficult: *Apply this entire procedure to the problem of the current step.*

Observe that such a program schema is essentially recursive; it uses itself as a subroutine (explicitly, in the last step) in such a way that its current state has to be stored, and restored when it returns control to itself.[34]

[33] Compare this with the "matching" process described in Newell and Simon (1956). The notions of "Character," "Character-Algebra," etc., originate in Minsky (1956) but seem useful in describing parts of the "GPS" system of Newell and Simon (1956) and Newell, Shaw, and Simon (1960*a*). The latter contains much additional material we cannot survey here. Essentially, GPS is to be self-applied to the problem of discovering sets of Differences appropriate for given problem areas. This notion of "bootstrapping"—applying a problem-solving system to the task of improving some of its own methods—is old and familiar, but in Newell, Shaw, and Simon (1960*a*) we find perhaps the first specific proposal about how such an advance might be realized.

[34] This violates, for example, the restrictions on "DO loops" in programming systems such as FORTRAN. Convenient techniques for programming such processes were developed by Newell, Shaw and Simon (1960*b*); the program state variables

Miller, Galanter and Pribram[35] discuss possible analogies between human problem-solving and some heuristic planning schemes. It seems certain that, for at least a few years, there will be a close association between theories of human behavior and attempts to increase the intellectual capacities of machines. But, in the long run, we must be prepared to discover profitable lines of heuristic programming which do not deliberately imitate human characteristics.[36]

V. Induction and Models

A. Intelligence

In all of this discussion we have not come to grips with anything we can isolate as "intelligence." We have discussed only heuristics, shortcuts, and classification techniques. Is there something missing? I am confident that sooner or later we will be able to assemble programs of great problem-solving ability from complex combinations of heuristic devices—multiple optimizers, pattern-recognition tricks, planning algebras, recursive administration procedures, and the like. In no one of these will we find the

---

are stored in "pushdown lists" and both the program and the data are stored in the form of "list structures." Gelernter (1959) extended FORTRAN to manage some of this. McCarthy has extended these notions in LISP (1960) to permit *explicit* recursive definitions of programs in a language based on recursive functions of symbolic expressions; here the management of program state variables is fully automatic. See also Orchard-Hays (1960).

[35] See chaps. 12 and 13 of Miller, Galanter, and Pribram (1960).

[36] Limitations of space preclude detailed discussion here of theories of self-organizing neural nets, and other models based on brain analogies. [Several of these are described or cited in *Proceedings of a Symposium on Mechanisation of Thought Processes,* London: H. M. Stationery Office, 1959, and *Self Organizing Systems,* M. T. Yovitts and S. Cameron (eds.), New York: Pergamon Press, 1960.] This omission is not too serious, I feel, in connection with the subject of heuristic programming, because the motivation and methods of the two areas seem so different. Up to the present time, at least, research on neural-net models has been concerned mainly with the attempt to show that certain rather simple heuristic processes, *e.g.,* reinforcement learning, or property-list pattern recognition, can be realized or evolved by collections of simple elements without very highly organized interconnections. Work on heuristic programming is characterized quite differently by the search for new, more powerful heuristics for solving very complex problems, and by very little concern for what hardware (neuronal or otherwise) would minimally suffice for its realization. In short, the work on "nets" is concerned with how far one can get with a small initial endowment; the work on "artificial intelligence" is concerned with using all we know to build the most powerful systems that we can. It is my expectation that, in problem-solving power, the (allegedly brainlike) minimal-structure systems will never threaten to compete with their more deliberately designed contemporaries; nevertheless, their study should prove profitable in the development of component elements and subsystems to be used in the construction of the more systematically conceived machines.

seat of intelligence. Should we ask what intelligence "really is"? My own view is that this is more of an aesthetic question, or one of sense of dignity, than a technical matter! To me "intelligence" seems to denote little more than the complex of performances which we happen to respect, but do not understand. So it is, usually, with the question of "depth" in mathematics. Once the proof of a theorem is really understood its content seems to become trivial. (Still, there may remain a sense of wonder about how the proof was discovered.)

Programmers, too, know that there is never any "heart" in a program. There are high-level routines in each program, but all they do is dictate that "if such and such, then transfer to such and such a subroutine." And when we look at the low-level subroutines, which "actually do the work," we find senseless loops and sequences of trivial operations, merely carrying out the dictates of their superiors. The intelligence in such a system seems to be as intangible as becomes the meaning of a single common word when it is thoughtfully pronounced over and over again.

But we should not let our inability to discern a locus of intelligence lead us to conclude that programmed computers therefore cannot think. For it may be so with *man*, as with *machine,* that, when we understand finally the structure and program, the feeling of mystery (and self-approbation) will weaken.[37] We find similar views concerning "creativity" in Newell, Shaw, and Simon (1958*c*). The view expressed by Rosenbloom (1951) that minds (or brains) can transcend machines is based, apparently, on an erroneous interpretation of the meaning of the "unsolvability theorems" of Godel.[38]

[37] See Minsky (1956, 1959).

[38] On problems of volition we are in general agreement with McCulloch (1954) that our *freedom of will* "presumably means no more than that we can distinguish between what we intend (*i.e.*, our *plan*), and some intervention in our action." See also MacKay (1959) and [the] references; we are, however, unconvinced by his eulogization of "analog" devices. Concerning the "mind-brain" problem, one should consider the arguments of Craik (1952), Hayek (1952), and Pask (1959). Among the active leaders in modern heuristic programming, perhaps only Samuel (1960*b*) has taken a strong position against the idea of machines thinking. His argument, based on the fact that reliable computers do only that which they are instructed to do, has a basic flaw; it does not follow that the programmer therefore has full knowledge (and therefore full responsibility and credit for) what will ensue. For certainly the programmer may set up an evolutionary system whose limitations are for him unclear and possibly incomprehensible. No better does the mathematician know all the consequences of a proposed set of axioms. Surely a machine has to *be* in order to perform. But we cannot assign all the credit to its programmer if the operation of a system comes to reveal structures not recognizable or anticipated by the programmer. While we have not yet seen much in the way of intelligent activity in machines, Samuel's arguments (circular in that they are based on the presumption that machines do not have minds) do not assure us against this. Turing (1956) gives a very knowledgeable discussion of such matters.

B. Inductive Inference

Let us pose now for our machines, a variety of problems more challenging than any ordinary game or mathematical puzzle. Suppose that we want a machine which, when embedded for a time in a complex environment or "universe," will essay to produce a description of that world—to discover its regularities or laws of nature. We might ask it to predict what will happen next. We might ask it to predict what would be the likely consequences of a certain action or experiment. Or we might ask it to formulate the laws governing some class of events. In any case, our task is to equip our machine with *inductive ability*—with methods which it can use to construct general statements about events beyond its recorded experience. Now, there can be no system for inductive inference that will work well in all possible universes. But given a universe, or an ensemble of universes, and a criterion of success, this (epistemological) problem for machines becomes technical rather than philosophical. There is quite a literature concerning this subject, but we shall discuss only one approach which currently seems to us the most promising; this is what we might call the "grammatical induction" schemes of Solomonoff (1957, 1958, 1959*a*), based partly on work of Chomsky and Miller (1957*b*, 1958).

We will take *language* to mean the set of expressions formed from some given set of primitive symbols or expressions, by the repeated application of some given set of rules; the primitive expressions plus the rules is the *grammar* of the language. Most induction problems can be framed as problems in the *discovery of grammars*. Suppose, for instance, that a machine's prior experience is summarized by a large collection of statements, some labelled "good" and some "bad" by some critical device. How could we generate selectively more good statements? The trick is to find some relatively simple (formal) language in which the good statements are grammatical, and in which the bad ones are not. Given such a language, we can use it to generate more statements, and presumably these will tend to be more like the good ones. The heuristic argument is that if we can find a relatively simple way to separate the two sets, the discovered rule is likely to be useful beyond the immediate experience. If the extension fails to be consistent with new data, one might be able to make small changes in the rules and, generally, one may be able to use many ordinary problem-solving methods for this task.

The problem of finding an efficient grammar is much the same as that of finding efficient *encodings,* or programs, for machines; in each case, one needs to discover the important regularities in the data, and exploit the regularities by making shrewd *abbreviations.* The possible importance of Solomonoff's work (1960) is that, despite some obvious defects, it may point the way toward systematic mathematical ways to explore this discovery problem. He considers the class of all programs (for a given general-purpose computer) which will produce a certain given output (the body of data in question). Most such programs, if allowed to continue, will add to that body of data. By properly weighting these programs, perhaps by length, we can obtain corresponding weights for the different possible continuations, and thus a basis for prediction. If this prediction is to be of any interest, it will be necessary to show some independence of the given computer; it is not yet clear precisely what form such a result will take.

C. Models of Oneself

If a creature can answer a question about a hypothetical experiment, without actually performing that experiment, then the answer must have been obtained from some submachine inside the creature. The output of that submachine (representing a correct answer) as well as the input (representing the question) must be coded descriptions of the corresponding external events or event classes. Seen through this pair of encoding and decoding channels, the internal submachine acts like the environment, and so it has the character of a "model." The inductive inference problem may then be regarded as the problem of constructing such a model.

To the extent that the creature's actions affect the environment, this internal model of the world will need to include some representation of the creature itself. If one asks the creature "why did you decide to do such and such" (or if it asks this of itself), any answer must come from the internal model. Thus the evidence of introspection itself is liable to be based ultimately on the processes used in constructing one's image of one's self. Speculation on the form of such a model leads to the amusing prediction that intelligent machines may be reluctant to believe that they are *just* machines. The argument is this: our own self-models have a substantially "dual" character; there is a part concerned with the physical or mechanical environment—with the behavior of inanimate objects—and there is a part concerned with social and psychological matters. It is precisely because we have not yet developed a satisfactory mechanical theory of mental activity that we have to keep these areas apart. We could not give up this division even if we wished to—until we find a unified model to replace it. Now, when we ask such a creature what sort of being it is, it cannot simply answer "directly"; it must inspect its model(s). And it must answer by saying that it seems to be a dual thing—which appears to have two parts—a "mind" and a "body." Thus, even the robot, unless equipped with a satisfactory theory of artificial intelligence, would have to maintain a dualistic opinion on this matter.[39]

[39] There is a certain problem of infinite regression in the notion of a machine

Conclusion

In attempting to combine a survey of work on "artificial intelligence" with a summary of our own views, we could not mention every relevant project and publication. Some important omissions are in the area of "brain models"; the early work of Farley and Clark (1954) [also Farley's paper in Yovitts and Cameron (1960), often unknowingly duplicated, and the work of Rochester (1956) and Milner (1960)]. The work of Lettvin *et al.* (1959) is related to the theories in Selfridge (1959). We did not touch at all on the problems of logic and language, and of information retrieval, which must be faced when action is to be based on the contents of large memories; see, *e.g.*, McCarthy (1959). We have not discussed the basic results in mathematical logic which bear on the question of what can be done by machines. There are entire literatures we have hardly even sampled—the bold pioneering work of Rashevsky (c. 1929) and his later co-workers (Rashevsky, 1960); Theories of Learning, *e.g.*, Gorn (1959); Theory of Games, *e.g.*, Shubik (1960); and Psychology, *e.g.*, Bruner *et al.* (1956). And everyone should know the work of Polya (1945, 1954) on how to solve problems. We can hope only to have transmitted the flavor of some of the more ambitious projects *directly* concerned with getting machines to take over a larger portion of problem-solving tasks.

One last remark: we have discussed here only work concerned with more or less self-contained problem-solving programs. But as this is written, we are at last beginning to see vigorous activity in the direction of constructing usable *time-sharing* or *multiprogramming* computing systems. With these systems, it will at last become economical to match human beings in real time with really large machines. This means that we can work toward programming what will be, in effect, "thinking aids." In the years to come, we expect that these man-machine systems will share, and perhaps for a time be dominant, in our advance toward the development of "artificial intelligence."

having a *good* model of itself: of course, the nested models must lose detail and finally vanish. But the argument, *e.g.*, of Hayek (see 8.69 and 8.79, 1952) that we cannot "fully comprehend the unitary order" (of our own minds) ignores the power of recursive description as well as Turing's demonstration that (with sufficient external writing space) a "general-purpose" machine can answer any question about a description of itself that any larger machine could answer.