AI Magazine Volume 4 Number 2 (1983) (© AAAI)

ON THE RELATIONSHIP BETWEEN STRONG AND WEAK PROBLEM SOLVERS

George W. Ernst
Computer Engineering and Science Department
Case Western Reserve University
Cleveland, OH 44106

Ranan B. Banerji
Department of Mathematics and Computer Science
St. Joseph's University
Philadelphia, PA 19131

Abstract

The basic thesis put forth in this article is that a problem solver is essentially an interpreter that carries out computations implicit in the problem formulation A good problem formulation gives rise to what is conventionally called a strong problem solver; poor formulations correspond to weak problem solvers Knowledge-based systems are discussed in the context of this thesis We also make some observations about the relationship between search strategy and problem formulation

DURING THE LAST DECADE the distinction between strong and weak problem solvers has been emphasized in the AI literature. Weak problem solvers are those that are relatively easy. Strong problem solvers, on the other hand, can solve relatively difficult problems but are specialized to a particular application domain. The usual explanation for the performance of strong problem solvers is that they can bring specialized knowledge from the application domain to bear on the problem. This distinction between problem solvers dates back to Newell (1969).

The question addressed in this article is "what is the relationship between weak and strong problem solvers?" One possible answer is that there are two different theories of problem solving: one for weak problem solving, the other for strong problem solving. We do not subscribe to this answer. However, if it is incorrect, there must be some relationship between the two that allows them to live harmoniously within a single theory. The nature of this relationship is the focus of this article. In passing we note that the theory of weak problem solvers has been well-developed for over a decade; see Nilsson (1971) for example.

This research was partially supported by the United States Air Force, RADC Contract F30602-82K-0045

MYCIN as a Weak Problem Solver

To start off the discussion, let us make a statement just to make a point: many expert systems can naturally be viewed as weak problem solvers As a concrete example, consider MYCIN (Shortliffe, 1976). Its state space is the set of atomic formulæ of the form

$$< \text{predicate function} > (< \text{object} >, < \text{attribute} >, < \text{value} >)$$

Each MYCIN production can be viewed as an operator in the problem reduction paradigm (Nilsson, 1971). For example, the production "if A & B then C" corresponds to the operator whose input state is C and whose output is the AND of the two states A, B. The goal states are patient data such as the results of lab tests.

Some aspects of MYCIN don't fit the problem reduction paradigm as naturally as the above, of course. For example, a

production whose action part is a conjunction of atomic formulae corresponds to a separate operator for each atomic formula in the conjunction. MYCIN's search strategy effectively applies such operators in a group. Certainty factors are best viewed as an extension of the problem reduction paradigm described in Nilsson (1971).

MYCIN's search strategy is a variation of the depth–first exhaustive search of AND / OR graphs described in Nilsson (1971). Descendents of "unsolvable nodes" are pruned, but descendents of "solved nodes" are not because MYCIN searches for multiple solutions.

Many knowledge–based systems have a similar translation into the problem reduction paradigm, particularly if it is extended by the addition of certainty factors. The thing that makes the translation so natural is that the knowledge base is often represented as a set of productions.

The above exercise shows that MYCIN is basically a weak problem solver, ignoring all of the human engineering that it possesses. That is, if we had a problem solver for the problem reduction paradigm which used a depth–first search and we formulated the MYCIN problem for it as described above, it should exhibit essentially the same problem solving capability as MYCIN. A similar analysis applies to many other knowledge–based systems. Why is it, then, that they are classified, and correctly so, as strong problem solvers?

Problem Formulation

The answer to this question is no surprise; it is the same explanation found in the literature. The productions contain lots of domain dependent knowledge to cope with special problem solving situations. However, we still have the apparent paradox that such problem solvers appear to be weak, according to the above analysis.

Our thesis is that what really makes a problem solver strong or weak is the problem formulation given to it. In the case of MYCIN, a medical diagnosis / therapy problem has been very carefully formulated for problem solving purposes. This causes the depth-first search employed by MYCIN to appear strong. An explicit statement of our thesis is:

> It is the formulation of a problem that causes a problem solver to appear weak or strong. The problem solver itself is merely an interpreter which carries out the computations implicit in the problem formulation

Hence, a good formulation of a problem gives rise to what is currently called a strong problem solver; weak problem solvers have poor problem formulations. It follows that a single problem solver can be weak for some problem formulations and strong for others.

Given this perspective let us look at the typical development of a knowledge– based system. After talking to an expert for some period of time, the knowledge engineer comes up with a formulation of the problem that looks reasonable to the expert. Actual use of this problem formulation gives results that the expert considers incorrect. The source of the difficulty is that the initial problem formulation is incorrect / incomplete. This is remedied by modifying some existing productions and adding some new productions.

Note that each production corresponds to a different operator in the problem formulation (see previous section). The operators of a problem or game correspond to our intuitive notion of the "rules of the game." Hence, what is happening is that the knowledge engineer is playing a game for which he doesn't know that rules. The expert knows the rules but he cannot tell them to the knowledge engineer, probably because some of the rules are in the right half of the expert's brain. So the knowledge engineer continues playing the game, making up the rules as he goes based on the advice of the expert. To be explicit each modification of a production modifies a rule of the game; adding a production adds a new rule to the game. The end result is an explicit statement of all of the rules of the game which is usually referred to as the system's knowledge base.

The above picture is consistent with Simon's (1973) contention that real world problem solving involves a lot of problem reformulation. In his model there are two basic kinds of activity: conventional problem solving (of the type described in Nilsson, 1971) and continual reformulation of the problem being solved, based on new information generated by the problem solving process.

The difference between Simon's model and the MYCIN effort is that the latter is attempting to generate a correct / complete formulation of the "medical game" once and for all This is probably a good deal of what is happening in medical school; students are trying to assimilate the rules of the "medical game." Note that when a person sees chess for the first time, it takes him a considerable amount of time to assimilate the rules of the game even though they are stated precisely. Of course, chess only has a few dozen rules as compared to the hundreds of rules in MYCIN.

Problem Formulations and Search Strategies

In what has gone above we have emphasized only one aspect of expertise / acquisition, i.e , that of getting a good formulation of the "rules of the game." This is not the only kind of problem dependent knowledge used by a problem solver. Often its search strategy also contains problem dependent knowledge There is a strong interaction between these two kinds of knowledge because, for example, a particular problem formulation may be appropriate for one search strategy and not for another.

MYCIN not only uses a problem formulation which is similar to that of a medical expert, it also uses a search strategy which is similar to that of the expert. Hence its problem formulation is appropriate for its search strategy The point is that one not only needs a good problem formulation, but a search strategy which fits the problem formulation must also be found. Using the problem formulation and search strategy of an expert is a very practical way to develop expert, limited domain problem solvers. Although this is an

important and very useful part of the AI, an equally important part is to understand the nature of intelligent processes. This section will emphasize the latter — how to find a problem formulation and a search strategy which fit one another.

Often we are given a problem formulation that is precise, correct and complete, but yet it is not appropriate for any reasonable search strategy. (We are ruling out strategies such as a table of all possible states together with their (optimal) solutions.) An example is Rubik's cube, a puzzle which is widely available, commercially. The difficulty is that the search space implicit in the given problem formulation seems to be too large and unstructured.

A typical strategy for solving Rubik's cube is

1 get the top plane correct;

2 get the middle plane correct;

3. get the corner cubes in the bottom plane in the correct position but not necessarily correct orientation;

4 get the remaining cubes in the bottom plane in the correct position but not necessarily correct orientation;

5 get the corner cubes on the bottom plane correct;

6 get the remaining cubes correct.

There are other strategies for Rubik's cube but all that we know of have the same form: get one set of cubes correct. Then get another set of cubes correct without changing the first set. Next get a third set of cubes correct without changing the first two, etc. Usually the first set is a face of the cube.

There is a difficulty in using such search strategies: none of the sets are invariant over most of the moves (operators) of Rubik's cube. The result is that in performing step i of the strategy, you undo the previous $i-1$ steps, unless you are very careful. To deal with this difficulty one develops sequences of moves which leave some of the sets of cubes unchanged. For example, the top plane is invariant over some move sequences. In applying such a move sequence, some cubes in the top plane are actually moved. But whenever this happens they are moved back to their original position before the end of the sequence.

At each step in the strategy, move sequences are used that leave the sets of cubes in the previous steps of the search strategy invariant. This implies that a sufficient number of such move sequences must be developed to allow an arbitrary initial state to be solved. Korf (1982) has written a program which generates such move sequences.

One can view this development of move sequences as a reformulation of Rubik's cube. In the new formulation each move sequence is a single operator. This formulation of the problem is good for the given search strategy if a sufficient number of new operators have been developed. At each step in applying the search strategy, the problem solver only uses those operators which leave the previous steps in the search strategy invariant. With the exception of the first step, most of the operators will be the new operators generated in reformulating the problem Different search strategies (e.g., different orders in which the cubes are fixed) may give rise to different reformulation because the search strategies may require different invariant properties for the operators.

This kind of problem reformulation was studied over a decade ago by Amarel (1970); he called it "changing the representation of a problem." His macro–moves are the move sequences described above.

The kind of strategy discussed above is a GPS (Ernst and Newell, 1969) based strategy. Each GPS difference corresponds to the set of cubes in a step of the strategy. The difference is present when one or more of the cubes in the set are incorrect. The differences are ordered by the steps in the strategy; i.e., the first difference to be removed corresponds to step 1; the second difference to be removed corresponds to step 2; etc.

So we see that the basic ideas in the above approach to solving Rubik's cube date back a long ways. Yet today we do not know how to automate such problem reformulations in a general way. Although Korf's (1982) method does a beautiful job in reformulating Rubik's cube, apparently it requires the differences to be state components such as the position of a cube. For some problems such simple differences are not sufficient. Goldstein has written a program (Ernst and Goldstein, 1982) which can discover more complicated differences that are appropriate for a given problem formulation. The limitation of Goldstein's program is that the given problem formulation may need to be changed. Such difficulties can only be avoided by looking for a good problem formulation and good differences at the same time.

Conclusion

The basic thesis put forth in this article is that a problem solver is essentially an interpreter that carries out computations implicit in the problem formulation. A good problem formulation gives rise to what is conventionally called a strong problem solver; poor formulations correspond to weak problem solvers. Of course, there is a whole spectrum of strength corresponding to how good the problem formulation is.

According to this view, much of what is conventionally called a system's knowledge base is really part of its problem formulation. This implies that research on knowledge-based systems is a form of research on problem formulation as opposed to what is conventionally called problem solving. We like this view because problem formulation is a "higher" conceptual level than that of problem solving In fact, we believe that this is the philosophical reason for the success / performance of knowledge–based systems — they focus on a higher conceptual level than previous work in AI. From a philosophical point of view it is very important to understand the basic nature of such research.

Problem formulation is central to research other than knowledge–based systems. Rubik's cube was used to exemplify the relationship between problem formulation and search strategy. This relationship must be taken into account

in looking for either a good problem formulation or a good search strategy. For this reason the mechanical discovery of either one should be done together with the mechanical discovery of the other.

References

Amarel, S. 1970. On the representation of problems and goal-directed procedures for computers. In R. B. Banerji and M D. Mesarovic (Eds.), Theoretical approaches to non-numerical problem solving. Springer-Verlag, 197–244.

Ernst, G.W. and Goldstein, M.M. 1982. Mechanical discovery of classes of problem-solving strategies. *Journal of the ACM* VOL:1-23.

Ernst, G.W. and Newell, A. 1969. GPS: A case study in generality and problem solving. New York: Academic Press.

Korf, R.E. 1982. A program that learns to solve Rubik's cube. *Proc of the National Conference on Artificial Intelligence* 164-167.

Newell, A. 1969. Heuristic programming: Ill-structured problems. In J. S. Aronofsky (Ed.) Progress in operations research. (Vol. III). New York: John Wiley and Sons, 361–414.

Nilsson, N.J. 1971. Problem-solving methods in artificial intelligence. New York: McGraw-Hill.

Shortliffe, E.H. 1976. Computer-based medical consultations: MYCIN, New York: North-Holland.

Simon, H.A. 1973. The structure of ill structured problems. *Artificial Intelligence* 4:181-202.