Computational Intelligence, Volume 21, Number 4, 2005

A CONSTRAINED ARCHITECTURE FOR LEARNING AND PROBLEM SOLVING

RANDOLPH M. JONES
Soar Technology, Inc., Waterville, ME 04901

PAT LANGLEY
Computational Learning Laboratory, Center for the Study of Language and Information, Cordura Hall, Stanford University, Stanford, CA 94305

This paper describes EUREKA, a problem-solving architecture that operates under strong constraints on its memory and processes. Most significantly, EUREKA does not assume free access to its entire long-term memory. That is, failures in problem solving may arise not only from missing knowledge, but from the (possibly temporary) inability to retrieve appropriate existing knowledge from memory. Additionally, the architecture does not include systematic backtracking to recover from fruitless search paths. These constraints significantly impact EUREKA's design. Humans are also subject to such constraints, but are able to overcome them to solve problems effectively. In EUREKA's design, we have attempted to minimize the number of additional architectural commitments, while staying faithful to the memory constraints. Even under such minimal commitments, EUREKA provides a qualitative account of the primary types of learning reported in the literature on human problem solving. Further commitments to the architecture would refine the details in the model, but the approach we have taken de-emphasizes highly detailed modeling to get at general root causes of the observed regularities. Making minimal additional commitments to EUREKA's design strengthens the case that many regularities in human learning and problem solving are entailments of the need to handle imperfect memory.

Key words: human and machine learning, problem solving, cognitive architecture, memory limitations, qualitative cognitive model.

1. INTRODUCTION

The subject of problem solving has been studied from a computational standpoint since the earliest days of research in artificial intelligence. Both human and computational problem solvers have limited resources, perhaps the most important of which is time. Problem solutions are only useful if they can be discovered in a timely manner. Thus, intelligent problem-solving systems have typically been designed with efficiency in mind, in that they use intelligent methods to control otherwise intractable searches through a problem space. In addition, learning in this framework usually involves improving the efficiency of the problem-solving methods.

Although efficiency concerns are important for both computational and psychological models, other types of limitations on problem solvers have not received as much attention. The research we report in this paper focuses on the impact of memory limitations on problem solving and learning. We have created a problem-solving architecture that imposes significant constraints on memory. Most significantly, the model does not have free access to its memory. That is, the problem solver may fail to solve some problems because it does not retrieve the appropriate knowledge, even if that knowledge exists in memory. Additionally, the architecture cannot rely on standard backtracking to search a problem space in a systematic manner; rather, it must begin every new search path by restarting from the top-level goal of the current problem.

We have selected these constraints because we believe they have a major impact on how humans learn and solve problems. We have tested this hypothesis by building minimal additional constraints in the architecture. The result is an architecture that does not provide detailed, quantitative accounts of specific human behaviors, but does provide qualitative explanations for general regularities in human learning. By modeling these regularities in a simple way, this minimalist architecture provides evidence that the regularities derive from the

need to deal with memory constraints, rather than arising from more complicated architectural representations and processes.

To develop an architecture centered on the assumed memory constraints required developing an appropriate framework for problem solving, integrating it with a model of memory, and implementing a learning algorithm based on the memory model. We have accomplished this effort by developing a system called EUREKA. Our evaluation of this model focuses on qualitatively replicating robust psychological findings on learning in problem solving.

The following section describes the overall framework of EUREKA's performance mechanism, its memory module, and its learning algorithm, together with some details of the model's actual implementation. We then present some experimental results on the system's behavior, and relate these results to well-known learning phenomena in the area of human problem solving. In the final section, we discuss these results and the system's relationship to other research.

2. AN OVERVIEW OF EUREKA

In this section, we provide a general overview of the EUREKA architecture and its implementation in a computer program. The memory limitations we have mentioned suggest another important design aspect of EUREKA: the system treats *all* problem solving as a form of analogical reasoning. Thus, it does not contain distinct methods for standard problem solving and analogical problem solving, like many other systems (e.g., Anderson 1983; Holland et al. 1986). However, unlike most case-based reasoners (e.g., Hammond 1988; Kolodner, Simpson, and Sycara 1985) and other strictly analogical problem solvers (Carbonell 1986), EUREKA takes advantage of the benefits provided by a standard problem-space search. We first discuss how these assumptions influenced our overall design decisions, and then present EUREKA's implementation in more detail.

2.1. Top-Level Design Decisions

To address the issue of memory limitations in problem solving, we focused on three distinct pieces of the system. These pieces include the problem-solving engine, the memory module, and the methods for learning from experience.

To take the role of memory seriously, we designed EUREKA to reason completely by analogy, but to solve problems by searching through a problem space. This contrasts with most existing analogical problem solvers, such as case-based reasoners, which attempt to solve new problems by transforming solutions from old problems. Such systems retrieve an entire problem solution from memory and use it as the basis for solving the current problem. In contrast, EUREKA creates analogies to individual subproblems each time it confronts a problem-solving decision (which typically occurs many times during a single problem-solving episode). We call this mechanism *analogical search control* (Jones 1993).

As EUREKA searches through a problem space, it must choose an operator to try next at each state it reaches. Instead of using a standard match against generalized operators in memory (which would introduce additional representation requirements), the system finds analogical situations in memory and maps the operators used there to the new situation. EUREKA stores only fully instantiated records of its problem-solving experiences in memory. None of the stored operators contain variables; all operators must be analogically mapped to new situations. The degree to which they must be mapped depends on the degree of match between the old and new situations. Thus, operators can map to other problems within the same domain or across domains, allowing transfer within and across domains to arise from a

single framework. Additionally, the contents of long-term memory increase monotonically; nothing is ever removed from memory. Thus, the forms of learning we examine do not address memory storage requirements, but focus on performance improvements in the ability to retrieve the most useful information from an ever-increasing store of long-term experiences.

To choose a memory model, we looked for one that was relatively generic, well known, had some psychological support, and was computationally well specified. The spreading-activation framework for memory meets these criteria. This framework was originally proposed by Quillian (1968), but it has since appeared in various guises (e.g., Norvig 1985; Charniak 1986; Granger, Eiselt, and Holbrook 1986; Hendler 1986; Anderson and Lebiere 1998). The particular implementation we have chosen for spreading activation is most similar to that used in Anderson's (1976, 1983) early ACT models.

Spreading activation has received considerable attention in the psychological literature, and there are conflicting reports on its accuracy as a model of human memory (Neely 1990). However, the basic mechanism has the qualitative aspects that we desire, and it has been successfully used in computational models. The spreading-activation framework also suggests a natural learning algorithm. As EUREKA acquires experiences, it stores them in memory in the form of a semantic network. Concepts are nodes that are related via links. Because activation spreads throughout the network in proportion to *trace strengths* attached to the links (Anderson 1976), it is natural for the model to learn by updating trace strengths, which in turn changes the system's retrieval patterns. However, experience shows that strengthening retrieval patterns alone is not powerful enough to account for observed human problem-solving behavior. After retrieval, there appears to be a *selection* phase, in which people decide which of the retrieved actions would be best to try next. Learning about selection allows greater improvement based on problem-solving successes, and it also lets the problem solver avoid making bad decisions that have failed in the past. A number of problem-solving models focus on learning selection knowledge (e.g., Laird, Newell, and Rosenbloom 1987; Minton 1990), but we are more interested in the effects of learning on the retrieval phase. Therefore, we have given EUREKA a simple selection method that learns by strengthening and weakening a numeric parameter associated with problem-space states.

An additional important aspect of EUREKA's design is its method for making multiple attempts at solving a problem. Most problem-solving systems require only a single attempt to solve a problem. Problem solving may involve a large amount of searching and backing up from dead ends, but the system will eventually either solve the problem or be able to say with certainty that it cannot solve the problem. This is possible because such systems incorporate systematic search methods for problem solving, such as exhaustive breadth-first or depth-first search. We have intentionally not incorporated such a systematic search mechanism into EUREKA, because it implies additional unrealistic assumptions about memory. Rather, our system makes a series of individual attempts at solving a given problem. If the system encounters a dead end, it starts from scratch with the initial problem description. On subsequent attempts on the problem, EUREKA will likely try a slightly different search path, because its learning mechanism encourages it to avoid past failed paths. However, there is no systematic backtracking and exhaustive search ability, and EUREKA never knows whether it has searched all possible avenues.

We built the nonsystematic search method primarily because systematic search makes unrealistic assumptions about human short-term memory capacity (Altmann and Trafton 2002 have made similar arguments). Systematic search also generally assumes that a system has perfect retrieval abilities for long-term knowledge, so it can "know" when a particular area of the search space has been exhausted. EUREKA does not always have access to all the appropriate long-term knowledge, so it can never make such a definitive judgment. Besides our theoretical concerns, it is interesting to note that there are also functional reasons to

incorporate nonsystematic search methods into a problem-solving system. Using our experiences with EUREKA as an inspiration, Langley (1992) presented theoretical and empirical evidence that a nonsystematic search method similar to EUREKA's outperforms the systematic depth-first search method on some well-defined classes of problems.

The remainder of this section provides a more detailed description of EUREKA's components. We hope to describe the model in sufficient detail so that the reader can follow the evaluation and understand the results in terms of the overall architecture. Jones (1993) provides a more thorough treatment of the computational details of the model and its implementation.

2.2. Performance and Memory in EUREKA

EUREKA divides a problem solution into two types of tasks, based on the *means-ends analysis* framework for problem solving (Newell and Simon 1972). TRANSFORM tasks have the form "TRANSFORM the *current state* into a *new state* that matches a set of *goals*." If the current state satisfies the goals, the TRANSFORM is immediately satisfied; otherwise, the system must select an operator to apply in an attempt to satisfy one or more of the goals. If EUREKA finds and executes such an operator, it produces a new task to TRANSFORM the resulting state into one that satisfies the goals. The system then continues work on the problem by attempting to solve the new TRANSFORM. It is important to note that EUREKA can treat each TRANSFORM that it creates as an unsolved problem, independent of any larger problem that contains it. This fact is a key element for the success of the learning algorithm.

The second type of task that EUREKA encounters is to APPLY an operator. This task can be stated as "APPLY a given *operator* to the *current state*, generating a *new state*." If the current state satisfies the operator's preconditions, the system executes the operator and returns the resulting state; otherwise, it sets up a new task to TRANSFORM the current state into a state that satisfies the preconditions of the operator. After EUREKA recursively solves the new TRANSFORM, it sets up a new task to APPLY the operator to the resulting state.

As EUREKA attempts to solve a problem in this fashion, it stores a trace of its activity in memory. An example trace for a simple "blocks world" problem appears in Figure 1. Because problem solving divides into two types of tasks, the trace for a single attempt to solve a problem looks like a binary tree of TRANSFORMs and APPLYs. The system integrates multiple attempts at a problem into memory, and thus builds a directed, acyclic, AND/OR graph to represent the set of solution attempts on each problem.

At a high level of abstraction, these complete traces are similar to Carbonell's (1986) derivational traces, or a large case storing EUREKA's sum of experiences (Zito–Wolf and Alterman 1993). However, EUREKA's lower-level representation of these traces is a semantic network of concepts and relations. This representation lets EUREKA implement its retrieval process with spreading activation. In contrast, the high-level problem traces are analyzed by the system's decision mechanism to choose appropriate actions after retrieval has occurred.

So far, we have presented no details that distinguish EUREKA's problem-solving algorithm from standard means-ends analysis. Although the system's approach to problem solving is based on this framework, it also includes some important differences. The standard means-ends approach selects operators for application by examining the entire set of operators in memory and choosing one that directly addresses the goals of the current TRANSFORM (Ernst and Newell 1969; Fikes, Hart, and Nilsson 1972; Minton 1990). However, EUREKA is allowed to select *any* operator from memory, subject to the behavior of its retrieval algorithm. This lets the system generate goal-driven behavior, as in standard means–ends analysis, but also supports more opportunistic behavior when appropriate. We call this performance algorithm

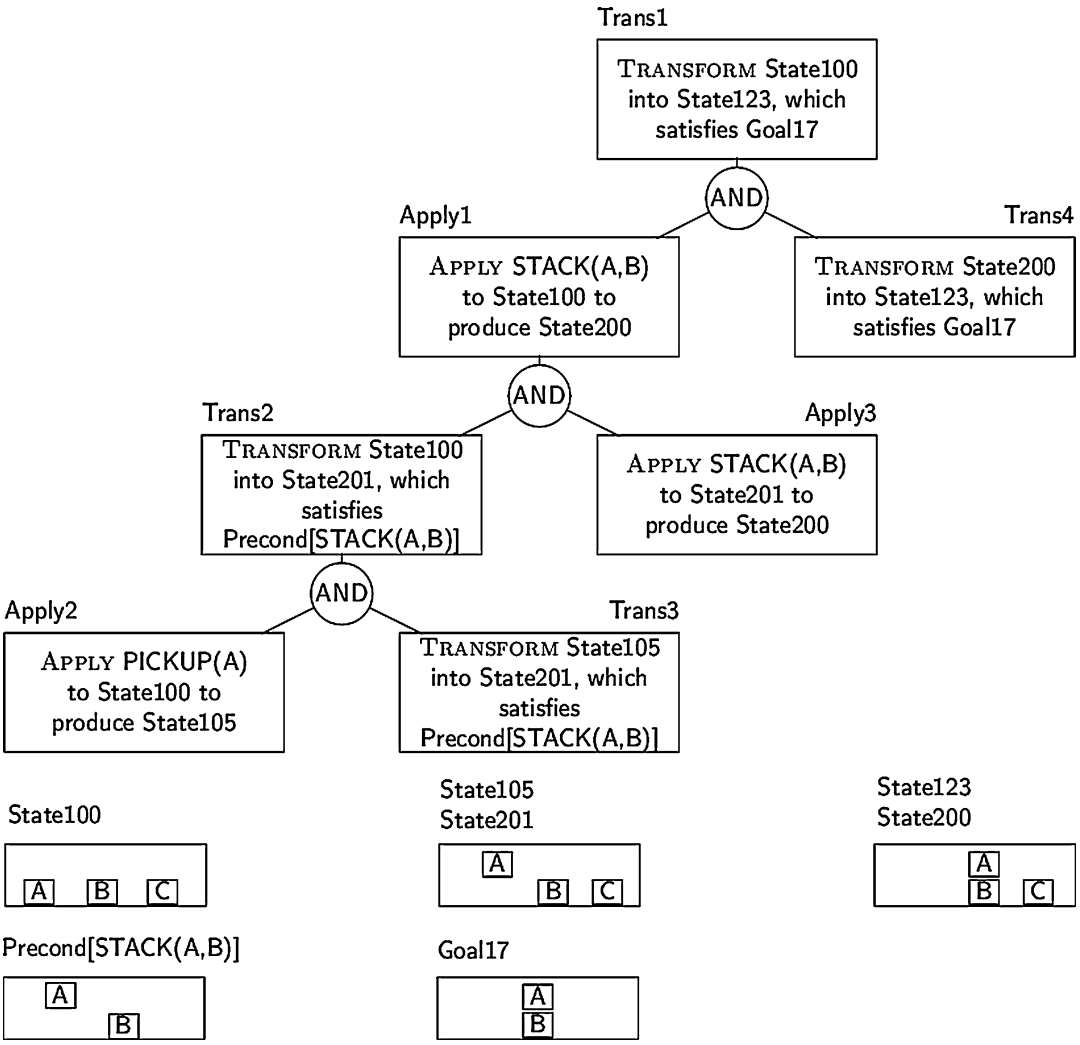

FIGURE 1. A memory trace for a problem from the "blocks world."

flexible means–ends analysis, and it has also been advantageously applied in other problem-solving systems (e.g., Jones and VanLehn 1991, 1994; Langley and Allen 1991).

The flexible nature of EUREKA's performance algorithm demands good algorithms for retrieving and selecting operators, so that it does not simply attempt to apply every known operator to a problem. In addition, as stated previously, EUREKA assumes that it cannot inspect all operators in memory to make a decision, so it requires a mechanism that considers only a small set of operators, but is able to find the "correct" operators for a situation in most cases.

EUREKA's solution is to retrieve operators and experiences from memory via spreading activation. When the system encounters a new TRANSFORM, it identifies the semantic-network nodes that represent that TRANSFORM and sets them up as temporary *sources* of activation. The spreading process begins by initializing each source node with an activation value of one. Each node then multiplies its activation value by a damping factor and passes the new value on to its neighbors in proportion to the trace strengths on the neighbors' links. The last

point is key to the retrieval mechanism. Nodes that are more strongly connected to activation sources will receive more activation themselves. Thus, those concepts that are most strongly connected to the initially activated nodes will most likely be retrieved from memory. After activation has been passed on to a new set of nodes, they repeat the spreading process until each activation "packet" falls below a fixed threshold.

After activation has spread throughout memory, the system examines the activation values of the nodes representing TRANSFORMs. EUREKA uses these values to retrieve a set of previously solved TRANSFORMs from memory. The assumption here is that, due to the associative nature of spreading activation, each TRANSFORM in the retrieved set will be similar in some way to the current problem.

However, the system must prune this set to find the retrieved TRANSFORM that is most similar to the current problem. To this end, EUREKA uses a partial matcher to evaluate the similarity between the current structure and each retrieved TRANSFORM, then probabilistically selects a single TRANSFORM based on this evaluation. The partial match used for selection favors matches in the goals, encouraging goal-directed behavior. However, the system also matches against the current state, so if it does not find a good match in the goal conditions, a forward-chaining strategy will emerge. Thus, a typical solution generated by EUREKA can contain a mixture of goal-directed and opportunistic (state-directed) behavior.

Finally, after the system has selected a single TRANSFORM from memory as most similar to the current problem, it takes the operator used to solve that problem and uses it for the current problem. However, EUREKA's memory traces and operators all describe specific object instances, and do not contain any variables. Thus, the selected operator will be specific to the retrieved TRANSFORM, and will most likely not be directly applicable to the current problem (even if the current problem and the retrieved TRANSFORM are in the same domain).

To overcome this, EUREKA attempts to find an analogical mapping between the current problem and the retrieved TRANSFORM. The current implementation accomplishes this with a simple mapping algorithm that attempts to maximize the number of relations in the two problems that can be matched. After generating one or more mappings, EUREKA selects one with probability proportional to the quality of the match and uses this mapping to construct an analogical image of the selected operator. This lets the system adapt its knowledge to novel situations.

To help understand EUREKA's performance algorithm, let us consider a simple example. Suppose the system has already solved the problem depicted in Figure 1, so that a trace of this solution is stored in memory. Now assume that we present the system with the problem shown in Figure 2. The top-level goal that EUREKA must achieve is to TRANSFORM `State117` into a state that satisfies the goals in `Goal323`. This requires retrieving an operator to APPLY from memory. The system first builds a representation of `Trans9` into its semantic network, and then spreads activation from the nodes that represent `State117` and `Goal323`. Activation

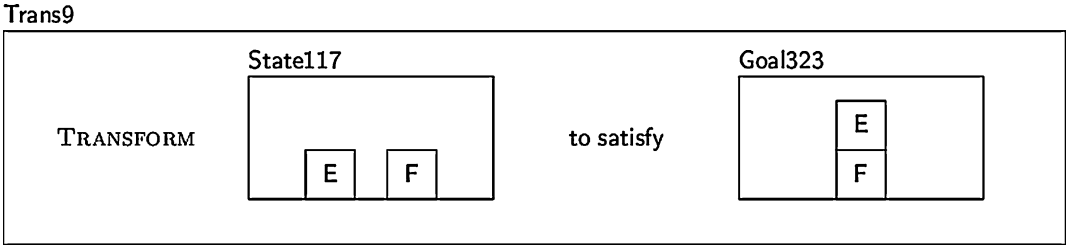

FIGURE 2. EUREKA receives a new "blocks world" problem.

spreads through various pathways (including the nodes for `On`, `Block`, `Table`, and others), and eventually reaches multiple TRANSFORM nodes.

After activation has spread, EUREKA retrieves all the TRANSFORMs with activation above a certain threshold. This threshold is not fixed, but is a percentage of the activation associated with the most active TRANSFORM node. For the sake of exposition, let us assume that the system retrieves `Trans1` and `Trans2` (TRANSFORMs from other problem traces may also be retrieved). The system next determines which of these TRANSFORMs most closely matches the current problem, in this case `Trans1`. In determining this match, EUREKA generates a mapping from `Trans1` to `Trans9` that includes $A \rightarrow E$ and $B \rightarrow F$. For the purposes of simplicity, the TRANSFORMs in this example match each other very closely. In general, EUREKA is able to find partial matches between TRANSFORMs that are quite different from each other. Finally, the system selects the operator STACK(A,B), which was applied to `Trans1`, analogically maps it to STACK(E,F), and recursively attempts to APPLY STACK(E,F) to `State117`. The system then continues work on the problem until it finds a solution or fails by entering a loop. If the system fails, it begins a new attempt at the problem from the top-level TRANSFORM, `Trans9`.

Loop detection is possible because EUREKA dynamically marks the problem situations representing the current solution path. On the surface, this might seem inconsistent with our decision to restrict EUREKA's ability to search systematically for problem solutions. One of our reasons for imposing non-systematic search methods on the system was our belief that humans cannot consistently hold an entire current search path in short-term memory, and can therefore not systematically backtrack. How is it, then, that we justify EUREKA's ability to detect loops in a problem-solving attempt? The answer to this question involves an idealization of demonstrated difficulties in human memory regarding recognition and recall (e.g., Shepard 1967). EUREKA is allowed to *recognize* a situation in the current problem-solving path, to check for loops. In contrast, EUREKA is not allowed to *recall* a situation on the path, so it cannot systematically backtrack.

2.3. Learning in EUREKA

In order for the system to improve its performance, it must use its experiences to alter future behavior at its decision points. EUREKA makes two major decisions each time it selects an operator. These decisions involve the *retrieval* of a set of TRANSFORMs from memory and the *selection* of the best matching TRANSFORM from the retrieved set.

Because we are primarily concerned with memory, the emphasis in EUREKA falls on the retrieval mechanism. Thus, the model focuses its learning strategy on changing the retrieval patterns associated with specific TRANSFORMs. When the system solves a problem, it strengthens the links involved in each TRANSFORM that it used to help solve the problem. In the previous example, the links connected to `A`, `B`, `C`, `Table`, and the `On` nodes would be among those strengthened. Recall that activation spreads to nodes in proportion to the trace strengths of the links on those nodes. Thus, the nodes with strengthened links will "soak up" more activation the next time activation reaches those sections of the semantic network. This in turn lets more activation reach the TRANSFORM nodes to which these concepts are connected. In this manner, retrieval depends on the combination of the features in the current TRANSFORM and the link trace strengths in memory, and the system improves performance by encouraging the retrieval of TRANSFORMs that have proved useful in similar situations in the past. As we will see, this relatively simple learning mechanism overcomes memory limitations in a manner that generates a number of important types of learning behavior.

The current model does not attempt to learn by changing the behavior of its partial matcher or the analogical mapping mechanism during the selection phase. However, it does

alter its behavior slightly in response to failed problem-solving episodes. When this occurs, EUREKA records the TRANSFORMs that it attempted to use to solve the problem, then uses this record to decrease the probability that it will select an offending TRANSFORM in the same situation, even if it matches the current problem well. This helps the system break out of cycles of unproductive behavior that may arise from repeatedly retrieving a TRANSFORM that proves useless. This is an intentionally simple learning mechanism. It almost certainly cannot account for certain types of human learning, but it allows us to determine and focus on the types of behavior changes that can be explained purely by the tuning of the analogical retrieval process. Essentially, our approach is a simple implementation of an extremely common approach to modeling learning. Thorndike's (1898, 1907) Law of Effect dictates that positive feedback causes an increase in memory trace strengths, and negative feedback causes a decrease. Various researchers have built learning models that incorporate one or both of these mechanisms (e.g., Holland et al. 1986; Ohlsson 1996; Ohlsson and Jewett 1997; Anderson and Lebiere 1998). We include very simple forms of both.

We should emphasize that EUREKA is not just learning by adapting weights on existing knowledge structures. Rather the system also records novel structures in a sort of episodic memory for all of the system's problem-solving experiences. EUREKA stores very TRANSFORM and APPLY that it encounters or generates (along with their constituent representations) into memory, making all experiences available for numerical tuning and subsequent retrieval and selection.

In summary, EUREKA uses flexible means-ends analysis to solve problems, and it divides the task of choosing an operator into two processes, retrieving a candidate set of similar TRANSFORMs (problems or subproblems) from memory and selecting the most similar TRANSFORM from the retrieved set. The system uses the selected TRANSFORM as an analogy to the current problem or subproblem to determine the next course of action. In this way, EUREKA combines advantages from learning search heuristics for problem spaces (Mitchell, Utgoff, and Banerji 1983; Langley 1985; Minton 1990) and from analogical or case-based reasoning (Kolodner et al. 1985; Carbonell 1986; Hammond 1988) in solving problems.

Combining a memory model based on spreading activation with a problem solver in this way also provides a considerable degree of flexibility and efficiency. The system can learn simply by updating trace strengths in memory to alter its retrieval patterns. In addition, EUREKA holds the computationally expensive processes of partial matching and analogical mapping in check by examining small portions of memory during each decision cycle. In this way, spreading activation accounts for human retrieval difficulties by allowing the retrieval of useful information without an exhaustive search of memory. In the following section, we will see how this integrated model accounts for a number of characteristics of human problem solving.

3. EMPIRICAL EVALUATION OF EUREKA

We designed EUREKA to explore how a problem solver can perform effectively and improve its behavior under certain assumptions about memory limitations. In evaluating the model, we are primarily concerned with learning, or the various ways in which the model can improve its performance with experience. To complete this evaluation, we analyzed how well EUREKA accounts for the main robust psychological findings on learning in problem solving. In this section, we present those findings and report a number of experiments on the implemented EUREKA system. We discuss the results of these experiments in terms of how well they account for the appropriate psychological behaviors.

TABLE 1. Robust Psychological Findings on Learning in Problem Solving (from VanLehn 1989)

1. Subjects reduce their verbalizations of task rules as they become more experienced with practice.
2. Improvement occurs quickly in knowledge-lean domains.
3. There is a power-law relationship between the speed of performance on perceptual-motor skills (and possibly problem-solving skills) and the number of practice trials.
4. Problem isomorphs do not become more difficult simply by changing surface features of the problems.
5. Other representation changes can make problem isomorphs substantially more difficult.
6. There is asymmetric transfer between tasks when one task subsumes another.
7. Negative transfer is rare.
8. "Set" effects (or *Einstellung*) can lead to negative transfer.
9. Spontaneous noticing of a potential analogy is rare.
10. Spontaneous noticing is based on superficial features.

The effects that EUREKA should model come from VanLehn's (1989) review of the psychological literature on learning in problem solving. He identified ten robust empirical findings in this area, which we have listed in Table 1. We will discuss each finding in turn as we describe the relevant empirical study with EUREKA. VanLehn also included nine results on expert-novice differences, but those fall outside of our current study.

These learning behaviors fall under four broad categories: practice effects, problem isomorphs, transfer across problems, and problem solving by analogy. Thus, our evaluation of EUREKA focuses on these areas. The remainder of this section describes the experiments we ran on the system, beginning with a description of the performance measures we use to gauge learning in the model. We follow with a presentation of the experiments and their results, discussing how the latter reflect EUREKA's ability to overcome memory limitations and exhibit the learning behaviors listed in Table 1.

3.1. Performance Measures

Our interest in analyzing how EUREKA improves its performance over time suggests three natural dependent measures, which we used in our experiments. All three of these measures relate to how much effort the system expends in its search for a problem solution. The measures include the number of attempts that EUREKA makes in trying to solve a problem (EUREKA must usually make multiple attempts, because it has no backtracking mechanism), the number of means-ends task nodes (APPLYs and TRANSFORMs) that EUREKA visits during a problem-solving trial (a measure of total search effort), and the number of *unique* APPLY and TRANSFORM nodes that EUREKA visits during a single trial (which can be viewed as a measure of "productive" search effort).

In our experiments, a problem-solving "trial" represents a single episode in which EUREKA tries to solve a problem. During such an episode, the system searches for a solution, possibly including restarting from the initial problem conditions. We label each restart as a problem-solving "attempt," which gives one measure of the search effort involved in a single trial. EUREKA completes a single trial either by solving the problem or be reaching some maximum number of attempts. EUREKA's current implementation allows only fifty attempts at a problem within a single trial. After fifty attempts, the system ends the current trial, considers the problem to be unsolved, and awaits the next problem. The measures of search effort are similar to those used to evaluate the performance of other problem-solving systems (e.g., Shavlik 1988; Minton 1990).

3.2. Practice Effects on Individual Problems

Items 1, 2, and 3 in Table 1 address perhaps the simplest type of learning that a problem solver can exhibit: improvement of performance on an individual problem during repeated attempts at its solution. Item 1 indicates that human subjects seem to automate their step by step reasoning as they gain experience on a task. This is sometimes interpreted to suggest that the subjects are shifting from more declarative representations of the task to more procedural representations. A similar interpretation is that experience gradually decreases the amount subjects must think about what to do, making deliberate steps more automatic. Items 2 and 3 suggest a pattern where initial learning is quite rapid, particularly in knowledge-lean domains, but the rate of learning slows with experience.

To evaluate EUREKA's ability to improve with practice, we created a number of problems, based on the "Towers of Hanoi" puzzle and the "blocks world" domain, in which an agent moves blocks of various sizes and colors to create specified configurations. These are not complex problem-solving domains. However, they are comparable to the types of tasks studied in the psychological literature, and they serve to illustrate the low-level learning effects we wish to explain. Other research (Jones and VanLehn 1991, 1994; VanLehn, Jones, and Chi 1992; VanLehn and Jones 1993) investigates the use of some of EUREKA's mechanisms in more complex domains.

In order for the system to solve problems from a particular domain, it must be given an initial set of operators for that domain. For example, in the "blocks world" domain, we supplied EUREKA with operators for stacking one block on another, unstacking one block from another, picking up a block, and putting down a block. Note that these are not generalized operators, containing variables. Rather, they are particular instances of each type of operator.

To store these operators in memory, EUREKA sets up a small problem trace for each operator. The problem trace starts with a goal to TRANSFORM the state containing the preconditions of the operator into a state in which the actions of the operator have been executed. This is a trivial problem that is immediately solvable by simply executing the operator that was used to set up the problem. By storing each of these miniature problems, EUREKA effectively loads the domain operators into memory so they become retrievable during problem solving. However, none of the operators initially appear in the context of any global problem-solving trace (which has a large effect on retrieval). As an example, the problem trace that initializes the operator STACK(A,B) appears in Figure 3.

After initializing EUREKA's memory in each domain, we tested it with a number of small problems. Problems in the "blocks world" consisted of various steps in building towers from a number of blocks. For the "Towers of Hanoi" puzzle, the problems consisted of moving different numbers of disks from one peg to another, given the standard set of constraints for moving disks. Each time we gave the system a problem, we presented the same problem 10 times in a row. In each of the 10 trials, EUREKA repeatedly attempted to solve the problem until it found the solution or it had made 50 attempts, at which point the system gave up and received the next trial. The system's learning mechanisms were turned on continuously, both within and across trials, and we collected performance measures for each trial of each problem. We then averaged these results for each set of 10 trials on a problem.

The results exhibited similarly shaped graphs for all the performance measures in both task domains. Each measure started with a fairly high value, quickly falling after the first and second trials, and then remaining fairly level. Our analysis compares these results with the psychological findings on practice effects (Items 1–3 in Table 1). The data demonstrated EUREKA's ability to learn quickly in knowledge-lean domains. Most of the improvement takes place during the early trials, particularly the first trial. After the early trials, the system shows some gradual improvement until performance eventually levels off. None of the dependent variables we measured exhibited a power-law relationship for improvement, but the graphs do

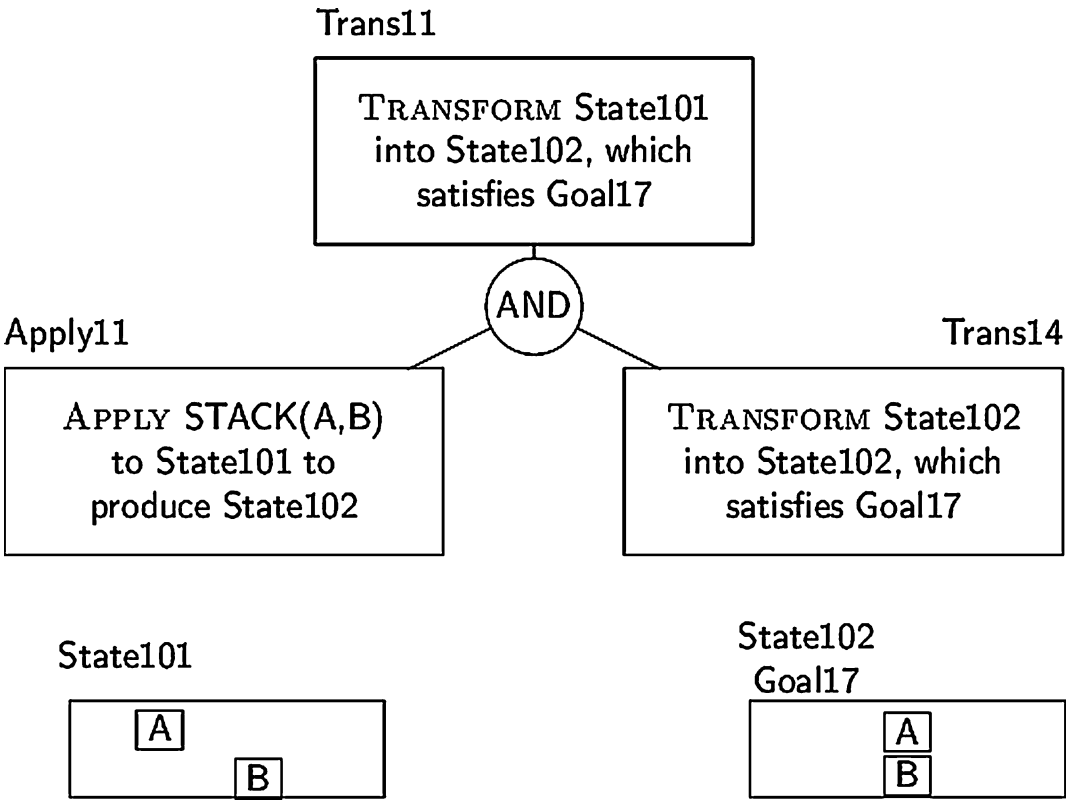

FIGURE 3. Initial trace for the operator STACK(A,B).

have the expected qualitative shape. It is worth noting that the jury is still out on whether the power law applies uniformly to cognitive skills as well as perceptual-motor skills. Matching precise power-law improvement in problem solving has proved to be a difficult task in general for computational systems on "natural" problem-solving tasks (Neves and Anderson 1981; Newell and Rosenbloom 1981). However, Ohlsson and Jewett (1997) in particular have characterized the conditions under which various models can (or cannot) exhibit power-law learning for certain abstract classes of problems.

The particular shapes of the curves from our experiments are indicative of how EUREKA acquires and tunes its knowledge. The first time EUREKA sees a problem, most of its learning involves adding new declarative structures to long-term memory. Before these structures exist, EUREKA has little experience to help guide its search on the novel problem. On subsequent trials, most of EUREKA's learning involves adjusting the strengths of links that already exist in memory. This leads to gradual improvement until the system reaches the point where it finds solutions with almost no wasted search.

3.3. Intradomain Transfer

Another type of learning involves 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-operator formation (Fikes et al. 1972; Iba 1989) rule composition (Anderson

1983), chunking (Laird et al. 1987), or any type of mechanism that generates typical sequences from individual operators. Ohlsson and Rees (1991) have also investigated this type of learning from a cognitive-modeling perspective. Approaches to intra-domain transfer rely on the fact that difficult problems can be regarded as sequences of smaller, simpler problems. The relevant psychological result on this type of learning is summarized by Item 6 in Table 1. The result basically states there will be differences in improvement based on the order in which a series of problems are presented.

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, having a solution to the small problem increases the probability that the difficult problem will be solved. This is the primary type of learning that intelligent problem-solving systems exhibit, and EUREKA also produces it through its hierarchical decomposition of problems.

We tested our predictions by again measuring EUREKA's performance on sets of problems from the "Towers of Hanoi" and "blocks world" domains. Each set contained problems of increasing optimal solution length, the earlier problems providing solutions to pieces of the later problems. In the control condition, we ran each problem separately, throwing out any learned memory structures between each trial. For the test condition, we ordered the problems by optimal solution length and ran them successively, allowing learning to take place. In each case, EUREKA was allowed only one trial (up to 50 attempts) per problem. Figure 4 compares EUREKA's ability to solve the "Towers of Hanoi" problems under the

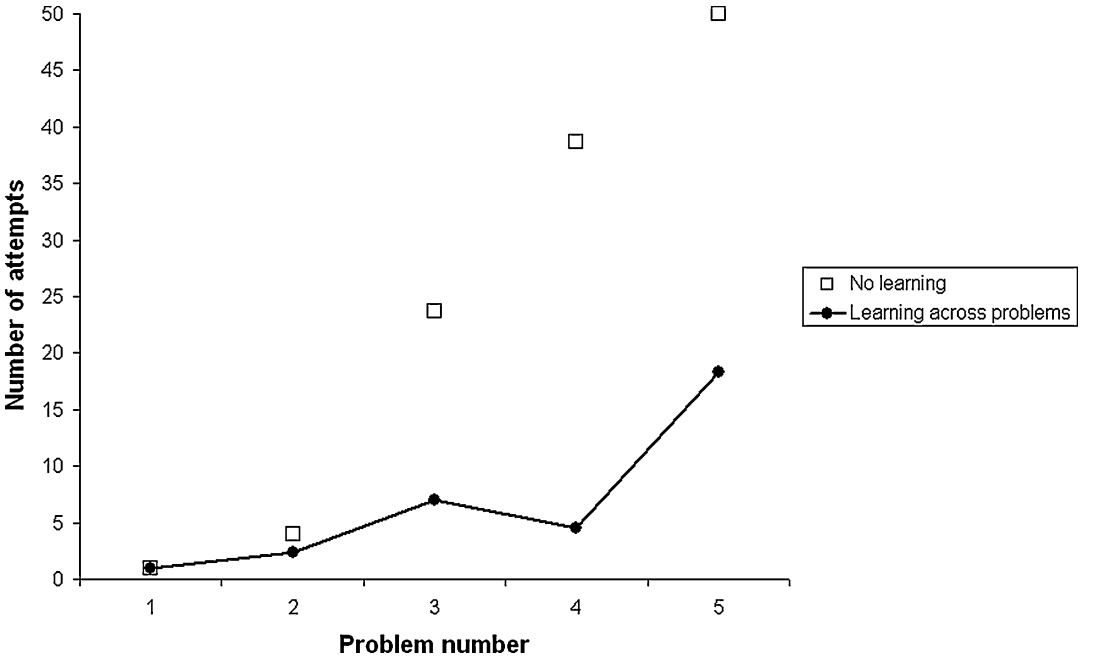

FIGURE 4. Improvement in number of attempts: solving "Towers of Hanoi" problems in order of optimal solution length.

control and test conditions. The results for the other performance measures, as well as for the "blocks world" domain, were similar.

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 not able to solve them after 50 attempts. These problems are marked in the figures by larger dots. 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. 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.

By examining the number of nodes visited and the number of unique nodes visited, we found that improvement also 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. In this manner, EUREKA shows the expected ability to improve from intra-domain transfer.

This result bears on VanLehn's observations concerning asymmetric transfer (Item 6 in Table 1). The experiment demonstrates that there is transfer from simple to more complex problems. We did not run the reverse experiment, solving difficult problems first, followed by simpler problems. However, we are convinced that EUREKA would exhibit two fairly robust properties. First, there is a good chance that the system would never even solve the difficult problems, meaning that it would have very little success to learn from. If this happened, the system would be only incrementally better off after attempting the difficult problems than before. It is possible that EUREKA would solve some of the subproblems, learning from them, but because difficult problems decompose "top down", the system might never get to the level of easily solved sub-problems. Assuming the system *could* solve the difficult problems after considerable effort, there would certainly be immediate transfer to the simpler problems, because complete solutions to the simpler problems would already be stored and strengthened in memory, making them easy to retrieve.

3.4. Interdomain Transfer or Analogy

Transfer is also possible between problems from different domains. This type of transfer usually involves the use of an analogy (in the more common sense of the term) from one domain to aid problem solving in another domain. EUREKA's ability to solve problems by analogy stems from the same mechanism that allows transfer within domains. The only differences are that concepts in the current problem and the analogy will be semantically farther apart, and more relations must be generalized when matching a retrieved TRANSFORM to the current problem.

Given the general ability of humans to solve problems by analogy, Items 4 and 5 in Table 1 list some specific characteristics for when such an ability can be successful applied. These results indicate that the real structure of a problem (as opposed to surface features) determines whether the problem will be easy or difficult to solve. Changing surface features does not change this. However, taking the "spirit" of a problem and casting it into an entirely new structural representation can have a dramatic impact on the ease of finding a solution. Items 9 and 10 in Table 1 also highlight some of the characteristics of human use of analogy. Item 9 indicates that humans generally fail to retrieve appropriate, useful analogies spontaneously (i.e., without a hint or some other retrieval aid). Item 10 indicates that when such retrieval does occur, it is usually based on superficial similarities rather than deep structural similarities between two problems or problem domains. In fact, the inability of humans to retrieve

analogies based on structural similarities may tell part of the story for why analogies are so difficult to retrieve spontaneously.

To demonstrate that EUREKA behaves in a similar manner, we selected domains that Holyoak and Koh (1987) used in their research on analogical problem solving by humans, which included two groups of subjects. The test group had been exposed to Duncker's (1945) "radiation" problem, in which a patient has a tumor that could be destroyed by high-powered X-rays. However, high power X-rays would also destroy any healthy tissue it went through, so the solution is to combine a number of lower intensity X-rays from different directions. The control group of subjects had not been exposed to Duncker's problem. Both groups were asked to solve an analogical problem (to the radiation problem), in which a broken lightbulb must be repaired using laser beams. Holyoak and Koh found that only 10 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 from memory an analogous, isomorphic problem that has already been solved.

For this experiment, we gave EUREKA representations of the radiation and broken lightbulb problems. EUREKA's representation of the lightbulb domain included operators for shooting a laser at a lightbulb, setting the intensity of a laser, shooting multiple lasers at a lightbulb, and a few other operators that would not aid the solution (to allow the system to retrieve something that will not work). In addition, we supplemented EUREKA's semantic network with knowledge relating concepts from the lightbulb and radiation domains. For example, radiation and lasers can both be considered as beams, and broken glass and dead tissue are both (somewhat dissimilar) types of damage. This supplemental knowledge allows weak paths of activation to connect concepts from both domains. We initialized the system with "mini-traces" for each of the "lightbulb operators," but gave no operational knowledge of the radiation domain (although the problem representations were isomorphic at some level of abstraction). After EUREKA had stored these initial operators in memory, we 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.

In the control condition, EUREKA had knowledge of the lightbulb operators, but had difficulty noticing their applicability to the radiation problem. In the test condition, EUREKA exhibited improvements in the average number of attempts taken to solve the radiation problem, the number of nodes visited, and the number of unique nodes visited. The system also showed improvement in its ability to find a solution for the test problem. 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 the 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.

Analogical transfer occurs in a manner similar to intradomain transfer. The difference lies in the degree of similarity between the problems. With intradomain transfer, different problems mention many of the same relations and often many of the same objects. Across domains, activation must spread through more abstract nodes to reach an appropriate target, so retrieval will be more difficult. In addition, more generalization must occur during the matching phase. Thus, transfer is still possible, just more difficult. This also accounts for the ability to solve problem isomorphs when only surface features are changed (Items 4 and 5 in Table 1). As long as the surface features are semantically similar enough for activation to connect them, it is a relatively simple matter for EUREKA to retrieve and select the appropriate operators. Deeper changes in problem representation lead to decreased chance of retrieval, as well as increased difficulty in matching during the selection phase.

3.5. Facilitating Retrieval of Analogies with Hints

Although spontaneous noticing of analogies is difficult for humans and EUREKA, there is evidence that humans can easily carry out analogies once they have been given hints to use them (Holyoak and Koh 1987). To examine the effects of hints on EUREKA, we again used the lightbulb and radiation problems. In this experiment, we provided EUREKA with the same operators for solving a "broken lightbulb" problem that were used in the previous experiment. 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 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.

The data for this experiment showed a clear beneficial impact from using the "hint" to aid retrieval. The control condition was identical to the previous experiment. In the test condition, EUREKA solved the target problem every time it tried, and did so using a dramatically reduced number of attempts and amount of search. The implication here is that spreading activation can lead to retrieval of appropriate knowledge regardless of the sources of the activation. In standard problem solving, activation leads to retrieval because the activation sources are somehow similar to the retrieved concepts. This experiment shows that hints or other external stimuli that spread activation into the network can also lead to improved behavior.

However, EUREKA's reliance on the appropriate sources for spreading activation also account for the difficulty of spontaneously retrieving useful analogies. Before EUREKA has learned the appropriate correspondences between two problems, it has no choice but to rely on surface features to find analogies, because those are the only cues for retrieval. Because two different problem domains likely share relatively few surface features, the retrieval of appropriate analogies is difficult. This accounts for the results described in Items 9 and 10 of Table 1. Only after EUREKA has had a chance to tune its retrieval knowledge (based on previous, successful retrievals), is it able to retrieve an appropriate analogy reliably and base the retrieval on deeper structural similarities.

3.6. Negative Transfer or *Einstellung*

In general, learning is not a bad thing. Item 7 in Table 1 indicates that experience with certain problems is not likely to make a human *worse* on similar problems. However, the flexibility that comes with being able to transfer past experience to new situations can sometimes have disadvantages. *Negative transfer* most often refers to cases where prior learning makes new learning more difficult. However, it can also include cases where prior learning makes new *performance* more difficult, and that is the aspect of transfer we have investigated with EUREKA, because performance difficulties translate directly to learning difficulties. In this view, negative transfer 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. Item 8 in Table 1 describes this type of situation, where specific problem-solving experiences lead to degraded behavior on certain new problems.

Although negative transfer is difficult to detect in EUREKA's normal learning behavior, it is possible to increase the likelihood of negative transfer by presenting degenerate sequences of problems, just as with humans. To demonstrate this, we ran the system on a series of "water jug" problems from one of Luchins' (1942) experiments on *Einstellung*. These problems involve a number of jugs of various sizes. The goal in each problem is to fill and empty jugs in various combinations to end up with a specific amount of water. For example, given a

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

29-gallon jug and a 3-gallon jug, 20 gallons of water can be obtained by filling the large jug and pouring it into the smaller jug three times.

All the problems in Table 2 can be solved by filling the center jug, pouring it into the rightmost jug twice, and then into the leftmost jug once. However, problems 6 and 7 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 7 involves filling the leftmost and rightmost jugs and pouring them into the center jug. In this particular experiment, Luchins found that *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.

To test how EUREKA would behave under similar conditions, we supplied the system with operators for filling jugs, emptying jugs, and pouring the contents of one jug into another. We then presented the system with Luchins' water-jug problems, in the order they appear in Table 2. We found that EUREKA suffers the same difficulties as the subjects in Luchins' experiment. EUREKA's learning mechanisms encourage it to retrieve TRANSFORMs that are familiar and that have been used successfully in the past. The same TRANSFORMs are retrieved even though there might be more useful information contained somewhere else in memory. In these cases, the more familiar information becomes too strongly stored in memory and blocks the retrieval and selection of information that would lead to a simpler solution of the current problem.

This summarizes EUREKA's account of Item 8 in Table 1. In contrast, there is nothing explicit in the EUREKA model to explain the rarity of negative transfer (Item 7 in Table 1). Rather, the explanation appears to lie in the types of problems that people generally encounter. Perhaps degenerate problem sequences such as those in the Einstellung experiments simply do not crop up that often. When they do, people do appear to struggle with inappropriate transfer.

4. DISCUSSION

Our results demonstrate how memory limitations impact even a relatively simple model of problem solving and learning. The primary lesson of this research is that we can account qualitatively for the main psychological effects on learning in problem solving (in Table 1), using a fairly minimal set of architectural assumptions together with realistic constraints on memory. The advantage of building an architecture is that we can analyze the principles

TABLE 3. Key Properties of the EUREKA Model

1. Experiences are initially stored verbatim as declarative memory structures.
2. Stroed experiences encode problem situation descriptions (which generally do not include deep structures or relationships).
3. Links between existing memory structures are tuned based on past experience.
4. Links are tuned in response to specific, successful retrieval episodes.
5. Retrieval is a process of spreading activation.
6. Activation sources can be concepts in the representation of a problem situation or concepts grounded in perception.
7. All reasoning occurs through the analogical mapping of old problem situations to new ones.

underlying the architecture and trace their impact on the effects the architecture models. We have listed EUREKA's primary design principles in Table 3. Figure 5 traces the relationships between these principles and the target phenomena, summarizing the accounts provided in the experimental presentation.

This study has also taught us other lessons. One major finding is that the memory limitations we imposed had a dramatic effect on the problem solver's ability to solve difficult problems. Most of our experiments involved problems from relatively simple domains, but we found that EUREKA simply could not solve many problems of high complexity.

We do not view this as a limitation of the architecture, but it is interesting to speculate on how EUREKA might be improved on more complex problems. It is possible EUREKA's computational power arises from some of the design decisions we made, such as the use of flexible means-ends analysis or analogical search control for reasoning. However, because these mechanisms were initially developed in EUREKA, both have been used in other problem-solving systems (Jones and VanLehn 1991, 1994; Langley and Allen 1991; VanLehn, Jones, and Chi 1992; Jones 1993; VanLehn and Jones 1993). These systems have demonstrated that the mechanisms can solve difficult problems in the absence of the kinds of memory limitations that we imposed on EUREKA. This leads us to the conclusion that memory limitations are indeed the source of EUREKA's relatively poor problem-solving ability.

In spite of these limitations, EUREKA was able to use a simple learning mechanism based on the memory model to improve its behavior in various ways. In fact, it is able to account for most of the robust psychological findings on learning in problem solving. This establishes an explicit connection between memory and problem solving. Assuming that humans (and future problem-solving systems working with massive knowledge bases) must deal with memory limitations, EUREKA explains how memory can adapt to mitigate those limitations.

4.1. Related Work

The motivations in our work overlap with the design philosophy behind Soar (Laird et al. 1987; Newell 1990). For many years, the Soar community has aimed to identify a parsimonious set of functional mechanisms that can produce the full range of intelligent behavior, including learning. However, even with its focus on parsimony, Soar incorporates more functional assumptions than EUREKA. Perhaps the biggest difference lies in Soar's assumption that long-term knowledge consists of generalized associative patterns cast as production rules. This representation has supported development of highly skilled models that incorporate large amounts of intelligence, but it has also meant that, in practice, most Soar models impose little or no limitations on working memory or retrieval of long-term

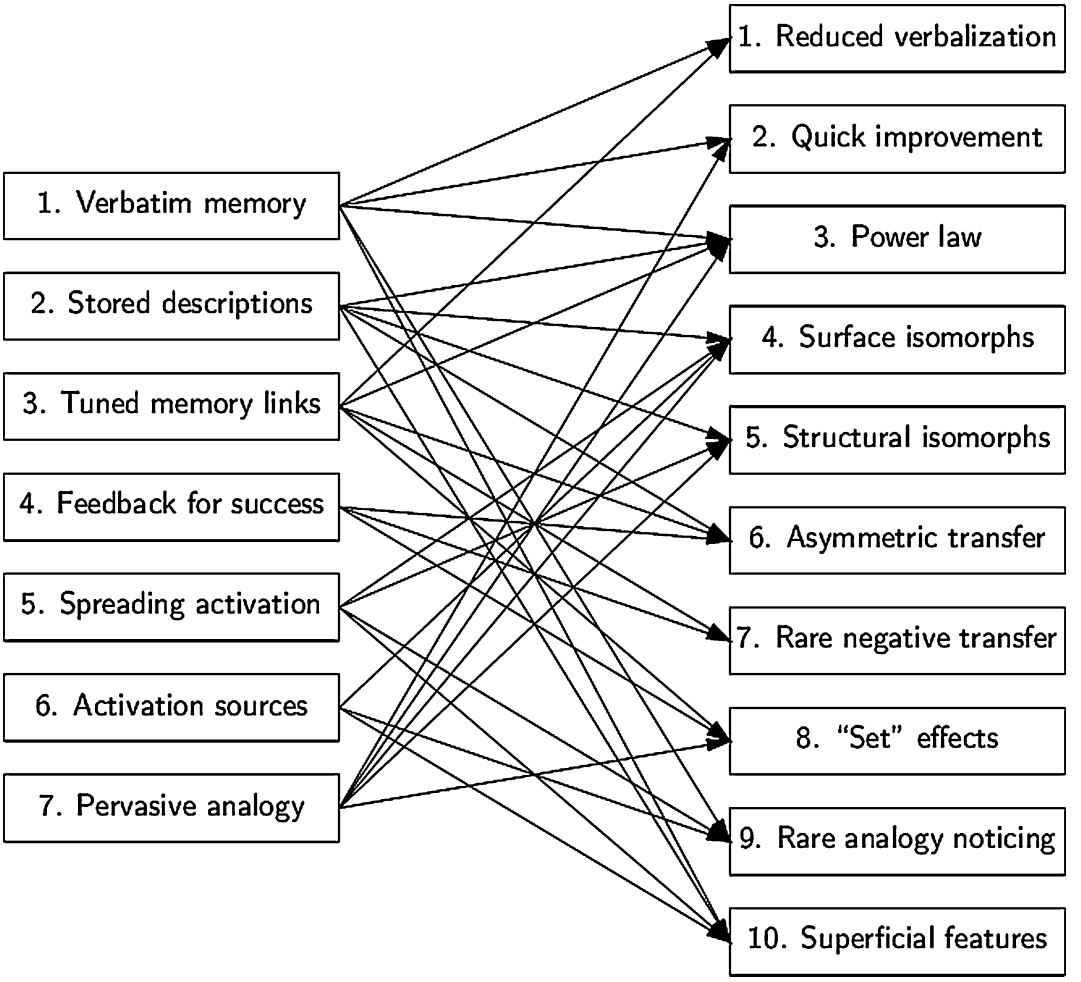

FIGURE 5. Relationships between key properties of EUREKA and the learning phenomena that EUREKA explains.

knowledge. In addition, the use of generalized relational patterns leads naturally to Soar's analytical learning mechanism, calling chunking, for composing such patterns. Combined with appropriate knowledge, chunking has been able to reproduce a number of different styles of learning, but inductive methods have proven quite difficult to implement. In contrast, one of EUREKA's core assumptions is *not* to include such powerful machinery. As a result, it stores only problem-solving instances rather than generalized knowledge, and utilizes relatively simple mechanisms for knowledge acquisition, generalization (via analogical retrieval and mapping), and tuning, which are precisely the areas in which Soar has difficulties. Although Soar is very strong for building knowledge-intensive systems, its assumptions alone may be insufficient to explain many learning phenomena with which we are concerned.

There is also some overlap between our goals and assumptions and those of the ACT research program (Anderson 1976, 1983; Anderson and Lebiere 1998). A major difference is that ACT models have generally provided detailed accounts of particular data, usually with quantitative comparisons. To achieve this, various incarnations of the ACT architecture have incorporated a variety of assumptions about processes and representations. The most

significant overlap between the two research efforts is in ACT's consistent attention to both memory and problem solving.[1] Perhaps the best example comes from Altmann and Trafton's (2002) work on representations of goal structures, which questioned the common assumption that problem-solving models can rely on structures such as goal stacks. They demonstrated a better match to psychological data by treating goals as "standard" knowledge structures that must be retrieved and maintained (using spreading activation) just as other knowledge. The ACT family has enjoyed enormous success in explaining various types of human behavior using a fairly consistent set of architectural assumptions. One drawback is that the variety of options for representing knowledge, structuring problem solutions, and modeling learning sometimes make it unclear which parts of a model are responsible for its explanatory power. We hope that EUREKA can narrow the possibilities for many observed regularities, thus informing the process of model development within other architectures, such as ACT-R.

In addition, our approach shares important features with the PRODIGY architecture (Minton 1990). Although not presented as a psychological model, this framework relies on means–ends analysis as its central problem-solving mechanism, which it augments with control knowledge for selecting, rejecting, or preferring given operators, bindings, states, or goals during the search process. Furthermore, whereas the initial PRODIGY system invoked a form of explanation-based learning to acquire this control knowledge, Veloso and Carbonell (1993) reported a successor that draws on derivational analogy to make such decisions. This mechanism stores the results of means–ends problem solving in an episodic memory and uses them to guide choices on future problems. One key difference is that PRODIGY retrieves entire solutions for reuse, as contrasted with EUREKA's reliance on analogical search control. Another distinction involves its retrieval scheme, which does not utilize spreading activation but is influenced by the current state and goal. A final difference is that PRODIGY uses systematic search rather than the iterative sampling method that EUREKA introduced.

EUREKA also bears clear resemblances to other systems that learn problem-solving knowledge from experience. One close relative is Langley and Allen's (1991) DAEDALUS, which uses subproblem cases stored in a probabilistic concept hierarchy to guide flexible means–ends analysis. Another kindred system is Jones and VanLehn's (1994) GIPS, which induces probabilistic knowledge for selecting means-ends operators and subgoals. Kambhampati and Hendler's (1992) PRIAR stores partial-order plans for analogical reuse on future tasks, Estlin and Mooney's (1997) SCOPE uses inductive logic programming to learn heuristics for partial-order planning, and Upal and Elio (2000) developed methods for learning rules for both search control and plan rewriting. Ohlsson (1996; Ohlsson and Jewett 1997) has characterized and studied a variety of approaches to strengthening and weakening in learning. We will not attempt to review all the work in this broad area, because Zimmerman and Kambhampati (2003) have recently published an extensive survey.

Other research related to EUREKA has focused on modeling analogical retrieval and mapping. Much of the early work on analogical reasoning ignored the process of retrieval, emphasizing instead elaborate mechanisms for generating mappings across representations. Perhaps the most established systems that take both retrieval and mapping seriously are ARCS (Thagard et al. 1990) and MAC/FAC (Gentner and Forbus 1991). In reviewing this work, we are most interested in how these more sophisticated models of analogical reasoning (particularly analogical retrieval) could be embedded into a problem-solving architecture like EUREKA, and especially how this would affect EUREKA's account of learning phenomena. However, to understand the potential impact on learning, we must first review the retrieval algorithms.

[1] However, we should note that, before ACT-R (Anderson and Lebiere 1998), there were few ACT models that combined memory mechanisms with the problem solver.

ARCS retrieves analogies through a two-stage process. First the system uses a lookup table to collect all concepts that are "immediately related" (e.g., by subordinate, superordinate, or part-of relations) to any concept in the target. ARCS then identifies those situations in memory that include at least one of these related concepts, which become the initial set of analogical sources. Next, for each potential target–source pair, ARCS constructs a constraint-satisfaction network that contains one node for each plausible concept match. Nodes connect through excitatory and inhibitory links that reflect which matches are consistent or inconsistent with each other. In addition, special nodes connect concepts that have been marked as important in various ways. Once ARCS has constructed this set of networks, it spreads activation until each network reaches quiescence. Finally, the system computes a retrieval score for each potential analogical source based on the activation of the nodes in the source's network. Unlike EUREKA, it is not clear whether ARCS distinguishes between sources that are "officially retrieved" and those that are not.

The retrieval component of the MAC/FAC analogy engine, MAC, takes a very different approach to retrieval. The system associates with each potential source situation a *content vector* that is computed and stored when the source situation is added to long-term memory. This vector contains a slot for each feature that is "nonsimple" in that it includes other features or concepts as arguments. The content vector for a situation records the presence or absence of each possible feature in the situation's representation, which means it contains slots for every non-simple feature that can ever appear in a situation. When MAC encounters a new target situation, it quickly computes the content vector for the target, then calculates the dot product of the target's vector with each content vector stored in memory. The resulting values are estimates of the rough similarity between the target and each potential source. MAC marks the source with the largest dot product as a retrieved candidate analogy and also marks any source with a value of at least 10 percent of the largest value as retrieved.

Of most interest to us is how these different retrieval mechanisms might influence learning results if they were embedded in our architecture. Like EUREKA, ARCS uses an activation-based, competitive mechanism for retrieval. In contrast, retrieval in the MAC/FAC system is, for the most part, *not* competitive, with retrieval scores for each source situation being computed independently. A competitive mechanism would make it fairly simple to use reinforcement-based tuning, as we have done in EUREKA, but a straightforward combination of EUREKA and MAC would not predict incremental improvement with practice. The most significant way to change a stored memory's retrieval score in MAC would be actually to change its representation, thus changing its content vector, which would in turn lead to new retrieval scores. This is an intriguing idea, because it provides a pragmatic context for a computer system to create new concepts and representations. Similar representation changes are a key part of Hofstadter's (1995) family of analogy systems. However, Hofstadter views the search for new representations as a short-term activity that does not involve learning, where we need long-term representation changes that have an impact on future behavior. In any case, there are certainly lessons to be learned from the variety of existing approaches to analogical retrieval and how they might be adapted to a problem-solving architecture like EUREKA.

5. CONCLUSIONS

In summary, we have learned how strict constraints on memory can strongly impact the ability of a problem solver to perform. In addition, a minimal set of architectural assumptions overcomes these limitations, simultaneously exhibiting the general types of learning behaviors that humans demonstrate when solving problems. Further architectural explorations might attempt to add theoretical commitments, to provide more quantitative accounts (as

with ACT-R), but minimizing the assumptions helps make the contributions of each more clear. In particular, however, we hope further to explore EUREKA's validity as a psychological model by examining alternative retrieval and learning mechanisms, as well as other ways in which the current mechanisms account for human performance.

Aside from psychological concerns, we are interested in how EUREKA might be extended as a computational system for problem solving. The system's current mechanisms indicate that it can acquire useful problem-solving knowledge for simple domains. EUREKA's inability to solve more complex problems arises in part from the strict constraints on memory, and in part from lack of knowledge and learning in the selection phase of the problem-solving cycle. Thus, some of our future efforts will focus on these aspects of the model, in an effort to improve the level of expertise that EUREKA can obtain and the complexity of problems it can solve.

We also have a continuing interest in investigating how aspects of EUREKA's design can be applied to other problem-solving architectures. As we have mentioned, we have had success translating analogical search control and flexible means–ends analysis into more traditional search-based architectures. This related research successfully adapts EUREKA's mechanisms to problems of more complexity, requiring more expertise and search to solve. It is our hope that these efforts will complement psychological studies with EUREKA, allowing us to study aspects of human learning in more complex domains. From the computational perspective, it would also be interesting to investigate how other analogical retrieval mechanisms in the cognitive science literature might be adapted to address EUREKA's concerns with memory limitations and modeling human learning in problem solving.

Overall, we have learned quite a bit from our experiences with the EUREKA model, and it has left us with a number of promising paths for future research. Using EUREKA's small set of commitments as a starting point, we hope that further study in these directions will lead us to an improved theory of learning and expertise in problem solving.

ACKNOWLEDGMENTS

We would like to thank Bernd Nordhausen and Don Rose for numerous discussions that led to the development of many of the ideas in this paper. In addition, we thank Kurt VanLehn, Paul Thagard, Keith Holyoak, and anonymous reviewers for their helpful comments on earlier drafts describing this work. 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.

REFERENCES

ALTMANN, E. M., and J. G. TRAFTON. 2002. Memory for goals: An activation-based model. Cognitive Science, **25**(1):39–83.

ANDERSON, J. R. 1976. Language, Memory, and Thought. Lawrence Erlbaum, Hillsdale, NJ.

ANDERSON, J. R. 1983. The Architecture of Cognition. Harvard University Press, Cambridge, MA.

ANDERSON, J. R., and C. LEBIERE. 1998. The Atomic Components of Thought. Lawrence Erlbaum, Mahwah, NJ.

CARBONELL, J. G. 1986. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. *In* Machine Learning: An Artificial Intelligence Approach, vol. 2. *Edited by* R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. Morgan Kaufmann, Los Altos, CA.

CHARNIAK, E. 1986. A neat theory of marker passing. *In* Proceedings of the Fifth National Conference on Artificial Intelligence. Morgan Kaufmann, Los Altos, CA, pp. 584–588.

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

ERNST, G., and A. NEWELL. 1969. GPS: A Case Study in Generality and Problem Solving. Academic Press, New York.

ESTLIN, T. A., and R. J. MOONEY. 1997. Learning to improve both efficiency and quality of planning. *In* Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, Yokohama, Japan, pp. 1227–1233.

FIKES, R. E., P. HART, and N. J. NILSSON. 1972. Learning and executing generalized robot plans. Artificial Intelligence, **3**:251–288.

GRANGER, R. H., K. P. EISELT, and J. K. HOLBROOK. 1986. Parsing with parallelism: A spreading-activation model of inference processing during text understanding. *In* Experience, Memory, and Reasoning. *Edited by* J. L. Kolodner and C. K. Riesbeck. Lawrence Erlbaum, Hillsdale, NJ.

GENTNER, D., and K. FORBUS. 1991. MAC/FAC: A model of similarity-based retrieval. *In* Proceedings, Thirteenth Annual Conference of the Cognitive Science Society, Lawrence Erlbaum, Hillsdale, NJ, pp. 504–509.

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

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

HOFSTADTER, D. R. 1995. Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought. Basic Books, New York.

HOLLAND, J. H., K. J. HOLYOAK, R. E. NISBETT, and P. R. THAGARD. 1986. Induction: Processes of Inference, Learning, and Discovery. MIT Press, Cambridge, MA.

HOLYOAK, K. J., and K. KOH. 1987. Surface and structural similarity in analogical transfer. Memory and Cognition, **15**:332–340.

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

JONES, R. M. 1993. Problem solving via analogical retrieval and analogical search control. *In* Foundations of Knowledge Acquisition: Machine Learning. *Edited by* A. L. Meyrowitz and S. Chipman. Kluwer Academic, Boston.

JONES, R. M., and K. VANLEHN. 1991. Strategy shifts without impasses: A computational model of the sum-to-min transition. *In* Proceedings, Thirteenth Annual Conference of the Cognitive Science Society, Lawrence Erlbaum, Hillsdale, NJ, pp. 358–363.

JONES, R. M., and K. VANLEHN. 1994. Acquisition of children's addition strategies: A model of impasse-free, knowledge-level learning. Machine Learning, **16**:11–36.

KAMBHAMPATI, S., and J. A. HENDLER. 1992. A validation-structure-based theory of plan modification and reuse. Artificail Intelligence, **55**:193–258.

KOLODNER, J. L., R. L. SIMPSON, and K. SYCARA. 1985. A process model of case-based reasoning in problem solving. *In* Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, Los Altos, CA, pp. 284–290.

LAIRD, J. E., A. NEWELL, and P. S. ROSENBLOOM. 1987. Soar: An architecture for general intelligence. Artificial Intelligence, **33**:1–64.

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

LANGLEY, P. 1992. Systematic and nonsystematic search strategies. *In* Proceedings of the First International Conference on AI Planning Systems, Morgan Kaufmann, Los Altos, CA.

LANGLEY, P., and J. A. ALLEN. 1991. Learning, memory, and search in planning. *In* Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society, Lawrence Erlbaum, Chicago, pp. 364–369.

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

MINTON, S. N. 1990. Quantitative results concerning the utility of explanation-based learning. Artificial Intelligence, **42**:363–391.

MITCHELL, T. M., P. E. UTGOFF, and R. BANERJI. 1983. Learning by experimentation: Acquiring and refining problem-solving heuristics. *In* Machine Learning: An Artificial Intelligence Approach. *Edited by* R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. Morgan Kaufmann, Los Altos, CA.

NEELY, J. H. 1990. Semantic priming an retrieval effects in visual recognition: A selective review of current findings and theories. *In* Basic Processes in Reading: Visual Word Recognition. *Edited by* D. Besner and G. Humphreys. Lawrence Erlbaum, Hillsdale, NJ.

NEVES, D. M., and J. R. ANDERSON. 1981. Knowledge compilation: Mechanisms for the automatization of cognitive skills. *In* Cognitive Skills and Their Acquisition. *Edited by* J. R. Anderson. Lawrence Erlbaum, Hillsdale, NJ.

NEWELL, A. 1990. Unified Theories of Cognition. Harvard University Press, Cambridge, MA.

NEWELL, A., and P. S. ROSENBLOOM. 1981. Mechanisms of skill acquisition and the law of practice. *In* Cognitive Skills and Their Acquisition. *Edited by* J. R. Anderson. Lawrence Erlbaum, Hillsdale, NJ.

NEWELL, A., and H. A. SIMON. 1972. Human Problem Solving. Prentice-Hall, Englewood Cliffs, NJ.

NORVIG, P. 1985. Frame activated inferences in a story understanding program. *In* Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, Los Altos, CA, pp. 624–626.

OHLSSON, S. 1996. Learning from performance errors. Psychological Review, **103**:241–262.

OHLSSON, S., and J. JEWETT. 1997. Simulation models and the power law of learning. *In* Proceedings of the 19th Annual Conference of the Cognitive Science Society, (pp. 584–589). *Edited by* M. G. Shafto and P. Langley. Lawrence Erlbaum, Mahwah, NJ.

OHLSSON, S., and E. REES. 1991. Adaptive search through constraint violations. Journal of Experimental and Theoretical Artificial Intelligent, **3**:33–42.

QUILLIAN, M. R. 1968. Semantic memory. *In* Semantic Information Processing. *Edited by* M. L. Minsky. MIT Press, Cambridge, MA.

SHAVLIK, J. 1988. Generalizing the Structure of Explanations in Explanation-Based Learning. Doctoral dissertation, University of Illinois.

SHEPARD, R. N. 1967. Recognition memory for words, sentences and pictures. Journal of Verbal Learning and Verbal Behaviour, **6**:156–163.

THAGARD, P., K. J. HOLYOAK, G. NELSON, and D. GOCHFELD. 1990. Analog retrieval by constraint satisfaction. Artificial Intelligence, **46**:259–310.

THORNDIKE, E. L. 1898. Animal intelligence: An experimental study of the associative processes in animals. Psychological Review Monograph Supplement, **2**(4):8.

THORNDIKE, E. L. 1907. The Elements of Psychology, 2nd edition. A. G. Seiler, New York.

UPAL, M. A., and R. ELIO 2000. Learning search control rules versus rewrite rules to improve plan quality. *In* Proceedings of the Thirteenth Canadian Conference on Artificial Intelligence, Springer-Verlag, New York, pp. 240–253.

VANLEHN, K. 1989. Problem solving and cognitive skill acquisition. *In* Foundations of Cognitive Science. *Edited by* M. I. Posner. MIT Press, Cambridge, MA.

VANLEHN, K., and R. M. JONES. 1993. Integration of explanation-based learning of correctness and analogical search control. *In* Machine Learning Methods for Planning. *Edited by* S. Minton. Morgan Kaufmann, Los Altos, CA.

VANLEHN, K., R. M. JONES, and M. T. H. CHI. 1992. A model of the self-explanation effect. Journal of the Learning Sciences, **2**:1–59.

VELOSO, M. M., and J. G. CARBONELL. 1993. Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. Machine Learning, **10**:249–278.

ZIMMERMAN, T., and S. KAMBHAMPATI. 2003. Learning-assisted automated planning: Looking back, taking stock, going forward. AI Magazine, **24**:73–96.

ZITO–WOLF, R., and R., ALTERMAN. 1993. A framework and an analysis of current proposals for the case-based organization and representation of procedural knowledge. *In* Proceedings of the Eleventh National Conference on Artificial Intelligence, MIT Press, Cambridge, MA, pp. 73–77.