# A Theory of Scientific Problem Solving

Randolph Jones (RJONES@CIP.ICS.UCI.EDU)
Pat Langley (LANGLEY@CIP.ICS.UCI.EDU)
Irvine Computational Intelligence Project
Department of Information & Computer Science
University of California, Irvine CA 92717 USA

## Introduction

We are interested in computational explanations of the nature of human problem solving. In the past, many artificial intelligence (AI) systems have implemented problem solving in a problem–space framework (Newell, 1980). In this paradigm, a problem consists of an initial state, a goal state, and a set of operators that can be used to transform the initial state into the goal state. These systems have been moderately successful in providing a formal analysis of problem solving, but they fail to exhibit many aspects of human cognition. We have developed a theory of problem solving that accounts for some of these phenomena. In addition, we have built EUREKA, a system that instantiates this theory, and we have tested the system in a variety of domains, including scientific reasoning tasks.

## Characteristics of Human Problem Solving

We have developed our theory in an attempt to account for many of the characteristics of human problem solving. Thus, we will begin by discussing some of these characteristics.

*Heuristic methods.* Humans do not attack problems blindly. In particular, when a person encounters a new problem, he will not start applying all his knowledge in random patterns until he solves it. Rather, he uses heuristics, or educated guesses and rules of thumb, to guide his search for a solution. One type of systematic heuristic problem solving that has seen some success in AI is *means–ends analysis* (Ernst & Newell, 1969; Fikes & Nilsson, 1971). At each decision point, these systems attempt to apply an operator that reduces the differences between the current problem state and the goal. In this way, the search for a solution is directed down promising paths.

*Non-systematic nature.* However, humans do not solve problems in a very systematic manner. If a person finds himself stuck at some point, he can usually not remember all the steps he took in reaching that point. In many cases, he will simply start the problem again from the beginning, often duplicating previously failed paths in his new attempts. In contrast, most of the AI work on problem solving has employed memory-intensive methods, such as depth-first and best-first search. These techniques assume a large memory in which they can store all previous goals and states. Using this memory, they can 'backtrack' to any earlier point in the problem, as well as avoid duplicating past failures. Ohlsson (1987) has studied the non-systematic nature of human problem solving, but most current systems will attempt to explore their entire problem space systematically if they cannot find a solution.

*Performance improvement and Einstellung.* Humans learn while they solve problems. One aspect of this learning involves improved performance. This can be seen when a human

transfers knowledge and methods from a previous problem to solve a new problem more easily. In general, we expect humans to get better as they solve a set of similar problems (Ohlsson, 1987), but there are also instances when learning causes negative transfer. One example of this is *Einstellung*, or the 'set' effect. In this case, a person has become so used to solving problems in a certain way that he ignores a much simpler solution (Luchins, 1942; Neves & Anderson, 1981).

*Response to external stimuli; Insight.* Finally, humans are influenced by their environment. External cues can often aid a person in solving a problem (Dreistadt, 1969). Cues can also cause people to experience flashes of *insight*. Hadamard (1949) has detailed four stages in episodes of scientific insight. The first stage is *preparation*, in which a scientist works on a problem for some time with out success. When he gets frustrated and ceases work on the problem, he enters the *incubation* stage. Some time later (from a few seconds to a few years), *illumination* or insight occurs, during which a potential solution suddenly pops into the scientist's head. Finally, during *verification*, he works out the details of his solution. We have argued elsewhere (Langley & Jones, 1988) that illumination occurs when an environmental cue causes the scientist to suddenly retrieve an operator which will aid in solving his problem.

## The EUREKA System

EUREKA is a running LISP program designed to model some of the processes of scientific discovery and problem solving. It consists of a memory component, a problem solver and a simple learning mechanism. The memory component can be described at various levels of abstraction, so we will begin with a low-level description and then discuss the higher level data representation as it applies to problem solving.

### Memory representation and retrieval

EUREKA includes a long-term memory, represented as a semantic network consisting of nodes (concepts) connected by a small set of labeled links (relations). Each link has an associated *trace strength*, which represents the strength of the connection between the two attached nodes. For example, a link connecting 'bird' to 'wings' would probably have a larger trace strength than a link connecting 'bird' to 'legs'. EUREKA does not embody the notion of a specific short-term or working memory. However, each node also has associated with it a *level of activation*, which exhibits how much attention the concept receives during problem solving. Using this representation, retrieval of concepts is implemented as a form of *spreading activation* (Quillian, 1968; Anderson, 1976, 1983).[1] When a node is activated, it 'spreads' its activation to nearby nodes in the semantic network. As activation spreads from a node, this activation is divided up between all the connected nodes in proportion to the trace strengths of the links involved.

EUREKA's memory is further organized into conceptual units that are used by the problem-solving component. Each of these units consists of a collection of nodes in the semantic network. These collections include operators, problem-space states, and deriva-

[1] The type of spreading activation we use is a bit different from that introduced by Quillian, in which activation was used to find pathways between two concepts. Our approach is more similar to that used by Anderson (1976) and by Holland, Holyoak, Nisbett, & Thagard (1986) in their work on analogy.

tional trace structures. Operators and problem-space graphs are well-documented concepts in problem solving (Newell, 1980), but the third notion bears a more in-depth explanation. We have borrowed the idea of derivational traces from Carbonell (1986). These are records of problem-solving episodes in which information is stored about the system's goals and its reasons for making certain choices. Derivational traces can be used to remember the details of previous attempts to solve problems and to aid in solving similar problems.

In EUREKA, a derivational trace is represented as a tree. Each node in the tree represents a *goal*, and its children represent the subgoals that must be satisfied in order to achieve that goal. The system distinguishes between two types of goals: TRANSFORM a problem-space state into another state, or APPLY an operator to a problem-space state (Ernst & Newell, 1969). If a node in the derivational trace has no children, it means that either no subgoals were required to satisfy the goal (success) or no subgoals could be found which would help satisfy the goal (failure). During problem solving, activation is spread throughout the derivational trace structure and the rest of memory to aid in choosing operators.

### Problem-solving component

EUREKA's basic problem-solving method is means-ends analysis, similar to that used in GPS (Ernst & Newell, 1969) and STRIPS (Fikes & Nilsson, 1971). A TRANSFORM goal can be satisfied by first applying an operator (i.e., setting up an APPLY goal) and then recursively TRANSFORMing the result. An APPLY goal can be satisfied by TRANSFORMing the current problem-space state to match the preconditions of the operator to be applied, and then APPLYing the operator to the resulting state. However, there are a few important differences.

Given a TRANSFORM goal, STRIPS would exhaustively search its set of operators and choose the 'best' one (using means-ends analysis) to APPLY. In contrast, EUREKA retrieves a small set of operators by spreading activation throughout its memory from nodes representing the current problem-space state and goal. The system then passes these operators on to a STRIPS-like matcher to decide which ones might be useful. The remaining operators are weighted according to how easily they were retrieved and how useful they have been in the past. Finally, the system selects a single choice at random based on these values. If no useful operators are found for a given goal, the problem solver fails.

Another important difference from STRIPS-like problem solvers is that EUREKA does not have the ability to backtrack. Instead, when the system fails to solve a problem, it starts over from the initial TRANSFORM goal for that problem. EUREKA attempts to solve the problem repeatedly, but it may duplicate previous problem-solving paths in the process. The model continues working until the problem is solved or until it becomes 'frustrated' and quits. Frustration occurs when the initial goal has a very high failure rate. Table 1 summarizes the the system's problem-solving component.

### Learning and memory maintenance

In addition to standard problem-solving actions, EUREKA maintains a large derivational trace structure in its long-term memory. Whenever the system encounters a new situation during problem solving (e.g. a new problem-space state or a new derivation-trace node), it adds this situation to the derivational trace. In this fashion, a record of all previous problem

Table 1. EUREKA's basic problem-solving algorithm.

```
TRANSFORM(State1,State2)
If State1 satisfies State2
     Then Return(State1)
     Else Let Remindings be a set of instantiated operators retrieved
               through spreading activation;
          Let OpA be an operator selected at random from any
               Remindings that reduce differences between State1 and State2;
          If OpA is empty
               Then Return(Fail)
               Else Let State3 be APPLY(OpA,State1);
                    If State3 is Fail
                         Then Return(Fail)
                         Else Return(TRANSFORM(State3,State2))

APPLY(OpA,State1)
If OpA can be applied to State1
     Then Return(EXECUTE(OpA,State1))
     Else Let State2 be TRANSFORM(State1,Preconditions(OpA));
          If State2 is Fail
               Then Return(Fail)
               Else Return(APPLY(OpA,State2))
```

solving behavior is stored. If a new situation has already been stored in memory, the trace strengths of the links involved in that situation are increased slightly. In addition, special actions are taken upon the success or failure of a goal. Counts are kept to record how often each goal has succeeded or failed, and when a goal succeeds the trace strengths of the nodes involved in the goal are increased. These counts are used to estimate the probability that the goal will succeed or fail in the future.

## Evaluation of the EUREKA Model

We have discussed a number of phenomena that our theory should handle. In order to test the theory, we have implemented it in the EUREKA system. Further, we have designed a number of problems in various domains to test the system along these lines. In this section we describe some experiments we have run with respect to the human problem-solving characteristics we described earlier.

*Non-systematic, heuristic methods.* EUREKA uses a variant of means-ends analysis that does not have the ability to backtrack. In addition, it uses spreading activation and counts of previous failures and successes to aid in conflict resolution. Since trace strengths are updated when familiar situations are encountered, the system can get stuck repeating old, unsuccessful behavior. However, the system tries to avoid previously failed states, so it can break out of this behavior. This encourages the system to explore a wide area in the problem space. We tested our system on a number of small 'blocks world' and 'chemical structure'[2] problems. EUREKA solved these problems after a small amount of exploration. When given more complicated problems (with no previous problem-solving memory), the system could not overcome the large problem space. It explored a large section of the space, but could

[2] These problems are based on Kekulé's problem of determining the structure of various molecules including benzene (Farber, 1966).

Table 2. 'Blocks-world' problems with two blocks.

| Problem Goal | Length of Optimal Solution | Number of Attempts | | Percentage of Space Searched | |
|---|---|---|---|---|---|
| | | Without learning | With learning | Without learning | With learning |
| A over Table | 3 | — | 1 | — | 5.0 |
| B over Table | 3 | — | 1 | — | 5.0 |
| A over B | 5 | — | 1 | — | 8.3 |
| A on B | 7 | 10 | 1 | 65.0 | 15.0 |

Table 3. 'Blocks-world' problems with three blocks.

| Problem Goal | Length of Optimal Solution | Number of Attempts | | Percentage of Space Searched | |
|---|---|---|---|---|---|
| | | Without learning | With learning | Without learning | With learning |
| A over Table | 3 | — | 3 | — | 6.2 |
| B over Table | 3 | — | 1 | — | 0.6 |
| C over Table | 3 | — | 1 | — | 0.6 |
| A over B | 5 | — | 2 | — | 4.0 |
| B over C | 5 | — | 1 | — | 1.0 |
| B on C | 7 | — | 1 | — | 1.4 |
| A over B on C | 11 | — | 1 | — | 2.2 |
| A on B on C | 13 | 10 | 1 | 30.0 | 2.6 |

not find the solution paths.

*Performance improvement.* EUREKA does have the ability to improve its performance based on previous experiences. We gave the system the same sets of 'blocks-world' and 'chemical-structure' problems. However, this time we ordered the problems from simplest to hardest and did not erase the system's memory after each problem. In this case, the system was able to solve all problems presented to it within three attempts. Tables 2 and 3 provide data from runs in the 'blocks-world' domain. Within each table, the initial states are the same. The optimal solution length is the size of the smallest derivation trace required to solve the problem.

*Einstellung.* Our theory explains Einstellung in terms of the trace strengths on links in the semantic network and the success counts kept for each state. Recall that trace strengths are increased whenever a goal is satisfied. This causes the system to retrieve the successful operators in similar situations. Combined with the record of success counts, this encourages the system to duplicate past successful behavior in new, similar situations. To test this effect, we gave EUREKA a series of 'water jug' problems (Luchins, 1942). The first few problems

required similar solution paths, and EUREKA exhibited improvement in duplicating this path with each new problem. The last problem could be solved using the same solution, or by using a unique solution that required only one operator application. The system chose to duplicate the solution path it had become familiar with from the previous problems, as humans often do in such situations.

*Response to external stimuli.* As shown in the earlier, there are times when EUREKA cannot solve a given problem, even if it has all the appropriate operators stored in memory. This can arise because the problem space is too large or because the appropriate operators are never retrieved from memory. However, the system can solve these problems in the presence of the appropriate external cues. We tested the system with the previous difficult problems, and with a problem simulating Archimedes' discovery of the principle of displacement[3] (Dreistadt, 1968). During each of these problems we activated useful concepts in long-term memory. The activation from these concepts caused the appropriate operators to be chosen to solve the problems. The system was thus able to solve problems it could not normally solve without cues from the environment. We feel this provides an initial account of the illumination stage of scientific insight. These experiments exhibit how a person might be unable to solve a problem and then suddenly discover a solution.

## Discussion and Future Work

We have presented a number of characteristics of human problem solving and a theory that accounts for them. These include heuristic and non-systematic problem solving, performance improvement, Einstellung, and stimulus-driven problem solving. Our theory explains these characteristics in terms of memory retrieval. We have implemented this theory in a computer simulation called EUREKA, and have tested its behavior in a number of situations. The results of these studies indicate that the model accounts for the characteristics we have discussed. We feel these results lend support to our explanation of human problem solving in terms of memory retrieval.

We plan to extend our theory and the EUREKA system to account for analogical problem solving and episodes of insight. We have shown that spreading activation has certain properties that allow transfer in problem-solving behavior. We believe that the same properties can be exploited to suggest and elaborate analogical solutions based on previous problems. We also believe that we will be able to account for insightful problem-solving experiences like those of Kekulé, Archimedes, and Darwin (Barlow, 1959). Elsewhere (Langley & Jones, 1988) we have described insight as a combination of effects from analogy and memory-limited problem solving. We plan to build these ideas into the EUREKA system, and we are hopeful that the results of future experimentation will further support our theory.

[3] In this problem, the goal is to prove that a crown is made of pure gold. The solution method involves determining the volume of the crown and comparing that to the volume of a gold brick of the same weight. However, the crown's volume cannot be measured by melting it down, because that would destroy the crown. Archimedes' key operator, which resulted in the formulation of the principle of displacement, was to measure the volume of the crown by immersing it in water and measuring the amount of water displaced.

## References

Anderson, J. R. (1976). *Language, memory, and thought.* Hillsdale, NJ: Lawrence Erlbaum.

Anderson, J. R. (1983). *The architecture of cognition.* Cambridge, MA: Harvard University Press.

Barlow, N. (1959). *The autobiography of Charles Darwin.* New York: Harcourt Brace.

Carbonell, J. G. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach, volume 2* (pp. 371–392). Los Altos, CA: Morgan Kaufmann.

Dreistadt, R. (1968). An analysis of the use of analogies and metaphors in science. *The Journal of Psychology, 68*, 97–116.

Dreistadt, R. (1969). The use of analogies and incubation in obtaining insights in creative problem solving. *The Journal of Psychology, 71*, 159–175.

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

Farber, E. (1966). Dreams and visions in a century of chemistry. In R. F. Gould (Ed.), *Kekulé centennial* (pp. 129–139). Washington, DC: American Chemical Society.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence, 2*, 189–208.

Hadamard, J. (1949). *The psychology of invention in the mathematical field.* Princeton, NJ: Princeton University Press.

Holland, J. H., Holyoak, K. J., Nisbett, R. E., & Thagard, P. R. (1986). *Processes of inference, learning, and discovery.* Cambridge, MA: MIT Press.

Langley, P., & Jones, R. (1988). A computational model of scientific insight. In R. Sternberg (Ed.), *The nature of creativity* (pp. 177–201). Cambridge, England: Cambridge University Press.

Luchins, A. S. (1942). Mechanization in problem solving: The effect of Einstellung. *Psychological Monographs, 54*(248).

Neves, D. M. & Anderson, J. R. (1981). Knowledge compilation: Mechanisms for the automatization of cognitive skills. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 57–84). Hillsdale, NJ: Lawrence Erlbaum.

Newell, A. (1980). Reasoning, problem solving, and decision processes: The problem space hypothesis. In R. Nickerson (Ed.), *Attention and performance VIII.* Hillsdale, NJ: Lawrence Erlbaum.

Ohlsson, S. (1987). Transfer of training in procedural learning: A matter of conjectures and refutations? In L. Bolc (Ed.), *Computational models of learning* (pp. 55–88). Berlin: Springer-Verlag.

Quillian, M. R. (1968). Semantic memory. In M. L. Minsky (Ed.), *Semantic information processing.* Cambridge, MA: MIT Press.