Retrieval and Learning in Analogical Problem Solving

Randolph M. Jones
Artificial Intelligence Laboratory
University of Michigan
1101 Beal Avenue
Ann Arbor, MI 48109-2110
rjones@eecs.umich.edu

Pat Langley
Robotics Laboratory
Computer Science Department
Stanford University
Stanford, CA 94305
langley@cs.stanford.edu

Abstract

EUREKA is a problem-solving system that operates through a form of analogical reasoning. The system was designed to study how relatively low-level memory, reasoning, and learning mechanisms can account for high-level learning in human problem solvers. Thus, EUREKA's design has focused on issues of memory representation and retrieval of analogies, at the expense of complex problem-solving ability or sophisticated analogical elaboration techniques. Two computational systems for analogical reasoning, ARCS/ACME and MAC/FAC, are relatively powerful and well-known in the cognitive science literature. However, they have not addressed issues of learning, and they have not been implemented in the context of a performance task that can dictate what makes an analogy "good". Thus, it appears that these different research directions have much to offer each other. We describe the EUREKA system and compare its analogical retrieval mechanism with those in ARCS and MAC/FAC. We then discuss the issues involved in incorporating ARCS and MAC/FAC into a learning problem solver such as EUREKA.

We are interested in the low-level memory, learning, and reasoning processes that give rise to improvement in problem-solving behavior over time. EUREKA is the problem-solving architecture we are using to study these processes. An explicit assumption within EUREKA's design is that all processes are aspects of analogical reasoning. In addition, we designed the system so that the low-level retrieval and matching processes would dominate its behavior. The system does not possess or learn the types of high-level control knowledge found in other problem-solving systems. Our intent is to investigate how much of human learning in problem solving can be modeled with such low-level mechanisms.

This paper presents an overview of EUREKA's architecture and some of the learning results it accounts for. We then turn our attention to two well-known analogical retrieval mechanisms in the cognitive science literature. ARCS (Thagard, Holyoak, Nelson, & Gochfeld, 1990) and MAC/FAC (Gentner & Forbus, 1991) model psychological findings on analogical retrieval and reasoning. However, neither has been examined in the context of a problem-solving system, or in a system that learns with experience. The remainder of the paper focuses on the issues of analogical retrieval and learning, and discusses the possibilities of incorporating these alternative analogical retrieval mechanisms into a problem-solving system.

Terminology

Before continuing, it is worth defining some terms to avoid future confusion. For analogical reasoning, a basic unit of knowledge is the *analogical case*, which is further decomposed into a set of concepts and relations between those concepts. For our purposes, every analogical case corresponds to a *problem situation*. A problem situation is a specific set of relations describing a state of the world, together with a set of goal relations that should be achievable by applying a sequence of operators to that state. Note that cases in EUREKA are a bit different from those in case-based reasoning, where "case" typically denotes an entire problem solution. At any given time, EUREKA will have a current problem situation, for which it must decide on an operator to apply. This is the *target problem situation*. The analogical reasoning process is generally divided into three stages. First, a *retrieval* mechanism identifies a number of *candidate sources* from the potential analogies stored in memory. Next, the set of candidate sources undergo further *elaboration* to fill out the potential mappings between each source and the target. Finally, *evaluation* of each candidate source determines how well each candidate will serve as an analogical source for the target. Let us now turn to a description of EUREKA in these terms.

An overview of EUREKA

Jones (1993) presents the computational details of EUREKA, but here we provide a general overview of the system. EUREKA adopts a reasoning formulation called *flexible means-ends analysis* (Jones & VanLehn, 1994; Langley & Allen, 1991). As described above, each problem situation includes a current world state and a set of goal conditions to which the state should be transformed. Operator selection creates a goal to apply a particular operator to the current state of the problem situation. If the preconditions of the operator can all be matched to the current state, the operator executes, leading to a new problem situation with a different state but the same goals. Otherwise, the system sets up a new problem situation with the same current state, but with the operator's preconditions as the new goals. EUREKA then treats this new problem situation in a recursive manner.

The difference between flexible means-ends analysis and standard means-ends analysis (Ernst & Newell, 1969; Fikes & Nilsson, 1971) is that the flexible form does not require selected operators to apply directly to the current goal conditions (i.e., it is not necessary that the selected operator obviously "reduce any differences"). Rather than using this heuristic to limit search, EUREKA relies on its retrieval and learning mechanisms to control which operators are suggested to apply to any particular problem situation. Because operator selection depends on the entire problem situation (and not just the goals), EUREKA can blend goal-driven and opportunistic behavior when appropriate.

Every time EUREKA generates a new problem situation, it stores a representation of the situation (as well as the operator the led to this situation) into its long-term semantic network. Each object and relation in a problem situation becomes a node in the semantic network. In addition, the network stores nodes representing instances of architecturally defined concepts, such as problem situations and operators. Items are never deleted from long-term memory, and memories are never stored in an abstract form. Rather, the semantic memory stores all the specific problem situations that it encounters. Situations become linked together in memory when they share objects, relations, or object types. If a particular concept from a problem situation already exists in memory, EUREKA increases the trace strengths of the links from the concept, rather than adding a new copy of the concept.

When EUREKA is working on a particular problem situation, it must select an operator to apply to the problem. To this end, EUREKA retrieves a subset of the stored problem situations from long-term memory. This small set of candidate sources is further elaborated and evaluated, to see which would provide the best candidate analogy for the current problem situation. EUREKA chooses one candidate stochastically, based on the evaluation score, and identifies the operator associated with that source analogy. Finally, the system creates a goal to apply to the newly mapped operator to the current state.

EUREKA proceeds in this manner until it solves the problem or the current solution path fails (by exceeding a time limit or detecting a cycle in the solution path). Upon failure, EUREKA does not have the luxury of backtracking, which would allow the system to search the problem space systematically and possibly exhaustively. Rather, EUREKA begins the problem anew from the initial problem situation. The inability to backtrack systematically greatly hinders the system's ability to solve problems, but we feel that this is a psychologically plausible limitation. The limitation also places further importance on effective learning.

The combination of EUREKA's learning mechanisms and its stochastic selection process encourage the system to explore alternative solution paths on subseqent attempts to solve a problem. However, there is no guarantee that a previous search will not be duplicated. If the system fails to find a solution after a preset number of attempts (50 in our experiments), it abandons the problem completely.

Analogical retrieval in EUREKA

EUREKA's analogical reasoner incorporates two stages. The first retrieves a set of candidate source problem situations from memory. The second involves a relatively expensive computation to elaborate the mapping between each candidate source and the target problem situation. Because the elaboration process is so expensive, it is important that the cheaper retrieval process return a relatively small set of candidates. However, the system must also do what it can to make sure it does not miss good candidates in memory. In EUREKA, we have focused on the retrieval phase, to analyze how changes in retrieval patterns can lead to higher-level changes in problem solving.

As mentioned above, EUREKA stores in its semantic network an episodic memory of every problem situation it encounters. Retrieval is implemented as a spreading-activation process similar to that found in ACT (e.g., Anderson, 1976). Each node in the representation of the target problem situation becomes a source of activation, which then spreads to other nodes according to the strengths of the links to those nodes. The activation algorithm appears in Table 1.

After the spread of activation terminates, EUREKA checks the "top-level" node for each problem situation stored in

Table 1: EUREKA's algorithm for spreading activation.

```
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 < 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 has the same type as Link
     Then Increase Total
           by X.trace_strength;
  Return Value *
          (Link.trace_strength / Total)
```

memory. This node contains a unique name for the problem situation and has links to all the nodes representing relations in the problem situation. The problem situation whose top-level node has the highest level of activation becomes a candidate source. In addition, any other problem situation becomes a candidate source if its top-level node has at least one percent of the level of activation of the strongest source.

Learning in EUREKA

As Table 1 indicates, the activation that spreads from a source node, i, to another node, j, depends on the number of nodes to which i is linked (because activation is divided among links of the same type), as well as the strength of the link between i and j. This highlights an important aspect of the retrieval process within EUREKA. That is, the spread of activation (and, therefore, patterns of retrieval) can change for primarily two reasons: new nodes and links being added to memory, and changes in link strengths. It follows that these are the two ways that learning can change behavior in EUREKA.

Thus, one way in which EUREKA learns is simply by adding new experiences by rote into memory. Because spreading activation is a competitive process, introducing more competitors into memory can change what gets retrieved in the future. However, simply adding experiences to memory will not necessarily *improve* problem-solving behavior, which is what we really want. Therefore, EUREKA also changes its behavior by updating link trace strengths. When the system solves a particular target problem situation, it checks which source analog was used to help solve the problem. The system then strengthens the links between the target problem situation and the successfully applied analogical situation. Note that a *problem* presented to EUREKA generally involves a set of *problem situations*, so EUREKA can learn about solved problem situations, even if the attempt to solve the global problem fails. In the long run, stored problem situations that help solve new problem situations become strongly connected to the problem situations that they help solve. Thus, they "soak up" more activation from future problem situations, and become more easily retrieved.

It is also worthwhile to note that EUREKA requires its learning mechanism to be noise tolerant. Because operator selection is based on the current structure of memory and the system cannot systematically backtrack, a search path that leads to failure now may turn into a successful path later. EUREKA might fail simply because it does not "remember" or retrieve the appropriate operator in a particular situation. As the system gathers experience, it may learn to retrieve such operators, turning bad search paths into good ones.

Qualitative behaviors exhibited by EUREKA

Table 2 presents VanLehn's (1989) list of a number of robust qualitative results that have been observed in humans learning to solve problems. EUREKA addresses these issues to varying degrees. Jones and Langley (1994; Jones, 1989) present a number of detailed experiments with EUREKA that address these results. Due to a lack of space, we will not present the details here so we can discuss other issues. EUREKA's retrieval and learning methods directly account for most of the behaviors identified by VanLehn. Behaviors 1 and 3 require a bit of extra interpretation, and are not modeled as well as the others. In general, the results indicate that these types of learning can indeed arise from rather low-level processes.

Table 2: Robust learning behaviors identified in human problem solvers (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.

There are other models of analogical problem solving (e.g., Hammond, 1986; Veloso & Carbonell, 1993), which rely on indexing methods for analogical retrieval, and generally focus on learning and reasoning at a higher architectural level than EUREKA. In contrast, there are other analogical mechanisms that share EUREKA's spirit in modeling analogical reasoning as a relatively low-level memory process. The following section discusses two of the more well-known models of this type.

Learning to solve problems with ARCS and MAC/FAC

ARCS (Thagard et al., 1990) and MAC/FAC (Gentner & Forbus, 1991) are the analogical retrieval algorithms associated with two relatively well-known and sophisticated systems for analogical elaboration and evaluation: ACME (Holyoak & Thagard, 1989) and SME (Falkenhainer, Forbus, & Gentner, 1989). It is attractive to consider incorporating these systems into a learning, problem-solving architecture such as EUREKA. We feel such an attempt could benefit research on both sides. On the one hand, EUREKA's analogical mechanisms have been used to model human learning in problem solving, but it is questionable whether they can model some of the psychological findings on analogical retrieval and evaluation (e.g., Gick & Holyoak, 1983). On the other hand, ARCS/ACME and MAC/FAC have both been demonstrated on the retrieval and evaluation results, but they have so far been used to model analogical retrieval in relative isolation from other tasks. Although both systems have built-in notions of what makes a good analogy, it is sometimes difficult to judge why other potential analogs or mappings might not be better in particular situations. A problem solver provides a context by which to judge the quality of analogies more objectively: A good analogy is one that helps solve a problem. In addition, our work with EUREKA has focused on how analogical reasoning can adapt with experience, but the two other systems have so far not incorporated mechanisms for changing their behavior over time. The remainder of this paper discusses the issues we foresee in incorporating the ARCS and MAC/FAC retrieval mechanisms into a problem-solving system that learns. First, let us provide a quick overview of the ARCS and MAC/FAC retrieval methods.

Retrieval with ARCS

ARCS divides the retrieval process into two stages, beginning with a table look-up for each concept in the target. This table provides a list of all the concepts in memory that are immediately related to a concept (e.g., by subordinate, superordinate, or part-of relationships). ARCS then considers retrieving any source that includes at least one of the collection of concepts related to the target. This happens by creating a constraint network of possible concept matches, linked together by excitatory and inhibitory links. ARCS only sets up match hypotheses between semantically similar concepts (from the look-up table). More complete match hypotheses are saved for the more expensive ACME matcher. In addition, special nodes are created to link concepts that have been marked as important in various ways. Finally, activation spreads throughout the network until the network settles. Each candidate source receives a retrieval score, computed from the activation of the concepts in the source. It is not clear, however, that the ARCS system makes a distinction between candidate sources that "are retrieved" vs. those that are not.

Retrieval with MAC/FAC

MAC/FAC[1] takes quite a different approach to retrieval. MAC computes a *content vector* for each source stored in memory. The content vector ignores concepts that represent simple objects, and records the number of occurrences of each concept that can take other concepts as arguments (e.g., relations and functions). This requires MAC to know the entire space of such concepts ahead of time. MAC then similarly computes a content vector for the target of the analogy. The retrieval score for each potential source is computed as the dot product of the source's content vector with the target's content vector. This gives an estimate of the degree to which relations are shared between the target and each source, and it is very quick to compute. The source with the largest dot product is marked as a retrieved candidate. In addition, any other source with a dot-product value of at least 10% of the highest value is retrieved.

Changes in retrieval as knowledge increases

Having a feel for how ARCS and MAC/FAC work, let us turn our attention to how their behavior might adapt with experience. We have stressed that our primary focus in EUREKA is on how learning can change retrieval patterns, leading to larger changes in problem-solving behavior. Thus, it is most important for us to examine the types of events that allow EUREKA's retrieval mechanism to learn, and how they would apply to MAC/FAC or ARCS. Let us first consider how the mere storage of new experiences can influence retrieval. There are two aspects of performance to examine when the knowledge base increases in size. First, new knowledge may change the time it takes for the retrieval algorithm to execute. Second, the resulting set of candidate sources may change as new potential sources are added to memory.

Taking the first issue, EUREKA's spreading-activation mechanism performs a limited search through memory, and many portions of memory (those distant from the target concepts) will be completely ignored by the retrieval process. Jones (1989, 1993) has demonstrated empirically and analytically that EUREKA's form of spreading activation takes a constant amount of time relative to the size of memory, even when implemented as a serial algorithm. On the other hand, the specific representation or "shape" of memory can sometimes have a significant impact on retrieval time. This is an important issue, related to the utility problem (Minton, 1988) in machine learning. It would not be desirable for a system to slow down merely because its memory is growing. However, both ARCS and MAC/FAC are guaranteed to take longer as new analogical sources are added to memory, because they both examine every potential source as part of the retrieval process. This means that processing is at least linearly related to the number of potential source analogs in memory. ARCS includes an even more expensive construction of the constraint network, which depends on the number of sources that include concepts similar to those in the target. Thagard et al. (1990) propose, however, that much of their algorithm can execute in parallel, so time will be constant if we assume an arbitrary number of processors (one per potential source stored in memory). Presumably the same is true of MAC/FAC.

Let us next consider how the mere addition of cases to the knowledge base can change retrieval patterns. New problem situations in EUREKA's memory imply new competitors for activation. Thus, if a newly stored situation shares concepts with other problem situations, activation levels necessarily

[1] Note that MAC corresponds to the analogical retrieval mechanism, while FAC is the more expensive elaboration and evaluation algorithm.

change. In contrast, MAC/FAC independently associates a content vector and retrieval score with each stored situation. The retrieval score has absolutely nothing to do with other stored cases. Thus, adding new cases will have a limited impact on the set of retrieved candidate sources. Because the retrieval threshold is based on a percentage of the highest-valued retrieval item, a newly stored problem situation can only significantly change retrieval patterns in the cases where it receives the highest retrieval score.

ARCS uses an activation process that is inherently competitive like Eureka's. Thus, the final retrieval values for each candidate source can certainly change as memory grows. Law, Forbus, and Gentner (1994) showed that increasing the number of cases can be quite detrimental to ARCS, but did not adversely affect MAC/FAC's behavior. Presumably, similar detrimental effects would be seen in EUREKA's retrieval mechanism as cases begin to compete with each other. On the other hand, EUREKA (and most likely ARCS) can also *benefit* from the addition of appropriate knowledge. MAC/FAC can only benefit in the sense that it may have a new case to retrieve, but it cannot benefit in any competitive sense. Thus, we interpret Law, Forbus, and Gentner's result not as a condemnation of competitive retrieval algorithms, but as further evidence of the importance of associating a learning method with retrieval.

Tuning retrieval with experience

The second aspect of learning has to do with tuning the retrieval process to improve and focus itself with experience. Again, EUREKA achieves this by increasing link trace strengths associated with stored problem situations when they aid in the solution of new problem situations. With appropriate experiences, the system will eventually retrieve fewer items, but they will have higher estimated quality. None of the presentations of ARCS or MAC/FAC have addressed the issue of learning. Thus, rather than comparing learning mechanisms, we are free to hypothesize the types of learning mechanisms that might be amenable to ARCS and MAC/FAC.

ARCS has a competitive activation-based retrieval mechanism, so it is tempting to assume that it would benefit from a learning mechanism similar to EUREKA's. However, it is important to note that activation spreads through a very different type of network in each case. EUREKA's semantic network is a long-term structure encoding the representation of problem situations and the relations and objects they share. In contrast, the constraint networks in ARCS are constructed anew each time a target is presented, and represent potential ways to match the concepts in each source to the concepts in the target. Despite these differences, there still seems to be some potential to altering link strengths in ARCS. Some link strengths are fixed measures of the degree of similarity between differently related concepts (e.g., synonyms have a similarity value of 0.6, superordinates a value of 0.3, and subordinates a value of 0.2). There does not seem to be any reason in principle that these similarity measures could not be learned for specific concept pairs, rather than fixed by these abstract types. This is an attractive option, because it would allow a more pragmatic view of similarity that adapts to problem-solving experience, rather than requiring a fixed table of similarities to be provided to the system. In addition, ARCS gives extra strength to concepts that are marked as "important" and mappings that are marked as "presumed". These marks are specified and fixed before retrieval begins. One of the benefits of a performance task, such as problem solving, is that it is possible to induce important concepts over time. Perhaps such an induction algorithm could use experience to adapt the measure of "importance" of concepts over time.

MAC/FAC is another story entirely, because it does not share the notion of competition between candidate analogical sources. As we have mentioned above, retrieval computations are independent for each target-source pair. However, the one thing that *is* common across candidate sources is the algorithm for computing the content vector. When constructing the vector, MAC/FAC counts each occurrence of every feature. These counts could instead be weighted by parameters for each feature, which would be tuned with experience.

There is a potentially more interesting alternative for learning. It almost seems that there is a built-in assumption to MAC/FAC: changes in analogical retrieval should only arise through a reformulation of the representation of the sources and targets. It is not clear whether the creators of MAC/FAC intend this to be a fixed, architectural constraint, but it is certainly interesting to view it that way. In this case, the only way for MAC/FAC to tune its retrieval patterns would be for it to change the representation of its stored cases. These changes would be based on knowledge of how the retrieval scoring mechanism works, and would be designed to award useful candidate sources higher scores in future similar situations. It is not clear to us what the details of such a mechanism would be. However, it would provide a pragmatic approach to changing representation within a cognitive agent. The agent would change its representations in response to problem-solving success (or failure), and would change them in such a way as to improve future behavior.

Summary

EUREKA provides a model of analogical retrieval in problem solving. In addition, it incorporates a learning mechanism that focuses on tuning the retrieval of candidate analogical sources from memory. These relatively low-level mechanisms give rise to larger qualitative changes in problem-solving behavior. We have used the EUREKA model to explain the primary learning effects that have been identified in human problem solvers. Because the system includes a memory-based mechanism for the retrieval of analogies, it is natural to compare this mechanism to ARCS and MAC/FAC, two well-known retrieval mechanisms in the cognitive science literature. We have used the lessons learned from building EUREKA to guide our analysis of how ARCS and MAC/FAC would fit into the context of a performance task (problem solving) where learning can and should take place. EUREKA demonstrates that low-level mechanisms can have a significant impact on high-level behavior. It will be interesting to see what qualitative differences arise in learning problem solvers that incorporate the ARCS or MAC/FAC algorithms.

References

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

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

Falkenhainer, B., Forbus, K., & Gentner, D. (1989). The Structure-Mapping Engine: Algorithm and examples. *Artificial Intelligence, 41*, 1–63.

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.

Gentner, D., & Forbus, K. (1991). MAC/FAC: A model of similarity-based retrieval. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Lawrence Erlbaum.

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

Hammond, K. J. (1986). *Case-based planning: An integrated theory of planning, learning, and memory*. Doctoral dissertation. Yale University.

Holyoak, K., & Thagard, P. (1989). Analogical mapping by constraint satisfaction. *Cognitive Psychology, 15*, 1–38.

Jones, R. M. (1989). *A model of retrieval in problem solving*. Doctoral dissertation. Department of Information and Computer Science, University of California, Irvine.

Jones, R. M. (1993). Problem solving via analogical retrieval and analogical search control. In S. Chipman & A. Meyrowitz (Eds.), *Machine learning: Induction, analogy, and discovery*. Boston: Kluwer Academic.

Jones, R. M., & Langley, P. (1994). *Learning and problem solving with limited memory*. Manuscript in preparation.

Jones, R. M., & VanLehn, K. (1994). Acquisition of children's addition strategies: A model of impasse-free, knowledge-level learning. *Machine Learning, 16*, 11–36.

Langley, P., & Allen, J. A. (1991). The acquisition of human planning expertise. In L. A. Birnbaum & G. C. Collins (Eds.), *Machine Learning: Proceedings of the Eighth International Workshop*. Los Altos, CA: Morgan Kaufmann.

Law, K., Forbus, K. D., & Gentner, D. (1994). Simulating similarity-based retrieval: A comparison of ARCS and MAC/FAC. In A. Ram & K. Eiselt (Eds.), *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Lawrence Erlbaum.

Minton, S. (1988). *Learning effective search control knowledge: An explanation-based approach.* Boston: Kluwer Academic.

Thagard, P., Holyoak, K., Nelson, G., & Gochfeld, D. (1990). Analogical retrieval by constraint satisfaction. *Artificial Intelligence, 46*, 259–310.

VanLehn, K. (1989). Problem solving and cognitive skill acquisition. In M. I. Posner (Ed.), *Foundations of cognitive science*. Cambridge, MA: MIT Press.

Veloso, M. M., & Carbonell, J. G. (1993). Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. *Machine Learning, 10*, 249–278.