A Model of Retrieval in Problem Solving

Randolph Jones*
Department of Information and Computer Science
University of California, Irvine, CA 92717

Dissertation
submitted in partial satisfaction of the requirements for the degree of
Doctor of Philosophy in Information and Computer Science

Technical Report 89–27

September 15, 1989

* Current address: Department of Psychology, Carnegie Mellon University, Pittsburgh, PA 15213

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Technical Report No. 14	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* A Model of Retrieval in Problem Solving		5. TYPE OF REPORT & PERIOD COVERED Interim Report 8/88–8/89
		6. PERFORMING ORG. REPORT NUMBER UCI–ICS Technical Report 89–27
7. AUTHOR*(s)* Randolph Jones		8. CONTRACT OR GRANT NUMBER*(s)* N00014–84–K–0345
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Information & Computer Science University of California, Irvine, CA 92717		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Computer Science Division Office of Naval Research Arlington, Virginia 22217		12. REPORT DATE September 15, 1989 13. NUMBER OF PAGES 99
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

Doctoral dissertation.

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

cognitive modeling
analogical reasoning
spreading activation
machine learning
memory retrieval
problem solving

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

OVER

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

20. ABSTRACT

In this research, we describe a model of memory and retrieval in problem solving, called EUREKA. This model incorporates a means-ends-analysis style of problem solver with a retrieval mechanism that is based on spreading activation. The model also includes a flexible method of operator selection that allows information from old problems to be analogically mapped onto new situations. Learning in EUREKA consists of storing a record of past problem solving behavior, updating trace strengths on links in memory, and recording successes involved in applying old knowledge to new situations. The latter mechanisms influence the retrieval and selection of information from memory, resulting in various forms of improved problem-solving performance. EUREKA is intended to be a psychological model, and we have run a number of experiments to verify that it accounts for various behavioral phenomena, including the ability to improve performance with practice on individual problems and to transfer knowledge within and across domains. The model also provides an explanation for episodes of negative transfer, or *Einstellung*, and the role of external cues in the environment. Our results indicate that a model of retrieval in terms of spreading activation, coupled with a problem-solving engine, provides a plausible explanation of human problem-solving behavior. In addition, the model provides an efficient method for retrieving operators from memory and a mechanism for the retrieval of analogies, both of which increase the problem solver's flexibility and performance capabilities from a computational standpoint.

Contents

List of Tables

List of Figures

Acknowledgements

There are many people who deserve my thanks for their role in helping me through my career at UCI. First, I would like to thank my committee members: Paul O'Rorke, Dennis Kibler, and particularly Pat Langley for helping to provide me with the knowledge, ideas, and challenges I needed to make a significant contribution to my field. I would also like to thank all the other members of the machine-learning research community at UCI for numerous discussions, comments, questions, arguments, etc. throughout the years. I would specifically like to thank Bernd Nordhausen (without whom this dissertation could never have been finally wrapped up), Don Rose, David Ruby, Piew Datta, David Aha, David Schulenburg, Jim Kipps, Tim Cain, Steve Morris, Jim Wogulis, John Gennari, Rogers Hall, John Allen, and José Ambros Ingerson. Whether you know it or not, you have all made a contribution to the ideas or details of this research. In addition, I thank Dennis Volper for providing me with the program to typeset my graphs.

My friends at UCI, including Jim Fradkin, Tedd Hadley, Nels Vander Zanden, Frank and Amy Vahid, Allen Wu, and many of those listed above made graduate life enjoyable (and probably extended it by a year or two), providing many hours of volleyball and various card games. Finally, I would like to thank my parents, Walter and Phyllis Jones, for making this all possible through their love, support, and twenty-one years of funding; and my wife, Sandy Brockman, for making everything worth doing (and putting up with a part-time husband for a year).

This research was supported in part by contract N00014–84–K–0345 from the Computer Science Division, Office of Naval Research, and a University of California Regents' Dissertation Fellowship.

CHAPTER 1

The Task and the Paradigm

Goals of the research

The principal goal of this research is to construct and test a model of the role of memory and retrieval in problem solving. We intend this model to account (at least qualitatively) for psychological results concerning human problem-solving behavior. We also wish to understand more about the factors that affect problem-solving ability in humans. We wish to examine the conditions under which humans can or cannot solve problems, and to study those cases in which they use creativity or past experiences to increase their ability. We have tested our model by implementing it as a running computer program and showing that it behaves similarly to humans under various conditions. We use the model and its implementation to explain how humans are able to perform certain tasks, what causes some of the troubles they encounter, and how they overcome those problems.

An additional aim of the research is to provide suggestions for the design of computerized problem-solving systems. We feel that a good method for creating intelligent systems is to examine the processes that account for intelligent behavior in humans and use explanations of that behavior to guide our efforts in designing artificial-intelligence (AI) systems. In this framework, a first step toward creating intelligent systems is to develop models that explain human behaviors. If we can develop a reasonable model, we will be able to incorporate the processes involved into future problem-solving systems. Hopefully, these new systems will be more creative, flexible, and powerful than the systems that have been constructed and studied to date.

Artificial intelligence and problem solving

The subject of problem solving has been studied from a computational standpoint since the earliest days of artificial-intelligence research. One of the first problem-solving systems that was able to solve a large class of problems was called GPS, for General Problem Solver (Ernst & Newell, 1969). This was the first system to introduce the notion of *means-ends analysis* in problem solving. A later system, STRIPS (Fikes & Nilsson, 1971), expanded on the means-ends approach. Since that time there has been a wealth of research covering the

issue of problem solving. In general, this work has advanced along two dimensions. There are some who have attempted to model the behavior of humans at various levels and for various tasks (e.g., Anderson, 1976, 1983; Laird, Rosenbloom, & Newell, 1986a; Ohlsson, 1987). Others have been content to downplay psychological issues and concern themselves more with building computer systems that are "better" problem solvers in computational terms (e.g., Fikes & Nilsson, 1971; Hendler, 1986; Minton, 1988/1989; Sacerdoti, 1974).

A major paradigm for researchers in computational problem solving (and artificial intelligence in general) involves the *problem-space hypothesis* proposed by Newell (1980). The hypothesis states that all cognitive behavior can be expressed in the form of *search*. A problem space consists of a set of problem states and a set of operators that can apply to these states. Given any initial situation (or initial state) for a cognitive system that has certain goals (goal states), the system carries out a search for a path from the initial state to a goal state by applying operators appropriate to the task. These operators carry the system through various intermediate states in the process. The search problem is to find a path (or the best path) from the initial state to the goal state. This task is complicated by the fact that multiple operators may be applicable to any given state, which makes it is easy to follow spurious paths. The problem-space hypothesis has proved to be an extremely useful model in AI, and problem solving in particular is very nicely explained in these terms. As such, we have relied heavily on the hypothesis in our representation of problems.

Modeling retrieval in problem solving

Many systems have been developed that search problem spaces by applying operators and expanding new states until they find a path between the initial state and a goal state. These systems might be said to be psychological models to the extent that they rely on the problem-space hypothesis, which appears to provide a valid model of human cognition. However, along numerous dimensions, these systems exhibit behaviors and limitations that are very different from those exhibited by humans. In this research we account for a set of problem-solving behaviors by integrating a psychologically plausible model of memory and retrieval with a problem-solving system.

An important aspect of the role of retrieval in problem solving concerns the availability of information. Humans have a large capacity for information in long-term memory, and difficulties occur in bringing only the relevant information to bear on a given problem situation. In our view, the retrieval of information from memory plays a central role in one's ability to solve a problem.[1] In addition, any mechanisms that influence retrieval should have a strong effect on the types of learning that can occur. In our model, one

[1] This is by no means a unique view in AI research. Schank (1982), Carbonell (1986), and others have also argued for the strong role of memory in problem solving.

of the most important forms of learning involves the adjustment of memory structures to alter retrieval patterns. These adjustments are designed to facilitate the retrieval of useful information, although we will see that there is sometimes a trade-off between flexibility and performance improvement.

Retrieval is also an important issue from a computational standpoint. Most problem-solving systems have assumed that all the knowledge stored in memory can be brought to bear on a problem. However, like humans, machines can store extremely large amounts of information and it is often unreasonable to exhaustively analyze the entire memory until the most relevant information can be selected.[2] In this respect, a good retrieval mechanism must be able to recall relevant information without letting the amount of retrieved information become too large. Once again, this type of mechanism involves certain trade-offs due to the imposed restrictions and constraints.

Contents of the dissertation

This dissertation describes a model of the role of memory and retrieval in problem solving, together with an implementation and evaluation of the model. This section provides a guide for readers who are interested in various specific aspects of this research.

Chapter 2 describes a number of characteristics of human problem solving that have been explored by psychologists, including performance aspects, learning, and creativity. These characteristics provide a general framework in which we develop our model and a set of phenomena that the model must explain.

The third chapter provides a discussion of *spreading-activation* models of memory retrieval. Spreading activation is the retrieval mechanism used in the current work and it is central to our model of problem solving. In this chapter we discuss spreading-activation models in general and give arguments for the utility of spreading activation from both psychological and computational standpoints.

In chapter 4 we introduce and describe in detail the EUREKA model of problem solving. The chapter begins with an overall view of the model and then discusses specific details concerning the representation of knowledge, performance and retrieval, and learning. Sections of this chapter also discuss many of the details involved in implementing EUREKA as a computer system.

Chapter 5, which contains an experimental evaluation of the EUREKA model, is divided into two main sections. The first reports tests of EUREKA's adequacy as a psychological model. The experiments in this section are designed to account for the characteristics discussed in chapter 2. The second section contains an empirical study of the model as a

[2] This becomes an important problem when considering high-performance problem solvers that work in large and complicated domains. Our current model does not address problems of this complexity, but provides suggestions for improving those that do.

computational system. Experiments in this section test EUREKA's behavior under various settings of its system parameters.

In chapter 6 we discuss our model in relation to some of the past research in problem solving in artificial intelligence. To this end, we identify and analyze a number of important features concerning problem-solving systems. We classify some past research along these dimensions and describe EUREKA's relationship to that research.

Finally, in chapter 7 we review the contributions of the dissertation and discuss the EUREKA model in terms of our experimental results. We also describe a number of directions in which we would like to extend our model, along with some other directions for future related research.

CHAPTER 2

Human Problem Solving

Much of the work in AI on problem solving has been based on the methods humans use to solve problems. Some early work was motivated by a desire to model human behavior with problem-solving systems (Ernst & Newell, 1969; Newell & Simon, 1972). Other researchers have been concerned with computational utility rather than psychological principles. Their work focuses on getting computers to solve problems in interesting and general ways, without necessarily addressing issues of psychological plausibility (Fikes & Nilsson, 1971; Hendler, 1986; Minton, 1988/1989; Sacerdoti, 1974). Recently, there has been a resurgence of interest in developing computational implementations of psychological models in an attempt to explain human problem-solving behavior (Anderson, 1983; Newell & Rosenbloom, 1981; Ohlsson, 1987).

The main focus of this dissertation is the integration of a memory retrieval mechanism with a general problem solver in an attempt to account for a number of behavioral phenomena. We have developed a model of human problem solving, along with a computer implementation, that accounts for these phenomena. Later chapters of this dissertation describe the details of our model, its implementation, and its behavior under various conditions. However, before we discuss these details, we introduce a number of characteristics of human problem solving that have been observed and studied by psychologists. These characteristics serve as the basis for the construction and evaluation of our model.

Influences and limitations on problem-solving behavior

One topic of interest in cognitive psychology involves the qualitative aspects of simple problem solving. This concerns the general characteristics that humans exhibit in mundane problem-solving episodes. The characteristics described here include the general approaches that humans take in solving problems and some limitations in their ability.

Use of heuristics

We have already described Newell's problem-space hypothesis, which claims that solving a problem consists of carrying out a search through the states in a problem space. There are many possible methods for searching such a problem space. However, problem spaces

are usually quite large, and humans must use some guiding principles to direct their search for a solution.

Newell and Simon (1972) found that humans use certain types of constraints on search, called *heuristics*. Heuristics usually involve special information about problems that keep one from exploring obviously fruitless paths. Heuristics also help direct the problem solver in making decisions when presented with a number of choices. Naturally, using heuristics can sometimes be misleading; a problem that might be solved using exhaustive search might not be solved with the use of heuristics. However, good heuristics will prove useful in solving most problems in a reasonable amount of time.

Newell and Simon also identified a particular type of heuristic problem solving called *means-ends analysis*, which involves a goal-directed search through a problem space. Rather than starting from the initial state and carrying out a forward search until the goal is found, means-ends analysis involves examining the differences between the initial state and the goal state. Then, an operator is selected that will reduce some of those differences. After this operator is selected, a new subproblem is set up with the goal of making that operator applicable. Once that operator has been applied, it is quite likely that other differences remain to be reduced.

As an example, consider the following section of a well-known protocol taken from Anzai and Simon (1979). In this experiment, the subject was asked to solve the "Towers of Hanoi" problem, in which one must transfer a stack of disks from one peg to another without placing a larger disk on top of a smaller one. Part of the subject's protocol contained the following statements:

> If it were five, of course, 5 will have to go to C, right?
>
> So, 4 will be at B.
>
> 3 will be at C.
>
> 2 will be at B.
>
> So 1 will go from A to C. (p. 139)

In this example, the subject used differences between the initial and goal states (e.g., Disk 5 must end up on Peg C) to drive her problem solving. The subject repeatedly set up subgoals for each disk until she was able to satisfy one by moving Disk 1 to Peg C. In this way, means-ends analysis works backwards from a number of goals in the goal state. Subgoals are set up until connections are made to the initial state. This approach limits the search through a problem space, making it more tractable.

LIMITATIONS ON MEMORY AND RETRIEVAL

In the previous section, we saw that humans use heuristic methods to limit their search when solving problems. However, even with the use of heuristics, one has no

guarantees about finding the correct path through a problem space. In order to be sure of solving a "solvable" problem, a person would require the capability to exhaustively search the problem space. Computational problem solvers can easily perform this type of search by *backtracking* whenever they make a mistake. Backtracking involves remembering the choices made at each decision point and trying each possibility until a successful combination is found. The lack of this ability in humans arises partly from memory limitations. We can also see an example of this behavior in Anzai and Simon's protocol. When the subject ran into trouble, she eventually elected to begin the problem anew, rather than merely backing up to an intermediate point.

Although humans do not have much difficulty storing information about their past, they do have troubles selectively retrieving large amounts of data on demand (Miller, 1956). If a human finds that he has failed to solve a problem, it is often difficult for him to remember his last decision and the choices he had already made at that point. Getting bogged down in the details of remembering where he was, he will often try to "clear the slate" and begin a new attempt at solving the problem, rather than taking up the attempt somewhere in the middle where he cannot recall the details.

The human problem solver may also duplicate certain paths he has taken previously even though they failed, because he cannot remember which paths he has taken previously. An added complication is that a human can never be sure he has exhausted all the possible decisions at a given decision point.

We argue that such memory limitations exert an important influence on problem solving. We have just considered the intractability involved in the amount of information that would need to be readily accessible to perform a systematic search of a problem space. However, there is also a retrieval problem in deciding which information to use to solve a given problem. Human memory contains an enormous number of facts and rules that are relevant to many distinct situations. When one comes to a decision point during problem solving, one cannot simply search one's entire memory looking for information appropriate to the task at hand.

This is also a practical problem for computer systems. Searching the entire memory each time a decision must be made is simply impractical. Rather than carrying out this type of search, humans retrieve much smaller groups of information and use only what they remember to help solve a problem. This allows them to perform memory-related tasks, like problem solving, efficiently. However, this approach has the drawback that one may not retrieve information necessary to complete a task.

REACTION TO THE ENVIRONMENT

A final important issue in human problem solving involves the interactions between the problem solver and the environment in which he works. Humans constantly receive input

cues from their surroundings, which in turn provoke reactions. These reactions can appear in the form of distractions that cause the problem solver to turn to a different task or to follow an incorrect path. In other cases, an environmental stimulus may help a person solve a problem by directing him down useful paths and steering him away from paths that lead to failure. We will return to this issue in our discussion of learning characteristics.

Learning in human problem solving

In this section, we examine aspects of human problem solving that involve learning. These characteristics provide the primary source of data that we use to evaluate our model in chapter 5. Each of the learning phenomena reported here is also discussed later in terms of our model. Before discussing these characteristics, however, we must first define precisely what we mean by the term "learning."

LEARNING AS IMPROVED PERFORMANCE

An important feature of human problem solving is that humans learn from their experiences. We view learning as a change in the ability to perform a task. In the current research, the task at hand involves solving problems. We say that a person or problem-solving system has learned something when its ability to solve problems has changed. Ideally, this change involves an *improvement* in the ability to perform the task, but this need not always be the case. Under this view, learning can be evaluated by measuring how well a person performs a task before and after learning has taken place. Defining learning as a change in performance still allows for a wide range of specific types of behavior. Performance can improve or degrade, and the conditions under which change occurs can be quite different. In the remainder of this section we describe some specific types of change in performance that humans exhibit when solving problems.

POSITIVE TRANSFER

When we speak of improving performance for a task, it is still unclear under which conditions improvement occurs. One type of improvement involves practicing a specific task and becoming increasingly better at it. In problem solving, this corresponds to becoming familiar with an individual problem. On the first attempt to solve a specific problem a person might be expected to do rather poorly. But after he has seen the problem repeatedly, he becomes more familiar with the details and remembers the difficult parts more clearly. We would expect the person's ability to solve this single problem to improve with each repetition.

Newell and Rosenbloom (1981) have investigated practice effects in problem solving in terms of the "power law of practice." This hypothesis states that the amount of effort

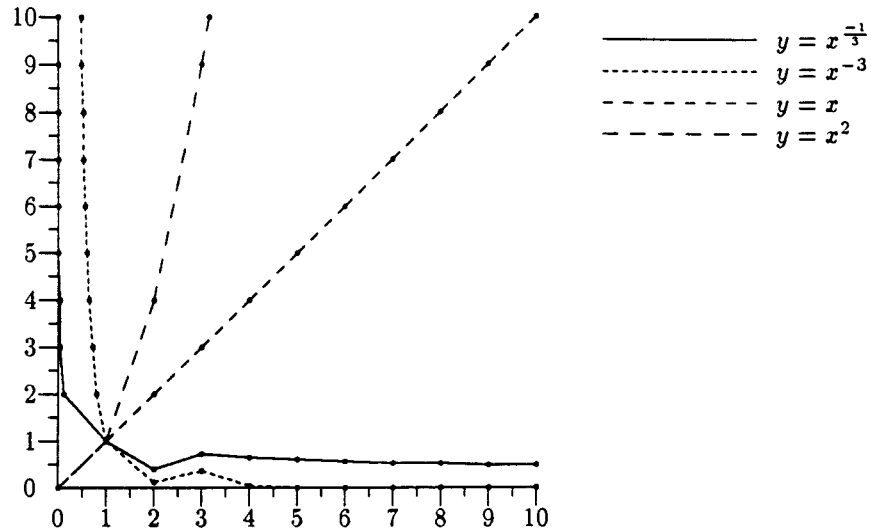

Figure 1. Some power-law curves in linear coordinates.

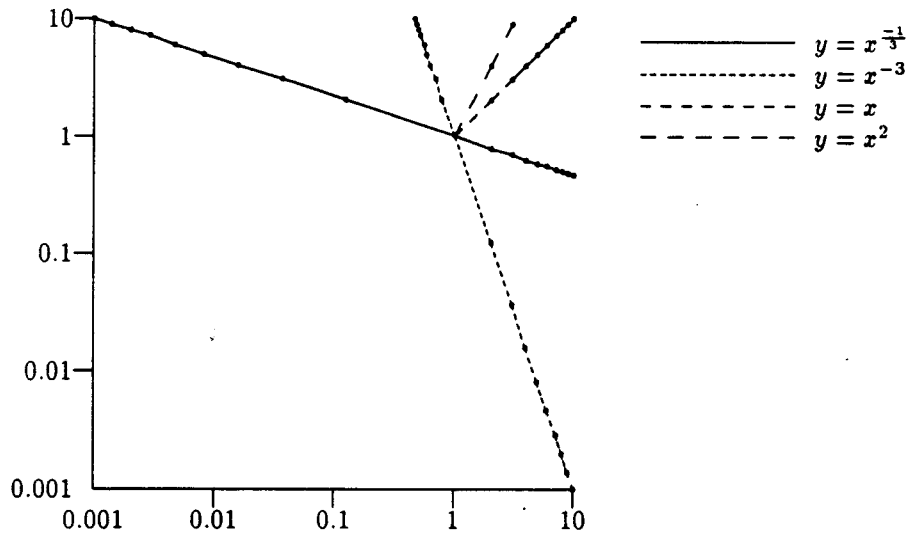

Figure 2. Some power-law curves in log-log coordinates.

required to complete a task reduces as a power law with respect to the number of times the task has been repeated. That is, graphing the logarithm of the measure for effort against the logarithm of the number of trials will yield a straight line. Newell and Rosenbloom discuss a number of motor-skill and retrieval tasks that have been shown to follow the power-law hypothesis, and they propose that problem-solving activities should follow the same law. This is not an outrageous expectation, because there is a relatively large family of curves that obey the power law. A small set of examples is provided in Figures 1 and 2 in both log-log and linear coordinates.

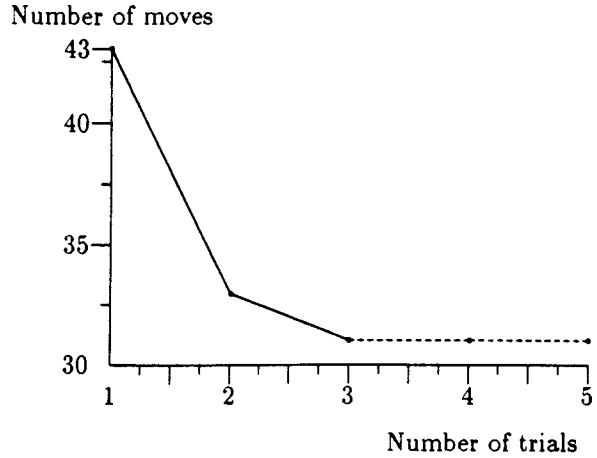

Figure 3. Improvement with practice in number of moves to solution.

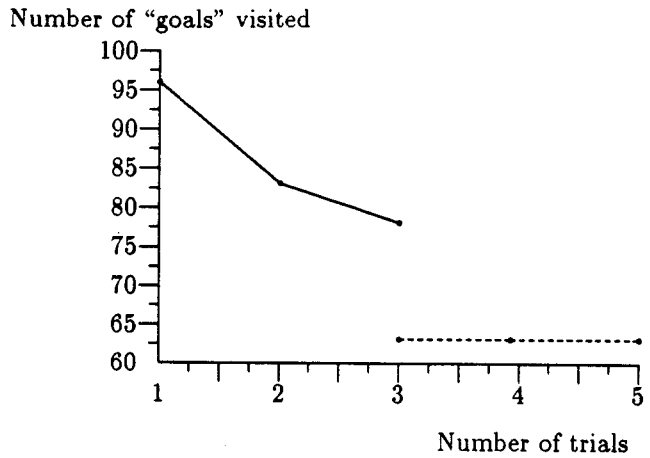

Figure 4. Improvement with practice in amount of search performed.

In testing their hypothesis, Newell and Rosenbloom measured a subject's speed in consecutive trials on a solitaire card problem. They found that this particular measure did follow a power law. We should note that the subject was not exactly practicing a single task, because each initial configuration of cards was different. However, their results do lend support to the hypothesis that the power law of practice applies to problem solving as well as low-level performance characteristics.

We have also examined the effects of practice on an individual problem by further studying Anzai and Simon's (1979) "Tower of Hanoi" protocol. In this protocol, the subject solved the same problem three separate times. We translated the protocol into a problem-history representation that could have been generated by our system (described in chapter 4). We then used the representation for each individual trial to measure the subject's effort in some of the same terms we use to evaluate our system in chapter 5.

Table 1. Improvement from transfer within the "Towers of Hanoi" domain.

CONDITION	# MOVES	# NODES
CONTROL	43	96
TEST	33	83

Figures 3 and 4 record the amount of effort expended by Anzai and Simon's subject during each successive trial. The dashed lines represent the minimum amount of effort required to solve the problem.

It is clear that these curves have the same general shape as some of the power-law curves. In particular, much less improvement occurs after each successive trial. Although only three trials were recorded, it is easy to see that the subject was rapidly approaching optimal behavior, after which no improvement would be possible.

It is interesting to note that, although these curves have the right general shape, they do not precisely obey a power law. Therefore, it appears that at least some practice episodes in problem solving do not obey this law. Rather, they seem to follow some more general law that specifies that the rate of improvement decreases with each trial.

A more interesting type of learning occurs when a person is able to practice one individual problem and have that practice lead to improved performance on other similar problems or on more difficult, related problems. This is called *positive transfer*. Humans exhibit the ability to solve problems they have never seen before after they have been exposed to similar problems. Positive transfer involves a bit more than simply getting better at solving an individual problem with practice. It requires that one generalize knowledge of specific problem-solving episodes to an extent that it becomes useful for a wide range of problems. Naturally, we would expect that the less similar two different problems are to each other, the less transfer there would be from one to the other. Nonetheless, we should expect at least a small amount of transfer in any case.

Again referring to Anzai and Simon's protocol, we observed that, after solving the problem once, the subject solved a series of smaller problems involving one disk, two disks, etc., before attempting the main problem again.[3] Thus, we were able to compare the subject's effort from the first trial with the effort in the second trial, after these smaller problems had been solved. As shown in Table 1, there is a noticeable improvement. It is not clear whether most of this improvement arises from transfer from the simpler problems or from the initial solution attempt. However, we assume that there was at least some

[3] The subject performed this activity spontaneously, without any prompting from the experimenter.

Table 2. Water-jug problems in Luchins' *Einstellung* experiment.

PROBLEM	GIVEN THE FOLLOWING EMPTY JARS			OBTAIN THIS AMOUNT
1	21	127	3	100
2	14	163	25	99
3	18	43	10	5
4	9	42	6	25
5	20	59	4	31
6	23	49	3	20
7	15	39	3	18

transfer from solving the simpler problems or the subject would have abandoned that approach.

NEGATIVE TRANSFER

The ability to generalize and transfer knowledge across problems can have drawbacks as well. When presented with a new problem to solve, it may not be possible to immediately distinguish a good strategy from a bad one. This is also something that must be learned. Just as solving one problem might help solve a new problem, it might just as easily hinder solving some other problem. This phenomenon has been exhibited in experiments by Luchins (1942) on *Einstellung* or the *set effect*.

In one of Luchins' experiments, subjects were presented with a series of "water jug" problems. These problems involve a number of jugs of various sizes. The goal in each problem is to fill and empty jugs in various combinations so as to end up with a specific amount of water. For example, given a 29-gallon jug and a three-gallon jug, 20 gallons of water can be obtained by filling the large jug and pouring it into the smaller jug three times.

Luchins gave subjects the problems listed in Table 2. All the problems can be solved by filling the center jug, pouring it into the rightmost jug twice, and then into the leftmost jug once. However, problems six and seven can also be solved with simpler procedures. For instance, the sixth problem can be solved by filling the leftmost jug and pouring it into the rightmost jug. Problem seven involves filling the leftmost and rightmost jugs and pouring them into the center jug. In this particular experiment, *every* subject failed to employ the simpler solutions on these test problems. This is a classic example of negative transfer. The knowledge of the complicated water-jug solution was useful in solving the

other complicated problems, but it actually proved a hindrance in solving what could have been an easier problem.

Creativity

One aspect of human behavior that might be considered the holy grail of AI researchers involves creativity. We will not delve into the philosophical aspects of what it means to be creative. Rather than attempting to define the term creativity, we will be content with considering behaviors that *seem* creative. In this section, we discuss two issues involved in problem solving that are generally thought to require some sort of creativity, in addition to being involved with learning or improved performance.

INTER-DOMAIN TRANSFER OR ANALOGY

The first type of creativity concerns the ability to use analogy. One powerful aspect of human problem solving is the ability to use knowledge that does not, at first glance, seem directly relevant to the problem at hand. Humans can conjecture relationships and facts about the current problem by drawing similarities to old problems and expanding on those similarities. This can sometimes allow one to solve otherwise unsolvable problems. We should note, that although humans certainly have the ability to use this mechanism, they often have difficulties recognizing and using analogies without help (Gick & Holyoak, 1980, 1983; Hayes & Simon, 1977; Reed, Ernst, & Banerji, 1974).

Holyoak and Koh (1987) ran an experiment that test subjects' ability to spontaneously retrieve and use an analogy to solve a new problem. Their experiment included two groups of subjects. The test group was taken from psychology classes that had discussed Duncker's (1945) "radiation" problem. In this problem, a patient has a tumor that could be destroyed by high-powered x-rays. However, the x-rays would also destroy any healthy tissue it went through. Lower intensity x-rays would not damage the healthy tissue, but would also be ineffective against the tumor. One solution to this problem involves aiming multiple low-intensity x-rays at the tumor from different directions. This allows the tumor to be destroyed without destroying any healthy tissue. The control group of subjects was selected from psychology classes that had not discussed this problem.

Every subject was asked to solve a "broken lightbulb" problem. In one version of this problem,

> the filament of an expensive lightbulb in a physics lab was broken. The lightbulb was completely sealed, but an intense laser could be used to fuse the filament. However, at that high intensity, the glass surrounding the filament would be broken. At lower intensities the laser would not break the glass, but neither would it affect the

> filament. The solution was to direct multiple low-intensity lasers toward the filament from different directions. (p. 335)

Holyoak and Koh found that only ten percent of the subjects in the control group could come up with the convergence solution to the problem. In contrast, this solution was found by 81 percent of the test group. This indicates the ability of humans to solve a new problem by retrieving an analogous problem that has already been solved from memory.

We view the use of analogy as an extension of the idea of positive transfer. Earlier we described how humans have the ability to generalize from specific problems in solving other similar problems. Analogy can be viewed as the same mechanism at a higher level of abstraction, in which one generalizes old problems to help solve new ones. The difference is that analogies generally take place across domains rather than within them. This means that the similarities between two problems will probably not be on the level of specific details, but on the level of more abstract relations and shared features (Gentner, 1983). We would expect that a general transfer mechanism capable of generalization within a domain would also be somewhat successful (though perhaps weaker) with higher-level generalizations or analogies. Naturally, any type of negative transfer that can occur on the lower intra-domain levels would have corresponding detrimental effects on the higher level of inter-domain analogies.

EFFECTS OF EXTERNAL CUES

A final interesting characteristic involves the effect the environment has on problem-solving behavior. Humans do not solve problems in a vacuum, and they are constantly reminded of things by the presence of various external cues. These cues can take many forms. For example, there might be an object in the area that is suddenly noticed by accident. At another extreme, something in the environment might be deliberately brought to the attention of the solver, as when a teacher gives a student a hint. The main feature of external cues is that, regardless of their nature or intent, they have some effect on the retrieval of information from memory. We are most interested in cases where external cues improve problem-solving performance. However, cues might be as detrimental in some cases as they are helpful in others.

The most benign form of external cue—the hint—usually comes from a teacher or a friend who is trying to help someone solve a problem. In this case, the teacher provides a cue to the solver in hopes of causing the solver to retrieve information that is useful for a given problem. Hopefully, this information helps the solver make a good choice for the current decision. A more malign case might involve someone who is purposefully trying to distract a solver's attention from whatever they are working on. Both types of intentional external cues can have dramatic effects on a person's ability to solve problems, whether that effect is helpful or detrimental.

To test the effects of hints on problem solving, Holyoak and Koh (1987) ran another experiment that focussed on how hints affect the retrieval and use of analogies. In this experiment, subjects were asked to read an summarize one of four stories. Each story was based on the "broken lightbulb" problem discussed above, but include small variations and the solution to the problem.

Next, the subjects were asked to solve Duncker's "radiation" problem. After an initial attempt to solve this problem, the subjects were given a hint to use the "broken lightbulb" story to suggest a solution. The subjects then attempted to solve the "radiation" problem again. After each attempt, Holyoak and Koh measured the percentage of subjects that successfully solved the "radiation" problem. They found the 39 percent of the subjects solved the problem before receiving a hint, while 66 percent solved the problem after the hint. This demonstrates the significant effect a benevolent environmental cue (in this case, a hint) can have on the ability to solve a problem, particularly when success hinges on the retrieval and use of an appropriate analogy.

For further psychological evidence concerning the effects of cues, we cite an experiment performed by Dreistadt (1969). He gave a set of "tricky" problems to four groups of people. The first group was given twenty minutes to solve the problems. The second group was given the same amount of time and the same problems, but was also presented with pictures that contained analogical hints to help find the solution. This group was not told the purpose of the pictures. The third group was given five minutes to concentrate on the problem, and then was given an eight-minute incubation period involving a distracting activity. This group was then given seven more minutes to solve the problem. The final group was presented with pictorial analogies *and* given an incubation period.

Dreistadt evaluated his subjects' behavior in terms of the number of correct answers in each group and the closeness of the incorrect answers to the correct solutions. The results indicated that pictorial analogies significantly aided the solution process, even though the subjects were not always aware they had been given help. Incubation alone did not seem to help in problem solving, but it did slightly enhance the effect of the pictorial analogies. These results suggest that analogies are an important part of insight, and that they can be cued by external stimuli that may be processed subconsciously.

Dreistadt's experiment also points out a more insidious type of external event, consisting of "incidental" cues. In this case, some part of the environment not directly related to the problem influences retrieval during problem solving. This effect may end up being helpful or detrimental, or it may have no noticeable influence on the solver's behavior at all. We usually notice an external cue having an effect on problem solving only when it aids in solving a problem. This phenomenon provides an explanation for some episodes of insight, which we discuss further in chapter 7.

Summary

In this chapter we explored a number of the issues involved in human problem solving. The characteristics we examined fall under the broad categories of problem-solving influences and limitations, learning, and creativity. On the surface, many of these phenomena appear to be unrelated and there could be a wide variety of cognitive mechanisms that account for them. However, we claim that each of these behavioral characteristics can be explained in terms of mechanisms involving memory and retrieval. The role of memory in problem solving plays a central part in the model we describe later in this dissertation. In chapter 5, we discuss a number of experiments we have run in an attempt to account for the phenomena covered in this chapter. Since memory issues play such a vital role in our research, we next introduce and discuss the specific memory model that we use.

usually involves setting a threshold that determines the amount of activation required to be a working-memory element.

Psychological support for spreading activation

In addition to being used in various computational applications, spreading activation has received attention from psychologists as a plausible model of human memory. Most of the psychological literature on this topic concentrates on accounting for the amount of time people take to perform memory-related tasks involving fact retrieval and word recognition.

RETRIEVAL OF FACTS FROM MEMORY

One of the first experiments testing the role of spreading activation in fact retrieval was run by Collins and Quillian (1969). Their model proposed that human memory is represented as a semantic network organized into a hierarchy of concepts. Each concept is connected to a set of properties and a set of specializations of the concept. These specializations in turn have properties and perhaps further specializations attached.

For example, the concept for animal includes the fact that animals have skin. One specialization of animal is bird, which includes among its properties the facts that birds have wings and fly. Finally, a specialization of birds is the canary, with the properties that canaries sing and are yellow. According to the theory of retrieval as spreading activation, relationships between objects that are farther away from each other in memory take longer to be retrieved and confirmed. For example, it should take a person longer to verify that "canaries have wings," than it would to verify that "canaries are yellow," because activation that spreads from *canary* and *wings* must pass through the intermediate *bird* node before retrieval of the proposition is complete. In contrast, activation markers spread from *canary* and *yellow* run into each other immediately. By the same argument, it should take even longer to verify the fact that "canaries have skin."

Collins and Quillian ran a set of experiments in which they presented subjects with a large set of sentences. Each sentence was in the form of a simple fact, such as "tennis is a game," or "an elm is a plant." The subjects were instructed to press one button if the sentence they saw was true in general, and to press another button if the sentence was false. Reaction times were recorded for each instance. In analyzing their data, Collins and Quillian divided the sentences according to the semantic distance between the concepts involved in them. For each group of sentences, they graphed the average reaction rime. Their results showed that reaction times increase linearly with the distance between concepts, matching the predictions of the memory model. This was the first support found for the idea of spreading activation in retrieval.

CHAPTER 3

The Spreading-Activation Model of Memory

The primary purpose of this research is to provide a computational explanation of a number of characteristics exhibited by human problem solvers. To accomplish this, we have integrated a model of memory and retrieval with a general problem solver. The memory model we have chosen is based on the use of *spreading activation* for retrieval.

The first spreading-activation model was introduced by Quillian (1968) in his work on natural-language processing. Along with his model, Quillian proposed the semantic-network representation of memory. In this framework, concepts are represented as nodes in a graph. Relations between concepts are represented as the links between nodes. Spreading activation involves passing information from a set of source concepts throughout a portion of the network along the various links.

Since Quillian's original proposal, spreading activation has enjoyed attention from psychologists and computer scientists. In this chapter we review the history of spreading-activation models in these fields. We also discuss the issues involved in integrating spreading activation into a psychological model of problem solving.

History and description of spreading activation

There are two related but distinct mechanisms that have come to be known as spreading activation. Quillian originally used this term interchangeably with *marker passing* to describe a process of passing tokens or markers throughout a network. He used these markers to find paths from source nodes to other nodes within a semantic network. When the paths from two different source nodes crossed, the retrieval process returned the path that connected the source nodes through that common node. This path was used as a semantic description of a relationship between the two nodes. In essence, marker passing provides a mechanism for carrying out a breadth-first search through the network. This type of marker passing is intended mostly to find relationships between concepts, which one can then analyze to make decisions about ambiguous situations.

For example, marker passing might be used to help understand the sentence "I saw a star in Hollywood." Marker passing would retrieve a path connecting the concepts "star" and "Hollywood." This path would suggest that the type of star being discussed is a

movie star rather than a star in the sky.[4] Although the users of such algorithms tend to use the terms "marker passing" and "spreading activation" interchangeably, we will use the former term for this specific mechanism to distinguish it from the type of spreading activation used in our research.

The type of spreading activation used in EUREKA is more similar to that introduced by Anderson (1976, 1983) in his ACT framework. In ACT, there are two types of memory. Procedural memory is represented as a number of productions that can fire when they match other active memory elements, and declarative memory represents facts using a semantic network similar to the type introduced by Quillian. One important difference is that the links between nodes in ACT's declarative memory have *trace strengths* associated with them. These values indicate the relative strengths of the specific relations that connect concepts. For example, a link representing the statement "fish are swimmers" might be stronger than a link for "fish have scales." This represents the notion that the concept of *fish* is more likely to remind us of *swimming* than of *scales*.

Anderson also makes another augmentation to the semantic network representation. He gives each node in the network a numerical *level of activation*. The nodes with the highest levels of activation are considered to be in *short-term* or *working* memory. As concepts receive higher levels of activation, production rules can match against them and fire more quickly. Thus, the ACT framework uses a spreading-activation mechanism to retrieve information from its declarative memory. However, this mechanism differs from marker passing in two important ways. First, no high-level knowledge structures are passed between nodes; only numerical activation strength is passed. Second, the type of information retrieved is different. Whereas marker passing retrieves a set of paths connecting source concepts, Anderson's type of spreading activation retrieves the set of concepts in the network with the highest activation levels.

In more detail, when a concept becomes activated (perhaps from being added by a production rule), activation spreads from it throughout a portion of the network. Activation from a number of activated concepts is thus distributed along different paths and collects in various nodes that are related to the multiple source nodes. In the most recent implementation of the ACT model (ACT*), the activation levels of concepts influence the amount of resources devoted to matching production rules against them. Under this particular view, there is no strict definition of a retrieved concept. However, in variations on this approach, levels of activation are used to determine new working-memory elements (e.g., Anderson, 1976; Holland, Holyoak, Nisbett, & Thagard, 1986). These variations require some algorithm for determining which elements are in working memory. This

[4] This marker-passing approach has also been used by Charniak (1983, 1986), Granger, Eiselt, and Holbrook (1986), and Norvig (1985) in their work on natural-language understanding and inference and by Hendler (1986) in his work on problem solving.

usually involves setting a threshold that determines the amount of activation required to be a working-memory element.

Psychological support for spreading activation

In addition to being used in various computational applications, spreading activation has received attention from psychologists as a plausible model of human memory. Most of the psychological literature on this topic concentrates on accounting for the amount of time people take to perform memory-related tasks involving fact retrieval and word recognition.

RETRIEVAL OF FACTS FROM MEMORY

One of the first experiments testing the role of spreading activation in fact retrieval was run by Collins and Quillian (1969). Their model proposed that human memory is represented as a semantic network organized into a hierarchy of concepts. Each concept is connected to a set of properties and a set of specializations of the concept. These specializations in turn have properties and perhaps further specializations attached.

For example, the concept for animal includes the fact that animals have skin. One specialization of animal is bird, which includes among its properties the facts that birds have wings and fly. Finally, a specialization of birds is the canary, with the properties that canaries sing and are yellow. According to the theory of retrieval as spreading activation, relationships between objects that are farther away from each other in memory take longer to be retrieved and confirmed. For example, it should take a person longer to verify that "canaries have wings," than it would to verify that "canaries are yellow," because activation that spreads from *canary* and *wings* must pass through the intermediate *bird* node before retrieval of the proposition is complete. In contrast, activation markers spread from *canary* and *yellow* run into each other immediately. By the same argument, it should take even longer to verify the fact that "canaries have skin."

Collins and Quillian ran a set of experiments in which they presented subjects with a large set of sentences. Each sentence was in the form of a simple fact, such as "tennis is a game," or "an elm is a plant." The subjects were instructed to press one button if the sentence they saw was true in general, and to press another button if the sentence was false. Reaction times were recorded for each instance. In analyzing their data, Collins and Quillian divided the sentences according to the semantic distance between the concepts involved in them. For each group of sentences, they graphed the average reaction rime. Their results showed that reaction times increase linearly with the distance between concepts, matching the predictions of the memory model. This was the first support found for the idea of spreading activation in retrieval.

THE FAN EFFECT

Later experiments on fact retrieval were run by Anderson (1974). He had subjects memorize sets of propositions of the form "a person is in a place," such as "a hippie is in the park." After the subjects had committed a large number of these facts to memory, Anderson tested the amount of time it took the subjects to recall if new sentences presented to them were members of the memorized set. The experiments indicated that it took longer to retrieve information about a concept (e.g., hippie) when that concept was present in more of the memorized sentences. This phenomenon is known as the *fan effect.* In addition, Anderson demonstrated that, in humans, activation seems to spread from all concepts involved in a presented sentence, contradicting the competing hypothesis that activation is spread from a single concept followed by a search for the other concepts.

PSYCHOLOGICAL RESEARCH ON PRIMING

Other psychological research designed to test the spreading-activation theory has focused on the priming effect on word recognition. As an example, Meyer and Schvaneveldt (1971) ran experiments in which they presented subjects with a large number of pairs of letter strings. The strings were divided into groups that included pairs of associated words, pairs of unassociated words, pairs of nonwords, and pairs including a word and a nonword. For each pair, both strings were presented on a screen with one above the other. In one experiment, the subjects were asked to push a *yes* button if both strings were words, and a *no* button otherwise. In the second experiment, they were asked to push a *same* button if the strings were both words or both nonwords, and a *different* button otherwise.

The spreading-activation theory predicts that if both of the strings are words that are related to each other, the activation of the first one should prime the second one and facilitate its retrieval. This implies that subjects should have the fastest response times for this task when the strings are two related words. Meyer and Schvaneveldt found that having related words did indeed speed up reaction time in the task. In addition, presenting a word before a nonword significantly increased the response time for completion of the task. The results in priming and fact retrieval are consistent with the type of spreading-activation mechanism used in the ACT framework. In fact, Anderson (1983) was able to account for these data with his spreading-activation mechanism in ACT*. The retrieval mechanism in our model is based on Anderson's algorithm.

RETRIEVAL OF ANALOGIES IN PROBLEM SOLVING

Spreading activation has also been proposed as a mechanism for the retrieval of analogies. Holyoak and Koh (1987) ran an experiment to determine the types of problem features that had the greatest effect on the retrieval of an analogy. In this experiment, they presented subjects with one of four versions of the "broken lightbulb" story that we

discussed in chapter 2. The four stories were created by altering the surface and structural features of the original story. In one story, surface features were changed by replacing the lasers with ultrasound waves. In another, structural features were changed by omitting the restriction of breaking the glass and adding the constraint that no single laser had enough intensity to repair the filament. In the final story, both alterations were made. After reading the lightbulb story and solving a set of unrelated problems, the subjects were asked to solve the "radiation" problem. The percentages of subjects who were able to solve the radiation problem were recorded before and after a hint was given to use the lightbulb story as an analogy.

The results of this experiment suggest that surface and structural similarities influence the spontaneous retrieval of analogies, whereas structural similarities influence the ability to apply an analogy once it has been retrieved. The results concerning retrieval are consistent with a spreading-activation mechanism. However, spreading activation would most likely imply that surface similarities have a greater impact on retrieval. However, Holyoak and Koh did not explicitly test this hypothesis. To do this, one would need to run variations on this experiment in which there is a more obvious distinction between the retrieval and the application of analogies.[5] Clearly, more experiments are required to specifically test the role of spreading activation in analogical problem solving. One of the hypotheses of this dissertation is that analogical retrieval can be explained by a single spreading-activation process that also accounts for many other psychological aspects of human problem solving.

Computational advantages of spreading activation

Aside from the use of spreading activation as a psychological model, we argue that this retrieval mechanism also has advantages from a computational point of view. In particular, search by spreading activation is only influenced by the structure of memory and not by its specific contents. Other types of retrieval algorithms (e.g., Laird et al., 1986a; Minton, 1988/1989; Ohlsson, 1987) can require an extensive analysis of portions of memory. In contrast, spreading activation uses a local algorithm that does not require attending to multiple complex concepts at one time. Spreading activation also imposes certain limits and constraints on the type and depth of search that will occur. More knowledge-intensive types of retrieval usually do not limit the size of memory that may be examined during retrieval. Finally, spreading activation specifies a paradigm under which the retrieval of objects occurs, placing a bias on which types of information will be retrieved under different conditions. Our argument is that this bias is consistent with human behavior under similar conditions.

[5] We should note, however, that Gentner (1988) *has* found that surface similarities have a greater impact on the retrieval of story analogies.

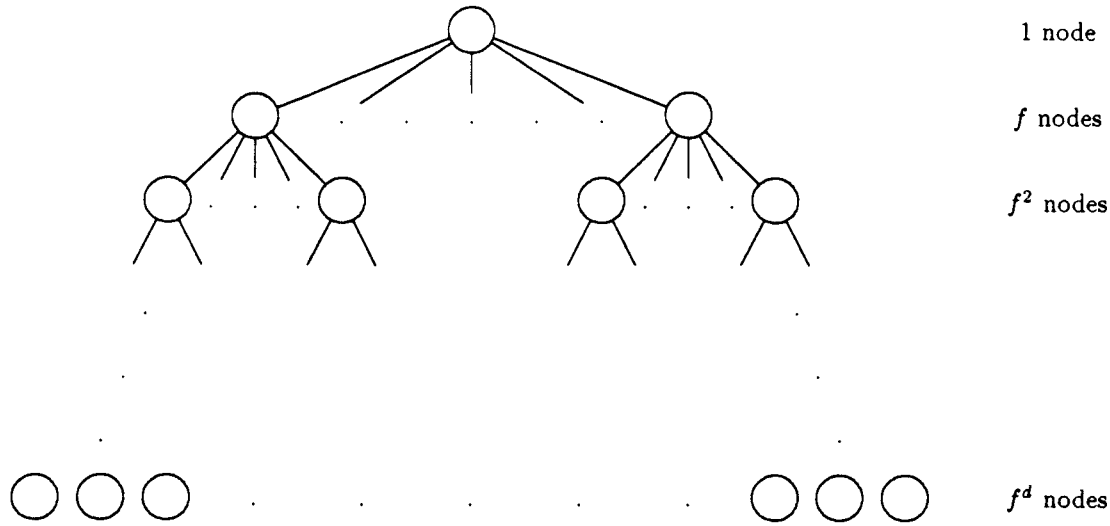

Figure 5. Spreading activation viewed as a tree traversal.

Time and memory complexity are important considerations when dealing with the computational characteristics of spreading activation. Consider a typical algorithm that implements this mechanism.[6] At first glance, it appears that there is an exponential growth in the number of semantic-network nodes visited during spreading activation. This growth is based on the branching factor of the network and the depth to which activation is spread. However, with most spreading-activation systems (including the one we describe in chapter 4), the amount of activation spread from a node is inversely proportional to the number of nodes connected to the source.

Consider an idealized network in which the fan from each node is f and the trace strengths of all links are the same. Even if there are cycles in the semantic network, the activation process treats each branch of its traversal separately. Thus, we can view a spreading-activation process as the exhaustive traversal of a tree, where multiple tree nodes may correspond to a single node in the semantic network (see Figure 5). To determine how long spreading takes, we derive a formula for the number of nodes visited during this traversal. The total number of nodes visited, T, is the summation of the number of nodes in each level of the tree up to a certain distance d. This distance is determined by the specific mechanism's parameters. For a network with fan factor f, we get

$$T = 1 + f + f^2 + \ldots + f^d \quad . \tag{1}$$

[6] Although spreading activation is well-suited for parallelism, we consider a serial algorithm for this analysis because the model presented in chapter 4 is implemented in this manner.

We can simplify this equation to

$$T = \frac{f^{d+1} - 1}{f - 1} \quad , \tag{2}$$

which is exponential with respect to d, the depth of the spreading process. However, d is determined by the amount of activation received by the furthest nodes and the threshold for stopping the spreading process. If we let a_n represent the amount of activation that is received by a node n levels away from the source, then we have

$$a_n = \frac{a_0}{f^n} \quad , \tag{3}$$

where a_0 is the initial activation given to the source node.

If we define the threshold for stopping the spread of activation as h, then activation will spread until $a_n < h$. In Equation 2, we used d as the number of levels that activation spread to. Therefore, we must have

$$a_d = \frac{a_0}{f^d} \leq h \tag{4}$$

and

$$a_{d-1} = \frac{a_0}{f^{d-1}} > h \quad . \tag{5}$$

These equations can be rewritten as

$$f^d \geq \frac{a_0}{h} \tag{6}$$

and

$$f^{d-1} < \frac{a_0}{h} \quad . \tag{7}$$

Substituting Equations 6 and 7 into Equation 2 gives us upper and lower bounds for the number of nodes visited:

$$\frac{f^2 \frac{a_0}{h} - 1}{f - 1} > T \geq \frac{f \frac{a_0}{h} - 1}{f - 1} \quad . \tag{8}$$

Notice that Equation 8 does not involve any exponential relationships, because d has been factored out. The equation also implies that the time and space required for spreading activation is independent of the size of memory, close to linear with respect to the inverse of the threshold $\left(\frac{1}{h}\right)$, and nearly constant with respect to f. Naturally, we have made some simplifying assumptions. However, any pattern of activation can be viewed in terms of a tree traversal, as we have done. This leaves only one variable that can complicate things: the fan factor f. It is important to note that, for a single step of activation, if f is high then a large number of nodes receive relatively small amounts of activation. If f is low, a small number of nodes receive a relatively large amount of activation. This balancing effect causes the time required to spread activation to remain approximately

constant when the threshold h is fixed. Chapter 5 contains some empirical results that support this claim.

In fact, in our implementation of spreading activation, there is an additional decay factor that attenuates the amount of activation that is passed from one node to another. This can further decrease the number of nodes visited during spreading. The advantage of these results is that we can make reasonable assumptions about the time and space required for the implementation to run. In addition, we can expect to integrate large amounts of knowledge into memory without degrading the efficiency of the system's retrieval mechanism.

A final argument with respect to the computational advantages of spreading activation concerns the lack of knowledge used by the mechanism itself. All the knowledge of the system exists within the semantic network. This knowledge is not consulted while carrying out the spreading activation process. In these terms, spreading activation can be considered a knowledge-free process. This means that the processing for spreading activation can be highly localized, because decisions made on where to spread activation and how much to spread to any given node need not consider the state or knowledge content of other parts of the network. This localization ability lends itself nicely to a simple algorithm for spreading activation that could easily be implemented in a parallel fashion. This would further increase the efficiency of the retrieval mechanism in terms of time. This advantage would be limited in a system where the search for retrievable information depends on other knowledge in memory.

Transfer in terms of spreading activation

As we have already suggested, spreading activation plays a central role in our research. Our intent is to show that the spreading-activation model is not limited to explaining the type of low-level memory issues concerned with the fact retrieval and priming paradigms. Rather, we believe that many aspects of human problem-solving behavior arise from memory and retrieval characteristics. Our model is based on the assumption that a spreading-activation retrieval process is consistent with the types of phenomena that have been demonstrated in the psychological literature on problem solving.

One of the main points to this assertion involves learning, or the effect of past experience on performance in solving new problems. A problem solver's behavior arises from its reactions to the environment and interpretations of its past problem-solving experiences. To account for this behavior, we must address the specific types of *transfer* that take place across problem-solving attempts. Because of this focus of study and the choice of a spreading-activation mechanism as the model for retrieval, we must show that these types of transfer can be modeled by spreading activation. The majority of the psychological literature, including Anderson's model of spreading activation, has been concerned with

explaining low-level characteristics of human memory. In contrast, we wish to extend support for the spreading-activation model by showing that it actually encourages the retrieval of useful information. This involves demonstrating that the types of transfer exhibited in human behavior can, in fact, be modeled by a spreading-activation process.

The simplest type of transfer arising from spreading activation comes from the implicit assumptions about the fundamental functions of spreading activation. One assumption is that activation of certain concepts in memory will cause concepts that are somehow similar to be retrieved. The expectation is that these similar concepts involve knowledge that has some bearing on new situations. However, it seems clear that strict similarity is not all that is required for the retrieval of helpful information. In terms of problem solving we need to broaden this view. We want the activation of certain concepts to cause retrieval of the most *useful* information (by some definition of utility) concerning a situation. This notion of utility relies heavily on similarity, but it must also be based on other factors of associative strength between concepts.

Perhaps the major claim of our research is that spreading activation allows the retrieval of useful information for problem solving. In fact, we make the stronger claim that the simple types of memory maintenance associated with spreading activation can be exploited to increase the utility of retrieved information in general. Transfer arises from setting up connections and relations between concepts and adjusting the strengths of the relations. To provide the maximum benefit, the adjustments are made in such a manner as to encourage the retrieval of the most appropriate information when the system is presented with specific problem situations. Naturally, we also expect to encounter negative transfer, but as we have noted, this is an unavoidable situation given the flexibility of the mechanism. In fact, we argue that negative transfer is a desirable characteristic for a psychological model, provided it occurs under conditions similar to those in which humans exhibit the same behavior. In the next chapter we provide a detailed description of our model of problem solving, including the role of a retrieval mechanism based on spreading activation.

CHAPTER 4

The EUREKA Model of Problem Solving

In chapter 2, we discussed a number of psychological phenomena related to problem solving. In this chapter we describe EUREKA, a model of problem solving that incorporates a set of mechanisms that explain these behaviors. In addition, we discuss many of the specific details involved in the implementation of this model. The intent of this chapter is to describe the EUREKA model from a technical standpoint. We begin the chapter with an overview of the model, followed by an in-depth discussion of its knowledge representation, performance, and learning.

Theoretical basis for the EUREKA model

At EUREKA's heart lies a problem-solving system that relies heavily on the means-ends-analysis (MEA) framework. To this system, we have added a long-term memory represented as a semantic network and the ability to retrieve information from the memory using a spreading-activation process.

As we discussed in chapter 1, the MEA approach relies on the problem-space hypothesis. Recall that in this framework, a problem is viewed as an initial state and some goal conditions, together with a set of operators that can be used to transform the initial state into a state that satisfies the conditions. The standard MEA approach selects operators for application by examining the entire set of operators and choosing one that reduces some of the differences between the current state and the goal conditions (Ernst & Newell, 1969; Fikes & Nilsson, 1971; Minton, 1988/1989). This procedure is repeated until the problem is solved or it becomes clear that the problem cannot be solved. In EUREKA, the search for an operator is replaced with a retrieval method that considers only a small subset of the (potentially very large) operator memory. In addition, the strict method for selecting operators is relaxed to allow the system more flexibility and the ability to generalize its knowledge.

As we have already suggested, the retrieval mechanism is key to the EUREKA model. The system demonstrates that a retrieval process based on spreading activation can be combined with relatively simple mechanisms for updating long-term memory to produce interesting behavior. In the context of problem solving, these mechanisms change the retrieval patterns associated with specific situations. Combining this memory model with

a problem solver provides more flexibility and efficiency. In addition, spreading activation accounts for human memory limitations by allowing the retrieval of useful information without an exhaustive search of memory. Finally, the memory model lets EUREKA explain various aspects of human problem-solving behavior.

In the following sections we describe three important aspects of the EUREKA model in detail. First, we discuss the representation and organization of knowledge in the system. Next we describe EUREKA's performance engine, including the problem-solving component, which employs a variant of means-ends analysis, and the retrieval mechanism, which helps the problem solver make decisions at critical points. Finally, we discuss EUREKA's learning mechanisms, which let the system improve its performance in a number of different ways.

EUREKA's knowledge representation

A primary concern for AI systems is the representation of knowledge. There are various levels at which it is useful to view the knowledge structures in EUREKA. This section describes EUREKA's knowledge from two perspectives, including the structures used by the retrieval mechanism and those used by the problem-solving engine.

SEMANTIC NETWORK

At a low level, EUREKA's long-term memory is a semantic network consisting of nodes (representing concepts) connected by small sets of labeled links (representing relations). The semantic network is functionally equivalent to predicate-calculus, attribute-value, or schema-based representations. That is, the same information can be stored using any of these representations. We have chosen the network representation to facilitate the use of spreading activation as a retrieval mechanism.

There are additional features of the memory that play roles in the retrieval process. Each node has an associated *level of activation*, which represents its current relevance to the system. In addition, each link has a *trace strength*, which represents the semantic strength of a specific relation between two concepts. These features are similar to those used by Anderson (1976, 1983), and we discuss them in more detail in the section on performance and retrieval.

PROBLEM-SOLVING TRACES

The semantic memory may sometimes appear to be a tangled network of concepts and relations with little obvious organization. However, the problem-solving component of EUREKA attributes special meanings to certain types of concepts and relations. Thus, the network encodes a large amount of knowledge that only becomes apparent when viewed

Table 3. Some nodes in the EUREKA's semantic network.

Trans1	State100	Goal17	And1
isa: Transform	isa: State	isa: State	isa: And
state: State100			arg1: Apply1
goal: Goal17			arg2: Trans2
child: And1			

On1	On2	On3	On4
isa: On	isa: On	isa: On	isa: On
partof: State100	partof: State100	partof: State100	partof: Goal17
arg1: A	partof: Goal17	arg1: C	arg1: A
arg2: Table	arg1: B	arg2: Table	arg2: B
	arg2: Table		

Status1	Status2	Status3
isa: Status	isa: Status	isa: Status
partof: State100	partof: State100	partof: State100
partof: Goal17	arg1: B	arg1: C
arg1: A	arg2: Clear	arg2: Clear
arg2: Clear		

at a higher level. As an example, Table 3 describes a number of nodes that could appear in the network. Because pictorial descriptions of semantic networks can become very messy, the table uses a slot-value representation. Each slot-value pair represents a link to a neighboring node. This table provides a partial representation of the top-level goal in Figure 7, so it may become more clear when compared to that figure.

Nodes are grouped together to represent a record of EUREKA's past problem-solving experiences. Since the memory records the problem solver's actions, we will briefly describe the system's performance component here, reserving the details until the following section. To represent problem-solving episodes, we have borrowed the idea of derivational traces from Carbonell (1986). EUREKA represents each attempt to solve a problem as a binary tree, in which each node represents an MEA goal.[7] The two children of a given node represent two subgoals that may be satisfied in an attempt to satisfy the parent goal. A series of attempts to solve a single problem results in a superimposition of the binary trees for each attempt. Some of the states in the individual trees may be shared with other

[7] For any particular model, there can be many different types of MEA goals, depending on how tasks are divided. For example, some systems include REDUCE-DIFFERENCE goals, which are implicit in EUREKA's TRANSFORM goals. In our work, we divide problem solving into two tasks, giving rise to a binary tree.

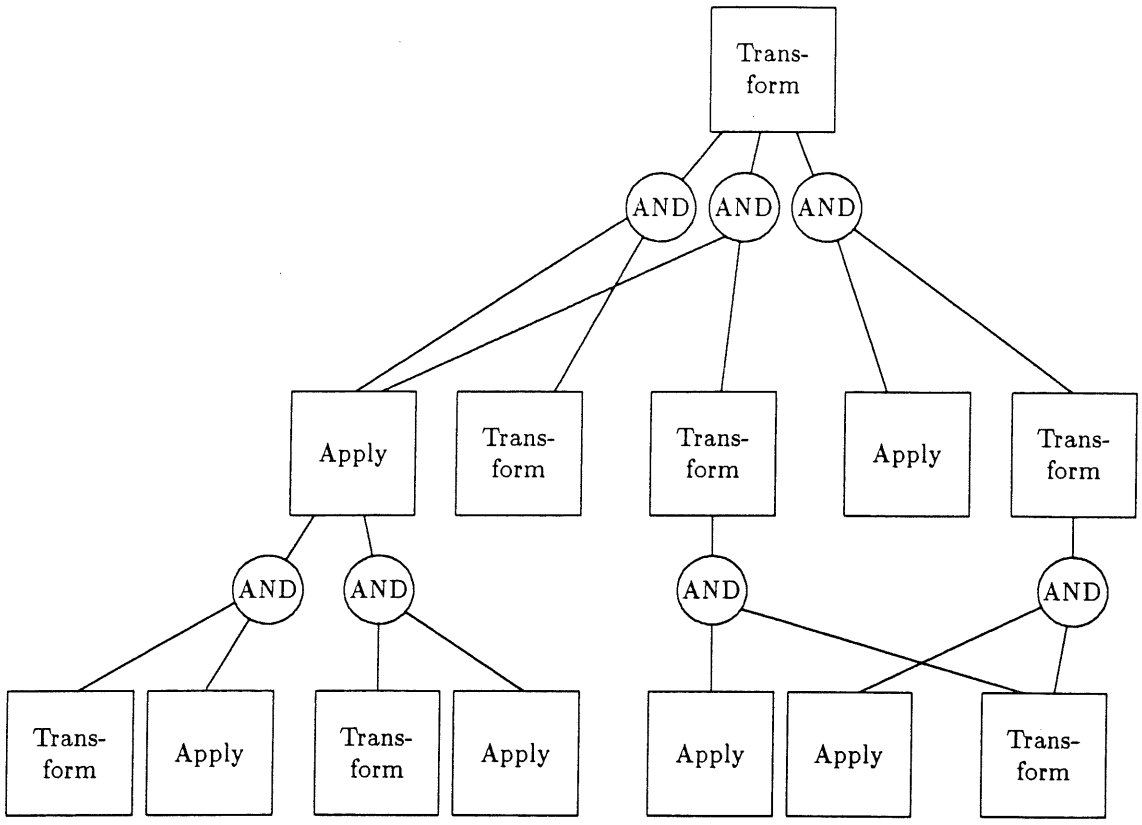

Figure 6. Abstract representation of multiple attempts at a single problem.

trees and some may not. The final product is a directed, acyclic, AND/OR graph, as shown in Figure 6. In the final graph, each state may have multiple pairs of children, each representing an attempt to satisfy the parent goal with two subgoals.

To clarify this situation, let us first consider a single attempt at solving a problem, which is presented to EUREKA as a TRANSFORM goal. That is, the problem is stated in terms like "Starting in the current state, somehow TRANSFORM (through the application of the appropriate operators) that state into a state that satisfies a certain set of goal conditions."[8] More precisely, an example goal might look like "Transform State100 into State123, which satisfies Goal17," where State100 represents a description of some "blocks-world" situation, Goal17 contains the condition that Block A is stacked on Block B, and State123 represents a state in which that condition is true. Figure 7 gives a pictorial description of this problem, which includes the various states and goal conditions that

[8] Throughout this dissertation we have attempted to be consistent in our use of terms for the various objects involved in a problem representation. We use the terms *goal* and *situation* to describe the goals for means-ends-analysis. The *current state* of a TRANSFORM goal is the state that must be transformed, and the *goal conditions* are the constraints that the transformation must satisfy.

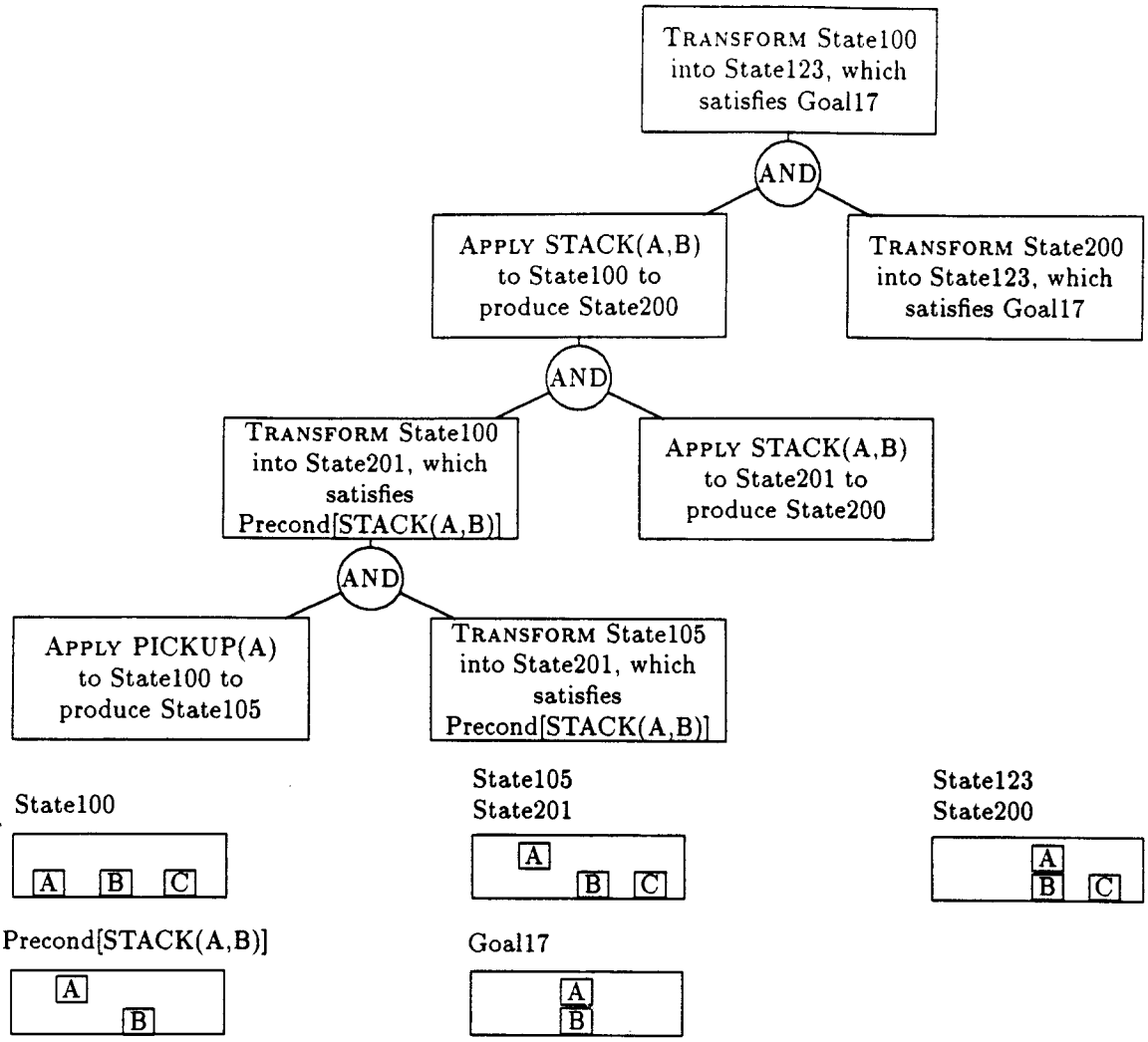

Figure 7. EUREKA's representation for a single attempt at a "blocks-world" problem.

may be encountered. The figure also provides an example solution, which we will examine in detail.

There are two ways that a TRANSFORM goal can succeed. First, the current state may satisfy the goal conditions. This would be the case in our example if Block A were on Block B in State100. If the conditions are met, the TRANSFORM goal always succeeds and has no children. The more interesting situation occurs when the current state does not already satisfy the goal conditions. In this case at least one operator must be applied in order to complete the transformation. In the MEA framework, one attempts to solve this TRANSFORM goal by setting up two subgoals. The basic sequence is to apply an operator that achieves part of the transformation and then set up a secondary TRANSFORM

goal to complete the transformation. We explain this process in more detail in the next section. For now, suppose EUREKA chooses the operator "Stack Block A on Block B," or STACK(A,B). In this case, it creates the subgoals "APPLY operator STACK(A,B) to State100 to produce State200," and "TRANSFORM State200 into State123, which satisfies Goal17."

When presented with an APPLY goal, there are also two situations that may occur. First, the chosen operator may be directly applicable to the current state. If this is true, the operator is executed and the resulting state becomes the current state of the following TRANSFORM goal. If the operator cannot be directly applied, one must set up a subgoal to make the operator applicable. In the means-ends paradigm, one must first transform the current state into a state that satisfies the preconditions of the chosen operator. Once this has been done, one can then APPLY the operator to the resulting state. Following the example, one sets up two subgoals: "TRANSFORM State100 into State201, which satisfies the preconditions of STACK(A,B)" and "APPLY STACK(A,B) to State201 to produce State200." If the preconditions are really satisfied as expected, then the APPLY goal will always be immediately satisfied and will not have any children.[9]

Whenever an MEA subgoal is created, it becomes the current goal in a recursive fashion. We explain this in more detail in the following section. The important point for understanding the memory structure is that goals in a single problem-solving episode can have either zero or two children. The children for a TRANSFORM goal consist of an APPLY goal followed by another TRANSFORM goal. The children for an APPLY goal consist of a TRANSFORM goal followed by another APPLY goal. A third possibility occurs when a goal cannot be satisfied for some reason. In this case the goal has no children, but it is marked as a failed goal. Figure 8 shows an example of a failed problem-solving episode.

As mentioned earlier, EUREKA compiles multiple attempts at solving a single problem into an AND/OR graph representing the system's entire experiences with that problem. There is often more than one way to attempt to solve a given TRANSFORM or APPLY goal and the system records each attempt, whether it is successful or unsuccessful. Therefore, after multiple attempts at solving a problem, each goal in memory will have a number of *pairs* of children, rather than just two children. Some of these pairs represent previous successful attempts at satisfying the current goal and some represent failed attempts. The pairs representing successful attempts denote solutions that can be used in the future to solve similar problems. However, all the pairs represent attempts that have been tried by the system, and an OR link connects each pair to the parent node. This represents the fact that, for any particular individual solution attempt, exactly *one* of these pairs has been set up in trying to solve the problem.

[9] However, one can imagine versions of this system where expectations are not always met. In that case the APPLY goal may have more subgoals associated with it. This type of situation arises in systems that do *reactive planning* (e.g., Ambros-Ingerson & Steel, 1988; Georgeff & Lansky, 1987; Schoppers, 1987).

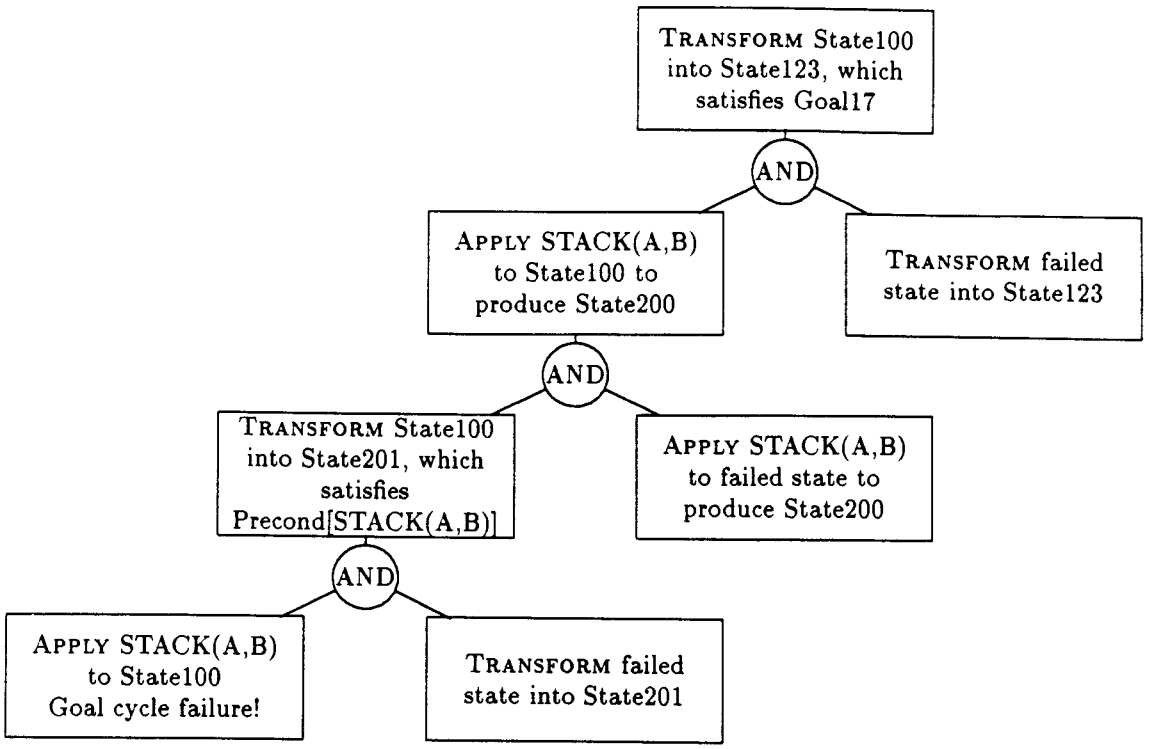

Figure 8. A failed problem-solving attempt.

A given problem may have more than one solution, which is one reason why EUREKA allows multiple pairs of child nodes to be connected to any given goal node. However, different solution paths could very easily share some subgoals. In this case, the representation of experiences for the problem will not be a tree, but a directed graph. Problem-solving actions may also bring the solver back to a situation it has seen before in a single solution attempt, thus creating a cycle. However, EUREKA views cycles as failures and does not allow them. Therefore, the total representation of the history of experiences for a single problem can be viewed as a directed, acyclic, AND/OR graph.

Performance and retrieval

Because EUREKA's high-level memory organization records the system's past problem-solving experiences, it is naturally very closely tied to the problem-solving component itself. As we have mentioned, the performance component is based on a means-ends problem solver that is reminiscent of early planning systems, such as GPS (Ernst & Newell, 1969) and STRIPS (Fikes & Nilsson, 1971). In this section, we examine the details of EUREKA's problem-solving component. Since the retrieval of information from long-term memory is an integral part of the problem solver, we discuss those issues in this section as well.

Table 4. EUREKA's two main problem-solving functions.

```
TRANSFORM(StateX,Conditions): Returns StateZ

  If StateX satisfies Conditions
     Then Return StateZ as StateX
     Else Let Operator be SELECT_OPERATOR(StateX,Conditions);
          If Operator is empty
             Then Return StateZ as "Failed State"
             Else Let StateY be APPLY(StateX,Operator);
                  If StateY is "Failed State"
                     Then Return StateZ as "Failed State"
                     Else Return StateZ as
                                  TRANSFORM(StateY,Conditions)


APPLY(StateX,Operator): Returns StateY

  Let P be PRECONDITIONS(Operator);
  If StateX satisfies P
     Then Return StateY as EXECUTE(StateX,Operator)
     Else Let StateW be TRANSFORM(StateX,P);
          If StateW is "Failed State"
             Then Return StateY as "Failed State"
             Else Return StateY as APPLY(StateW,Operator)
```

THE BASIC PROBLEM SOLVER

EUREKA's performance component deals with two main tasks. The basic problem solver makes individual attempts at solving a given problem, but there is also an outer control loop to guide repeated attempts at solving a single problem. In this section, we first describe the more detailed basic problem solver and then cover how this problem solver is used repeatedly to find a solution to a problem.

When presented with a TRANSFORM or APPLY goal, EUREKA's problem solver attempts to satisfy that goal using a set of actions based on the means-ends framework. However, the system differs from the standard means-ends approach used in STRIPS in the way it selects operators to satisfy TRANSFORM goals. In a strict means-ends framework, operators are chosen strictly according to their abilities to reduce differences between the current state and goal conditions. EUREKA allows a more liberal selection of operators, which we discuss in detail later.

We can best describe the actions of the problem solver in terms of two recursive functions: TRANSFORM and APPLY. As might be expected, each of these functions has the task of satisfying a goal of the same name. Each of these functions may call itself and the other function in a recursive manner, and each recursive call involves setting up a subgoal as explained in the section on knowledge representation.

Table 4 gives a description of the two basic functions. First, let us consider the TRANSFORM function, which is presented with a goal in the form of "TRANSFORM StateX into StateZ, which satisfies Conditions." The value returned by the function is either a state that satisfies the Conditions or a special "failure" state. TRANSFORM first checks to see if StateX already satisfies the Conditions. If this is the case, then TRANSFORM succeeds and returns with StateZ bound to StateX. If this is not the case, then TRANSFORM must set up an APPLY goal followed by another TRANSFORM goal in an attempt to complete the transformation.

To set up an APPLY goal, TRANSFORM selects an operator to try to apply. Ideally, the chosen operator will reduce some differences between the current state StateX and the Conditions. Although this would necessarily be true in a standard means-ends system, it is not always the case in EUREKA. As we describe in the next section, the model selects an operator and sets up an APPLY goal using that operator. This subgoal may fail, in which case the current TRANSFORM goal also fails. However, if the APPLY subgoal succeeds, it returns a new state, StateY, resulting from the application of the Operator to StateX. With this new state, TRANSFORM sets up a recursive subgoal of the form "TRANSFORM StateY into StateZ, which satisfies Conditions." When the current TRANSFORM goal finishes, StateZ receives the value returned by the subgoal, whether it is a failure or a successful state.

The APPLY function has a similar form to the TRANSFORM function, but is somewhat simpler because APPLY does not have to worry about selecting an operator to set up its subgoals. The form of such goals is "APPLY Operator to StateX to produce StateY." The function first checks the preconditions of the Operator to see if they are satisfied in StateX. If so, APPLY simply executes the Operator in StateX and returns the resulting state in StateY. If the preconditions are not met, the current state, StateX, must be further transformed until the Operator can be applied. More precisely, EUREKA must set up a subgoal of the form "TRANSFORM StateX into StateW, which satisfies the preconditions of Operator." If this subgoal fails, then APPLY also fails. If the subgoal succeeds, it returns a state, StateY, to which the Operator should now be applicable. At this point, APPLY calls itself in an effort to execute the operator in the new state. The final subgoal is of the form "APPLY Operator to StateW to get StateY," and the value returned by the recursive call becomes the return value of the parent APPLY goal.

So far we have seen how EUREKA approaches a single attempt at solving a problem. However, there is an additional mechanism that determines the system's behavior when it fails to solve a problem during one of these attempts. As we suggested previously, EUREKA does not have the ability to backtrack when it fails to solve a problem. Rather, the system must restart its work from the top-level TRANSFORM goal. In this manner, the system continues attempting to solve the problem until it is successful or a predetermined limit on the number of attempts is reached. Each individual attempt at solving the problem does

not necessarily duplicate past work because the system includes mechanisms for making decisions randomly and encouraging (but not guaranteeing) search down new paths. In the current implementation, EUREKA attempts to solve a problem 50 times before it gives up.

To summarize, we can discuss EUREKA's basic problem solver in terms of three functions. TRANSFORM and APPLY work together to manage individual attempts to solve problems. The system's outer control loop provides the top-level mechanism for making new attempts at a problem after failure and for giving up when the system does not seem able to solve a problem. In the remainder of this section, we describe the details involved in selecting an operator.

THE ROLE OF RETRIEVAL IN PROBLEM SOLVING

The retrieval of information is a central issue in the EUREKA model. In terms of performance, learning, and memory, the critical point of problem solving concerns choosing an appropriate operator to help satisfy a TRANSFORM goal. In order to accomplish this, the system must retrieve a set of operators from memory and evaluate them in some manner to choose the "best" one. In many standard problem solvers, retrieval consists of an exhaustive search through the operator memory, with the final selection being made according to some predetermined evaluation function. A number of other systems use control rules or preference rules to retrieve and evaluate operators (Hendler, 1986; Laird, Rosenbloom, & Newell, 1986b; Minton, 1988/1989; Ohlsson, 1987).

In EUREKA, retrieval is modeled by a process of spreading activation. Every time the TRANSFORM function is executed, activation is spread from the nodes in the semantic network that represent the current state and goal conditions. This activation is used to retrieve a set of related TRANSFORM goals from long-term memory. After these states have been retrieved, one is selected as a model for solving the current TRANSFORM goal. This selection is based on two factors: the degree of match to the current state and goal conditions, and the history of success or failure in using each of the retrieved TRANSFORM goals as a model in the past, which we detail later. After EUREKA has selected a candidate TRANSFORM goal, it carries out an analogical mapping to enable the use of the retrieved goal as a model for the current goal. Finally, it selects an operator that was used successfully in the retrieved situation for application to the current goal.

There are two distinct steps in the process of operator selection: the retrieval of a set of past goals, and the evaluation and selection of a model and an operator from the retrieved set. Most problem-solving systems downplay the aspect of retrieval by making all operators candidates for evaluation and selection. However, this becomes cumbersome and inefficient when a large number of operators are stored in memory.

It also seems clear that humans do not retrieve every operator they have ever used when presented with a problem. Rather, they consider a small set of operators that are likely to be relevant to the problem, and they make the final selection from this small set. We have selected a retrieval mechanism with these properties for use in EUREKA.

Naturally, there are drawbacks to this type of retrieval framework. It is possible that the appropriate operators for a given situation may not even be retrieved, much less selected. In this case, the system may not be able to solve a "potentially solvable" problem. However, there is evidence that humans also suffer from this type of difficulty (Dominowski & Jenrick, 1972; Gick & Holyoak, 1980). The possibility of failing to retrieve necessary information comes with the ability to solve many different types of problems efficiently. An accurate model of human behavior should explain the conditions under which humans *cannot* solve a problem. As we have noted above, failure can occur even if the appropriate knowledge is stored somewhere in memory; the knowledge may just be inaccessible to the retrieval mechanism.

SPREADING ACTIVATION IN EUREKA

Although we have already provided a description of spreading activation in chapter 3, we should also give the details of spreading activation as implemented in EUREKA. As we have seen, this mechanism is used to retrieve TRANSFORM goals from memory when the system must choose an operator to help solve the current TRANSFORM goal. At this point, EUREKA spreads activation from the concepts involved in the current goal, including concepts in the current working state, the current goal conditions, and information about the TRANSFORM goal. process is begun.

The spreading-activation algorithm is provided in Table 5. In EUREKA's current implementation, activation spreads in a depth-first manner, although there is no compelling reason why another type of algorithm could not be used. When a node in the semantic network receives an amount of activation, it is first added to any other activation associated with the node and stored. Next, that activation is divided and passed on to the neighboring nodes. Finally, the spreading process is recursively applied to the new set of nodes.

If the spreading process were allowed to proceed in an unrestricted manner, activation would simply spread throughout the network forever, so there are certain restrictions and assumptions associated with the process. First, the amount of activation passed to any neighboring node is always less than the amount of activation in the source node. This is achieved by multiplying the activation value by a "damping" factor. Another restriction requires that if a concept receives an amount of activation that is close to zero (specified by a threshold), activation stops spreading at that point. These two conditions guarantee that the amount of activation being spread decreases and that each activation

Table 5. EUREKA's spreading-activation algorithm.

```
Let ACTIVATION_THRESHOLD be 0.01;
Let DAMPING_FACTOR be 0.4;
Let INITIAL_ACTIVATION be 1.0;

SPREAD_INIT(Source)
  SPREAD(Source,INITIAL_ACTIVATION,NIL)

SPREAD(Source,Value,Path)
  If Value is less than ACTIVATION_THRESHOLD or Source is in Path
     Then EXIT
     Else Increase Source.Activation by Value;
          For each link X from Source
             Let Target be the node connected to Source by X;
             Let Newvalue be
                 SPREAD_VALUE(Source,X,Value)×DAMPING_FACTOR;
             PUSH Source onto Path;
             SPREAD(Target,Newvalue,Path)

SPREAD_VALUE(Source,Link,Value)
  Let Total be 0;
  For each link X from Source
     If X is the same type of link as Link
        Then increase Total by X.trace_strength;
  Return Value×(Link.trace_strenth/Total)
```

path eventually terminates. The current implementation uses a damping value of 0.4 and a threshold of 0.01. Initially activated concepts start with an activation value of 1.0. The implementation includes one additional simplifying assumption: each path of activation is temporarily recorded to ensure that it never cycles. In this way, activation that is spread from a given node can never be spread back to that node after traveling through a cyclical path in the network.

A final detail of the spreading-activation process concerns the manner in which activation is divided when passed from a source node to a number of neighboring nodes. A simple algorithm would divide the activation evenly among all neighboring nodes. If this were the case, however, all neighboring concepts would be equally likely retrieved in spite of their degree of relatedness to the source concept. Anderson (1976, 1983) overcomes this problem by associating trace strengths with the links between nodes and dividing the activation among neighboring concepts in proportion to these trace strengths. In the EUREKA model, we rely heavily on the use of trace strengths to influence retrieval patterns. The system's major learning mechanism involves adjusting these trace strengths

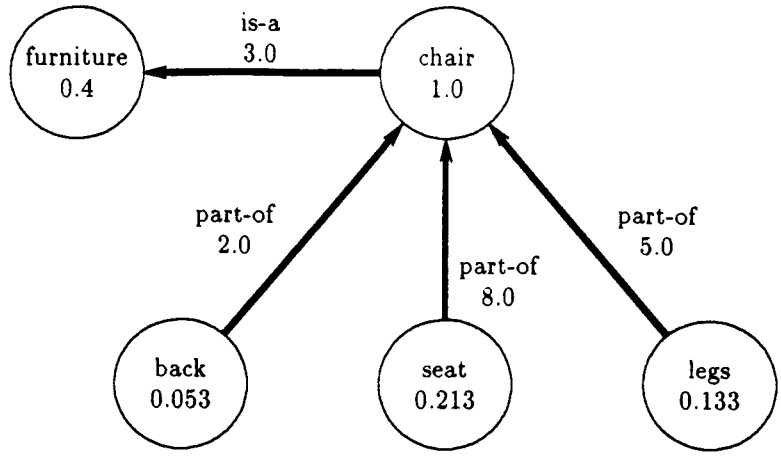

Figure 9. Competition for activation is limited to links of the same type.

to encourage the retrieval of familiar and useful information. Therefore, EUREKA's spreading mechanism also divides activation in proportion to trace strengths, but there is one important additional consideration.

A useful metaphor to describe spreading-activation processes involves the competition of nodes in the network for activation from neighboring source concepts. The competition is based on the various strengths of the links to those concepts. However, in EUREKA many different types of links connect concepts. For instance, the node representing a chair has *part-of* links to its various parts, an *is-a* link to the node for *furniture*, and various links to situations in which a *chair* is involved. For example, it would be undesirable for the node for *furniture* to compete for activation with the node for *chair-legs*, since these two concepts bear very different relationships to the concept of a *chair*. Therefore, EUREKA's spreading mechanism only allows competition for activation between nodes connected to a source by links of the same type.[10]

For example, consider Figure 9, in which *chair* is given an activation level of 1.0, and there are three concepts that are *part-of* a *chair*. The activation available for spreading from *chair* is the initial amount of activation times the damping factor, or $1.0 \times 0.4 = 0.4$. Since *furniture* is the only node connected to chair by an *is-a* link, it receives the full activation value of 0.4. However, each of the three *part-of* nodes receive a share of the 0.4 value based upon the strength of its link to *chair* relative to the other *part-of* links. For example, *seat* gets an activation value of $0.4 \times \frac{8}{15} = 0.213$, whereas its competitors receive less activation.

[10] Neches (1981/1982) provides another alternative for separating the competition for activation. His approach spreads activation through different types of links depending on the system's current situation.

SELECTING AN OPERATOR TO APPLY

In the standard MEA framework, there are two main points at which decisions need to be made. The first involves selecting an operator to APPLY in order to satisfy a TRANSFORM goal. The second involves selecting from a number of possible variable bindings to satisfy a TRANSFORM goal.[11] No decisions are necessary to satisfy APPLY goals because they are handled in a deterministic fashion. Such a goal is either immediately satisfied, or it must have a TRANSFORM goal set up to satisfy the preconditions of the operator. This choice is determined by the current state and the particular operator. Naturally, the points where decisions occur are also the points where mistakes might be made when solving problems. They are also the only points at which a system's behavior can change as a result of learning. In our research, we are primarily concerned with the decision of which operator to select in satisfying a TRANSFORM goal. Choosing a set of bindings with which to satisfy a TRANSFORM goal can also introduce interesting problems, but this is not the focus of the EUREKA model.

The choice of an operator to aid in satisfying the current TRANSFORM goal actually consists of a set of decisions. First, activation is spread throughout the long-term memory, as described in the previous section. This leads to the retrieval of a small number of stored TRANSFORM goals with the most activation. The cutoff point for retrieving goals is set by the activation of the most strongly activated goal. In the current implementation, any goal that has less than one percent of the activation of the most active goal is not retrieved. From empirical studies with the system, we have found that this algorithm usually causes anywhere from one to ten goals to be retrieved. This set of goals is then pruned to include only those TRANSFORM goals that have been previously satisfied, since previous problems that have not been solved are not useful for solving new problems.

After the system has pruned the retrieved set of TRANSFORM goals, it has a small set of situations that are related to the current goal in some way. EUREKA chooses one of these situations and then attempts to APPLY one of the actions taken in that case to solve the current goal. This choice is made by creating a partial match between the current TRANSFORM goal and each of the retrieved goals. Each retrieved goal is assigned a value between zero and one that signifies its degree of match with the current goal. This score is calculated by taking the ratio of the amount of shared structure between the current goal and the retrieved goal to the size of the total structure of the retrieved goal. After assigning a partial-match value, p, to each goal, EUREKA weights the value by multiplying a *selection factor* that represents how useful the particular goal has proven in the past. This factor is calculated from a measure of how often the goal has been chosen as a model (t), and a measure of how often it has actually succeeded in helping to solve a problem (s).

[11] This type of decision is necessary when variables are allowed in the goal and operator conditions.

This leaves each retrieved goal with a numerical attribute, defined as $\frac{s}{t} \times p$, that predicts the relevance of the goal to the current problem.

After all the calculations are done, one of the retrieved goals is randomly selected based on the goals' associated values. The selection is based on a weighted distribution that gives extra precedence to the goals with the highest scores. However, since the selection is random, no candidates are completely ruled out unless they share absolutely no structure with the current goal. It is always possible that a goal that matches poorly will be selected. This allows the system a degree of flexibility, but for the most part it encourages selection of highly matched goals. Once EUREKA has chosen a similar goal, it examines the operators applied in the past to satisfy that goal. The system then chooses one of these operators[12] and analogically transforms it according to the partial match between the goals in an attempt to make the operator applicable to the current goal. Finally, EUREKA sets up the transformed operator in a new APPLY goal to satisfy the current TRANSFORM goal.

The analogical-mapping mechanism computes a number of partial transformations from concepts in the stored goal to concepts in the current goal. The number of transformations is limited by the requirement that at least some structure can be matched between the two goals after transformation. This greatly limits the number of analogical matches that are derived, and prevents the worst-case scenario involving all concepts in one goal mapping to all concepts in the other. After the transformations have been computed, each one is scored according to the new degree of match between goals and the number of analogical assumptions involved. Finally, the best analogical match is chosen in a random manner similar to the selection of a goal from the retrieved set. This mechanism lets EUREKA use stored goals as models even when they do not completely match the current problem.

As an example, suppose the system has already solved the problem shown in Figure 7 and it is given a new problem with an initial state consisting of Blocks E, F, G, and H sitting on a table, and a goal condition of having Block E stacked on Block F. Assuming EUREKA has already retrieved and selected the stored goal, a number of transformations would be proposed by the analogical mapping mechanism, including A→E and B→F, along with others. These two specific transformations would provide the best match between the goal conditions in the current and retrieved TRANSFORM goals. Therefore, EUREKA would select the operator it applied in the old case and analogically map it to STACK(E,F) for the current problem. We should stress that the evaluation of degree of match gives more weight to matches on the goal conditions, giving rise to an MEA type of operator selection. However, the system can select other matches when there are no retrieved goals that match on the conditions well. This can lead to a forward-chaining type of behavior or, in the worst case, a semi-random walk through the problem space.

[12] Usually there is only one operator that led to satisfaction of the retrieved TRANSFORM goal. If there is more than one, a random selection is made.

Note that if EUREKA's current TRANSFORM goal is one that it has previously solved, it has a higher chance of being retrieved by the spreading-activation mechanism. It will also have the highest possible degree of partial match because it is matching against *itself*. This means that the system will tend to repeat whatever it has done successfully in the past to solve the goal. However, we should stress that EUREKA's decisions are based on probabilistic choices, so even in this case it may select a different state, though it would be highly unlikely.

In addition, retrieved goals that are likely to be more relevant to the current situation should have a higher degree of match to the current goal. Because of the high degree of match, the retrieved goal is more likely to be selected. This argument is based on the assumption that structural similarity implies greater relevance. Along with the retrieval mechanism, this discourages EUREKA from selecting operators that work in the domain of chemistry, for example, when it is busy working on a problem in the blocks world. Although this type of selection is discouraged, it is not ruled out completely. In this way, the mechanism allows the selection of a useful situation from another domain that can be used as an analogy to solve a problem in the current domain. Therefore, EUREKA has a single algorithm involving the retrieval, selection, and analogical mapping of stored goals that accounts for a number of types of problem solving. These include cases of straightforward operator application, simple generalization within a domain, and broad analogies across domains.

One remaining question concerns the kinds of knowledge the system has available initially. If EUREKA started without any knowledge in its long-term memory, it would never be able to solve any problems, because there would be no previously solved problems on which to base future decisions. Therefore, EUREKA must start with a set of operators that it can apply to new problems. To be consistent with the problem-solving mechanism, these operators are stored in the form of simple problems that require only one operator application to be solved. Each problem is represented as a simple satisfied goal that is not connected to any other goal in any type of sequence. In this way, each operator initially stands on its own, rather than being involved in a more complicated problem-solving episode. This gives EUREKA the ability to solve new problems before it has seen any other problems.

Learning mechanisms in EUREKA

There are two dimensions along which EUREKA can be said to learn. The first involves a simple type of rote memory that increases the syntactic knowledge of the system. The second is the more interesting process of tuning the strength of relationships between concepts in memory. Both mechanisms respond to past performance in an attempt to

improve future performance. In this section, we discuss these two types of learning and their implementation details.

STORING PROBLEM-SOLVING TRACES

The first type of learning in EUREKA involves the storage of all past behavior in long-term memory. As each problem is solved, a problem-solving trace is integrated into memory. We mentioned earlier that the traces stored in memory record the actions performed by the problem solver, and we also described how these memories are organized. Here, we detail the processes involved in adding a problem-solving trace to memory.

When EUREKA is presented with a problem in the form of a TRANSFORM goal, it first checks to see if it has seen this goal before. If so, the system finds the memory structure representing the goal. This goal is passed to the problem solver and no new structures are added to memory.[13] If the goal cannot be found in memory, a new structure is built to record its presence.

Presumably this goal will require a number of operator applications to be solved. If so, EUREKA invokes its decision procedure to choose an operator with which to set up an APPLY goal. Once this goal has been determined, the APPLY function is recursively called to solve it. Again, EUREKA checks to see if the goal has been seen before, adding it to memory if it has not. In addition, this goal is linked to its parent (the initial TRANSFORM goal). If the current operator can be applied to the current state of the APPLY goal, the operator is executed and the resulting state is passed back to the calling function. If it cannot be applied, further subgoals are set up with recursive calls. For each goal, a new goal structure is added to memory if necessary, and the children are linked to their parents.

Eventually the system returns to the top level TRANSFORM goal. At this point, it has satisfied the APPLY goal that it set up, and has returned that resulting state. The system creates a new TRANSFORM goal in case more transformations need to be carried out to achieve the top-level TRANSFORM goal. This leads again to a recursive call on the TRANSFORM function, with more links and goal structures being added to memory if necessary. After the TRANSFORM goal's children have been processed, they are linked by an AND node to the parent goal. After a number of attempts at solving a problem, the memory structure develops into a directed, acyclic, AND/OR graph similar to the one shown in Figure 6.

The ultimate use of these memory structures is to provide information about EUREKA's past behavior. This lets the system use its past experience with problems and operators to help guide its future behavior. If EUREKA encounters a problem that is similar to an old problem, it can retrieve and examine the goals for the old problem, using these old goals to suggest actions to take in solving the new problem.

[13] Some memory maintenance actually does take place here, but we describe it in the next section.

MAINTENANCE OF THE SEMANTIC NETWORK

As we have discussed, the storage of past problem-solving behavior provides a reference to help guide future problem solving. In contrast, the maintenance of the actual connections between concepts in the semantic network representation provides the ability to influence the use of this information in (hopefully) productive ways. This maintenance consists exclusively of building and strengthening links between concepts in long-term memory. The strengthening activity is designed to facilitate problem solving by encouraging the retrieval of concepts that are familiar and have proven useful in the past.

The simplest task of semantic memory maintenance involves recording familiar concepts. If EUREKA encounters a situation that it has not seen before, it simply adds this problem to memory, as discussed earlier. However, if parts of the situation are already stored in memory, the trace strengths on the existing relations are increased by a small factor. The justification for this is that increased familiarity with a concept should increase that concept's likelihood of retrieval. For example, if the system always ends up with a certain goal while solving a given problem, it will be more likely to retrieve that goal in the future.

The second type of situation in which relations are built and strengthened occurs when a problem (or subproblem) is successfully solved. In this case, strengthening can be viewed as a reward for successful behavior. In particular, goals that were retrieved and aided in solving the current problem are strengthened so that they will more likely be retrieved in similar situations in the future. If certain goals prove to be useful in a large number of situations, they receive large increases in trace strength and become much more likely to be retrieved in appropriate situations. This is a desirable characteristic, since the goal has proven useful so often in the past. Naturally, this can lead to instances where undesirable behavior is produced (as in cases of *Einstellung*), when a goal becomes so easily retrieved that it causes the system to overlook a potentially more useful goal.

RECORDING PAST SUCCESSES AND FAILURES

The final influence on learning in EUREKA concerns the selection of an old TRANSFORM goal from a retrieved set. This involves the selection factor that is multiplied by the degree of match during selection, as described earlier. Recall that we defined the selection factor as $\frac{s}{t}$, where t measures the number of times a goal has been selected as a model in problem solving, and s measures how often this selection has resulted in success. Whenever EUREKA selects a stored goal it increases by one the value t associated with using that goal in the current situation. If the current problem is successfully solved, both s and t are incremented by a *selection learning factor*. In the current implementation, this factor is set to one, but in chapter 5 we will examine changes in EUREKA's behavior due to changes in this value.

This learning factor serves two primary purposes. First, it encourages the selection of situations that have proven useful in the past. More importantly, it discourages EUREKA from duplicating past failed behavior, counteracting effects that can arise from becoming very familiar with an operator that is not useful in the current situation. This lets the model avoid "getting caught in a rut," and it encourages the system to explore more of the problem space.

Discussion

This ends our description of the EUREKA model of problem solving. As we have seen, there are three distinct components in EUREKA, involving the system's memory, performance and retrieval, and learning mechanisms. Although the model is based on a means-ends framework, it has a number of features that distinguish it from standard MEA systems.

First, EUREKA records all of its past behavior in memory to use as a model to guide future problem solving. The model also relaxes MEA's strict requirements for operator selection. EUREKA chooses operators by performing an analogical match on past problems that it has solved. This allows the system to exhibit MEA-style characteristics in general, but also allows the flexibility to break out of that pattern. As we have mentioned, this mechanism lets EUREKA make generalizations within or across domains, and it lets the system semi-randomly search its problem space when it cannot find operators that are clearly relevant to the current problem.

Finally, EUREKA incorporates a model of retrieval based on spreading activation, which provides the ability to focus on local areas of long-term memory and to learn by influencing retrieval patterns. We claim that these mechanisms combine to create a model of problem solving that can account for many aspects of human behavior. In the next chapter, we provide empirical studies of the EUREKA system to support that claim.

CHAPTER 5

Experimental Evaluation of EUREKA

As we have mentioned, the primary purpose of this research is to account for the role of memory and retrieval in human problem solving. However, a theory is useless unless it is testable. Our approach to testing the EUREKA model is to implement it as an artificial-intelligence system. By running experiments on this implementation and analyzing its behavior, we can empirically test the validity of the model. Since we intend EUREKA to explain human behavior, here we present experimental results on the system with respect to the psychological characteristics discussed in chapter 2. For each aspect of problem solving, we discuss some of the relevant psychological research, followed by our predictions about the system's behavior under similar circumstances. Each section also describes the experiments we have run on EUREKA and a discussion of their results.

Psychological validity is not the only interesting aspect of an AI model that can be tested. The EUREKA implementation contains algorithms that depend on certain parameters, and one can examine the limits of its behavior as the values of the parameters change. For a robust system, we expect that small changes in parameter values would not drastically change its behavior. Rather, performance would gradually degrade (or at least change in character) as the parameter values drift farther from their "optimal" values. By varying the parameter values, one can compare the behavior of the system along these lines. In addition, one can explore the behavior of the system with respect to features of the problems it is given. The final section of this chapter describes experiments of this type. They are designed to explore the nature of the system as a computational mechanism rather than its validity as a psychological model.

Evaluation of EUREKA as a psychological model

PERFORMANCE MEASURES

The experiments we have run on psychological aspects of problem solving are primarily concerned with learning. As stated earlier, we define learning in problem solving as the improvement of problem-solving performance. This suggests that the dependent measures in our experiments should involve various aspects of the system's performance. Most of the

experiments in this section are described in terms of the same three dependent measures, so will discuss these measures here.

The first performance measure we examine involves the number of attempts that EUREKA makes in trying to solve a problem. Since the system does not have the ability to backtrack, it usually must make a series of attempts during a single trial before it solves a given problem. EUREKA's current implementation allows only fifty attempts at a problem within a single trial. After fifty attempts, the system gives up and awaits the next problem. It is clear that a reduction in the number of attempts required to solve a problem indicates an improvement in performance, so this is one of our primary dependent variables when we consider learning.

The second measure records the number of means-ends goals that EUREKA visits during a problem-solving trial. This measure represents the amount of time that the system actually spends searching for a solution. In general, we expect that learning would lead to a reduced amount of search, and thus to a quicker solution. This type of measure is often used to compare the performance of problem-solving systems (e.g., Minton, 1988/1989; Shavlik, 1988).

Our final dependent measure concerns the number of *unique* goals that EUREKA visits during a single trial. This measure represents how much territory within the search space the system actually examines when attempting to solve a problem. As we have mentioned, it is possible for EUREKA to duplicate past failed paths during problem solving, so this measure is usually less (and never greater) than the total number of goals visited.

We might assume that improved performance would involve a decrease in the last measure, as with our other two dependent measures. However, we will see that there are times when learning actually causes an increase in the amount of the problem space that is searched. This allows the system to break out of unproductive behavior. Perhaps the best type of learning would involve a decrease in the total number of goals visited, but an increase in the number unique goals visited. This would indicate that EUREKA is spending most of its time exploring new territory. In the following sections we discuss our experiments in terms of these three measures.

PERFORMANCE IMPROVEMENT ON INDIVIDUAL PROBLEMS

Perhaps the simplest type of learning that a problem solver can be expected to exhibit concerns the improvement of performance on an individual problem during repeated attempts at its solution. This is analogous to improving a skill with practice. A degenerate case of this behavior occurs when a system solves a problem once, then always succeeds in solving the problem in the same way during later attempts. In fact, many problem-solving systems behave in this manner (e.g., Fikes, Hart, & Nilsson, 1972; Minton, 1988/1989;

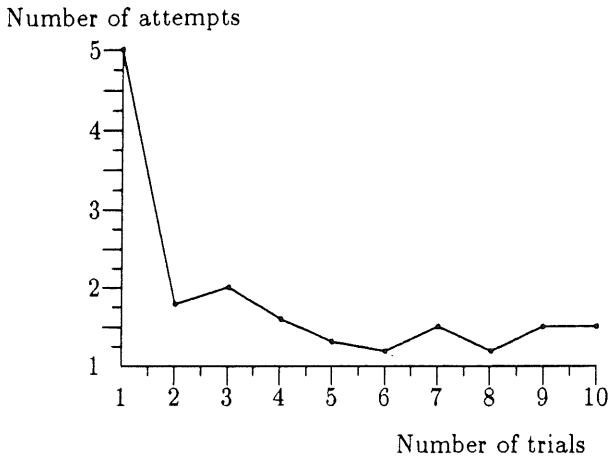

Figure 10. Improvement in number of attempts: practice on individual "Towers of Hanoi" problems.

Mitchell, Utgoff, & Banerji, 1983). However, this would not be a desirable characteristic for a psychological model because humans do not typically behave this way.

In accounting for this characteristic of problem solving, we expect EUREKA to improve as it attempts to solve a problem repeatedly. We measure improvement in terms of the work required before the system solves the problem. Again, we characterize this work by the number of attempts the system makes, the amount of time it spends, and the amount of the problem space that it searches before finally reaching a solution. We expect all three of these factors to be influenced as the system becomes more familiar with the problem. A limiting factor on improvement will naturally arise when the system always finds a solution to the problem without any wasted effort.

To evaluate EUREKA's ability to improve with practice, we created a number of problems based on the "Towers of Hanoi" puzzle used by Anzai and Simon (1979), and the "blocks world" domain, in which a robot hand moves blocks of various sizes and colors to create specified configurations. Each time we gave the system a problem, we presented the same problem ten times in a row. In each of these trials, EUREKA repeatedly attempted to solve the problem until it found the solution or it had made fifty attempts, at which point the system gave up and received the next trial. Learning carried over between trials, and we collected performance measures for each trial of each problem. We then averaged these results across ten sets of ten trials.

The data for the experiment in the "Towers of Hanoi" domain are displayed in Figures 10, 11, and 12, and the data for the "blocks world" problems are provided in Figures 13, 14, and 15. These data match the qualitative behavior we expected. Learning does not take place all at once, but performance steadily improves until a certain level of proficiency is reached. These graphs have the same general shape as a power-law curve.

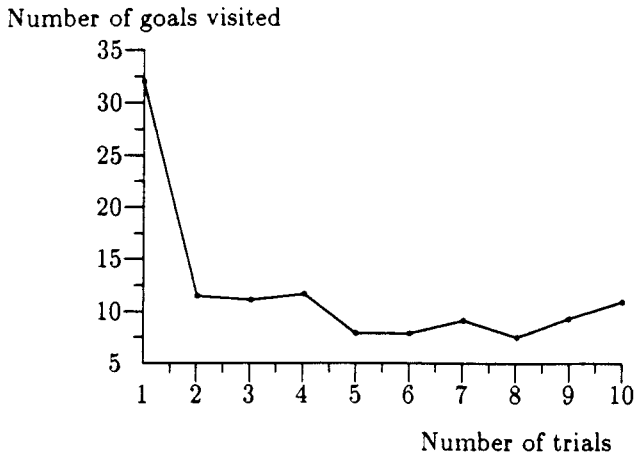

Figure 11. Improvement in number of goals visited: practice on individual "Towers of Hanoi" problems.

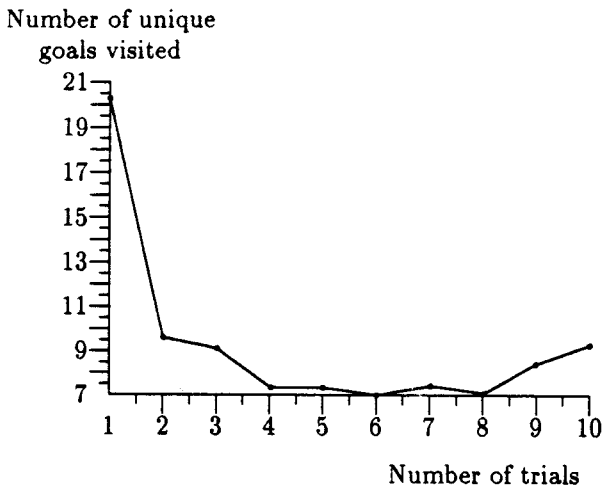

Figure 12. Improvement in amount of problem space searched: practice on individual "Towers of Hanoi" problems.

In particular, notice that most of the improvement occurs in the first one or two trials. However, these curves do not fit a power law. Rather, they are more similar in shape to those calculated from Anzai and Simon's protocol, as shown in Figures 3 and 4.

INTRA-DOMAIN TRANSFER

Another type of learning concerns the improvement of problem-solving ability on difficult problems in a particular domain after solving simpler problems from the same domain. This is a classic type of learning that has been exploited by problem-solving systems that use macro-operators (Fikes et al., 1972; Iba, 1985, 1989; Korf, 1985), operator composition

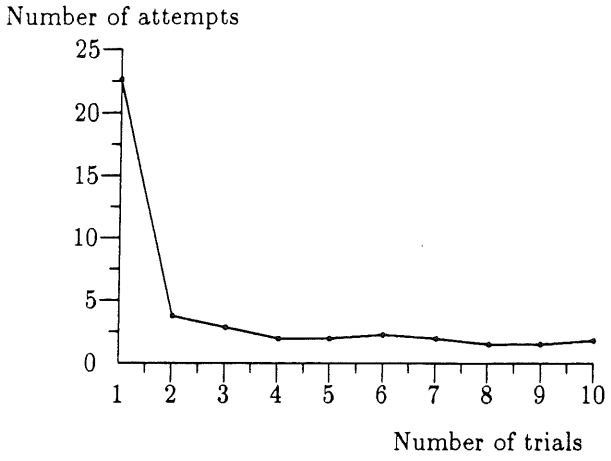

Figure 13. Improvement in number of attempts: practice on individual "blocks world" problems.

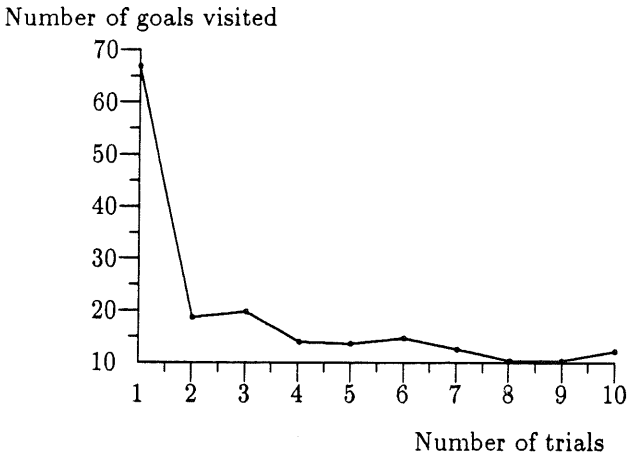

Figure 14. Improvement in number of goals visited: practice on individual "blocks world" problems.

(Anderson, 1983), chunking (Laird et al., 1986a, 1986b), or any type of mechanism that compiles operator sequences into individual units. This approach relies on the fact that difficult problems can be regarded as sequences of smaller, simpler problems. The basic idea, from a computational standpoint, is that if one solves a simple problem that involves a short sequence of operators, one can use that sequence of operators in the future to solve the same problem without having to search for the solution again. This can greatly reduce the amount of search involved in solving a new problem. This occurs when the previously solved problem appears as a small piece of the new problem. The recorded solution reduces the amount of work necessary to solve the new problem and therefore increases the speed with which it is solved. In a system that does not search exhaustively,

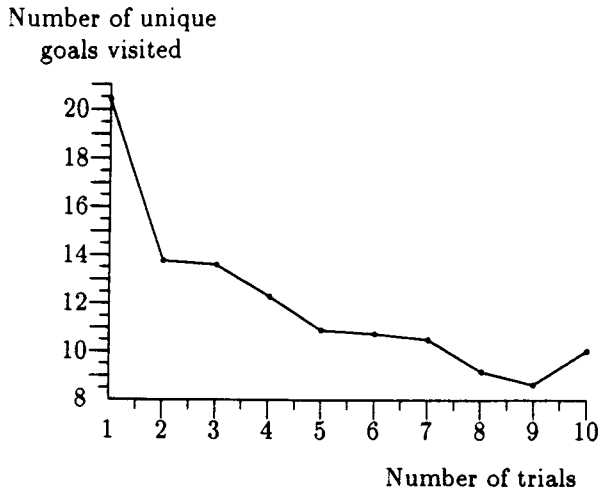

Figure 15. Improvement in amount of problem space searched: practice on individual "blocks world" problems.

having a solution to the small problem increases the probability that the difficult problem will be solved at all. One can argue that this is the primary type of learning that AI problem-solving systems have exhibited so far. For this reason, we should expect EUREKA to display this type of improvement, as well as because it is a type of learning exhibited by humans.

Each goal in EUREKA's memory is treated as a problem that may or may not have been solved. If a given goal has been solved, then the memory structure has been strengthened to reflect the solution. This adjustment should affect future retrieval patterns in such a way that EUREKA is likely to repeat the successful behavior it used to solve the problem. This means that the system should not waste time searching for solutions to goals it has already satisfied, although there is a potential for some search due to the system's random nature. Given this influence, when EUREKA works on a difficult problem for which various pieces have been solved, the likelihood of solving the problem should increase. In addition, the amount of search required to solve the problem should decrease, since the system will not spend time searching portions of the problem space that involve pieces of the problem that are already familiar.

We tested this prediction by measuring EUREKA's performance on small sets of problems from the "Towers of Hanoi" and "blocks world" domains, each problem having a different optimal solution length. In the control condition, we ran each problem separately, throwing out any learned memory structures between each trial. For the test condition, we then ordered the problems by optimal solution length and ran them successively, allowing learning to take place. Figures 16, 17, and 18 compare EUREKA's ability to solve the "Towers of Hanoi" problems under the control and test conditions, whereas Figures 19, 20, and 21 show similar results from the "blocks world" problems. Again, the dependent

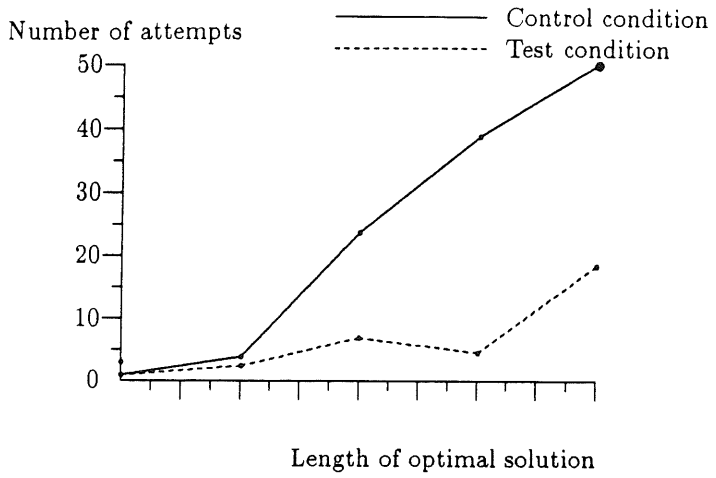

Figure 16. Improved performance in number of attempts: solving "Towers of Hanoi" problems in order of optimal solution length.

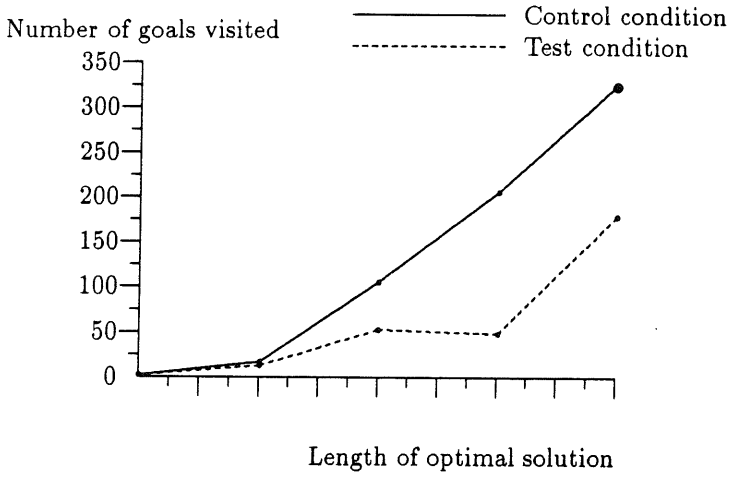

Figure 17. Improved performance in number of goals visited: solving "Towers of Hanoi" problems in order of optimal solution length.

measures of performance include the number of attempts required to solve the problems, the number of goals examined, and the number of unique goals examined.

For the control condition, the system's performance degrades as optimal solution length increases. In fact, the longest problems were so difficult that EUREKA was never able to solve them. These problems are marked by larger bullets. For the test condition, one can see that the ability to transfer knowledge from simpler problems to more difficult problems significantly improves the system's performance along all three dimensions. Under these conditions, EUREKA was able to solve even the most difficult problems easily, although it could not solve them at all under the control condition.

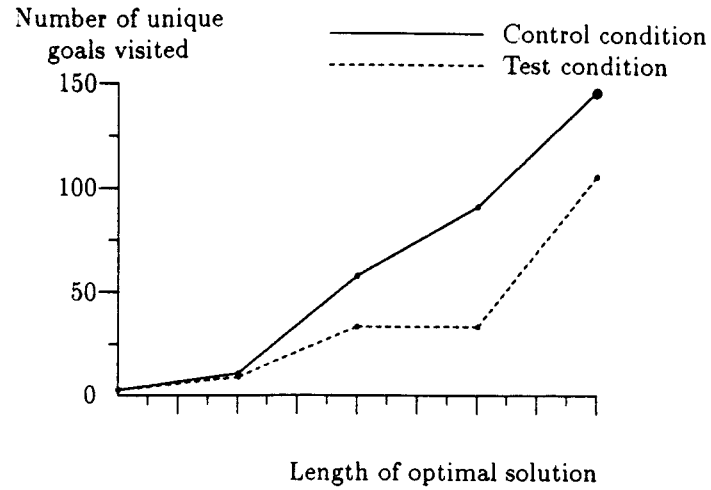

Figure 18. Improved performance in amount of problem space searched: solving "Towers of Hanoi" problems in order of optimal solution length.

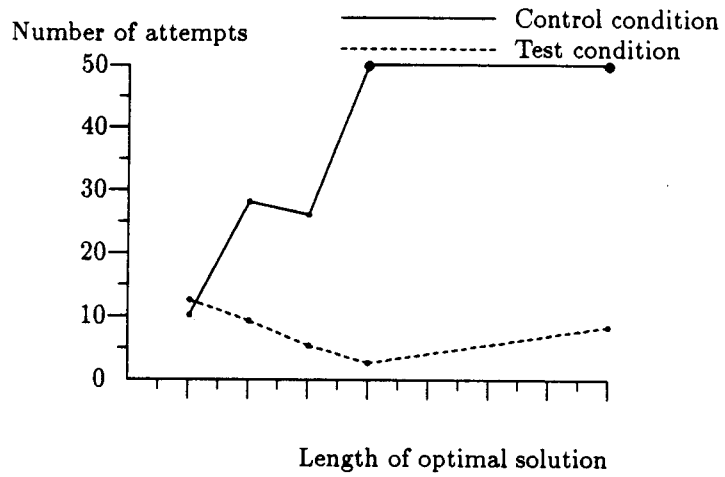

Figure 19. Improved performance in number of attempts: solving "blocks world" problems in order of optimal solution length.

By examining the number of goals visited (Figures 17 and 20) and the number of unique goals visited (Figures 18 and 21), one can see that improvement occurs in the form of a reduction in the total amount of search required to solve each problem. This arises from the fact that EUREKA has stored familiar, successfully solved pieces of the more difficult problems, making it less likely to wander down unproductive paths.. Each pair of points corresponds to a comparison between performance with and without learning, as exhibited in Table 1 from Anzai and Simon's protocol. EUREKA shows the same ability to improve from intra-domain transfer that the subject in the protocol showed.

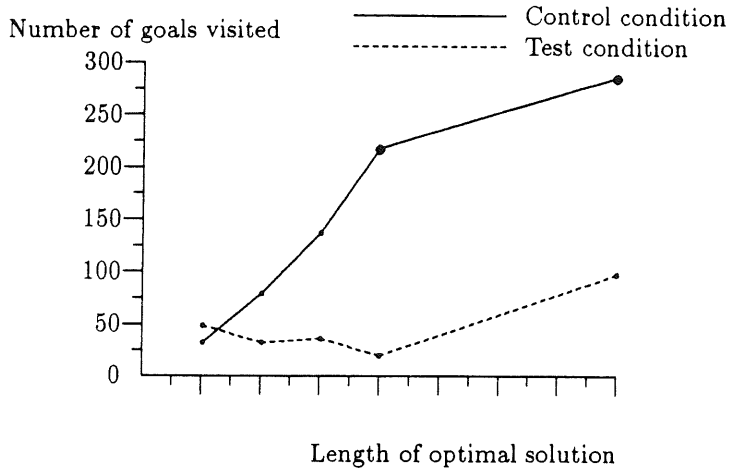

Figure 20. Improved performance in number of goals visited: solving "blocks world" problems in order of optimal solution length.

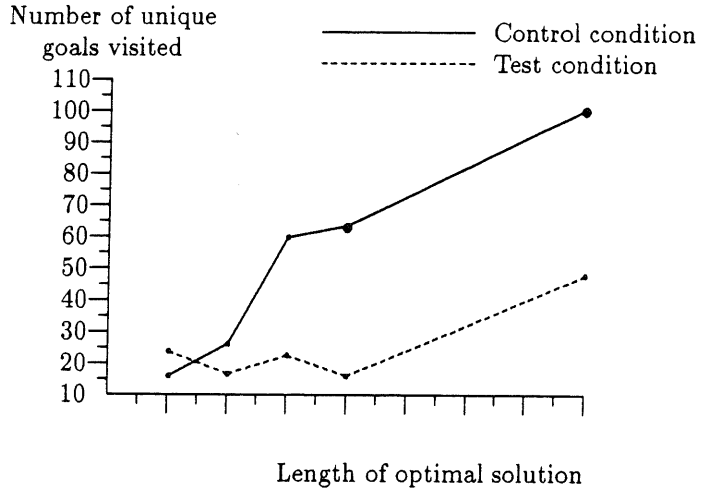

Figure 21. Improved performance in amount of problem space searched: solving "blocks world" problems in order of optimal solution length.

INTER-DOMAIN TRANSFER OR ANALOGY

Transfer is also possible between problems from different domains. This type of transfer usually involves the use of an analogy from one domain to aid problem solving in another domain.

The subject of analogy has received a large amount of attention within both AI and cognitive psychology. In fact, the term has been applied to so many different things that it is difficult to formulate a succinct definition. Hall (1989) provides a good review of research on many different uses and types of analogy. In the current research we are

Table 6. Results of the experiment on analogy in EUREKA.

CONDITION	% SUCCESSES	ATTEMPTS	NODES	UNIQUE NODES
CONTROL	50	37.7	117.2	38.2
TEST	80	13.2	48.2	18.7

strictly interested in the use of analogy in problem solving. To be even more precise, we are concerned with the ability to retrieve and generalize operators in an attempt to apply them to problems in new domains.

Gick and Holyoak (1980) have performed a series of experiments dealing with the ability to solve new problems by finding analogies with old stories and problems. They demonstrated that humans can solve otherwise difficult problems using analogies. They also showed that the specific nature of a given problem greatly influences the ease with which it can be applied in an analogical situation. However, they have also consistently found that humans are rather poor at analogical problem solving. Often, subjects must be almost "spoon fed" the analogy before they can generate an appropriate solution to a given problem.

EUREKA's ability to solve problems by analogy stems from the same mechanism that allows transfer within domains. The only difference is that some generalization must be done when matching the retrieved TRANSFORM goals. In this section, we report experiments designed to test EUREKA's ability to account for the psychological data on analogical problem solving.

For the experiment on analogy in EUREKA, we gave the system problems from the two separate domains provided by Holyoak and Koh (1987). Our test problem was the "radiation" problem, and the source analogy was the "broken lightbulb" problem. We supplied the system with operators capable of solving the lightbulb problem and ran it under two conditions. In the control condition, EUREKA attempted to solve the radiation problem without any previous experience. In the test condition, the system first solved the lightbulb problem, and then attempted to solve the radiation problem.

The results of this experiment are provided in Table 6. EUREKA exhibited improvements in the average number of attempts taken to solve the radiation problem, the number goals visited, and the number of unique goals visited. The system also shows improvement in its ability to solve the test problem at all. Given a maximum of fifty attempts to solve the problem, EUREKA was only able to complete the task fifty percent of the time under the control condition. However, after having solved an analogical problem, the system was

able to solve the test problem eighty percent of the time, showing an improvement from the use of analogy similar to that observed by Holyoak and Koh.

NEGATIVE TRANSFER OR *Einstellung*

As we have mentioned at various points throughout this dissertation, the flexibility that comes with being able to transfer past experience to new situations also has disadvantages. Perhaps the most important of these disadvantages arises when an attempt is made to transfer knowledge to a new problem inappropriately. This may happen when a problem solver sees a new situation that somehow looks similar to a familiar situation.

We found that EUREKA suffers the same problems as humans on the water jug problems from Luchins' experiments on Einstellung. Recall that the mechanisms in EUREKA are designed to encourage retrieval of useful information by increasing the strengths on relations in the semantic network. These adjustments are designed to increase the likelihood that old goals will be retrieved in the types of situations to which they have been successfully applied. This encourages the system to retrieve goals that are familiar and that have been used successfully in the past. The same goals are retrieved even though there might be more useful information contained somewhere else in memory. In these cases, the more familiar information blocks the retrieval and selection of information that is more appropriate to the current problem.[14] We ran EUREKA on the problems provided in Table 2. As in Luchins' experiment, EUREKA failed to find optimal solutions to the test problems after solving the series of control problems.

EXTERNAL CUES IN THE FORM OF HINTS

Earlier we mentioned the use of hints to improve problem-solving behavior. In the EUREKA model, we view hints as one instance of a more general category of external cues from the problem solver's environment. In the future, we plan to test EUREKA with "incidental" cues in an attempt to account for episodes of insight. In the current work, we have tested the system's behavior with respect to intentional, beneficial cues, or hints.

To test EUREKA along these lines, we again ran experiments with the problems used by Holyoak and Koh (1987). In this experiment, we provided EUREKA with operators for solving a "broken lightbulb" problem. However, we did not have the system actually attempt to solve that problem. Rather, in the control condition, we gave the system the task of solving the radiation problem in a normal manner. In the test condition, EUREKA was again required to solve the "radiation" problem. However, this time semantic-network

[14] Neves and Anderson (1981) have suggested an alternative explanation, in which Einstellung arises from the use of operator composition for learning.

Table 7. Improved performance through the use of hints.

CONDITION	% SUCCESSES	ATTEMPTS	NODES	UNIQUE NODES
CONTROL	50	37.7	117.2	38.2
TEST	100	11.7	39.5	19.7

nodes involved in the "broken lightbulb" problem were activated during every problem-solving cycle. This served as a hint in the form of an external cue to retrieve the appropriate operators to solve the problem.

Table 7 provides the results of this experiment. The data for the control condition are identical to the data for the experiment on analogy because the control conditions for each experiment were the same. We can also see from the table that the hints provided to the system dramatically reduced the number of attempts and the amount of search it required to solve the problem. In addition, when EUREKA received hints, it was always able to solve the problem. Thus, the effect of hints appears to be even more dramatic than that found by Holyoak and Koh with humans. This could be because the hints we gave the system were somehow stronger than the hints that they gave to their subjects.

Evaluation of computational characteristics

The experiments in the previous section concentrated on providing support for EUREKA's status as a psychological model of the role of memory and retrieval in problem solving. However, in addition to being a psychological model, EUREKA is implemented as a computer system, and it has a number of characteristics that can be tested systematically. Here we focus on some of those computational characteristics.

Recently, there has been an increasing interest in extensively testing AI systems (Kibler & Langley, 1988). At some level, every AI program is a formal system that contains two distinct parts. One component is the *system architecture*, which includes the definable assumptions and biases that are built into the system and not expected to change. Some examples of parts of EUREKA's architecture include the knowledge-representation schemes, the spreading-activation retrieval mechanism, and the decision algorithms.

The second part of an AI system involves the set of *parameters* that are associated with the architectural components. Most systems have these, although they are sometimes not obvious. We feel it is important to make these parameters explicit in order to test a system's computational characteristics. Ideally, the system's performance will not overly depend on particular values for its parameters. If behavior is very sensitive to particular

values, this suggests that the architectural principles of the system are not responsible for that behavior. Rather, it indicates that there is extra knowledge hidden in the parameter values, which can lead to brittleness or other questionable behavior.

Another way to test the computational characteristics of a system is to control the input given to the system and measure its behavior with respect to features of the input. This is the same type of experiment that we used in the psychological section of this chapter. However, there are also behaviors that EUREKA exhibits that are not specifically addressed in the psychological literature. Therefore, the remaining experiments in this chapter are of this type. The results of these experiments not only display interesting aspects of EUREKA's behavior, but may serve as the basis for specific predictions about human behavior that can be tested in the future.

SYSTEM PARAMETERS

The first two experiments in this section were designed with the intent of showing that EUREKA behaves well across a wide range of settings for the parameters involved in its decision algorithms. There are two primary parameters involved in the decision points: one concerns the amount that the trace strengths on links are increased during problem solving, and the other involves the amount of punishment or reward the system associates with selecting a retrieved goal as a model for future problems. These factors influence behavior in the *retrieval* of a set of TRANSFORM goals from memory, and in the *selection* of a single goal from that set. In principle, changing the values of these parameters could drastically change the amount of information retrieved from memory and the likelihood that it will be used as a model once it has been retrieved. It is not necessarily desirable (or possible) to come up with a "best" set of values for the parameters. Rather, the particular parameter values represent a specific bias in a continuum of possible behaviors. For these experiments, we wish to explore this behavior space and exhibit the tendencies of the system with respect to certain ranges of the parameter values.

Retrieval of information

Our first experiment measured EUREKA's behavior with respect to the parameter for increasing the weights of the links in memory after a problem has been successfully solved. There are two occasions in which trace strengths are incremented. Whenever a relation is encountered that is already stored in memory, the link representing that relation has its trace strength incremented by one. However, when the system succeeds in solving a problem, the trace strengths of all links associated in situations that helped to solve the problem are increased by a factor v. In the first part of this experiment, we tested the effects of this factor on intra-domain generalization in the blocks world.[15] That is, we gave the system overly-specific operators and measured its ability to solve a new problem

[15] More detailed experiments on intra-domain generalization are described later in this chapter.

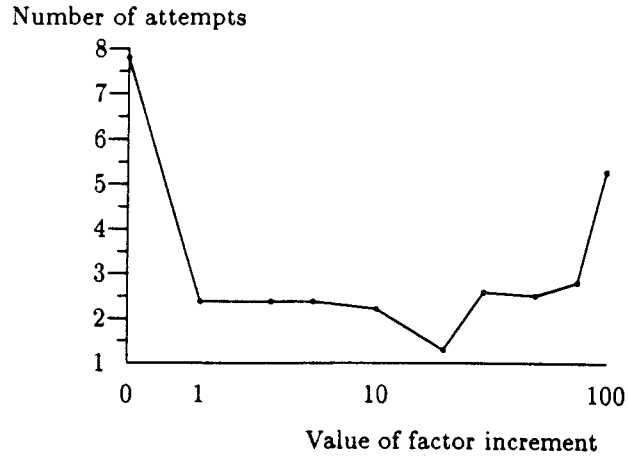

Figure 22. Number of attempts compared to the retrieval increment factor, v.

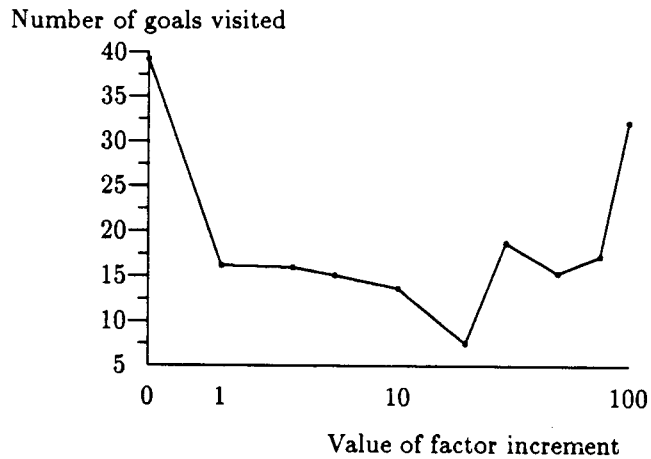

Figure 23. Number of goals compared to the retrieval increment factor, v.

after it had solved a similar problem to which the operators were directly applicable. We tested this system with a number of different values for the link strengthening factor, v, producing the results presented in Figures 22, 23, and 24.

Each of these graphs shows an initial decline in the effort spent on solving the new problem as the learning factor, v, increases. This occurs because the successful use of the overly specific operators on the test problem causes the likelihood that the operators will be retrieved in future similar situations to increase with v. However, it is interesting to note that after the point where $v = 20$, performance actually starts to degrade. An explanation for this is that operators receive too much reward for being successful in the training problem and they become easily retrieved even when they are inappropriate to

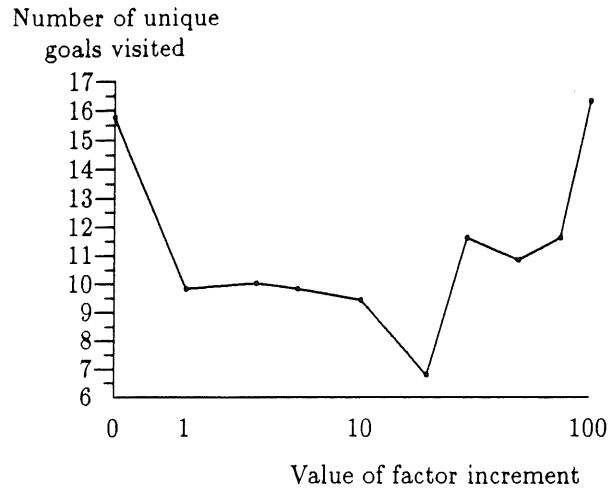

Figure 24. Number of unique goals compared to the retrieval increment factor, v.

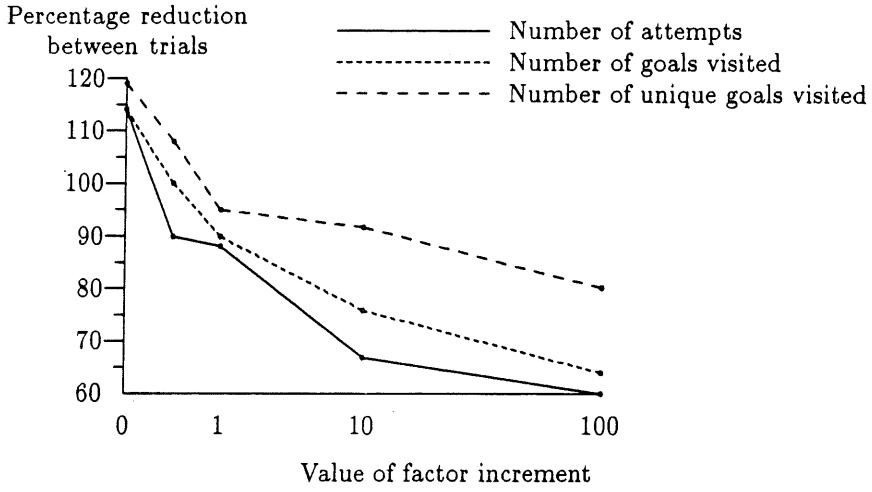

Figure 25. EUREKA's behavior with respect to the retrieval increment factor, v.

a new problem. This also indicates that Einstellung effects increase as v gets very large. In the psychological experiments in this chapter, we used a value of $v = 5$ in order to get effective learning without having too strong an Einstellung effect.

In the second part of this experiment, we repeated some of the experiments on practice effects with various values of v. Specifically, we had the system attempt to solve a problem multiple times. Then we compared the effort spent on the problem in the first trial with the effort spent on the last trial. This comparison was calculated as a percentage decrease in effort and the results are shown in Figure 25.

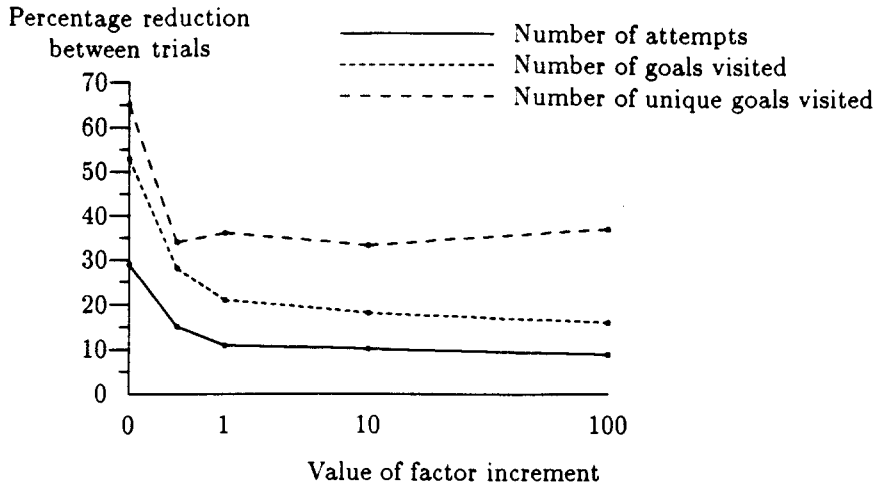

Figure 26. EUREKA's behavior with respect to the selection increment factor, w.

A lower value in this figure indicates a greater improvement in performance across trials. Along all the measures of system effort, performance improvement appears to increase with v, although this increase becomes less pronounced as v becomes large. There do not appear to be any negative transfer effects in this experiment, but that is to be expected since a single problem was being solved for each trial. Therefore, no transfer was occurring between distinct problems.

Selection of retrieved information

Our second experiment examined the factor used to select an old goal as a model once it has been retrieved. Recall that this factor is multiplied by the degree of match between two goals to derive a final factor for selection. The selection factor is computed by storing two values: t is a measure of how often a goal has been selected for use as a model in a particular situation, and s is a measure of how often a problem has been solved when the goal was chosen in that situation. When a problem is solved, each goal that was used as a model has its s and t attributes incremented by a fixed value w. When EUREKA fails to solve a problem, only t is incremented. The increment factor, w, is the variable of interest in this experiment.

As in the second part of the experiment on the retrieval parameter, we ran EUREKA on a subset of the trials from the experiment on improvement with practice in individual problems. This time, however, we varied the value of w between zero and 100, measuring the percentage change in each dependent variable between the first and last trial. Again, a decrease in this value represents an average increase in performance improvement. The results are graphed in Figure 26. The number of attempts made to solve each problem and the number of goals visited exhibit a gradual improvement as w becomes large, both

appearing to reach asymptotic values at about $w = 1$. The percentage decrease in number of unique goals visited achieves asymptote at an even smaller value of w.

These results are consistent with what we know of w's role in EUREKA. This factor's major purpose is to encourage the system to explore new paths after failures and to prefer old paths that have been successful. As such, we would expect improvement on individual problems to be more dramatic as the factor is increased. However, it is interesting that increases in w appear to have little impact as it becomes large. For the psychological experiments in this chapter we chose the relatively small value of $w = 1$, since this represents the point at which w's effect reaches an asymptote.

COMPARING RETRIEVAL TIME TO MEMORY SIZE

In chapter 3, we provided a simplified analysis that suggested that the time taken to execute a retrieval process based on spreading activation would be independent of the total size of memory. This is a desirable characteristic because it means that the problem solver will not slow down as knowledge is added to the system. However, our analysis contained a number of simplifying assumptions, and it is not clear that the results hold for the specific implementation of spreading activation in EUREKA.

In order to supplement this analysis, we ran an experiment in which we continuously added knowledge to EUREKA's semantic network. At various points throughout this process, we started the retrieval process by spreading activation from a small set of specified nodes. Finally, we graphed the time taken to spread activation from each source against the total number of nodes in the network. These results are provided in Figure 27. Each curve in the figure represents the spreading time from a single source node. The most obvious characteristic of this graph is that each curve eventually levels off, indicating the type of behavior that we predicted. For large networks, retrieval time does seem to be independent of the size of memory.

There are some other aspects of this graph that should be discussed. First, notice that sometimes spreading activation appears to visit more nodes than are in the network. This happens because there is a large number of cycles in a typical network, so individual nodes are visited many times during the spreading process. Thus, even if the total number of nodes visited is larger than the number of nodes in the network, it does not mean that every node in the network is visited.

Also, notice that the curves are very jagged for small network sizes. This suggests that retrieval time is influenced quite a bit by the specific structure of the network, rather than by its size. Apparently, adding a few links or altering the strengths on links can significantly influence retrieval time, at least for small networks. The curves eventually appear to smooth out, but this could be partly because our interval for measuring retrieval time increased as the network grew.

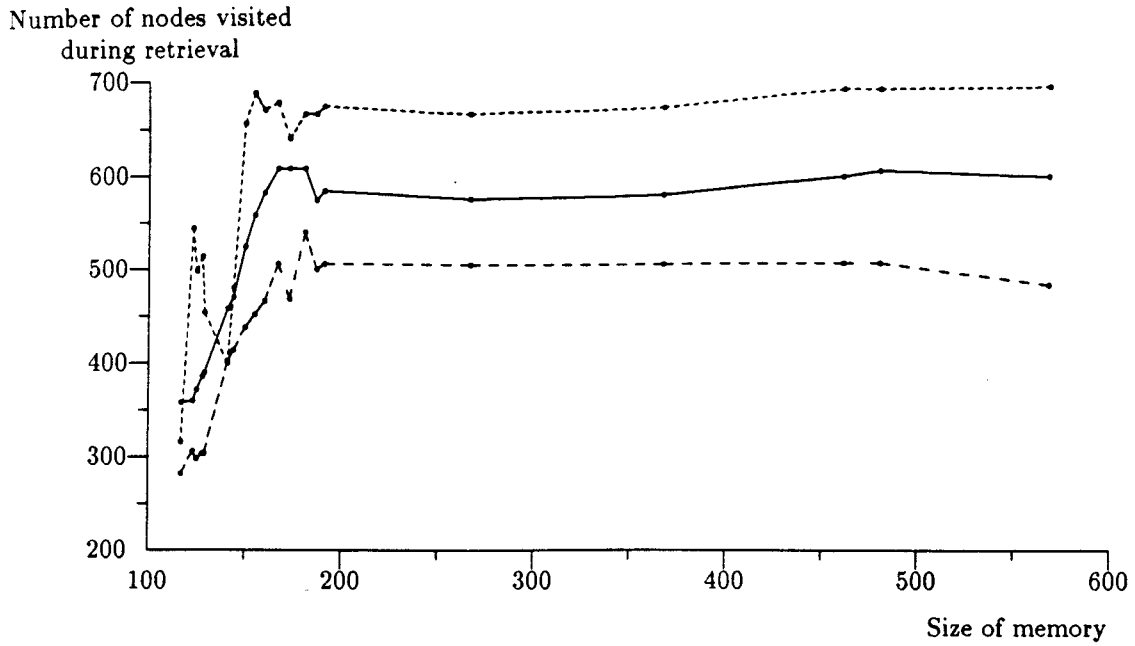

Figure 27. Comparing retrieval time to total network size.

INTRA-DOMAIN GENERALIZATION

An interesting phenomenon that may not seem directly related to analogy involves simple generalization within a domain. In our view, analogy is a form of transfer *across* domains, whereas generalization involves transfer *within* a domain. Since problems solved by analogy often look quite different from each other (by virtue of the fact that they are from different domains), this approach to problem solving is often thought to be a more creative act than generalizing knowledge from one problem to another within the same domain. In EUREKA, these two types of transfer are simply two manifestations of a single process. The same underlying mechanisms that allow using analogies across domains allow generalization within a single domain.

Generalization in problem solving has also received attention from other AI researchers (Anderson, 1983; Holland et al., 1986). However, generalization in systems such as ACT* and PI appears as one of a number of explicit learning mechanisms. In EUREKA, generalization arises from the same partial-matching algorithm that is used for straightforward problem solving and analogy. Thus, the model provides a mechanism for retrieving operators that can be generalized, and also specifies the conditions under which generalization occurs.

To test EUREKA's behavior with respect to generalization, we supplied the system with a set of very specific operators from the "blocks world" domain that contained no variables. For example, rather than giving the system a general operator for picking up any block,

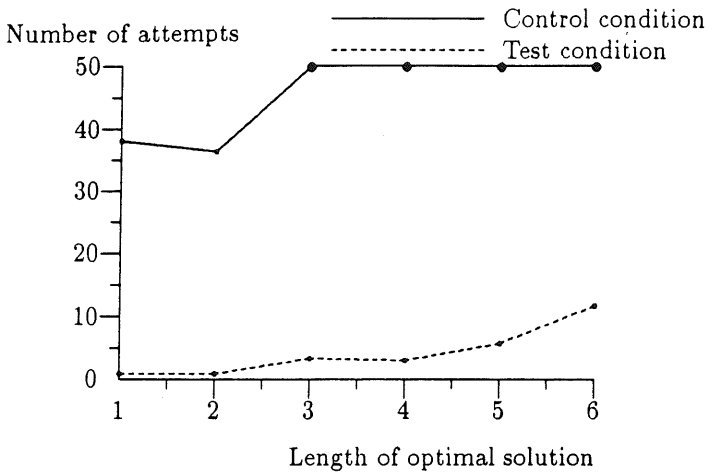

Figure 28. Improvement in number of attempts: generalization of past performance.

we gave it an operator for picking up Block A. EUREKA could not apply this operator directly to pick up any other blocks. More important, if the system had the goal to pick up Block B, it would be less likely to retrieve this operator because it would not receive much activation.

As a control, we ran EUREKA on problems that could only be solved by generalizing the constants in the operators to cover the new situations. In the test condition, we ran EUREKA on the same set of problems after first giving the system similar problems that could be solved without generalization. This experiment is similar to the experiment on analogy. However, we expect a much greater improvement between the control and test conditions, because the overly specific operators appear in very similar contexts once they have been successfully used to solve similar problems.

The data from this experiment are displayed in Figures 28, 29, and 30. As expected, the results of this study are similar in many ways to the results concerning the use of analogy. EUREKA had a very difficult time solving any of the problems in the control condition, and it was not able to solve the most difficult problems at all (again, problems that were not solved after fifty attempts are marked with larger bullets). However, after having solved similar problems for which directly applicable operators were available, performance improved dramatically. Both the number of attempts necessary to solve each problem and the amount of search required were reduced. In addition, for the more difficult problems, learning actually *increased* the amount of the problem space that was searched. In this case, it appears that learning has reduced the total amount of time required to solve new problems, but has increased the amount of productive time spent investigating new areas of the problem space.

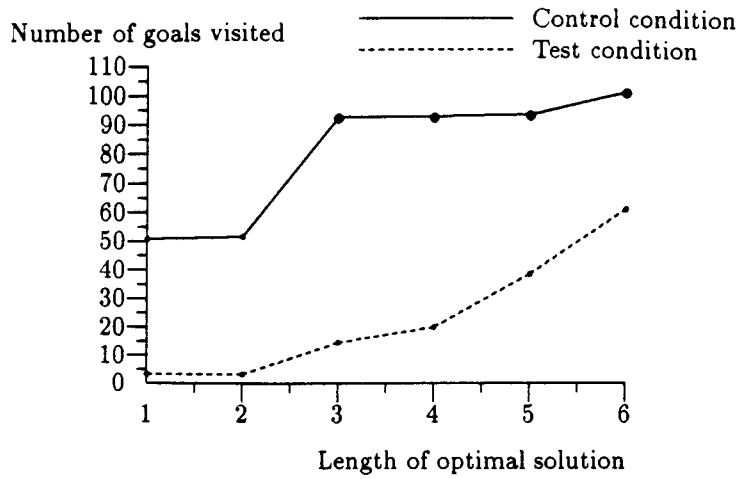

Figure 29. Improvement in number of goals visited: generalization of past performance.

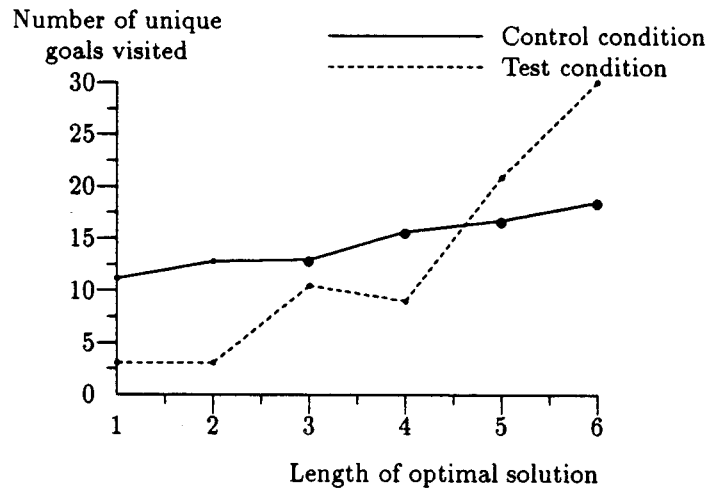

Figure 30. Improvement in amount of problem space searched: generalization of past performance.

These results again match our predictions of EUREKA's behavior. It is important to note that the system always had the potential to solve every problem under both the control and test conditions. However, in the control condition, performance was hampered by the inability to retrieve appropriate information from memory, because all the operators were overly specific.

EFFECTS OF GOAL INTERACTIONS

Our final computational study involving EUREKA concerns problems that contain various degrees of goal interactions. In some of our experiments, we used the independent

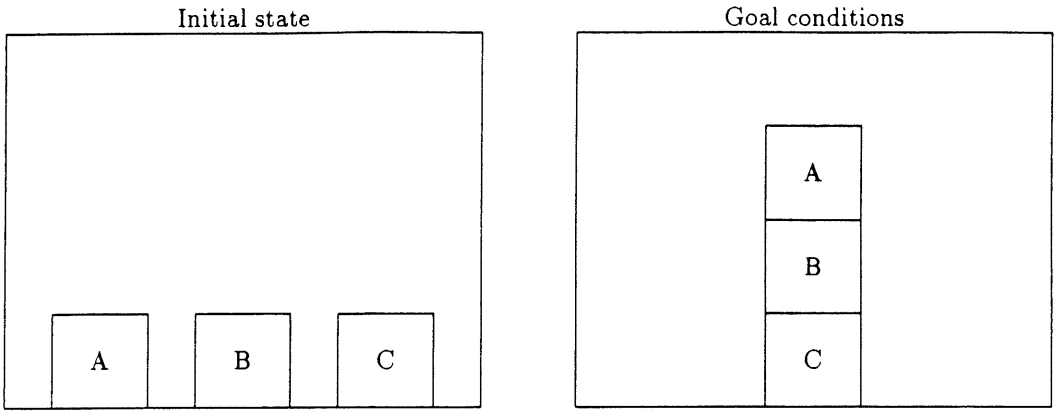

Figure 31. A "blocks world" problem with goal interactions.

measure of optimal solution length to estimate the difficulty of a problem. However, a common feature of humans and systems that use means-ends analysis is that they have a hard time solving problems in which various goals interact with each other.

A classic example of a goal interaction in the "blocks world" domain involves building a tower three blocks high (see Table 31). A means-ends system must decide which of the two goals to satisfy first. If the system chooses to stack A on B, it will not be able to stack B onto C without first taking A off of B again. This would lead to a suboptimal solution involving eight operator applications. The optimal solution to this problem involves first stacking B on C and then stacking A on B, requiring only four operator applications.

It is clear from this example that optimal solution length is not the only factor that determines the difficulty of a problem. We have developed a new measure of problem difficulty that takes both optimal solution length and degree of goal interaction into account. Certainly there may be other factors involved in problem difficulty, but this at least gives us a more accurate measure.

Our measure for problem difficulty involves the *expected solution length* of a problem. This measure assumes that, given a number of goals to solve, a means-ends system has an equal chance of choosing any particular goal to work on next. We can use this random-selection strategy to find the average solution length over all possible goal orderings. In addition, we can calculate the *degree of goal interaction* of a problem by dividing the expected solution length by the optimal solution length. In the example provided above, the expected solution length is $\frac{8+4}{2} = 6$, and the degree of goal interaction is $\frac{6}{4} = 1.5$.

We have used these measures to study how EUREKA behaves when given problems that contain goal interactions. We expect that the amount of effort the system spends solving a problem will depend both on the optimal solution length and the degree of goal interaction of the problem. However, after EUREKA has learned a solution to the problem, it should be able to avoid any extra work that arises from goal interactions.

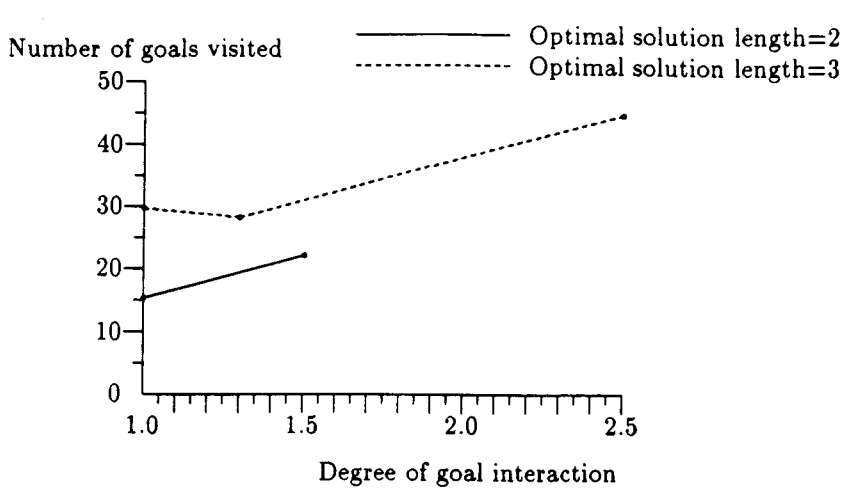

Figure 32. Number of goals visited: First trial on "blocks world" problems of varied complexity.

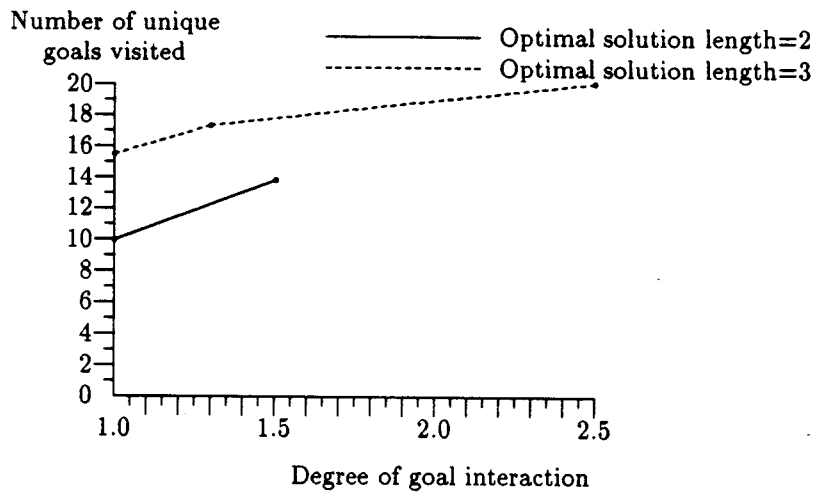

Figure 33. Number of unique goals visited: First trial on "blocks world" problems of varied complexity.

To test these predictions, we designed a number of "blocks world" problems with various optimal solution lengths and degrees of goal interaction. We then had EUREKA solve each problem twice, and we measured the number of goals and unique goals visited during each trial. The results are provided in Figures 32, 33, 34, and 35.

First, examining Figures 32 and 33, we see that the effort expended by EUREKA on a problem does indeed depend on both optimal solution length and degree of goal interaction. We do not make any predictions about the precise quantitative relationships between these measures because of the small number of data points. In addition, Figures 34 and 35 show that, once EUREKA has solved a problem before, the amount of effort expended no longer

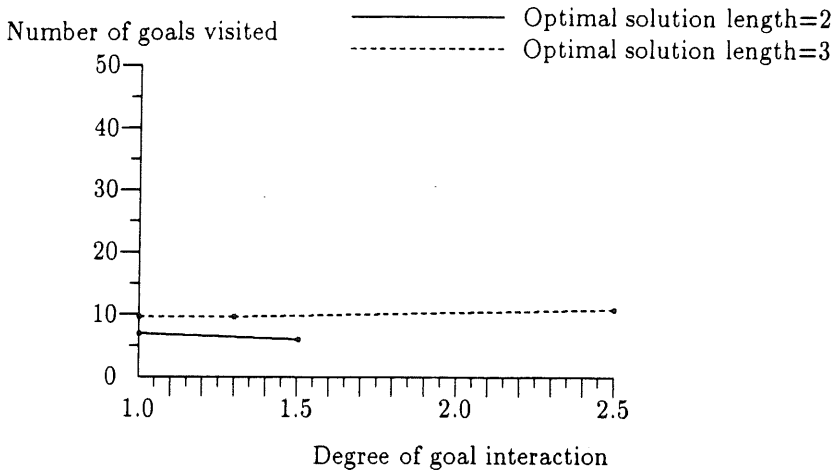

Figure 34. Number of goals visited: Second trial on "blocks world" problems of varied complexity.

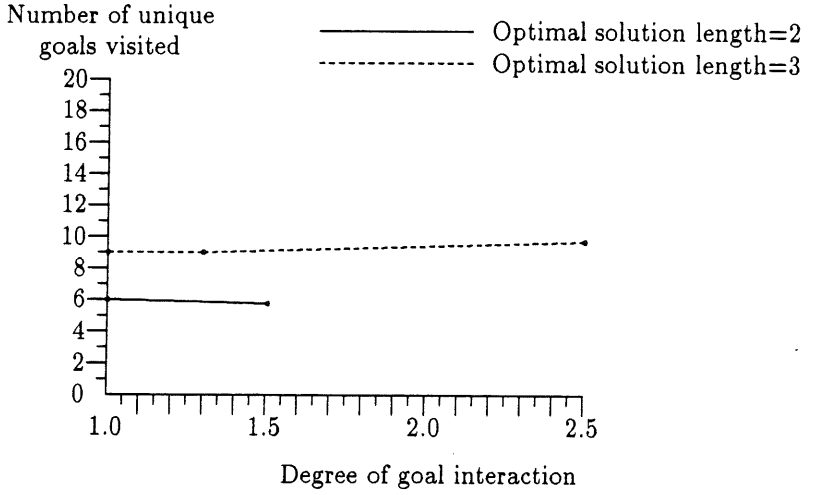

Figure 35. Number of unique goals visited: Second trial on "blocks world" problems of varied complexity.

depends on the degree of goal interaction. This indicates that EUREKA has at least some ability to overcome difficulties caused by goal interactions.

Although this is a small study, it provides us with some interesting information about the character of problems involving goal interactions. First, the experiment shows that, like humans, EUREKA has more difficulty with problems that have more goal interactions, but also has the ability to overcome those difficulties. Also, the experiment indicates that expected solution length may indeed provide a reasonable measure of problem difficulty.

Summary

The experiments in this chapter have given us insights into how EUREKA behaves under various conditions. In the section on psychological evaluation, we were able to show that the model exhibits a number of behavioral characteristics that humans also exhibit. In the computational section, we examined how EUREKA responds to various parameter settings and explored some of the system's behavior that does not specifically account for psychological phenomena. In chapter 7, we will further discuss the conclusions we draw from the results of these experiments.

CHAPTER 6

EUREKA in Perspective

Many of the ideas in our research have been influenced by other work in the area of problem solving. In addition, there is a large amount of work that has not directly influenced ours, but bears enough relation to justify discussing its differences and similarities. In this chapter, we introduce a number of dimensions that are useful for describing problem-solving systems. In addition, we analyze EUREKA and a number of other problem-solving systems along these lines to compare alternative approaches to addressing these issues. The dimensions we have chosen to describe problem-solving systems include the representation of knowledge, the basic problem-solving approach, mechanisms for retrieval and selection of long-term knowledge, learning issues, problem solving by analogy, and reaction to the environment.

Knowledge representation

A major decision in designing a problem-solving system concerns the representation of knowledge. Usually, a problem solver has at least two types of memory. One is a long-term memory that holds permanent, general knowledge about the world and particular problem domains. Another is a temporary, short-term, or working memory that is used to keep track of information being used to solve a current problem. In addition, a distinction is often made between *procedural* and *declarative* knowledge. Procedural knowledge is usually represented in the form of rules or productions, and represents knowledge of how to perform actions or to make specific inferences. In contrast, declarative knowledge involves information and beliefs about the state of the world, and is usually represented in the form of non-executable facts.

In many respects, this is a philosophical distinction rather than a technical one. Procedural knowledge is generally assumed to be a sort of compiled knowledge that is directly applicable. In contrast, declarative knowledge must be interpreted before manifesting itself in the system's performance. At the implementation level of a computer system, all types of knowledge end up being interpreted to some extent, and therefore are arguably declarative. In another sense, all knowledge is inherently procedural because it must be "executed" to bear on system performance. Therefore, the distinction appears to have less relation to implementation-level decisions than to assumptions about the human cognitive architecture or the computational approach.

There has been a wealth of research that uses production rules to model procedural knowledge in long-term memory. Systems that have taken this approach include SOAR (Laird et al., 1986a, 1986b), PRODIGY (Minton, 1988/1989), UPL (Ohlsson, 1983, 1987), PI (Holland et al., 1986), and ACT* (Anderson, 1983). Most of these systems limit the contents of long-term memory to procedural knowledge. That is, all long-term knowledge is encoded in the form of productions. Anderson's ACT* is a notable exception in that it contains both a procedural long-term memory and a declarative one. In this way, Anderson distinguishes between facts and procedures. For example, ACT* might store the information that all birds have wings as a declarative fact. In contrast, a procedural approach, such as that used by SOAR, might store the same information as the production rule: $\forall x$, if x is a bird then x has wings. Both approaches represent the same information; the difference is a matter of interpretation as to how the knowledge is used by the system.

Although these systems use productions to represent long-term knowledge, they all use declarative knowledge to represent short-term memory. This reflects the notion that knowledge does not become proceduralized unless it is integrated into long-term memory. This approach also treats short-term memory as a temporary blackboard that only contains knowledge in the form of facts and relations involved in the current problem. However, this declarative knowledge can be implemented in quite different ways. For example, SOAR stores a stack of contexts that represent the subgoals being worked on in solving a top-level problem, whereas PRODIGY stores a tree that records all the actions that the system has taken so far. The latter approach is similar in some ways to EUREKA's traces, but in PRODIGY this is only a temporary structure in short-term memory. None of this tree is ever stored directly in long-term memory.

An alternative approach to representing long-term memory views knowledge as primarily declarative in nature. This type of representation can be found in frame-based or case-based planners, such as SCRAPS (Hendler, 1986), DAYDREAMER (Mueller, 1987), CHEF (Hammond, 1986/1988), and Carbonell's derivational and transformational analogy methods (Carbonell, 1983, 1986; Carbonell & Veloso, 1988). In these approaches, knowledge about the world is stored mostly or entirely in the form of facts and (specific or abstract) cases that have been seen or used to solve problems in the past.

One major difference between this declarative approach and the procedural approach is that, in production systems, each piece of knowledge is independent of the others. This allows greater flexibility in adding and deleting knowledge, and allows possibilities for interesting interactions between pieces of knowledge. The declarative approach relies on the notion that pieces of knowledge that are relevant to each other should be somehow connected together. For example, a declarative frame or plan for cooking a specific dish might contain an ordered list of tasks that must to be carried out to satisfy the problem. Taken to an extreme, as in case-based reasoning, each plan is simply an exact memory of

a case that has been seen before, rather than a set of abstract rules and goals that must be satisfied.

Although these systems rely heavily on the declarative representation of long-term knowledge, they rarely omit the use of procedural knowledge entirely. For example, SCRAPS contains rules that propose plans to use given certain goals, and DAYDREAMER includes planning and inference rules to fall back on when it cannot find any appropriate declarative cases in memory. Even a pure case-based reasoner such as CHEF contains rules for modifying past cases and enabling the use of new objects in new cases, although these rules can be viewed as part of CHEF's architecture rather than as part of memory. Once again, all of these systems use a declarative approach to represent working memory.

Long-term knowledge in EUREKA is also represented declaratively. In fact, the current model does not include *any* explicitly procedural knowledge. That is, even knowledge that might eventually be used in a procedural manner is stored declaratively, in the context of where and how it is used, then interpreted when necessary. In addition, EUREKA does not have a separate working memory. Rather, the working memory at any given time is always a subset of the long-term declarative memory traces. ACT* approaches this idea, having its short-term memory partly consisting of information from the declarative long-term memory. But ACT*'s short-term memory also holds temporary structures that disappear as soon as the problem has been solved. In summary, EUREKA views all knowledge as declarative in nature. However, portions of this memory can be retrieved into working memory and interpreted to produce actions. Finally, retrieval of useful declarative knowledge is facilitated when it appears in a problem-solving context and has proven useful in the past.

Basic approach to problem solving

The different approaches to modeling basic problem-solving processes are closely tied to the representation of knowledge in long-term memory, although there is a bit more variation in problem-solving approach. For example, many of the systems that use a production-based long-term memory also rely on a strict production-system or problem-space model of reasoning processes. These include SOAR, UPL, and ACT*. In these models, reasoning always takes place in a match and execute cycle. All productions in memory with satisfied conditions become available for firing, and some subset of these are executed. This cycle continues until no productions can fire or the goals of the system are satisfied.

Production systems have proven to be reasonably flexible and they are not generally restricted to specific approaches to problem solving. For example, standard production matching results in a forward-chaining style of problem solving, but adding conditions to productions can lead to heuristic search. In addition, adding conditions concerning the

system's current goals can be used to implement a backward-chaining style of search or means-ends analysis. In these cases, the production-system models are used as interpreters to implement higher-level problem-solving approaches.

SOAR augments the standard production-system model with the capability of universal subgoaling. Under this view, any decision that the system must make is a new problem that needs to be solved. For example, when SOAR must decide which operator to apply to a situation, it sets up a new problem for itself to determine the best solution, rather than relying on a built-in decision procedure. Of course, the decision procedure must be implicit in the productions for solving the problem, but they are not directly implemented in the system architecture.

Holland et al.'s PI also augments the straightforward production-system approach with the ability to apply rules in two directions. Productions with matched conditions are fired normally. However, productions with actions that match current goals are set up with APPLY subgoals in means-ends fashion.

Other production-system models attempt to improve efficiency by limiting the conditions under which rules can fire. For example, PRODIGY is limited to working in a means-ends paradigm. That is, candidate rules must potentially reduce some of the differences between the current working-memory state and the current goal conditions. In addition, if a selected rule cannot fire in the current state, a subgoal is automatically generated to transform the current state into a state that satisfies the preconditions of the operator.

There are also a number of possible approaches to problem solving given a declarative long-term memory. A frame-based planner, such as SCRAPS, stores plans that group together relevant rules to achieve a goal, rather than relying on sets of completely independent rules. As we will discuss later, this can be viewed as a declarative approach to chunking, which can also be implemented in a production-system framework.

Case-based reasoners, on the other hand, rely on a very different approach to problem solving. In the production-system model, problem solving arises from selecting appropriate rules to apply in given situations. Case-based reasoning relies on finding cases in long-term memory that are similar to the current problem, and then transforming them (as in CHEF, DAYDREAMER, or transformational analogy) or using them to guide future problem-solving efforts (as in derivational analogy). For example, in CHEF this means that very little search takes place in trying to solve the problem directly. There is search involved in finding an appropriate case to use to solve the new problem, and there is search involved in transforming the old plan into a solution to the new problem. However, there is no clear analog to the search at the problem level that occurs in production systems. One criticism of case-based reasoning is that it provides no explanation for the origin of the initial set of cases, but DAYDREAMER is a notable exception in this regard. When it cannot find a

suitable case to use for a new problem, it falls back on a problem-space approach, using its production rules in long-term memory.

EUREKA uses an approach that is in some sense a hybrid between case-based reasoning and means-ends analysis. It attempts to limit search by using past cases to help solve new problems. However, the system does not rely on finding a full-blown problem-solving episode in memory and using it exclusively to guide behavior on a new problem. Rather, the system retrieves pieces of problems individually when they are appropriate. The system therefore gains the advantage of using past cases while allowing some of the flexibility that production systems get from applying independent pieces of knowledge to a problem. In this way, EUREKA can borrow knowledge from a number of different cases in solving a problem.

Also, EUREKA starts with individual operators stored as miniature trivial problems in memory, so it can fall back on a problem-space approach when it cannot find any cases appropriate to solving the current task. In this case, the system prefers matches in the goal structures of the current TRANSFORM goal and the retrieved goals. This encourages the selection of operators that reduce differences, as in a standard MEA framework. However, as we have noted, the system is not limited to considering those operators and can fall back on a more liberal operator-selection strategy like forward chaining if necessary. Thus, EUREKA provides a compromise between the strict problem-space approach and the case-based approach. The system retrieves pieces of cases in a problem-solving context, if possible, and individual operators when necessary.

Retrieval and selection

In our work, we have attempted to divide the decision-making process into two distinct phases. First, a small amount of information must be recalled from memory. Next, this information goes under more extensive analysis to decide which is most useful for the current problem. We refer to these two stages as *retrieval* and *selection*, respectively. These are stages that all problem-solving models must address, at least implicitly. The selection stage is sometimes referred to as *conflict resolution* (Forgy & McDermott, 1977), and has received quite a bit of attention in research on problem solving. However, many models have ignored the issue of retrieval by examining all long-term when they must make a decision. This has especially been the case in production-system models. Recently, however, some research with production systems has addressed the retrieval issue.

The idea here is that, since long-term knowledge is represented in the form of rules, there can be a special set of rules that propose which of the other rules are available for firing in any given situation. Ohlsson's (1987) UPL was the first system to make these retrieval rules explicit, so we will borrow his terminology in explaining this approach. In UPL, SOAR, and PRODIGY, rules are retrieved by the use of *proposers*, which suggest rules

that might be applicable to the current situation. This corresponds to retrieving a subset of the rules in long-term memory for consideration. Often the default proposers (those existing before any learning has taken place) retrieve all rules in memory as candidates, so that the system will not miss any potentially applicable rules. Frame-based systems such as SCRAPS also use this retrieval approach, although the proposer rules usually suggest entire plans to use in solving a problem, rather than individual rules to apply.

After an initial set of rules has been retrieved, a subset of these (often just one rule) must be selected for firing. In UPL, this rule set is pruned through the use of *censors*, which indicate which of the retrieved rules would be bad to apply in the current situation. SOAR, PRODIGY, and SCRAPS also have rules of this type, but SOAR and PRODIGY also include *preferences*, which order the remaining candidates in terms of their potential usefulness. After this process, there may be a single, unique rule to fire and the selection process is over. On the other hand, there may still not be a clear winner.

At this point, the various models go their separate ways in making a selection. PRODIGY and SCRAPS make a random selection from the remaining rules, although PRODIGY remembers the rules it did not choose and has the ability to backtrack in case the random choice fails. UPL uses more standard production-system conflict-resolution techniques for ordering the rules. These include preferring rules that have not fired before, preferring rules that are more specific, and choosing at random as a last resort. As mentioned previously, SOAR creates a subgoal in an alternative problem space to resolve the decision, and searches in this problem space until a clear choice is determined.

Case-based reasoners, such as DAYDREAMER, CHEF, and Carbonell's approaches, do not use productions to retrieve and select cases. Rather, they use an approach that has some ideas in common with spreading activation. The concepts mentioned in the current goal conditions and working state are used as *indices* to find previous cases from which to reason. In this way, only those cases that have abstract concepts in common with the current problem will be retrieved. Selection is then made based on the number of indices that the retrieved cases have in common with the current problem. DAYDREAMER goes a bit further, in that it also uses indices from concepts reflecting the system's current "emotions" to help retrieve cases and determine relevant goals. Also, unlike other case-based reasoners, DAYDREAMER falls back on production rules when it cannot find any cases to reason from. In this situation, a more traditional production-system style of selection is made with all the rules in memory as candidates, so there is no special retrieval mechanism for these productions.

ACT* is a production-system model that uses a quite different approach to the retrieval of operators. When this system is presented with a problem, some of the features are set up as sources of activation in the declarative long-term memory, thereby calling these concepts into working memory. Then a spreading-activation process similar to that used in EUREKA is run to retrieve more concepts from the declarative long-term memory into working

memory. Finally, the entire rule set in the procedural long-term memory is matched in parallel against the working memory elements. In this way, ACT* only retrieves those rules that involve active concepts. Then conflict resolution is carried out using standard production-system methods, such as preferring more specific rules and rules that have not fired before. Other factors that ACT* incorporates into selection include the degree of match in the rule conditions (rules that do not match completely are sometimes allowed to fire), the strength of the productions (productions that have proven useful in the past are preferred), and preferring productions that explicitly test the current goals of the system. This also encourages the system to favor a goal-directed behavior like means-ends analysis, while allowing it to fall back on forward chaining if necessary.

PI uses a more restricted form of spreading activation for retrieval. Whenever this system fires a rule, the concepts mentioned in its actions are called into working memory. Likewise, when a subgoal is created for a rule that cannot yet fire, the concepts mentioned in the subgoal are called into working memory. However, these concepts are not set up as sources of activation to initiate a spreading-activation process to neighboring concepts, as in ACT* or EUREKA. Rather, only these concepts are activated, and their activation decays with time. As in ACT*, rules are only allowed to match against working memory elements. In this manner, activation does not spread automatically throughout memory, but proceeds in one step each time a rule is fired. Finally, selection is made using types of evaluation similar to those used in ACT*.

EUREKA uses a spreading-activation mechanism similar to that used in ACT*, but it does not have a separate procedural memory to match against working-memory elements. Rather, all knowledge, including operator representations, is stored declaratively. Instead of retrieving a set of productions or operators, EUREKA retrieves a set of declarative TRANSFORM goals, which are connected to structures representing the operators that were used in those situations. The system then selects one TRANSFORM goal to use as a model according to the degree of match between the retrieved goals and the current goal. Selection is also influenced by a factor that indicates how useful each goal has proven in solving the current problem in the past. Finally, an operator that was used to solve the previous problem is analogically transformed in an attempt to apply it to the current problem. In this way, EUREKA makes an explicit distinction between the retrieval and selection stages. In addition, the model differs from both the problem-space and case-based approaches by retrieving a small set of partial cases, rather than a complete set of productions or a single, full problem-solving episode.

Learning

All the problem solvers we have discussed in this chapter attempt to learn from experience in some way. The approaches taken vary greatly in terms of the number of learning

mechanisms involved and the types of learning behaviors that they exhibit. In all of these approaches, past knowledge is used to influence the retrieval and/or the selection of information used in making decisions. For example, the systems that rely on productions to retrieve and select knowledge learn by forming new proposers, censors, and preferences. SOAR, PRODIGY, and UPL learn new proposers when they solve a problem. UPL simply follows the chain of states that were visited and creates a new proposer for each state. The idea is that if a particular rule was successful in a state before, it should be proposed the next time the system is in that state. These proposers are generalized to allow the rule to apply in situations that have not been seen before but are very similar to the successful problem. UPL forms censors when it is able to identify an action from a particular state that led to failure.

SOAR uses a "chunking" approach to create new proposers and preferences. When the system cannot make a clear decision given its current proposers, censors, and preferences, it sets up a subgoal to treat the decision as a new problem. After working on this problem, a clear result should be available. SOAR then creates a chunk describing its working-memory state before the subgoal was created and after it was solved. This chunk is used to create new rules of the form, "if the current situation is similar to the situation that created this chunk, prefer the solution that was found this time." This allows the system to make a clear decision in similar situations in the future without resorting to search in an alternative problem space.

PRODIGY uses an explanation-based approach to create retrieval and selection rules. This system includes separate domain theories that are used to explain various types of problem-solving episodes. For example, PRODIGY can use one set of rules to explain why a given problem was solvable. This explanation results in a general proof, which is stored in the form of proposers that suggest the use of the same operators in similar situations in the future. PRODIGY also has the ability to form censors by explaining goal failures and to form preference rules by explaining goal interactions when it detects them.

Case-based reasoners do not store special rules to retrieve and select long-term knowledge in appropriate situations. Rather, since they reason from specific cases stored in memory, they must retrieve the most appropriate case from memory to use as a model when presented with a problem. The important issue here involves how cases are stored and indexed in memory. Case-based reasoners usually index their cases by the concepts mentioned in their goals and initial conditions. Usually, the goals are given precedence. In DAYDREAMER, cases are also indexed by various emotional states that the system can be in. Again, these systems do not have to concern themselves with making specific pieces of procedural knowledge available at the appropriate times. Rather, they attempt to store each specific problem that they encounter so that it will be retrieved at the most appropriate time for future problems. The systems described here do not address the

issues of adapting these indices when a retrieved case does not turn out to be useful in solving a new problem.

PI and ACT* rely on the *tuning* of productions rather than learning specific rules to guide search. One type of production tuning involves adding, deleting, and generalizing the conditions under which productions can apply. This is similar to the approach of creating proposers and censors to retrieve and select productions. The difference is that the conditions are built into copies of existing productions rather than being stored as a separate rule set. A second type of tuning involves strengthening productions when they prove useful in solving problems. This leads to a "natural selection" view of learning, in which productions undergo mutations by discrimination and generalization, and then the best ones (those that prove useful often) become the strongest and are selected more often when appropriate in the future.[16]

SCRAPS also modifies its behavior by creating new proposers and censors, but it does not create them in response to past problem-solving behavior. In fact, Hendler (1986) does not provide any examples of transfer from one problem-solving trial to another, although it seems possible within his framework. Rather, new censors and proposers are created at the beginning of a problem-solving episode when certain aspects of the environment are brought to the attention of the system. For example, if SCRAPS is told to plan a suicide and to notice that it has a gun available, it finds a connection within its semantic network between suicide and gun and creates a proposer to prefer the use of a suicide plan that involves a gun. In addition, if the system is told to satisfy two interacting goals that involve buying a gun and taking a plane home, it will notice a connection involving the danger of having a gun on an airplane. Thus, SCRAPS will create a censor to keep the system from buying the gun before taking the plane ride. This type of learning is most similar to the type that EUREKA exhibits involving external cues. SCRAPS forms rules that influence the retrieval and selection of plans, based on aspects of the problem or cues in the environment that are potentially useful or detrimental to problem solving.

As we discussed in chapter 4, EUREKA learns primarily by altering its patterns of retrieval of information from long-term memory. The selection of information is also influenced, but it plays a less important part, particularly in transfer across different problems and domains. EUREKA learns by rewarding the retrieval of knowledge that proves useful in solving a problem. This is similar to the creation of proposers to retrieve information that has proven useful in the past, but it is implemented in the context of a retrieval mechanism based on spreading activation, rather than a production system. The use of spreading activation better suits the declarative representation of knowledge, and complements the use of a partial-matching mechanism for selection.

[16] This view is also adopted in Langley's (1985) SAGE system.

In chapter 5, we discussed one type of improvement that EUREKA exhibits under the heading of intra-domain transfer. This type of transfer can also be viewed as the learning of search-control knowledge. This type of learning has been the major focus of learning problem solvers in AI, and usually results from directly applying knowledge of past successes and failures to new problems without the type of inductive generalization that is found in analogy. For example, this type of learning is exhibited by systems that form search-control rules for retrieval and selection of information from long-term memory (e.g., Laird et al., 1986b; Minton, 1988/1989; Ohlsson, 1987). These systems all learn by using pieces of previously solved problems to eliminate search when those pieces are encountered in new problems. This type of learning was originally exploited by systems that use macro-operators (Fikes et al., 1972; Korf, 1985; Iba, 1985, 1989) and it seems to provide a reasonable mechanism for transfer within a domain. SOAR and UPL also use search-control learning to account for the gradual improvement of problem solvers with practice on individual problems.[17]

The other primary type of learning we have discussed has to do with the generalization of knowledge. This type of learning can also manifest itself in the form of restricted search, but the mechanisms and knowledge involved are quite different. When a system has the ability to generalize its knowledge, it also has the ability to overgeneralize and make mistakes. Systems with built-in generalization mechanisms (e.g., PI and ACT*) have this ability. Other systems (e.g., SOAR and UPL) perform some generalization when creating new search-control rules, but have not been extensively analyzed in terms of the effects of generalization. Finally, systems like PRODIGY forego the ability to generalize knowledge for the sake of deductive correctness. Because PRODIGY cannot generalize its knowledge (its control rules can only be as general as its original operators were, it cannot over-generalize and will not make errors of commission on new problems. This is an advantage in that it allows the system to ignore certain types of failures that can occur in systems that generalize. On the other hand, this approach also restricts the problems that the system can potentially solve.

The use of analogy

One of the important contributions of the EUREKA model is that it provides an explanation for the retrieval and use of analogies in problem solving.[18] In the previous section,

[17] SOAR exhibits this phenomenon with a restricted form of its general chunking mechanism, in which only some of the possible chunks are created after a single trial (Newell & Rosenbloom, 1981). Otherwise, SOAR (like PRODIGY) will always solve a problem the same way after its first success, without exhibiting any further improvement.

[18] In this section, we focus on the retrieval and use of analogies in the context of problem solving. However, there is a wealth of literature in psychology and AI that addresses various other issues in analogical reasoning (see Hall, 1989). In particular, Gentner (1983), Holyoak and Thagard (1988), and Falkenhainer (1989) have described theories for elaborating analogies once they have been retrieved.

we discussed a number of mechanisms that accounted for various types of learning. Much of this learning appears in the form of search control. That is, the learning is exhibited completely in terms of reducing search on new problems that share pieces with problems that have been seen before. However, an important aspect of human problem solving, and a useful mechanism for machine problem solving, involves the ability to generalize ones knowledge broadly across domains when appropriate. This ability can also limit search and suggest strategies for solving new problems.

The branch of research in problem solving that has best exploited the use of analogy in problem solving is case-based reasoning. In one view, case-based reasoning is almost synonymous with analogical reasoning. Individual cases describing entire problem-solving episodes must be adapted to appropriate levels of generality to solve new problems.

The problems that arise in reasoning by analogy include deciding when to attempt to use analogical reasoning rather than straightforward problem-solving, how to retrieve potential analogies, and how to best apply those analogies. In Hammond's CHEF and in Carbonell's framework for transformational analogy, the first problem is ignored. They view all problem solving as the use of analogy to previous cases. This is similar to the assumption made in EUREKA, but in this work it depends on the existence of full cases that can be used individually and completely to solve new problems through transformation or derivational replay. DAYDREAMER and derivational analogy use case-based or analogical reasoning as their primary problem-solving strategy, but they have the ability to fall back on production rules when they cannot find an appropriate case.

As discussed previously, retrieval of analogies in case-based reasoning involves following indices from the current goal conditions and working-memory descriptions into long-term memory. This leads to search through a portion of memory and partial matching between the problem statements in order to choose an appropriate analogy. An important issue here is that these systems usually rely on the use of a single case to solve a new problem. This means that retrieval and analogical matching only takes place with respect to the initial problem statement. If a problem initially appears to provide a good analogy, but fails to be useful as problem solving continues, the problem solver runs into trouble. Again, DAYDREAMER and derivational analogy fall back on problem-space search to complete the new problem if this happens, while CHEF and transformational analogy rely on massive transformations of the old case until it finally provides a solution to the new problem.

EUREKA also relies on indices into long-term memory to find old cases to use by analogy. One difference is that these indices start a spreading-activation process to search for appropriate analogies rather than providing direct links into a set of cases. A major difference between EUREKA and the case-based approach is that EUREKA does not attempt to force a single past case into a solution of a new problem. Rather, for each subproblem, EUREKA looks for a case that is likely to help solve that individual piece. This means that the system must repeat the retrieval-selection cycle for each subproblem, but it also

allows the system to change analogies in mid-problem if necessary and to fall back on the use of individual operators (stored as miniature cases) if no appropriate operators can be found in the context of a larger problem-solving episode. EUREKA can therefore ignore the decision of when to do straightforward problem solving and when to do analogical reasoning, because they both arise from the same mechanism.

Other problem-solving systems have been less successful at modeling the use of analogy in problem solving. In EUREKA, the use of analogy is a basic architectural assumption as the primary reasoning process. In other systems, the use of analogy must be provided as a distinct mechanism that includes decisions about when to switch into analogical-reasoning mode and how to pick a case from memory to use as a model or source. Consider PI and ACT*, which have the ability to generalize individual operators. This could allow the type of extreme generalization that occurs in analogy, but these systems usually make generalizations conservatively, discouraging the formation of broadly general rules even when they might be appropriate. Therefore, in order to reason analogically, productions that carry out the analogical-reasoning process step by step in a problem-solving manner must be added to long-term memory.

A similar situation occurs with SOAR. It could be provided with the ability to reason by analogy, but it would not arise directly from chunking, its sole learning mechanism. Rather, the system would need productions for detecting and deciding when it would be appropriate to go into an "analogical-reasoning problem space", picking appropriate analogs, and creating the mapping. In this view, analogy is interpreted as a high-level reasoning task rather than as an implicit architectural mechanism, as it is in EUREKA and the case-based work.

Focus of attention

A final aspect of human problem solving, which almost all of the past research (including EUREKA) has ignored, involves the ability to focus attention. Most systems are given a single problem to work on at a time and have precise algorithms for determining which pieces of the problem they should attend to and in what order. This excludes the possibility of switching attention to different aspects of a problem, or switching from one problem to a completely different problem when appropriate. Some systems have hinted at such a capability. For example, SOAR and PRODIGY both have the potential for search-control rules that determine which problem to work on next, but they have been mentioned merely in passing, with no suggestions about how they might be used to make decisions.

The DAYDREAMER model is a notable exception to the lack of research in this area. Mueller's system attempts to model the processes of human daydreaming when it is not busy trying to solve a problem it has been given. During daydreaming, the system decides which problems and pieces of problems it is going to "dream" about, and is even able to

set problems up for itself. In DAYDREAMER, focus of attention is primarily influenced by the system's current emotional state. For example, if the system fails to achieve a given goal, it might change its emotional state to one of anger. This causes it to generate a "revenge" goal and to attend to problems that cause this fanciful revenge situation to come about. These plans are only "dreamed," so they are not actually executed unless the opportunity presents itself. The system also has the capability of noticing connections to seemingly irrelevant environmental cues through a marker-passing algorithm similar to that used by SCRAPS. Given some input cues, the algorithm returns a set of paths, which are then analyzed to see if they can help to achieve certain goals. This provides the system with the ability to account for some types of insight.

The ability to focus attention is particularly important in dealing with the ability to react to one's enironment. We have shown that external cues can alter EUREKA's behavior within a problem-solving episode, but cues could also serve to transfer the system's attention to different portions of a problem or to completely different problems altogether, as in cases of cue-driven insight. We will discuss the possibility of such behavior in EUREKA in the final chapter.

CHAPTER 7

Discussion and Future Research

The EUREKA model provides contributions to research on problem solving along a number of distinct dimensions. In this final chapter we examine the model along each of these dimensions, discussing the contributions this work has made and outlining directions for future research.

Psychological evaluation

Our primary means of testing the EUREKA model was to see if it exhibits characteristics similar to those found in human problem solving. We did this in an effort to show that the system can account for a portion of human behavior and can exhibit some of the same types of flexibility and creativity as humans.

The types of phenomena we have used in testing our model include the ability to improve performance on individual problems with practice, the ability to transfer knowledge between problems within a domain, and the ability to generalize knowledge across domains, as in the use of analogy. We have also demonstrated that the system may suffer from Einstellung, a type of negative transfer that is exhibited by humans, and that its problem-solving abilities are influenced by the existence of external cues in the environment.

The dependent measures we used to measure performance in many of these experiments involved the time and effort the system required to solve the given problems. These were measured in terms of the number of attempts the system needed to solve each problem, the time it spent searching the problem space, and the amount of productive work the system did by exploring new areas of the problem space. EUREKA combines a retrieval mechanism based on spreading activation with a problem solver, with the latter including an analogical-matching mechanism and a simple learning method that involves updating link trace strengths in memory. Our experimental results show that this combination can account for many of the types of learning that humans exhibit. Although there has been other work in AI that accounts for some of these phenomena, they have each typically addressed a smaller set of behaviors. In particular, none have smoothly integrated analogical and inductive problem solving with more straightforward reasoning techniques.

In testing the system, we have relied on accounting for past data from psychological experiments. However, we could further evaluate the theory by running psychological

experiments to test the predictions it makes about human behavior. Success along these lines would weaken the argument that EUREKA had been built to match a small specific set of phenomena. Some of these experiments would attempt to account for the predictions EUREKA makes about intra-domain generalization and the effects of goal interactions. Other experiments would examine the ability to retrieve and use analogies, and the effects of environmental cues on performance. We will discuss our ideas for the latter experiments in more detail later in this chapter.

Problem solving

In addition to providing a theory of human problem solving, EUREKA can be viewed and described strictly as a computational problem-solving system. Currently, the system's major purpose is to provide a psychological model and it is far from being a complete and powerful problem solver. However, it does suggest a number of techniques and mechanisms that should prove useful in computational problem solving.

First, EUREKA views problem solving as a task involving the retrieval of useful information from memory. This is similar to the view provided in case-based reasoning. Under this view, problem solving is less involved in examining productions or operators and evaluating their utility for the current problem, and more concerned with retrieving past experiences that will suggest useful approaches to apply to the current problem. However, EUREKA borrows techniques from both paradigms without relying completely on either one. In this way, the system exploits the benefits of case-based reasoning, while allowing problem-space search to take place when necessary.

Another contribution of EUREKA is that it suggests a method for the efficient retrieval of information from a large database. Many of the most powerful contemporary problem solvers (e.g., SOAR and PRODIGY) rely on the ability to perform an exhaustive search of memory, if necessary. This approach provides these systems with the ability to solve wide ranges of problems of non-trivial complexity. However, these systems should suffer when presented with problems involving large amounts of domain knowledge, or when provided with general knowledge from large numbers of problem domains in which most of the knowledge in memory is irrelevant to each particular problem. Minton (1988/1989) has called one facet of this issue the "utility problem" for explanation-based learning.

EUREKA's spreading-activation mechanism provides the ability to focus on small portions of memory and provides a decision-making mechanism that does not slow down as the size of memory increases. Thus, the system has no utility problem, but there is naturally a tradeoff involved. Since EUREKA has an implicit limit on the amount of memory it will examine, there will be cases when the system cannot solve a problem even though it has the appropriate knowledge stored in memory. However, we predict that this type of mechanism will have strong heuristic value, providing a solution most of the time.

In addition, EUREKA has a single mechanism for making decisions about which course of action to take next. Depending on the knowledge in memory and the current problem, this mechanism manifests itself in seemingly different types of problem-solving behavior. These include straight-forward operator application (or deductive reasoning), the ability to generalize operators within a domain, and the ability to draw broad analogies across domains (inductive reasoning). As suggested previously, this is a desirable characteristic because the system does not have to make any high-level decisions about which type of performance mode it should use on each problem. Rather, the most appropriate method arises from the general mechanism, based on the system's current knowledge base and the demands of the current problem.

As we have suggested, EUREKA is somewhat weak as a computational problem solver, but it contains a number of mechanisms that should be useful in the context of more powerful problem solvers. In the future, we plan to extend EUREKA to take advantage of this potential. For example, one important factor in EUREKA's weakness is its lack of higher-level control knowledge of the type found in UPL, SOAR, or PRODIGY. In addition, we built in the assumption that the system could not backtrack for the sake of psychological validity. However, we expect that supplying EUREKA with a limited backtracking ability, along with the ability to learn higher-level control knowledge that operates on the retrieved knowledge, will greatly increase the complexity of the problems that it can solve.

Analogy

EUREKA also provides a context for the retrieval and use of analogies in problem solving. Although the use of analogy has received a large amount of attention (see Hall, 1989), it has yet to be incorporated in a problem solver in an elegant and general way. In addition, most research has focussed on how to elaborate analogies once they have been suggested, (e.g., Carbonell, 1983, 1986; Falkenhainer, Forbus, & Genter, 1986; Holyoak & Thagard, 1988) and not on how to retrieve them in the first place. EUREKA retrieves analogies using the same memory mechanism that it uses to retrieve other knowledge from memory. The system also provides a mechanism for decision making in problem solving that includes analogy as one activity in a continuum of possible problem-solving behaviors, allowing analogies to arise naturally when they are useful. The system does not need to switch from straightforward problem-solving mode into analogy mode, as has been the case in other work on analogical problem solving (e.g., Anderson, 1983; Holland et al., 1986; Mueller, 1987).

One extension of this ability would involve the use of alternative analogical-mapping mechanisms. EUREKA's matcher is a relatively simple one that generates a number of partial matches and evaluates them. The evaluation function involves the degree of match between two structures and the number of assumptions required to achieve the match.

As we have mentioned, the elaboration of analogies is a well-studied problem, and we might expect EUREKA's performance to improve if equipped with a smarter analogical transformation mechanism, such as the structure-mapping engine (Falkenhainer, 1989; Falkenhainer et al., 1986). Since this component of the system is independent of the other components, it should be easy to replace it with alternative mechanisms.

We have also mentioned our desire to test some of EUREKA's predictions concerning the use of analogy by humans. Specifically, our model predicts that the use of analogy in solving problems will be facilitated by analogies involving uncommon concepts,[19] and by analogies that have proved useful in the past. We hope to test these predictions by studying human subjects and comparing their behavior with that produced by EUREKA.

Another area for future work concerns the development of the single matching mechanism that manifests itself in terms of different types of problem solving. We believe that using a general analogical method as the sole reasoning method can provide further benefits in problem solving and other parts of AI. For example, using this approach should be useful in concept induction tasks, in which similar objects form natural classes. In addition, in the areas of reasoning and explanation-based learning, a single analogical method should be able to account for the three primary methods of reasoning: deduction, induction, and abduction.[20] We want to explore the benefits that can be realized by viewing various forms of reasoning as special cases of analogy.

External cues and insight

A final area of interest involves EUREKA's account of the impact of external cues. We have demonstrated the effect cues can have on the retrieval and use of analogies. We also believe that the effects of cues on the retrieval mechanism can provide an explanation for episodes of scientific insight. In this account, cues cause the retrieval of information that leads to the solution of a difficult problem. In fact, we have successfully modeled one type of this behavior, which we call *problem-driven* insight, but other forms of insight remain to be considered.

Elsewhere (Langley & Jones, 1988; Jones & Langley, 1988), we have proposed a theory of scientific insight in terms of environmental cues,[21] but we will provide a short description of the theory here. First, we should note that there are two distinct types of phenomena in which sudden flashes of insight appear to aid problem solving. The two phenomena

[19] By "uncommon," we do not mean concepts that are unfamiliar to the system. Rather, we mean concepts that appear in relatively few problems, and therefore are directly connected to fewer other concepts in the semantic network.

[20] In fact, this type of approach has been suggested independently by Falkenhainer (1989).

[21] Yaniv and Meyer (1987) have proposed a similar psychological model for the role of memory and cues in episodes of insight.

arise from very different types of processing, although the heart of each type of insight can be explained in terms of a single retrieval mechanism.

A simple form of insight, which we call *problem-driven insight*, occurs when a problem solver has difficulty solving a problem and cannot find a solution, but continues work on the problem. Under these conditions, an insight may occur with the appearance of an appropriate external cue that causes the retrieval of useful information. The external cue behaves much as a hint does, causing the solver to retrieve some knowledge that was previously inaccessible. In fact, this type of insight may be involved in Holyoak and Koh's (1987) experiment, in which an external hint facilitates the retrieval of an appropriate analogy. This suggests that EUREKA's account of the role of hints in problem solving can provide the basis for a model of problem-driven insight.

A more involved type of insight, *cue-driven insight*, is insight in the classic sense as described by Wallas (1931). In these cases, insight is a four-stage process. The first stage involves *preparation*, or the careful study of a problem and repeated attempts to solve it. The second stage, *incubation*, begins when the problem solver becomes frustrated at being unable to solve the problem and ceases work on it altogether. After a period of time (anywhere from a few seconds to a few years), an *illumination* occurs, in which the problem solver realizes in a flash that he can solve the problem a certain way. The final stage, *verification*, involves working out the details of the insight and finishing the solution of the problem.

In this type of insight, illumination occurs only after the problem solver has given up work on the difficult problem. Again, our theory states that this insight occurs when some environmental cue causes the solver to retrieve information that can be used to solve the problem. The retrieved information may be directly relevant knowledge that was just not remembered during the problem-solving activity, or it may be information that can be used by analogy to help solve the problem.

However, in cue-driven insight, the problem solver is not currently paying attention to the problem when illumination occurs. This cannot be explained simply in terms of the retrieval of useful information, because the information is only useful in the context of a problem that is no longer at the focus of attention. Therefore, any cue that causes insight in this case must also cause the problem solver to refocus attention on the unsolved problem. We hypothesize that either the original cue or the retrieved information reminds the solver of the unsolved problem. This causes the solver to begin work on the problem again with the new information at hand. We would like the EUREKA model to eventually account for this type of insight as well, but to do this we must explain how focus of attention switches back to the original problem

We feel that a problem solver's focus of attention can be explained in terms of memory and retrieval. At this point, attention is focussed on those concepts that are most active in memory, but choosing an explicit focus merely involves adding another decision point to

the model. Naturally, we believe that the memory and retrieval mechanisms for attention will be similar to those involved in standard problem solving. The problem that remains is to implement a principled decision procedure that determines the focus according to the activation of memory elements. This is a major step in our effort to construct a general model of problem solving that includes distraction, attention, and both types of insight.

A full model of insight will provide a number of predictions about this phenomenon in humans. For example, the EUREKA model predicts that the incubation period does not actually influence an episode of insight unless some external event during that time alters retrieval patterns in memory. This can be manifested by external cues or by learning that is not obviously relevant to the current problem. Both of these hypotheses can be tested on humans to see if they manifest themselves in the manner that EUREKA predicts.

Concluding remarks

Our experiences in constructing and evaluating the EUREKA model have been encouraging. We have found that we can explain a number of human learning behaviors by incorporating a theory of memory with a problem solver based on means-ends analysis. In addition, EUREKA provides a number of predictions about human behavior that have yet to be tested. Finally, the model addresses a number of issues in computational problem solving and suggests methods for improving systems in that area. These issues include providing a mechanism for the retrieval of information from long-term memory and using analogy as a general reasoning framework. Our research has also opened a number of interesting new questions. By examining these questions, we feel that EUREKA can eventually provide the basis for a complete theory of human problem solving, as well as an architecture for computational problem solving.

References

Ambros-Ingerson, J. A., & Steel, S. (1988). Integrating planning, execution, and monitoring. *Proceedings of the Seventh National Conference on Artificial Intelligence* (pp. 83–88). St. Paul, MN: Morgan Kaufmann.

Anderson, J. R. (1974). Retrieval of propositional information from long-term memory. *Cognitive Psychology, 5*, 451–474.

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.

Anzai, Y., & Simon, H. A. (1979) The theory of learning by doing. *Psychological Review, 86*, 124–140.

Carbonell, J. G. (1983). Learning by analogy: Formulating and generalizing plans from past experience. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Los Altos, CA: Morgan Kaufmann.

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* (vol. 2). Los Altos, CA: Morgan Kaufmann.

Carbonell, J., & Veloso, M. (1988). Integrating derivational analogy into a general problem solving architecture. In J. Kolodner (Ed.), *Proceedings of the Case-based Reasoning Workshop.* Clearwater Beach, FL: Morgan Kaufmann.

Charniak, E. (1983). Passing markers: A theory of contextual influence in language comprehension. *Cognitive Science, 7*, 171–190.

Charniak, E. (1986). A neat theory of marker passing. *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 584–588). Philadelphia, PA: Morgan Kaufmann.

Collins, A., & Quillian, M. R. (1969). Retrieval time from semantic memory. *Journal of Verbal Learning and Verbal Behavior, 8*, 240–248.

Dominowski, R. L., & Jenrick, R. (1972). Effects of hints and interpolated activity on solution of an insight problem. *Psychonomic Science, 26*, 335–338.

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

Duncker, K. (1945). On problem solving. *Psychological Monographs, 58*(270).

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

Falkenhainer, B. C. (1989). *Learning from physical analogies: A study in analogy and the explanation process.* Doctoral dissertation, University of Illinois at Urbana-Champaign.

Falkenhainer, B., Forbus, K. D., & Genter, D. (1986). The structure-mapping engine. *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 272–277). Philadelphia: Morgan Kaufmann.

Fikes, R. E., Hart, P., & Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence, 3*, 251–288.

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.

Forgy, C., & McDermott, J. (1977). OPS, a domain-independent production system language. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence* (pp. 933–939). Cambridge, MA: Morgan Kaufmann.

Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science, 7*, 155–170.

Gentner, D. (1988). Analogical inference and analogical access. In A. Prieditis (Ed.), *Analogical.* Los Altos, CA: Morgan Kaufmann.

Georgeff, M. P., & Lansky, A. L. (1987). Reactive reasoning and planning. *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 677–682). Seattle, WA: Morgan Kaufmann.

Gick, M. L., & Holyoak, K. J. (1980). Analogical pròblem solving. *Cognitive Psychology, 12*, 306–355.

Gick, M. L., & Holyoak, K. J. (1983). Schema induction and analogical transfer. *Cognitive Psychology, 15*, 1–38.

Granger, R. H., Eiselt, K. P., & Holbrook, J. K. (1986). Parsing with parallelism: A spreading-activation model of inference processing during text understanding. In J. L. Kolodner & C. K. Riesbeck (Eds.), *Experience, memory, and reasoning.* Hillsdale, NJ: Lawrence Erlbaum.

Hall, R. P. (1989). Computational approaches to analogical reasoning: A comparative analysis. *Artificial Intelligence, 39*, 39–120.

Hammond, K. J. (1988). Case-based planning: An integrated theory of planning, learning, and memory (Doctoral dissertation, Yale University, 1986). *Dissertation Abstracts International, 48*, 3025B.

Hayes, J. R., & Simon, H. A. (1977). Psychological differences among problem isomorphs. In N. J. Castellan, Jr., D. B. Pisoni, & G. R. Potts (Eds.), *Cognitive theory*. Hillsdale, NJ: Lawrence Erlbaum.

Hendler, J. (1986). Integrating marker-passing and problem-solving: A spreading-activation approach to improved choice in planning (Doctoral dissertation, Brown University). *Dissertation Abstracts International, 47*, 2059B.

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

Holyoak, K. J., & Koh, K. (1987). Surface and structural similarity in analogical transfer. *Memory and Cognition, 15*, 332–340.

Holyoak, K. J.,& Thagard, P. (1988). *Analogical mapping by constraint satisfaction.* Manuscript submitted for publication.

Iba, G. A. (1985). Learning by discovering macros in puzzle solving. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 640–642). Los Angeles: Morgan Kaufmann.

Iba, G. A. (1989). A heuristic approach to the discovery of macro-operators. *Machine Learning, 3*, 285–317.

Jones, R., & Langley, P. (1988). A theory of scientific problem solving. *Proceedings of the Tenth Annual Conference of the Cognitive Science Society* (pp. 244–250). Montreal: Lawrence Erlbaum.

Kibler, D., & Langley, P. (1988). Machine learning as an experimental science. In D. Sleeman (Ed.), *Proceedings of the Third European Working Session on Learning* (pp. 81–92). Glasgow, Scotland: Pitman.

Korf, R. E. (1985). Macro-operators: A weak method for learning. *Artificial Intelligence, 26*, 35–77.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986a). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning, 1*, 11–46.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986b). *Universal subgoaling and chunking: The automatic generation and learning of goal hierarchies*. Hingham, MA: Kluwer Academic.

Langley, P. (1985). Learning to search: From weak methods to domain-specific heuristics. *Cognitive Science, 9*, 217–260.

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

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

Meyer, D. E., & Schvaneveldt, R. W. (1971). Facilitation in recognizing pairs of words: Evidence of a dependence between retrieval operations. *Journal of Experimental Psychology, 90*, 227–234.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review, 63*, 81–97.

Minton, S. (1989). Learning effective search control knowledge: An explanation-based approach (Doctoral dissertation, Carnegie Mellon University, 1988). *Dissertation Abstracts International, 49*, 4906B–4907B.

Mitchell, T. M., Utgoff, P. E., & Banerji, R. (1983). Learning by experimentation: Acquiring and refining problem-solving heuristics. In R. S. Michalski, J. G. Carbonell, T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* Los Altos, CA: Morgan Kaufmann.

Mueller, E. T. (1987). Daydreaming and computation: A computer model of everyday creativity, learning, and emotions in the human stream of thought (Doctoral dissertation, University of California, Los Angeles). *Dissertation Abstracts International, 48*, 813B.

Neches, R. (1982). Models of heuristic procedure modification (Doctoral dissertation, Carnegie Mellon University, 1981). *Dissertation Abstracts International, 43*, 1645B.

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.* 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.

Newell, A., & Rosenbloom, P. S. (1981). Mechanisms of skill acquisition and the law of practice. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition.* Hillsdale, NJ: Lawrence Erlbaum.

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

Norvig, P. (1985). Frame activated inferences in a story understanding program. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 624–626). Los Angeles: Morgan Kaufmann.

Ohlsson, S. (1983). A constrained mechanism for procedural learning. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (pp. 426–428). Karlsruhe, Germany: Morgan Kaufmann.

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

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

Reed, S. K., Ernst, G. W., & Banerji, R. (1974). The role of analogy in transfer between similar problem states. *Cognitive Psychology, 6*, 436–450.

Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence, 5*, 115–135.

Schank, R. C. (1982). *Dynamic memory.* Cambridge, England: Cambridge University Press.

Schoppers, M. J. (1987). Universal plans for reactive robots in unpredictable environments. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 1039–1046). Milan, Italy: Morgan Kaufmann.

Shavlik, J. (1988). *Generalizaing the structure of explanations in explanation-based learning.* Doctoral dissertation, University of Illinois.

Wallas, G. (1931). *The art of thought.* London: Jonathan Cape.

Yaniv, I., & Meyer, D. E. (1987). Activation and metacognition of inaccessible stored information: Potential bases for incubation effects in problem solving. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 13*, 187–205.