Some Recent Work in Artificial Intelligence

R. J. SOLOMONOFF, MEMBER, IEEE

Abstract—**This paper will review certain approaches to artifical intelligence research—mainly work done since 1960. An important area of research involves designing a machine that can adequately improve its own performance as well as solve other problems normally requiring human intelligence. Work in heuristic programming that seems most relevant to this goal will be discussed at length.**

Important subproblems are devising techniques for self-improvement, the general problem of deciding what task to best work on next in a network of tasks, and the general problem of how to mechanize learning or inductive inference. Some work in linguistics and pattern recognition is directly concerned with the induction problem.

Another area of research that will be treated is simulation of organic evolution.

I. Introduction

AN IMPORTANT GOAL of artificial intelligence research is to devise machines to perform various tasks normally requiring human intelligence. Proving mathematical theorems, learning to translate languages, playing good games of chess, and learning to improve its own performance are a few of the kinds of things such a machine is expected to perform. Although each of such tasks has certain peculiarities that characterize it uniquely, many workers in this field feel that there are certain characteristics that are common to most tasks requiring intelligence, and they have tried to work on problems in which these characteristics are prominent. Some tasks common to many artificial intelligence problems are: initial description and transformation of a problem into a more readily solvable form, heuristic search and its associated subproblems, breaking a difficult problem into several easier problems, and learning through induction from past experience.

The present paper will emphasize work in two principal areas. Because of its access to a source of powerful, though neglected, heuristics for artificial intelligence, simulation of organic evolution will be discussed at some length. Some recent work of this sort looks promising in its ability to solve simple problems using only the simplest heuristic devices. Solution of more difficult problems will be possible when more of the devices used in organic evolution are understood and applied.

Heuristic programming will also be discussed in some detail. In this particular approach to artificial intelligence, the researcher usually tries to program a computer to solve problems in ways similar to those that he uses himself. I feel that the path of research most likely to achieve a high level of intelligent machine behavior, is that of devising a machine that can effectively work on the problem of improving its own performance, as well as work on problems that are more directly useful.

I will deal mainly with heuristic programming work that seems most directly related to this goal, as well as important associated problems in inductive inference and the solution of networks of tasks.

Simulation of networks of neurons, optical character recognition, and other biological systems have inspired a large part of present-day artificial intelligence research, but I will not discuss such work at any length, since at the present time, it appears that none of it is capable of dealing with problems as complex as self-improvement.

Minsky [44] has written an excellent critical review of artificial intelligence work up to about September, 1960. This paper will be largely limited to work done since then.

II. Heuristic Programming

Heuristic programming started with Newell, Simon, and Shaw's "Logical Theorist" [52]. Gelernter's geometry theorem prover [31], Newell, Shaw and Simon's "General Problem Solver" [53], and Samuel's [58] checker playing program followed.

The programs of Newell et al. and Gelernter had little learning in them, other than use of theorems that had been previously proved by them. Except for this, their ability to work problems did not improve with experience.

Samuel's program, on the other hand, employed two important kinds of learning. The first kind was rote learning. It remembered board configurations that it had analyzed in the past, as well as the results of the analyses. It also had a set of measurements for the quick, approximate evaluation of board configurations, such as the number of pieces ahead or the "centralness" of the white pieces. Each such measurement was given a weight in this approximate evaluation. To decide on a possible move, its possible implications were played out several moves into the future, the approximate evaluation function was applied to the resultant board configurations, and the results were used to evaluate the current board position more accurately. As the machine played more games, it used this experience, as well as an interesting kind of internal calculation, to modify the weights in its evaluation function.

Samuel's program has been remarkably successful. After some improvements, it now plays a master level game. It did not, however, come close to beating the world champion. Samuel feels that this would require a marked increase in program proficiency. He is now attempting to improve its learning capacity by taking triplets of the measurements used for board configurations, and assigning learnable weights to them (Samuel [59]).

Another important step was the invention of GPS (General Problem Solver) of Newell et al. [53]. This was a pro-

Manuscript received August 17, 1966; revised October 17, 1966. This research was sponsored by USAF Contract AF-19(628)-5975; AFSOR Contract AF-49(638)-376, Grant AF-AFOSR 62-377, and Public Health Service NIH Grant GM 11021-01.

The author is with the Rockford Research Institute, Cambridge, Mass.

gram meant to generalize the methods of their "Logical Theorist," so that it might be applied to a greater variety of problems. The problems it could solve were of the following form: One is given an initial state of a system—say a set of mathematical postulates. One is given a final desired state of the system—say a theorem to be proved. One is given a set of operators, such as rules of inference, that can be successively applied to the system to yield new states. Such states might be new theorems or new rules of inference. The problem is to find a sequence of operators to be applied successively to the original state to yield the desired state.

GPS uses a set of observations, called "differences" between the current state and the desired state, to decide which operator to try next. Application of an operator yields a new current state and a new set of differences. This recursively yields a new set of operators to try. If one operator doesn't "reduce" the differences, another is alternatively tried.

This hill-climbing technique (see Minsky [44] and Minsky and Selfridge [47] for good discussions of hill climbing) is augmented by various devices that devise intermediate sub-goals, or sets of sub-goals.

Newell, Shaw, and Simon applied GPS principally to theorem proving at first, but since then, their approach has been generalized in various ways and applied to a variety of problems. Newell and Ernst [51] have written a review of the various generalizations and kinds of heuristic devices used by various workers. Some examples are Slagle's symbolic integration program (Slagle [66]) and Tonge's assembly line balancing procedure (Tonge [75]). Ernst [18] has generalized GPS in several ways and applied it to eleven different kinds of problems.

At this point, it seems clear that heuristic programming can be used to solve a great variety of problems. We may then ask whether such programs could be given the task of improving themselves in speed or otherwise optimizing their own operation. The criterion of success at self-improvement will not be vacuously circular if the program also has to solve problems different from self-improvement.

While Samuel's checker program does indeed improve its performance with time, its general methods remain about the same. An important step toward self-improvement was the modification of GPS by Newell et al. [54] so that it might learn to improve the set of "differences" that it uses in helping to decide what operator to try next. I will not describe the system in detail, but an analogy with hill climbing will help explain its method of operation.

In the general hill-climbing problem, one has several parameters, x_i, discrete, continuous, or a mixture of both. One has an evaluation function $F(x_1, x_2, \cdots)$. What is the best strategy of trials to use, if one wants a set of x_i's that will maximize F? A common strategy is to pick a set of x_i's, and obtain its F value. Then try random points "near" the original x_i's for the next trial. If a better F is obtained, use this new set of x_i's for one's next base. If not, then try a new random point in the neighborhood of the original set of x_i's.

This random mutation method will be discussed later in the section on induction with respect to evolutionary models.

In most hill-climbing problems, F is a continuous function of its arguments. In Newell's system, however, there is no F, but instead, a method for telling which of two sets of X values are better—so one can tell if one trial mutation is better than or worse than another. This was done by devising a good set of criteria to decide if one set of differences was an improvement over another.

One of the advantages of Newell's scheme is that it pointed out *which* of the x_i's were most likely to need changing. This technique, when it is applicable, is a great advance over the random choice of changes in the x_i's.

While Newell's scheme has been hand simulated with some success, there were some initial limitations (described in Newell [49]) in the GPS system that made it impractical to run his program within GPS on a computer. Since then, these difficulties appear to have been resolved. In Newell [50] another difference improving scheme for GPS is briefly outlined, but neither method has yet been programmed. Some further evidence that GPS can deal with problems *like* self-improvement, is afforded by Simon's [65] heuristic compiler. The problem to be solved here is that of writing a computer program. The desired program can be described to the compiler in several forms. First, as the input and desired output. Second, as a description of what the program must do, using a set of terms different from the set of operators the machine will use to implement the program. Third, as a mixture of the first two methods. For the second method, an algorithmic compiler *could* be used, but is not.

The main point of this work was to show the generality of GPS. It might, however, be possible to use this program with the first mode of input only, give it a small set of desired input-output pairs, and ask that it construct a minimal program to transform the inputs into the outputs. If there is an incomplete set of examples, the compiler will have to perform some inductive inference upon this set.

The induction problem is treated in just this form by Amarel [4], [5]. There are many important ideas in his work. First in his realization of the importance that the language (in this case, a set of operators) be adequate for the task of interest. In much work in induction (pattern recognition, in particular), the languages used to describe regularities in the data are woefully inadequate for any but the simplest concepts. One is reminded by Ashby's "principle of requisite variety." Paraphrased, it says that if you are looking for a door knob, and you have a barrel of assorted junk, it is well to have evidence that there *are* door knobs in the barrel, before looking for them there. Minsky [45] also discusses the adequacy of languages.

Another important element is Amarel's use of subroutines that have been useful in old programs as good trial components for new programs. Also, he develops transition probabilities between these subroutines, so he can build up larger subroutines. Both of these learning techniques can reduce tremendously the time spent in search-

ing for an adequate program. An earlier form of this was proposed by Selfridge [62].

Importance of ordering of the data given to a machine is emphasized. The order in which examples are given can contain just as much heuristic information as any trick that more directly reduces search time (Solomonoff [71]).

He tries to find *short* programs that link the known input to the known output examples. He does this because it takes less time to find such programs. There is, however, a much more important reason that he overlooks.

The short programs are more likely to extrapolate better. Given a training sequence of input-output pairs, it is easy to construct a table rapidly that will map one into the other, but will have negligible chance of extrapolating properly to new data. Such a table can be viewed as a "longer program" than a short sequence of operators connecting the input and output. This point will be discussed further in the section on inductive inference.

Although Amarel hasn't programmed any of his theories, his ideas and his analysis of them are important.

A general problem solving system with integrated learning features has been programmed by Hormann (the 1965 paper [36] gives latest description; [35] is an abridged version of it; [33] and [34] give important details).

Hormann's program may be broken down into several mechanisms or routines. Some of these are sets of abstraction and characterization routines that take the input problem and translate it into a form that the rest of the program is more likely to be able to work with. Proper "representation" can often determine whether a problem is solvable or not (Amarel [6], [7]). Ideally, if a problem is not solvable in reasonable time by the rest of the system, it should be possible for the administrative routine to ask the abstraction and characterization routines for a better (hopefully) representation of the problem.

The programming and problem oriented mechanisms taken together form a small GPS-like device—perhaps most similar in form to Simon's [65] heuristic compiler. It attempts to sove problems in a very direct way without using sub-goals. Its own goals may, however, have been devised for it by the planning mechanism, and can be sub-goals toward some larger goal.

The planning mechanism looks at a problem, then tries to find a sub-goal or a plan with several sub-goals to simplify the problem. The importance of sub-goals in drastically reducing heuristic search time is discussed by Minsky [44].

The induction mechanism incorporates all of the learning routines in the program. It does the learning for the abstraction and characterization routines, as well as the planning, programming, and problem oriented mechanisms. Learning is implemented through a comparison of the present problem of one of the mechanisms or routines with earlier problems. Since all induction is performed by the same induction mechanism, it should be ultimately possible for the machine to bring all past experience in any mechanism or routine to bear on any present problem in the same or any other mechanism or routine.

While Hormann uses several interesting induction devices, they are few in number, and certainly not capable, even in theory, of recognizing all conceivable regularities in a body of data. This limitation could, however, be overcome by using a "complete" language to describe regularities to be used in induction. It is also necessary to have methods for devising trial "regularity recognition devices" within this language, that do not limit the power of the language. Up to the present time, there have been three systems that have used complete languages to construct inductive hypotheses—the finite state machines used by Fogel et al. [26] for simulation of evolution, the rewrite rules used by Solomonoff [71] for arithmetic induction, and the general set of computer instructions used by Kilburn et al. [38] to extrapolate alphanumeric sequences. While each of these systems is, in theory, capable of discovering any describable hypothesis, in all cases the limited heuristics used make only the simplest kinds of hypotheses practically accessible.

Hormann's device is administrated by a "mechanism coordinator" that calls on various subroutines when they are needed.

Much recent work on this program has been in developing "secondary learning" capabilities—in which the trainer can "tell" things to the machine directly, rather than give information only in the form of a training sequence of problems of increasing difficulty. This is in the spirit of McCarty's [43] idea that in order for a machine to be able to learn something, it must be capable of being told it.

There are, however, difficulties associated with secondary learning. If the machine is told how to do a problem, there is no assurance that it will generalize this ability in any useful way—unless the trainer is familiar with the particular generalization devices used by the machine and has picked the tutorial example accordingly.

Usually if the trainer knows that much about the machine, he can devise a suitable ordinary (primary) training sequence for it to accomplish at least the same things.

The above remarks apply equally well to machine or human learning.

Altogether, Hormann's seems to be the most general problem solver programmed that has much learning in it.

Slagle's [68] program is also very general, and though its learning is of a very restricted kind, it is still important. To place this work in proper perspective, let us consider Slagle's 1964 paper [67] on networks of tasks. I shall describe a problem type that is a sub-class of the type that Slagle considers. This sub-class is the type we usually find in heuristic search problems.

We are given N subsets of M different tasks. The M tasks are A_i $(i=1, 2, \cdots M)$ and the kth subset of tasks is $[A_{ajk}]$, $k=1, 2, \cdots N$ and $j=1, 2, \cdots N_k$, there being N_k members of the kth subset. A given task may occur in more than one subset.

The goal of interest is either to work all of the tasks in one subset, or to know for certain that at least one task in each subset is impossible.

It takes effort E_i to work on task A_i, and after that effort

has been expended, one has a probability p_i of having *successfully* completed that task, and a probability $1-p_i$ of finding that the task is impossible.

The problem is to find a strategy for choosing the first and all subsequent tasks to work on, such that one's expected total effort in attaining the goal is minimal.

It is possible to write a formal solution to the problem but, for any reasonable number of tasks, the computation time is prohibitive.

The relevance to heuristic search is that an ordinary search tree with various sub-goals in it presents a problem similar to Slagle's.

Slagle's 1964 paper [64] gives an easily implemented solution to certain important cases, and good approximations for a larger set of cases.

In order to make good statistical estimates of E_i and P_i in a real task net, it is useful to generalize Slagle's problem somewhat. We can associate with each task, A_i, two functions of effort. $G_1^i(E)$ is the probability that after effort E has been expended, the task will have terminated with either success or knowledge of impossibility of success. $G_2^i(E)$ is the probability that if it terminates with effort E, the result will be success.

For many tasks of interest, $G_1(\infty)<1$; i.e., it is possible to expend an arbitrary amount of effort on the task without its terminating.

Some reasonable functional forms for the G's are:

$$G_1^i(E) = a(1 - e^{-E/T}),$$

$$G_2^i(E) = b, \; a \text{ and } b \text{ being constants.}$$

A constraint commonly occurring in real task nets is that the tasks are partially ordered, so that certain tasks cannot be worked before certain others have terminated.

Of much importance are correlations between various tasks. Here the knowledge of success or failure after effort E on the ith task will affect the forms of G curves for other tasks.

There are two very important kinds of correlation. Consideration of correlations between ancestral tasks can prevent one from going into loops or staying in one area of the tree for an inordinately long time.

Consideration of "sibling" correlation prevents the parent task from appearing to be spuriously attractive.

In Slagle's task net solution, one chooses the best task to work on and continues until it is completed, before starting on a new task. If the more general $G(E)$ functions are used one will often take up a new task and temporarily suspend work on an old one that has not resolved itself after an excessively large amount of effort has been spent on it.

Slagle's 1965 paper [68] considers a very general problem solver in which he has, by statistical study, obtained the E_i's and p_i's for all of the tasks in the net. Superficially, the problem looks identical to that of his 1964 paper [67], but a deeper analysis seems to indicate some differences. At any rate his method of approximate solution differs markedly from that of his 1964 paper [67], and seems to be far more complex. For each task that might be worked on, he computes a figure of "merit," then works on that task with most "merit." The merit of each task depends on the E_i of that task, and on the p_i's of that and various other tasks in the net. It is, however, independent of all other E_i's in the net. I believe that this later independence is wrong, however, and have been able to discover a commonly occurring counter-example.

Some work in operations research involving networks of tasks of the sort discussed here has been done by Eisner [17] and De Baum [14]. Scott and Hopkins [60] review some of this work.

Evans has written a program for recognizing geometric analogies such as are used in "intelligence tests" ([19] is the main reference; [20] is an abridgement that leaves out many important ideas). A sequence of 3 pictures is given, labeled A, B, and C, respectively. Each picture will consist of one or more simple geometric figures in various relative positions. The question to be answered is "Figure A is to Figure B as Figure C is to which of the answer?"—there are 5 possible answer pictures shown.

The program first takes the 8 pictures as given, and describes them in its own internal language—usually telling what the various geometric figures are, their relative sizes, which are similar to which, their relative positions, which are inside which, etc. From this data, it is usually possible to describe several ways in which figure A is related to figure B. The machine then tries to apply these relations between C and each of the 5 possible answers in turn. If there are several answers that fit, the one involving the "most specific rule" is chosen.

The problem worked is an induction problem of a very useful kind and the type of reasoning Evans uses can be applied to very difficult problems in heuristic programming. Suppose we have found that if we are in situation A, the best thing to do is B, and we have a lot of experience with A and B.

Next, situation C arises, and we have had little experience with it. What shall we do? One of the best trial actions can be obtained by finding or creating an action D, such that C is to D as A is to B. Ordinarily there will be much ambiguity in the choice of D, but it is often resolvable by other criteria.

A field related to heuristic programming is the computer simulation of human cognitive behavior. Up to the present time, the kinds of behavior simulated have been fairly simple. The models used certainly do not yet add significantly to the work that has been discussed in the present paper.

There has been somewhat of a division in artificial intelligence work between those, such as H. Simon, who are primarily interested in how the human mind works, and others, such as Minsky, who are mainly interested in writing a program for a very intelligent machine. My own orientation is with Minsky, but I feel that we should try to make our machines' operations correspond with those of humans, for the preliminary part of the work, since there is somewhat more likelihood that we can debug a complex machine if it thinks a bit like we do.

In review of the work on heuristic programming, GPS, Hormann's, Slagle's, and Amarel's systems all seem capable of sufficient generality to work on the problem of devising new heuristics for themselves. In addition, Hormann, Slagle and Amarel do have learning features integrated into them. Though in Slagle's system, this consists only of parameter readjustment, its application to networks of tasks touches on problems common to all heuristic search programs.

In all of the learning systems mentioned, the kinds of self-improvement accessible to the machines have been quite limited. Most of the heuristics we can describe (in fact most of the heuristics used in these programs) could not have conceivably been discovered by any of these systems. The effective languages for describing learnable heuristics within each system have been too weak, and the heuristics for finding heuristics have been too weak. The discussions of Newell et al. [54] and Amarel [4], [5] of hierarchies of languages and Minsky [45] on power of languages are of some value here. We still need to know the kinds of heuristics we need to find heuristics, as well as what languages can *readily* describe them. We must then devise suitable training sequences for our systems, so that using the heuristic-finding heuristics we have given them, they can find new heuristics.

It is not difficult to devise languages that are "universal" in that they can describe *anything* that is describable by any other language. Since a universal language has facilities for defining arbitrary things, it can, by making suitable definitions, describe any object or set of objects very briefly. One must, however, exercise care in devising definitions. Each must be fairly short, usually using the names of other entities that it has been found useful to define. If, in this sense, the definitions are not "short," then it is less likely that these definitions will be useful in the future. Inductions made using concepts having "long" definitions will tend to be poor. Solomonoff ([73], pp. 231–240) describes an induction system which devises definitions of subsequences of symbols. The question of when a definition is too "complex" or "unlikely" is discussed quantitatively with respect to its use in prediction.

With respect to task nets, Slagle has only begun work on a very difficult, complex problem. A few areas of needed research are: how best to parametrize the statistical characteristics of a task with respect to success and failure probabilities after a certain time; how to parametrize correlations between various tasks in the net; obtaining reasonable approximation methods for deciding what task to work on next; and how to obtain the statistical parameters of a task on the basis of past experience with related tasks.

There are several important possible developments of Evans' geometric analogy program. The first would be to apply it to induction problems that occur as subtasks in various problem solving systems. His methods could be used to devise trial solutions to new problems that are analogous to known solutions of corresponding problems. Another very difficult line of development would be to work verbal analogy problems. A very effective program of this sort would require an understanding of semantics that is well beyond our present knowledge. However, an attempt at such a program could do much toward indicating clearly many of the problems of semantics.

III. Inductive Inference

Learning from experience, discovering patterns in pictures or sequences of words, and the recognition of analogies are all aspects of induction, a problem that runs through almost all artificial intelligence research. I will first discuss some theoretical work in this field and then some attempts to mechanize induction.

There are several important problems involved in devising a machine that can, in principle, discover and use an arbitrarily chosen concept for prediction and decision making.

The first problem is to find regularities in a body of data. Amarel [4], [5], Simon [65], Kilbrun [38], Hormann [33]–[36], and Abrahams et al. [2] have done some of the more interesting work in this area.

Next, having found this "regularity," how is it to be used to predict the future, or be used to influence future decision? If several regularities are found, how can they be combined to give better predictions? If we have one regularity with a small a priori likelihood but much empirical data for it, what weight shall we give to its predictions relative to those of another regularity of very high a priori likelihood, for which we have much less empirical data? How are these a priori likelihoods to be computed?

A very important problem is that of describing "regularities" or "concepts" in the most general possible way, so one is certain that the nature of concepts accessible is not being limited by purely linguistic constraints.

Solomonoff [72], [73] has devised several theories of induction that attempt to answer these questions in a general manner. Later work has made it seem likely that these theories are all mathematically equivalent.

All induction problems can be shown to be equivalent to extrapolating a long sequence of symbols, S, this sequence containing all data to be used in the prediction.

Consider a universal Turing machine, or a general purpose computer with an infinitely expandable memory. Certain strings when presented to this computer as input, will produce a string that starts out with the sequence S, and continues. Each such continuation can be regarded as a prediction obtained by its input string. To obtain the probability of a specific continuation, we must weigh all of the predictions we have obtained. One of the theories gives them a weight proportional to the "likelihood" of the input string that caused them. For a binary input string of length N, this weight will be about 2^{-N}.

An approximate prediction can be obtained using only the minimum length input. This is equivalent to saying that the continuation is most likely which has the shortest "description"—"description" being equivalent to computer input string. This single best prediction approach is equivalent to that of Van Heerden [78].

Another of the four theories extrapolates a sequence by

hypothesizing the sequence was produced by a universal Turing machine with random input. An important kind of approximation to this is the work of Booth [10]–[12] on finite state machines with random inputs.

The present status of Solomonoff's formulation of induction is rather uncertain. It has been applied in an approximate way to several prediction problems, and appears to give reasonable answers. It seems directly applicable to several problems in pattern recognition and discovery, but has not yet been applied to these or any other practical problems.

A problem in which it might be used to obtain quantitative probability values, is in a more exact analysis of the work of Kilburn et al. [38], on extrapolation of sequences of symbols. They start with a short sequence of letters or numbers and try to predict the next member of the series. Short sequences of specially devised instructions are used as computer inputs, in attempts to produce the known sequence as output. If they succeed they use the successful input sequence to predict the next symbol of the sequence to be extrapolated.

Their search for satisfactory sequences of symbols is essentially random, with strong bias toward short codes. As the machine is used to extrapolate larger numbers of sequences, statistics on various instructions and on parts of instructions are collected and used to guide future searches for codes. These statistical constraints increase the search efficiency a great deal.

It is not altogether clear as to whether the set of computer instructions they use is "complete"—in the sense of being able to express any conceivable program. However, if they did use a complete set of instructions, their method of induction would be an approximation to Solomonoff's induction theory.

The methods used by Kilburn et al. would not be directly applicable to the extrapolations of much complexity (i.e., those that involve many instructions to describe them), since the search process would take too long. However, they had many good ideas on how to continue the research and it is unfortunate that they did not do so.

A more recent attempt to extrapolate sequences of symbols was made by Abrahams et al. [2] (for earlier, more accessible work, see Pivar [57] and Fredkin [28]). This program consists of a set of "workers" which are called upon in turn to look for a special kind of regularity in the sequence. The kinds of "workers" they use are rather interesting. They recognize polynomial sequences and some exponential types with no difficulty, but they also try to "fit" programs that don't fit exactly, then, using the errors as a new sequence, try to fit a program to them.

As it is, the workers are fixed. If none of the workers find a good fit for a sequence, the program does *not* try to devise new workers to do the job by combining old ones. Also, the program makes only single extrapolations—it does not give probabilities—even when it is clear that there are several programs that will fit the data about equally well—each extrapolating somewhat differently.

Fogel et al. [25], [26] have used an interesting variant of Kilburn's [38] sequence extrapolation system. Instead of using a randomly constructed sequence of computer instructions to express the known data and extrapolate, they use a small finite state machine, also somewhat randomly constructed using a hill-climbing technique.

A finite state machine having N input symbols and S different states, can be characterized by a $N \times S$ matrix, M_{ij} $(i=1, 2, \cdots N, j=1, 2, \cdots S)$.

M_{ij} has 2 components. When the machine has input symbol i and is in state j, the first component tells what state the machine will go to next. The second component tells what its output symbol will be.

Since any digital computer is a finite state machine, we see that the possible input-output relationships in this device are completely unrestricted—if we allow enough states.

The program operates by feeding a sequence of symbols containing certain regularities into a small (having few states) finite state machine. A "score" is given to the machine which is high if the machine's output usually predicts the next input symbol.

The original machine is then "mutated" slightly, by changing the number of states or otherwise modifying the state transition matrix. The new "offspring" machine is then tested. If its score is better than that of the parent, it becomes the new parent of mutated offspring. If its score is not better, then the original parent has another mutated offspring.

This hill-climbing process continues until a machine is obtained with a sufficiently high score.

Various refinements of this technique are also used—e.g., a parent may be allowed to have children until 2 or more have a higher score than the parent. This branching can be allowed to continue, retaining the "best" 100 machines, say, at all times—which reduces the likelihood of local maxima, a disease common to hill climbers.

This mutation and testing process can be simulated *very* rapidly on a computer, since finding the next state and output symbol requires only a table look up—normally a *very* fast operation.

In 1958 and 1959, Friedberg [29], [30] devised programs that mutated sequences of computer instructions, one instruction at a time, in attempts to find a sequence that would consistently accomplish a certain simple task. The program was, however, unable to solve rather simple problems in a reasonable time. This has been interpreted by Minsky and Selfridge [47] as being due to the "Mesa" phenomenon. In certain hill-climbing situations, such as Friedberg's, small changes in one's parameters produce little or no changes in the result, but changes over a certain size produce *very* large changes in the results. In such cases, hill climbing of the usual sort is not effective. One method to solve such problems is to use different representations of them—essentially what Fogel et al. have done.

There are, however, several difficulties in their system. A relatively minor one is the penalty for complexity of machine that is incorporated into the "score" that the machine makes on a prediction run. At the present time, this penalty is rather arbitrarily set at a level proportional to the number

of states used. Fogel says that one can make this penalty depend on whatever is "costly" if one likes, so if one has limited memory, one can penalize a machine for using much of it.

It is my impression that if the machine is to be used for induction, then the penalty for complexity is by no means arbitrary, but can be computed—in some cases with exactness. Use of any other penalty function will result, on the average, in poorer predictions (Solomonoff [72], [73]).

Perhaps the most serious difficulty in the entire system is its lack of mechanisms corresponding to some of the processes involved in sexual reproduction and recombination of useful sets of characteristics. While it is possible to develop machines to solve simple problems by mutating a few states at a time, this process is far too slow to obtain solutions to complex problems. In natural evolution this difficulty is overcome by many special mechanisms (Darlington [13]; see also Stebbins [74] ch. 3 for a simple discussion of a few of the mechanisms). Some of the more important of these involve the preservation of subsequences of genes (which correspond to subroutines) that have been useful. In sexual reproduction if we neglect cross-over, the offspring will acquire long sequences of genes as units—i.e., the chromosomes.

On the average, half of these units will be from one parent, half from the other.

In terms of machines, this *may* mean that the child machine will acquire certain submachines from one parent, and certain from the other. If "crossing over" is allowed, the chromosomes themselves contain shorter substrings of instructions that tend to remain together in the offspring.

The mechanisms of chromosome breakage, inversion, and crossing over work in a manner that tends to push synergistic[1] genes together on the chromosome.

So far, Fogel et al. have not tried any of the mechanisms used by organic evolution other than mutation. They do, however, discuss the possibility of including some of them. For "sexual reproduction" of machines that have binary signals for their prediction outputs, they propose to "mate" 3 of the machines, making a larger machine that uses a majority decision for prediction. Clearly, this kind of mating cannot occur very often, since the number of states per machine would be cubed each generation! One might follow each sexual mating by many generations in which the "score" for performance gives a large reward for reduction in the number of machine states. It is not clear what the overall effect would be.

Another idea in the direction of recombination is their suggestion that the Krohn-Rhodes [41] theory of decomposition of automata be used to divide machines into parts, so that the parts could be recombined as are the sets of characteristics described by chromosomes. Certainly this is an interesting idea. Though at first the decompositions would tend to be useless for recombination they could later become useful in this way if a suitable evolutionary milieu were provided. Such a milieu would have to provide a large enough pool of organisms to mate with one another, as well as suitable individual survival criteria.

The method of Fogel et al. should not be regarded in its present state as being particularly good for working any but the simplest problems. To work more difficult problems, the same difficulties must be overcome that are inherent in other problem solving systems. It is necessary to devise a training sequence of tasks with small enough "conceptual jumps" in the sequence so that the learning entity can follow them. Heuristic devices must be found for each kind of conceptual jump, and these heuristics must be expressed within the learning system being used.

In simulation of evolution much of the successful research will probably center around devising good kinds of mutations and finding methods to select and preserve good sets of characteristics for recombination.

The promise of artificial evolution is that many things are known or suspected about the mechanisms of natural evolution, and that these mechanisms can be used directly or indirectly to solve problems in their artificial counterparts. For artificial intelligence research, simulation of evolution is incomparably more promising than simulation of neural nets, since we know practically nothing about natural neural nets that would be at all useful in solving difficult problems.

Research in simulation of evolution has, however, been very limited in both quantity and quality. The few workers who have used this approach to solve problems have usually used mutation alone, or have used mutation with sexual reproduction, ignoring all but the most blatant (and usually the least important) features of sexual reproduction.

Simon and Katovsky [64] and Feldman [24] have studied human extrapolation of sequences of symbols. Their goals in these studies were to understand human behavior and not to discover how to do very difficult extrapolations. The models they used for induction were very simple ones, and of only a little interest for the solution of difficult problems.

Foulkes [27] extrapolates sequences of symbols by using symbol transition probabilities from n-tuples of symbols that have occurred in the past. As the sequence being extrapolated grows in length, better statistical data is obtained for longer n-tuples. The value of n can vary for the various n-tuples, and is kept as large as is consistent with the amount of statistical data available for each n-tuple.

A system related to Foulkes' but much more complex is that of Uhr [77], which will be discussed in the next section.

The problem in induction is to find regularities in a body of data and use them for prediction. In psychology, the discovery of such regularities is called concept formation. In much research on human concept formation, the subject is given a set of objects that have been divided into two classes by some classification system unknown to him. The task of concept formation is to discover the unknown classification rule.

Hunt and Horland [37] work with very simple classification schemes, the most complex being the alternation of two

[1] Two things are "synergistic" if their occurrence together produces more beneficial effect than the sum of the benefits they give when they occur separately.

conjunctions of properties. An example would be "The set of all objects that are either red and large and sweet or blue and triangular." They found that human subjects readily discover concepts of this sort.

Kochen [40] has devised a program for discovering more complex concepts of the same general type as those used by Hunt and Horland—e.g., a classification scheme that is the alternation of five conjunctions of six binary properties.

Such a concept can be readily represented by a computer program that classifies short strings of randomly selected binary symbols in accord with this concept. If with each such random string, this classification information is given to Kochen's program, it will eventually determine (usually for a nonexhaustive set of examples) the exact nature of the classification system.

If there is any noise in the classification data given to Kochen's program, it will not find the proper concept and there is no clear way to modify it to deal with noisy data.

This is a very serious deficiency of Kochen's work. Very few induction problems in the real world are of the type that his programs deals with. On the other hand, the most complex sources of data can often be approximated by simple concepts corrupted by noise. These simple approximation concepts can then be modified to construct more complex concepts to fit the data better.

A program for discovering concepts such as the ones Kochen deals with, but that have been corrupted by noise, could be an important advance in artificial intelligence.

IV. Linguistics and Pattern Recognition

We would like to be able to communicate directly with an intelligent machine, i.e., to have our written English immediately transformed by the machine into a form that it could use. One may regard this input problem as an important kind of mechanical translation, from human langugage into a model within the machine that is used to store information in more usable form. I will not deal further with this particular problem but will refer to reviews by Simmons [63] on answering English questions by computer and by Bobrow [9] on syntactic analysis of English by computer.

V. M. Glushkov, in an address at the 1965 IFIP meeting in New York, mentioned an interesting kind of mechanical translation by induction, that was done in the Soviet Union. A small set of translation pair sentences from Hungarian to Basque were given as data to extrapolate from. A minimal (in some sense) machine was then constructed to transform the sample Hungarian sentences into the corresponding Basque. This minimal machine was then found to translate *new* Hungarian sentences into Basque with some accuracy. This work is much in the spirit of Fogel et al.

Another interesting inductive approach to mechanical translation is that of Faulk [21], [22]. Very loosely speaking, his idea is: Suppose we are given sentences A, B and C in English, with their corresponding translations a, b, and c in Russian. We note that if A is "similar" to B (in an as yet undefined sense) then a will tend to be "similar" to b. If we quantify the idea of similarity, then we expect S_{AB}, the similarity of A to B, to be about the same as S_{ab}.

Suppose we are given a new English sentence, D, and we want to find its translation. Then it is Faulk's idea that we should try to find a Russian sentence d, such that

$$S_{ad} \approx S_{AD}, \qquad S_{bd} \approx S_{BD} \quad \text{and} \quad S_{cd} \approx S_{CD}.$$

To find such a sentence, we start with a first approximation, and continue to make slight changes in it, retaining those changes that give a better fit—a kind of hill-climbing procedure.

Faulk has had success in translating simple sentences using this procedure and a relatively simple form for S. Whether it can be applied to more complex sentences will depend in part on what sort of characteristics the similarity criterion S is based on. The hill-climbing procedure can be faster than most, since by analyzing S, we can determine to some degree the cause of the "misfit," and make appropriate corrections.

Whether this method will ever give very good translations for much ordinary text is uncertain, but as a method of induction, it can be generalized to a great variety of problems.

Another induction system built around language translation is that of Uhr [77]. It is meant to be a preliminary model for a very general mechanism to learn the relationship between any given set of input-output pairs. In the models discussed, these input-output pairs are, for example, sentences in English and the corresponding sentences in French and the task of the program is to find a set of translation rules between them. The most advanced of the programs discussed forms strings of characters, which it regards as "patterns." At first, such patterns are formed of strings that have simple mappings into corresponding patterns of the translation language. Classes of such patterns are formed that can be used for ambiguity resolution of translations or as contexts to control other translation rules. These classes are concatenated with one another to form tentative new classes to be used in similar ways. Certain rules are given for inventing new translation rules and discarding others.

Many of the ideas suggested by Uhr seem very good. Two of his preliminary systems have been programmed. A third had not been at the time of his paper. The paper is rather unclear on many points, so it is difficult to make any detailed criticism of his most advanced system, but if he succeeds in satisfactorily implementing the ideas he mentions, he will have made an important step toward artificial intelligence.

Grammatical models may also be used for induction. Suppose one is given a set of strings of symbols, and the problem is to extrapolate this sequence—to devise a rule to determine if an arbitrary new string is a member of the set or not. Early approaches of Solomonoff [69], [70] to this problem, consisted of devising a grammar for a language in which the sample strings were "acceptable sentences." The idea of language was generalized somewhat, so that the translation *pairs* of sentences from two ordinary languages, constituted the "acceptable sentences" in a "translation language," which could then be used to translate between the two ordinary languages. An example was given of how such a language could have something like a phrase structure grammar.

These two papers tried to show how to find a set of grammar rules if one is given some initial sentences and a "teacher" who is allowed only to tell if a sentence proposed by the machine is acceptable or not. Later, Solomonoff ([73], pp. 240–253) devised a criterion for optimization of a grammar for a set of strings in which no "teacher" is available, with some suggestions as to how this grammar might be found. This approach obtains probability values for membership of arbitrary strings in the class to be extrapolated. Kirsch [39] also discusses the use of grammars for induction.

A good generalization of the grammar concept has been made by Narasimhan [48], who uses it for multidimensional pattern recognition. Preliminary work has been with recognition of letters and of nuclear events on bubble chamber photographs. Getting rid of "noise" on bubble chamber photos before using the grammar to recognize events has been an important initial problem for Narasimhan. Ideally, however, the grammar should be of sufficient power to be able to generate the noise as well as the signal. Nevertheless, tentative removal of noise before analyzing the data with respect to an approximate "noiseless" grammar might be a more practical approach to finding the optimum analysis for the more general, noise-including grammar.

Similar in spirit to Narasimhan is the approach of Eden [15] to the recognition of characters in cursive handwriting. He devises a set of elementary strokes and possible distortions of them, then asks how the observed writing could have been created from these strokes and distortions. This is equivalent to using a generative grammar to categorize the characters.

Uhr and Vossler [76] describe a pattern recognizing system that devises its own operators and then tries to assign optimum weights to them.

Their input patterns are in the form of characters or other two dimensional patterns and are represented on two dimensional 20×20 binary arrays. Patterns on the binary array are first centered to some extent, then processed by various 5×5 template operators. These operators are moved over all positions in the 20×20 array, and a "1" is assigned to those positions in which the templates match adequately. The resultant patterns of "1's" is blurred somewhat by describing it in terms of four moments. Classification (i.e., recognition) of an input pattern is made by comparing the four moments obtained for it using a given operator, with corresponding moments obtained on patterns of known classification. These comparisons are made for several operator types, and the judgments of each are given suitable weights to make a final classification.

Learning is of two types. First, the weight of each of the operators is slowly changed to reflect its effectiveness in classification—mindful of Samuel's work on checkers.

Another important kind of learning is in the program's devising its own operators or templates. The 5×5 templates have 25 trinary symbols in them—0, 1, and "don't care"—so there are 3^{25} such templates possible. It is not necessary to examine all of them to obtain useful ones, since only the template forms that actually occur as parts of input patterns are of any possible utility. A template is usually formed by selecting a 5×5 array at random, from the 20×20 input patterns. If it is found that this pattern is not good for pattern discrimination, it is discarded and a new template is selected.

The system had been originally designed for recognition of two dimensional graphic patterns and was fairly successful in this. It was then tried on two dimensional sound spectrograms of spoken words, and was to some extent able to classify them properly.

While the system does indeed improve its own performance in a useful manner, and it is at least as good as any other template matching system, it must be realized that only a very narrow range of pattern types can be classified by a single layer of template matching.

Much has been made of the system's effectiveness for speech recognition—a field in which it was not specifically designed to operate. A more conservative conclusion is that recognition of words through speech spectrograms is but another area where single layer template transformations are of value.

Template matching systems correspond to "rewrite rules" in mathematical logic. Any describable transformation can be expressed as a set of such rewrite rules, and any transformation can be expressed by a sufficiently large number of layers of template matching transformations.

Uhr and Vossler combine their template operators to make new operators that may conceivably be more powerful than single-layer template transformations, but their paper does not give enough detail to tell whether this is indeed so.

V. Hardware and Software

Since 1960 there have been continuing developments of both hardware and software. Some developments that have most affected artificial intelligence research are the continued decrease of cost per operation of computers, the development of list processing languages of many different sorts, the development of time shared facilities and related software, and the greater availability of very large fast memories.

The overall effect of these has been to make experimental research in this field less expensive, and much easier to implement for both initial programming and debugging. Larger fast memories have made possible larger, more complex programs than were feasible before.

It is my feeling, however, that much of the very important work that needs to be done is of a purely theoretical nature, and is quite independent of these more tangible developments.

VI. Other Views and Reviews

The opinions and predilections of the author have biased this review toward certain areas of artificial intelligence. I want to mention some other reviews, each having an emphasis somewhat different from my own.

The reference volume [79], edited by Feigenbaum and Feldman, contains a broad coverage of reprints of important papers on heuristic programming. Most of these are of the same general type as the research discussed in the present review, but papers on simulation of human behavior are also included. This volume has the important

Minsky [44] review, as well as his well-indexed bibliography with references up to 1962.

Feigenbaum's 1963 review [23] covers about the same type of research as the Feigenbaum-Feldman book, but discusses a larger number of papers rather briefly, concentrating on the period 1960 to 1963.

Newell and Ernst [51] again cover the same type of research but discuss the generality of the various methods used thus far in heuristic programming, and list the general heuristic types used by different researchers.

Pask's [55] long review is on self-organizing systems and artificial intelligence. A self-organizing system consists of a set of elements (typically these might be natural or artificial neurons, or some forms of automata) initially loosely organized. When placed in a suitable environment, the organization of the elements becomes more constrained in a manner that enables the system to solve various problems.

Although he is relatively uncritical of the research he discusses, Arbib [8] gives a good exposition of many of the ideas involved in Turing machines, finite automata, neural nets, reliable automata and information theory. He gives simple proofs for Gödel's theorem and for the unsolvability of the halting problem. It is often claimed that these two results demonstrate conclusively the necessary superiority of man's brain over any possible machine. It would be well for the worker in this field to be familiar with the exact nature of these theorems, so that he may judge for himself the validity of such claims.

Sebestyen's [61] book deals with a now popular approach to character recognition, in which classes of objects in hyperspace are formed by dividing it into sections with various hypersurfaces.

Abramson et al. [3] review pattern recognition and some types of machine learning for the period 1960 to 1963. The general methods of Sebestyen are discussed, as well as methods more narrowly applicable to recognition of two-dimensional characters and human speech. Both Sebestyen and Abramson et al. have extensive bibliographies.

Simmons [63] describes in some detail fifteen different systems for answering questions in English by computers. Bobrow's [9] review of computer syntactic analysis gives some background necessary for the work discussed by Simmons.

Eden's [16] review of human information processing covers the period 1959 to 1963. Prominent in this research are the methods of information theory, linguistics and statistical decision theory. The observation of "property filters" in the perceptile apparatus of various animals has confirmed some early theoretical models by Pitts and McCulloch [56].

Current Research and Development in Scientific Documentation 14 (Anonymous, [1]) contains reviews, usually by the principal investigator, of a very large number of research projects on various approaches to artificial intelligence and closely related studies in linguistics, mechanical translation, and information retrieval. The report contains extensive bibliographic references, is well indexed, and covers work in many foreign countries as well as in the United States.

Current journals and conference proceedings having frequent papers on artificial intelligence are:

1) *Information and Control*
2) *Proc. SJCC and FJCC* (Spring and Fall Joint Computer Conferences)
3) *Proc. IFIP* (International Federation for Information Processing)
4) *Journal of the Association for Computing Machinery* (J. ACM)
5) *Communications of the Association for Computing Machinery* (C. ACM)
6) *Behavioral Science*
7) *IEEE Transactions on Information Theory*
8) *IEEE Transactions on Electronic Computers*
9) *IEEE Transactions on Human Factors*
10) *IEEE Transactions on System Science and Cybernetics.*

Bibliography

[1] Anonymous, "Current research and development in scientific documentations, 14," National Science Foundation, Washington, D. C., Rept. NSF-66-17, 1966.

[2] P. Abrahams, J. Hansen, and M. Pivar, "Final report, research in sequence analysis," Information International, Inc., Cambridge, Mass., April 1965.

[3] N. Abramson, D. Braverman, and G. Sebestyen, "Pattern recognition," *IEEE Trans. on Information Theory*, vol. IT-9, pp. 257–261, October 1963.

[4] S. Amarel, "An approach to automatic theory formation," in *Principles of Self-Organization*, Von Foerster and Zopf, eds. New York: Pergamon Press, 1962.

[5] ——, "On the automatic formation of a computer program that represents a theory," in *Self-Organizing Systems*, Yovits, Jacobi, and Goldstein, eds. Washington, D. C.: Spartan Books, 1962.

[6] ——, "On the mechanization of creative processes," *IEEE Spectrum*, vol. 3, pp. 112–114, April 1966.

[7] ——, "On machine representations of problems of reasoning about actions," RCA Labs., Princeton, N. J., preliminary rept., 1966.

[8] M. Arbib, *Brains, Machines and Mathematics*. New York: McGraw-Hill, 1964.

[9] D. Bobrow, "Syntactic analysis of English by computer—a survey," *Proc. JFCC*, vol. 24, pp. 365–387, 1963.

[10] T. Booth, "Random input automata," presented at the International Conference on Microwaves, Circuit Theory and Information Theory, Tokyo, 1964.

[11] ——, "Statistical properties of random digit sequences," presented at the 7th Annual Symp. on Switching and Automata Theory, 1965.

[12] ——, "Random processes in sequential networks," *1965 Proc. IEEE Symp. on Signal Transmission and Processing*, pp. 19–25.

[13] C. Darlington, *The Evolution of Genetic Systems*. New York: Basic Books, 1958.

[14] R. M. De Baum, *Chem. and Engrg. News*, vol. 42, no. 25, 1964.

[15] M. Eden, "Handwriting and pattern recognition," *IRE Trans. on Information Theory*, vol. IT-8, pp. 160–165, February 1962.

[16] ——, "Human information processing," *IEEE Trans. on Information Theory*, vol. IT-9, pp. 253–256, October 1963.

[17] H. Eisner, *Operations Res.*, vol. 10, p. 115.

[18] G. Ernst, Ph.D. dissertation, to be published.

[19] T. Evans, "A heuristic program of solving geometric analogy problems," Ph.D. dissertation, Mass. Inst. Tech., Cambridge, Mass., 1963. Also available from AF Cambridge Research Lab., Hanscom AFB Bedford, Mass.: Data Sciences Lab., Phys. and Math. Sci. Res. Paper 64, Project 4641.

[20] ——, "A heuristic program to solve geometric-analogy problems," *1965 Proc. SJCC*, vol. 25, pp. 327–339.

[21] R. Faulk, "An inductive approach to mechanical translation," *Commun. ACM*, vol 1, pp. 647–655, November 1964.

[22] ——, "The phenomenon of interlingual correspondence: a quantitative formulation of the translation problem for natural languages," IBM Watson Research Center, Yorktown Heights, N. Y., res. rept.

[23] E. Feigenbaum, "Artificial intelligence research," *IEEE Trans. on Information Theory*, vol. IT-9, pp. 248–253, October 1963.

[24] J. Feldman, "Simulation of behavior in the binary choice experiment," in *Computers and Thought*, Feigenbaum and Feldman, eds. New York: McGraw-Hill, 1963, pp. 329–346.

[25] L. Fogel, A. Owens, and M. Walsh, "Artificial intelligence through a

simulation of evolution," in *Biophysics and Cybernetic Systems*, Maxfield, Callahan, and Fogel, eds. Washington, D. C.:Spartan Books, 1965.
[26] ——, *Artificial Intelligence Through Simulated Evolution*. New York: Wiley, 1966.
[27] J. D. Foulkes, "A class of machines to determine the statistical structure of a sequence of characters," *1959 IRE WESCON Conv. Rec.*, vol. 3, pt. 4, pp. 66–73.
[28] E. Fredkin, "Techniques using LISP for automatically discovering interesting relations in data," in *The Programming Language LISP*, Information International, Inc., Cambridge, Mass., March 1964.
[29] R. Friedberg, "A learning machine, pt. I," *IBM J. Res. and Dev.*, pp. 2–13, January 1958.
[30] R. Friedberg, B. Dunham, and J. North, "A learning machine, pt. II," *IBM J. Res. and Dev.*, pp. 282–287, June 1959.
[31] H. Gelernter, "Realization of a geometry theorem-proving machine," in *Information Processing*. Paris: UNESCO, 1960. Also in *Computers and Thought*, Feigenbaum and Feldman, eds. New York: McGraw-Hill, 1963.
[32] J. Hawkins, "Self organizing systems—a review and commentary," *Proc. IRE*, pp. 31–48, January 1961.
[33] A. Hormann, "Programs for machine learning," pt. I, *Information and Control*, pp. 347–367, December 1962.
[34] ——, "Programs for machine learning," pt. II. *Information and Control*, pp. 55–77, March 1964.
[35] ——, "How a computer system can learn," *IEEE Spectrum*, vol. 1, pp. 110–119, July 1964.
[36] ——, "Gaku, an artificial student," *Behavioral Sci.*, p. 88, January 1965.
[37] E. B. Hunt and C. I. Horland, "Programming a model of human concept formation," in *Computers and Thought*, Feigenbaum and Feldman, eds. New York: McGraw-Hill, 1963, pp. 310–325.
[38] T. Kilburn, R. Grimsdale, and F. Sumner, "Experiments in machine learning and thinking," in *Information Processing Proc. of ICIP, 1959*. Paris: UNESCO, 1960.
[39] R. Kirsch and B. Rankin, "Modified simple phrase structure grammars for grammatical induction," Nat'l Bureau of Standards, Washington, D. C., Rept. 7890, May 1963.
[40] M. Kochen, "Some mechanisms in hypothesis selection," in *1962 Proc. Symp. on Mathematical Theory of Automata*. Brooklyn, N. Y.: Polytechnic Press, 1963, pp. 593–614.
[41] K. Krohn and J. Rhodes, "Algebraic theory of machines," in *1962 Proc. Symp. on Mathematical Theory of Automata*. Brooklyn, N. Y.: Polytechnic Press, 1963.
[42] T. Marill, A. Hartley, D. Darley, T. Evans, B. Bloom, D. Park, and T. Hart, "Cyclops-I: a second generation recognition system," in *AFIPS Conf. Proc.*, vol. 24. Washington, D. C.: Spartan Books, 1963.
[43] J. McCarthy, "Programs with common sense," in *Mechanization of Thought Processes*, vol. I. London: Her Majesty's Stationery Office, 1959.
[44] M. Minsky, "Steps toward artificial intelligence," *Proc. IRE*, vol. 49, pp. 8–30, January 1961. Also in *Computers and Thought*, Feigenbaum and Feldman, eds. New York: McGraw-Hill, 1963.
[45] ——, "Descriptive languages and problem solving," *Proc. WJCC*, vol. 19, pp. 215–218, 1965.
[46] ——, "Matter, mind and models," in *Information Processing*, vol. I. Washington, D. C.: Spartan Books, 1965.
[47] M. Minsky and O. Selfridge, "Learning in random nets," in *Proc. 4th London Symposium on Information Theory*, C. Cherry, ed. London: Butterworths, 1961.
[48] R. Narasimhan, "Syntax-directed interpretation of classes of events," *Commun. ACM*, pp. 166–172, March 1965.
[49] A. Newell, "Some problems of basic organization in problem solving programs," in *Self-Organizing Systems*, Yovits and Cameron, eds. New York: Pergamon, 1963.
[50] ——, "Learning, generality and problem solving," in *Information Processing 1962*, C. M. Popplewell, ed. Amsterdam: North Holland, 1963.
[51] A. Newell and G. Ernst, "The search for generality," *Proc. IFIP Congress*, vol. I, W. Kalenick, ed. Washington, D. C.: Spartan Books, 1965.
[52] A. Newell and H. Simon, "The logic theory machine—a complex information processing system," *IRE Trans. on Information Theory*, vol. IT-2, pp. 61–79, September 1956.
[53] A. Newell, J. Shaw, and H. Simon, "Report on a general problem solver," in *Information Processing*.Paris: UNESCO. Also in *Computers and Thought*, Feigenbaum and Feldman, eds. New York: McGraw-Hill, 1963.
[54] ——, "A variety of learning in a general problem solver," in *Self-Organizing Systems*, Yovits and Cameron, eds. New York: Pergamon, 1960.
[55] G. Pask, "A discussion of artificial intelligence and self organization," in *Advances in Computers*, vol. 5, Alt and Rubinoff, eds. New York: Academic Press, 1964, pp. 110–218.
[56] W. Pitts and W. McCulloch, "How we know universals," *Bull. Math. Biophys.*, vol. 9, pp. 127–147, 1949.
[57] M. Pivar and M. Finkelstein, "Automation, using LISP, of inductive inference on sequences," in *The Programming Language LISP*, Information International, Inc., Cambridge, Mass., pp. 125–136, March 1964.
[58] A. Samuel, "Some studies in machine learning, using the game of checkers," *IBM J. Res. and Dev.*, vol. 3, pp. 210–229, 1959. Also in *Computers and Thought*, Feigenbaum and Feldman, eds. New York: McGraw-Hill, 1963.
[59] ——, personal communication.
[60] J. N. D. Scott and D. K. Hopkins, "Anticipating risks in research and development," *New Scientist*, pp. 159–161, January 21, 1965.
[61] G. Sebestyen, *Decision Making Processes in Pattern Recognition*. New York: Macmillan, 1962.
[62] O. G. Selfridge, "Pattern recognition and modern computers," *1955 Proc. WJCC*, pp. 91–93.
[63] R. Simmons, "Answering English questions by computer; a survey," *Commun. ACM*, pp. 53–70, January 1965.
[64] H. Simon and K. Katovsky, "Human acquisition of concepts for sequential patterns," *Psychol. Rev.*, vol. 70, pp. 534–546, 1963.
[65] H. Simon, "Experiments with a heuristic compiler," *J. ACM*, vol. 10, pp. 493–506.
[66] S. Slagle, "A heuristic program that solves symbolic integration problems in freshman calculus," *J. ACM*, vol. 10, pp. 507–520, 1963. Also in *Computers and Thought*, Feigenbaum and Feldman, eds. New York: McGraw-Hill, 1963.
[67] J. Slagle, "An efficient algorithm for finding certain minimum-cost procedures for making binary decisions," *J. ACM*, pp. 253–264, July 1964.
[68] ——, "A multipurpose theorem proving heuristic program that learns," Lawrence Radiation Lab., Livermore, Calif., UCRL-12342 Rev. II, AEC Contract W-7405-Eng-48, June 22, 1965.
[69] R. Solomonoff, "The mechanization of linguistic learning," in *Proceedings of the Second International Congress on Cybernetics*. Namur, Belgium: 1960, pp. 180–193.
[70] ——, "A new method for discovering the grammars of phrase structure languages," in *Information Processing*. Paris: UNESCO, 1960, pp. 285–289.
[71] ——, "Training sequences for mechanized induction," in *Self-Organizing Systems*, Yovits, Jacobi, and Goldstein, eds. Washington, D. C.: Spartan Books, 1961, pp. 425–434.
[72] ——, "A formal theory of inductive inference," pt. I, *Information and Control*, pp. 1–22, March 1964.
[73] ——, "A formal theory of inductive inference," pt. II, *Information and Control*, pp. 224–254, June 1964.
[74] G. Stebbins, *The Process of Organic Evolution*. Englewood Cliffs, N. J.: Prentice-Hall, 1966.
[75] F. Tonge, "A heuristic program for assembly line balancing," in *Computers and Thought*, Feigenbaum and Feldman, eds. New York: McGraw-Hill, 1963.
[76] L. Uhr and C. Vossler, "A pattern recognition program that generates, evaluates and adjusts its own operators," *1961 Proc. WJCC*. Also in *Computers and Thought*, Feigenbaum and Feldman, eds. New York: McGraw-Hill, 1963.
[77] L. Uhr, "Pattern string learning programs," *Behavioral Sci.*, vol. 9, pp. 258–270, July 1964.
[78] P. Van Heerdan, *A General Theory of Prediction*. Polaroid Corp., Cambridge, Mass., privately circulated report, 1963.

The following volumes each contain several papers from the preceding bibliography:

[79] *Computers and Thought*, E. Feigenbaum and J. Feldman, eds. New York: McGraw-Hill, 1963.
[80] *Self-Organizing Systems*, Yovits and Cameron, eds. New York: Pergamon, 1960.
[81] *Self-Organizing Systems*, Yovits, Jacobi, and Goldstein, eds. Washington, D. C.: Spartan Books, 1962.
[82] *Information Processing 1962*, C. M. Popplewell, ed. Amsterdam: North-Holland, 1963.
[83] *IEEE Trans. on Information Theory*, vol. IT-9, October 1963.