DESCRIPTIVE LANGUAGES AND PROBLEM SOLVING

Marvin Minsky
Dept. of Mathematics and Computation Center
Massachusetts Institute of Technology

Advances in machine problem solving may depend on use of internal languages for description and abstraction of the outcomes of experiments. As more complex problems are attempted there will have to be less trial and error and more systematic analysis of the results of each trial. Learning on the basis of experience will require a phase of refinement in which the machine will attempt, by analysis and inductive inference, to get as much as possible from each experiment.

Introduction

Work on artificial intelligence is proceeding at a slow, apparently steady rate. The complexity of problems being attacked is growing slowly, as is the complexity of the successful programs themselves. In the past it seems to have taken two or three years for each significant advance and one may ask why progress is so slow. Much of this time has been spent on the development of programming languages and systems suitable for the symbol manipulation processes involved. But much of the difficulty has been conceptual also. The methods which worked quite well on easy problems did not extend smoothly to the difficult ones. Continued progress will require implementation of new ideas, for there are some very tough problems in our immediate path. It seems to us that solution of these problems will require the use of non-trivial formal and descriptive language systems. These are only beginning to appear as a working part of the problem solving machinery and it will take much ingenuity to bring current notions into usable form.

The two papers of this session represent important, and very different, phases in the development of machine usable language systems. In one case we have a system which can process, to find the meaning with respect to a small universe, expressions in a very life-like fragment of ordinary language.

In the second paper we find an ambitious attempt at the beginnings of a Theory of Computation, based in part on the use of a symbol manipulation language suited at once for both theoretical analysis and for practical programming use.

Our purpose here is to indicate a few of the considerations that seem to point toward the incorporation of complex linguistic processes into the next generation of heuristic programs. Some of these difficulties have arisen in the author's work, jointly with McCarthy, on the Advice-Taker system.

In a recent paper the author discussed the principles and mechanisms of a variety of problem solving systems, but did not dwell on the question of extending these to really complex problems. We assume the terminology of that paper. When one attempts to apply the techniques described there one discovers that

1. The search problems become very serious. One is faced not only with greatly enlarged problem trees but also with a greater variety of plausible methods.

2. The problem of learning from experience becomes qualitatively more difficult. To learn the lesson of a complex experience requires shrewd, deliberate, analysis that cannot be approximated by any of the simple learning models based on averaging or on correlation.

3. The classification and pattern recognition methods must be on a descriptive level. Again, correlation or matching methods must be replaced by more sophisticated symbol-manipulation processes.

4. Planning methods, Character and Difference algebras, etc., threaten to collapse when the fixed sets of categories adequate for simple problems have to be replaced by the expressions of a descriptive language. The use of look-up tables for choosing methods will have to be supplemented by something more like reasoning.

When we call for the use of "reasoning" we intend no suggestion of giving up the game by invoking an intelligent subroutine. The program that administers the search will be just another heuristic program. Almost certainly it will be composed largely of the same sorts of objects and processes that will comprise the subject-domain.
programs. Almost certainly it will be recursively applied to itself so that the system can be finite. But it does seem clear that the basic (non-recursive) part of the structure will have to be more complex than is any current system.

The Need for Analysis

The simplest problems, e.g., playing tic-tac-toe or proving the very simplest theorems of logic, can be solved by simple recursive application of all the available transformations to all the situations that occur, dealing with subproblems in the order of their generation. This becomes impractical in more complex problems as the search space grows larger and each trial becomes more expensive in time and effort. One can no longer afford a policy of simply leaving one unsuccessful attempt to go on to another. For each attempt on a difficult problem will involve so much effort that one must be quite sure that, whatever the outcome, the effort will not be wasted entirely. One must become selective to the point that no trial is made without a compelling reason; just as in any research, expensive experiments must be carefully designed. One must do a good deal of criticism and analysis between experiments so that each will be a critical test of a significant portion of the search space.

The ability to solve a difficult problem hinges on the ability to split or transform it into problems of a lower order of difficulty. To do this, without total reliance on luck, requires some understanding of the situation. One must be able to deduce, or guess, enough of the consequences of the problem statement to be able to set up simpler models of the problem situation. The models must have enough structure to make it likely that there will be a way to extend their solutions to the original problem.

The construction of less difficult subproblems will be useful, by definition, only if one has already a very good chance of solving them efficiently. Otherwise the search tree will grow beyond bounds. This means we must have already built up adequate solution methods for the lower order problems, e.g., as a set of more or less packaged subroutines. This entails some formidable requirements:

Training Sequences

The machine is presumed to have acquired its good subroutines through earlier solution of less complex problems. (We are not interested here in the case in which these methods are provided at the start.) Thus the machine must have been exposed to a graded sequence of problems. To be sure, given time-limits, a machine will select a graded subsequence from an unorganized variety of problems. But a careful arrangement will be necessary to insure that methods learned in the problems that the machine does manage to solve will be useful on more difficult problems met later. In any case one cannot rely on making large jumps, either in machines or in humans.

Refinement Phase

Solving simpler problems is not enough. To make progress one needs also to "package" the successful method for effective later use. We are not interested in the trivial case of recognizing a problem once before solved, though this can be difficult enough when there is some disguise. The success must be generalized to cover a substantial variety of situations. To do this it would seem that there should be a phase of exploration and consolidation in which the successful method is refined—its central innovation (if any) isolated and packaged, in terms as general as possible. One must explore its range of application and construct an expression describing this range. This may involve inventing similar problems on which the method, or close variant, works; then constructing a plausible generalization.

Certainly people must go through such phases. One cannot usually solve hard problems with once-used but still unfamiliar methods. One must first "understand" the methods quite well; this means becoming able to recognize situations in which they are applicable. It is probably misleading to think of this as "practice"—acquisition of facility through repetition. Exercise in, e.g., mathematical technique is probably different from exercise in weight-lifting. Its effect is not so much in reinforcing methods, or paths already weakly laid down, but is rather to provide the necessary data for some Inductive Inference technique. The latter will replace the special method by one of somewhat greater generality.

Failure of the refinement phase to yield a precise, abstractly stated conclusion can be concealed to a point. One often encounters mathematical situations in which one can answer particular questions quickly, yet is unable to state a satisfactory formal generalization. This can happen through the assembly of a set of different models.
or examples which, as a group, show most or all of the features of the unformulated general theorem. One can answer some question in the negative, by finding inconsistency with an example. Consistency with all leads one to the affirmative. Often the examples themselves are not formulated clearly, or completely consciously. In such cases one will find some statements seem "obvious" yet (because of the incomplete understanding which precludes giving any precise explanation) are also felt to be "intuitive." An incomplete formalization or conceptualization, e.g., such a set of examples, can be very powerful when used at or near the top level. But if not understood or "packaged" it could become a serious nuisance later when, because of its informality, it cannot be used in deduction or in the construction of further abstractions.

**Coding and Retrieval Problems**

The compact representation of results of previous experience requires an adequate descriptive language. This language must permit general statements about both problem-domain matters and about the problem-solving methods. It must permit logical deductions to be made. This raises several problems.

One problem that has been a great nuisance to us arises in connection with non-mathematical problems in which actions affect the state of some subject domain. Thus a move affects the positions of pieces in a board game. When this happens, some statements formerly deduced about the situation cease to be true. (In a mathematical domain in which actions affect the state of some subject domain, such a set of examples, can be very powerful when used at or near the top level. But if not understood or "packaged" it could become a serious nuisance later when, because of its informality, it cannot be used in deduction or in the construction of further abstractions.) One must then deduce all the consequences of an action in so far as it affects propositions that one is planning to use. This might be done through some heuristic technique which can assess relevancy, or it could be done through a logic which takes such consequences into account. The trouble with the latter is that the antecedents of all the propositions must contain a condition about the state of the system, and for complex systems this becomes overwhelmingly burdensome. Other systematic solutions to the problem seem about equally repellent. It is a problem that seems urgently to require a heuristic solution.

Independent monitors are then set up to detect when such actions are proposed. The normal problem solving exploration process proceeds independently of these monitors, and is interrupted when one of them detects a threat to the proposition it is defending. This model has a certain introspectively attractive character; it suggests a free conscious exploration with more or less subconscious trouble-detectors. Unfortunately, its essentially parallel nature threatens to make its use in serial computer programming rather expensive. We hope someone will come up with a better idea.

In any case, the retrieval problem has to be faced. The problem of making useful deductions from a large body of statements (e.g., about the relevance of different methods to different kinds of problems) raises a new search problem. One must restrict the logical exploration to data likely to be relevant to the current problem. This selection function could hardly be completely built-in at the start. It must develop along with other data accumulated by experience.

Another rather serious problem centers around the problem of abbreviations, or proper names. The language must be used together with an abbreviative technique so that the most useful notions can be designated by reasonably convenient (short) representations. This is not only a matter of convenience and compactness; it is a more or less inescapable requirement of known inductive inference techniques and thus requires for formation of hypotheses or generalizations. Unfortunately an abbreviation cannot show all of the structure of the longer expression it designates. This seriously limits the possibilities of making formal logical deductions. Ultimately the machines will have to use mnemonic codings in their internal languages, just as we need to do this when we use their external languages.

The systematic solution to the abbreviation problem is, again, to revise the whole body of propositions in current use, in so far as they are going to be used in the same deductive operations. All the alternatives to this that we can envision are of somewhat stopgap nature. We content ourselves with the observation that it is equally a major problem for humans to make substantial changes in basic abstractions, ways of classifying or perceiving, and the like. Once one has built up a structure depending on a certain conceptual commitment, he will stake off a revision of its foundation as though the cost of changing
it were high. Otherwise, perhaps, people would not argue so much. One may view that phenomenon, if one likes, as a matter of ego involvement. But it would be well to remember that being wrong (and having to change) has a real intellectual cost, and not merely a social cost.

In any case, our ideas on this subject are not yet in presentable condition.

Conclusion

The need to be able to make abstractions in symbolic language is already urgent in current attempts to make machines prove theorems, play games, etc. There are some very difficult problems to be faced in this area. We still maintain, with McCarthy, that "in order for a program to be capable of learning something it must first be capable of being told it." Results on "self-organizing systems" without explicit provision for such abilities show very little promise to date, and systematic attempts in the direction of internal language processing should be promoted.

References
