A Computational Model of Analogical Problem Solving

Jaime G. Carbonell

Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

This paper outlines a theory of analogical reasoning based on a process-model of *problem solving by analogy* and the hypothesis that problem solving and learning are inalienable, concurrent processes in the human cognitive system. The analogical problem solver exploits prior experience in solving similar problems, and, in the process, augments a hierarchically-structured epsiodic long term memory. An analogical transformation process is developed based on a modified version of Means-Ends Analysis in order to map past solutions from similar problems into solutions satisfying the requirements of the new problem.[1]

1. Introduction

Analogical reasoning has been a sparsely-investigated phenomenon in Artificial Intelligence [7, 12, 8, 20]. Nonetheless, analogy promises to be a central inference method in human cognition as well as a powerful computational mechanism. In this paper, I discuss a computational model of problem solving by analogy based on an extension of means-ends analysis (MEA). My central hypothesis (based in part on Schank's theory of memory organization [18, 17]) is the following: When encountering a new problem situation, a person is reminded of past situations that bear strong similarities (at different levels of abstraction) to the present situation. This type of reminding experience serves to retrieve behaviors that were appropriate in earlier situations, whereupon past behavior is adapted to meet the demands of the current situation.

Commonalities among previous and current situations, as well as successful applications of modified plans can serve as the basis for generalization. Similarly, performing an inappropriate behavior in a new situation can lead to discrimination (among the ways in which situations are organized in memory and/or the mechanisms that adapted an existing plan to the new situation). However, a *reactive environment* that informs the problem solver of success, failure, or partial success is an absolute requirement for any generalization or discrimination process to apply. I consider problem solving by analogy and experiential learning to be inalienable components of a unified cognitive model. Analogical problem solving exploits knowledge of plans indexed under situations similar to the current one, generating new purposive behavior potentially relevant to future problem solving. And, the learning component creates the memory structures to encode experiential knowledge (generalizing where appropriate) that enable the problem solver to retrieve and compare relevant situations and plans from memory. The bulk of this paper focuses on developing a computationally-effective mechanism for problem solving by analogy. A detailed discussion of experiential learning processes is beyond the scope of this presentation (but see [3]).

2. Problem Solving by Analogy

Traditional AI models of problem solving (e.g., GPS [13], STRIPS [6], and NOAH [15]) approach every problem almost without benefit of prior experience in solving other problems in the same or similar problem spaces. Hence, GPS will perform the same Means-Ends Analysis (MEA) process to solve the monkey-and-bananas problem (where a monkey needs to place a box underneath some bananas suspended from the ceiling in order to climb on the box, reach the bananas and eat them) and to solve the experimenter-and-bananas problem (where the experimenter must move the same box enabling him to reach the hook in the ceiling where he will hang the bananas -- then he must move the box away to create a problem for the monkey). However, an intelligent monkey observing the experimenter hang the bananas, may directly conclude which parts of the experimenter's behavior he should replicate in order to reach the bananas. Similarly, an experimenter who becomes hungry after watching an unenlightened monkey repeatedly fail in his attempts to reach the bananas, should know how to reach the bananas himself without planning from ground zero.

Clearly, the bulk of human problem solving takes place in problem spaces that are either well known or vary only slightly from familiar situations. It is rare for a person to encounter a problem that does not remind him of potentially applicable solutions to similar problems solved or observed in past experience. New puzzles (such as Rubik's magic cube) are such exceptional problems, where initially the only tractable solution procedure is the application of weak methods [13] without benefit of (non-existent) past experience. Therefore, my investigations center on simplified versions of real-world problems, rather than more abstract mathematical puzzles.

Now, let us turn to problem solving in familiar problem spaces. What makes a problem space "familiar"? Clearly, knowledge of solved problems in that space is a major aspect. This knowledge must have been acquired, and must be brought to bear in the problem solving process. There is no other way to account for the fact that humans solve problems in familiar situations much faster, and with more self-assurance. A computer model should exploit the same skill-acquisition process; i.e., it should learn to adapt its problem-solving behavior to known problem spaces -- falling back on the application of weak methods when more direct recall-and-modification of existing solutions fails to provide an answer. How might a problem solver be augmented to exhibit such adaptive

[1] This research was sponsored in part by the Office of Naval Research (ONR) under grant number N00014-79-C-0661. The author would like to thank Allen Newell, David Klahr and Monica Lam for their useful suggestions in various discussions.

behavior? Below, I illustrate a tractable mechanism for problem solving by analogy.

2.1. The Plan-Transformation Problem Space

First, consider a traditional Means-Ends Analysis (MEA) problem space [13], populated by:

- A set of possible problem states.
- One state designated as the *Initial State*
- One or more state(s) designated as *goal states* -- for simplicity, assume there is only one goal state.
- A set of operators with known preconditions that transform one state into another state in the space.
- A difference function that computes differences between two states (typically applied to compute the difference between the current state and the goal state).
- A method for indexing operators as a function of the difference(s) they reduce (e.g., the table of differences in GPS).
- A set of global path constraints that must be satisfied in order for a solution to be viable. (E.g., a path constraint may disallow particular subsequences of operators, or prevent an operator that consumes K amount of a resource from applying more than N times, if there is only NxK amount of the resource available to the problem solver.)[2]

Problem solving in this space consists of standard MEA: Compare the current state to the goal state; choose an operator that reduces the difference; apply the operator if possible -- if not recurse on the subgoal(s) of establishing the unsatisfied precondition(s) of that operator.

How can we exploit knowledge of solutions to previous problems in this problem space? First, consider the simplest case; knowledge consists only of solutions to previous problems. Each solution consists of a sequence of operators and intermediate states, including the initial and final states, together with the path constraints that the solution was designed to satisfy. One rather simple idea is to create "macro-operators" from sequences and sub-sequences of atomic operators that have proven useful as solutions to earlier problems. For instance, STRIPS with MACROPS exploited this idea [6] using its "triangle table" to store all partial sequences of operators encountered in a solution to a previous problem. However, the simple creation of macro-operators suffers two serious shortcomings. First, the combinatorics involved in storing and searching all possible subsequences of all solutions ever encountered becomes rapidly unmanageable. Searching for applicable macro-operators can become a more costly process than applying MEA to the original problem. Second, path constraints are ignored in this process. If the new problem must satisfy a different set of path constraints, most previous macro-operators may prove invalid. Therefore, let us think not in terms of creating more and more powerful operators that apply to fewer and fewer situations, but rather think in terms of gradually transforming an existing solution into one that satisfies the requirements of the new problem.

Consider a reminding process (a search for solutions to problems similar to the one at hand) that compares differences among the following:

1. The initial state of the new problem and the initial state of previously-solved problems
2. The final state of the new problem and the final state of previously-solved problems
3. The path constraints under which the new problem must be solved and path constraints present when previous similar problems were solved.
4. The differential applicability of the retrieved operator sequence in the old and the new problem situations.

The difference function used in comparing initial and final states may be the very same function used for difference reduction in standard MEA. Here, I advocate using the difference function as a *similarity metric* to retrieve the solution of a previously-solved problem closely resembling the present problem. The difference function applied to path constraints is a generalization of the problem-state difference function, as it must address operator-sequence differences in addition to state information. Hence, *reminding* in our problem-solving context consists of recalling a previously solved problem whose solution may transfer to the new problem under consideration. A more sophisticated method of computing similarities among episodic memory structures is based on a *relative-invariance hierarchy* among different components of recalled problem solutions, as discussed in [4].

Reminding is only the first phase in analogical problem solving. The second phase consists of transforming the old solution sequence into one satisfying the criteria for the new problem. How does this transformation process proceed? I submit that it is equivalent to problem solving in the space of solutions.[3]

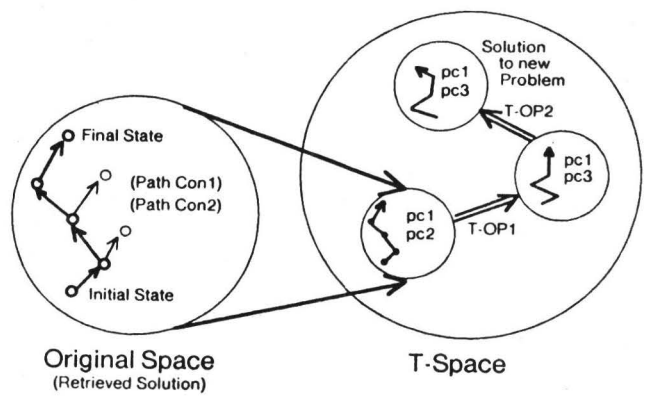

Figure 2-1: A solution path in the original problem space becomes a state in the *analogy transform problem space*.

[2] The introduction of path constraints in this manner constitutes a slight modification of the standard MEA problem space.

[3] Here I apply my previous definition of a solution to be a sequence of operators and intermediate states together with the set of path constraints that sequence is known to satisfy. Thus, I advocate applying MEA to the space of potential solution sequences rather than the original problem space. However, the reminding process should generate an initial solution sequence close to the goal solution sequence, where closeness is determined by the difference metric above.

Finding an appropriate analogical transformation is itself a problem solving process, but in a different problem space. The states of the transform problem space are solutions to problems in the original problem space, where the initial state is the retrieved solution to a similar problem, and the goal state is a solution satisfying the criteria for the new problem. The operators in the transform problem space are the atomic components of all solution transformations (e.g., substitute an operator in the solution sequence for another operator that reduces the same difference, but requires a different set of preconditions or entails different side effects, etc. -- see below). The differences that the problem solver attempts to reduce in the new problem space are precisely those computed by the similarity metric in the reminding process. In other words, progress towards a goal is determined by transitions in the solution space towards "solution sequences" corresponding to problems increasingly similar to the original problem. Intermediate states in the transform space need not correspond to *viable* solutions in the original (object) space, in that intermediate solution sequences may not be executable due to unsatisfied operator preconditions. The diagram in figure 2-1 gives an intuitive flavor of this problem-solving process. More precisely, the analogy transform problem space (T-space) is defined as follows:

- States in the transform space are potential solutions to problems in the original problem space (i.e., sequences of states and operators including the initial and final states, plus the path constraints under which those solutions were computed.)
- The initial state in the transform space is the solution to a similar problem retrieved by the reminding process.
- A goal state in the transform space is the specification of a solution that solves the new problem, satisfying its path constraints.
- An operator in the transform space (labeled a "T-operator" to avoid confusion) maps one solution sequence into another potential solution sequence in the untransformed problem space. The following is a partial list of useful T-operators:
  - ***General Insertion***. Insert a new operator into the solution sequence.
  - ***General deletion***. Delete an operator from the solution sequence.
  - ***Subsequence Splicing***. Splice a solution to a new subproblem into the larger established solution sequence. This T-operator is useful in the following situation: If an operator in the original problem sequence cannot be applied under the new problem specification because one of its preconditions is not satisfied, problem solve in the original (object) space to find a means of satisfying this precondition. If successful, splice the precondition-fulfilling subsequence into the original solution sequence.
  - ***Subgoal-preserving substitution***. Substitute an operator in the original solution sequence by another operator (or sequence of operators) that reduces the same difference. This T-operator is particularly useful if either a precondition of an operator in the original sequence cannot be satisfied, or if the presence of a particular operator in the solution sequence violates a path constraint.[4]
  - ***Final-segment concatenation.*** Treat the solution sequence as a macro-operator in the original problem space and apply MEA to reduce the difference between the old final state and the new final state. If successful, concatenate the solution to this subproblem at the end of the original solution sequence.
  - ***Initial-segment concatenation.*** Apply the process above to find a path in the original problem space from the new initial state to the old initial state. If successful, concatenate at the beginning of the original solution. [Note that in this case we *start* with the initial state for the new problem and seek a path to the initial state for the retrieved solution, whereas in the final segment-concatenation operator the inverse process applies.]
  - ***Sequence meshing.*** If two solutions to previous problems were retrieved in the reminding process, each solution differing from the new problem specification in non-identical aspects, merge their operator sequences. The resultant solution sequence should differ from a solution to the new problem by the intersection of the differences between each retrieved solution and the new problem specification.[5] If the differences between the two retrieved solutions and the new problem specification form disjoint sets, sequence meshing yields a complete solution.
  - ***Operator reordering***. Often a path constraint in the new problem specification can be met by simple reordering of operators (when allowed by their preconditions) in the retrieved solution.
  - ***Parameter substitution***. The objects to which operators were applied in the retrieved solution may be substituted for objects in the new problem (that do not violate the preconditions of the operators).
  - ***Solution-sequence truncation***. If the final state of an operator subsequence of the retrieved solution exhibits a smaller difference with the new problem specification, use this subsequence as the new basis for mapping into the desired solution sequence.
- The difference function in the transform space (D_T) is a combination of the difference measures between initial states (of the retrieved and desired solution sequences), final states, path constraints, and degree of applicability of the retrieved solution in the new problem scenario. Hence, the values of D_T are 4-vectors, with the interpretation that all four component differences must be reduced (independently or jointly) in the transform space (T-space) problem-solving process.

$$D_T = \langle D_O(S_{I,1},S_{I,2}),\ D_O(S_{F,1},S_{F,2}),\ D_P(PC_1,PC_2),\ D_A(SOL_1,SOL_2)\rangle$$

D_O is the difference function between states in the original space. D_P computes differences between path constraints (PC's). D_A measures the applicability of the old solution in the new scenario by determining the fraction of operators in the initial solution sequence (SOL_1) whose preconditions are not satisfied under the new problem specification. S_I denotes an initial state, and S_F denotes a final state. The subscript 1 indexes the retrieved solution, and 2 indexes the specifications on the desired solution to the new problem. D_T is reduced when any of its four components is independently reduced.

[4]Note that a subgoal-preserving substitution is much more restrictive than a general delete T-operator followed by a general insert T-operator. Therefore, this T-operator is more apt to yield useful transformations, a fact reflected in the ordering of operators under each appropriate entry in the difference table.

[5]Merging two partial operator sequences is an interesting and potentially complex problem in itself. Procedural networks, developed in the NOAH system [15], facilitate computations of operator interactions when meshing two plans. It is not always the case that two partial solution sequences can be merged effectively (e.g., each subsequence may violate necessary preconditions for the other subsequence). Non-algorithmic T-operators, such as ***sequence meshing***, define their own internal problem space.

The problem-solving process in T-space succeeds when D_T = <NIL, NIL, NIL, NIL>. Interesting search problems occur when, in order to reduce one component in the difference vector, one or both of the other components must be increased. For example, the insertion of new operators into the solution sequence may have the unfortunate side-effect of violating an established precondition of an operator in the original sequence. In this case reducing $D_O(I)$ or $D_O(F)$ results in increasing D_A. Our first-pass solution is to define a (linear) combination of the four components and choose the operator that maximally reduces this value, backtracking when necessary. Fortunately, it is often the case that differences in the 4-vector can be reduced in a componentwise-independent manner. Moreover, a modified version of the Δ-MIN method [2] may apply, focusing the backtracking process when backtracking proves necessary.

- A difference table for indexing the T-operators is needed. Entries in the difference table take the form **"To reduce <DIFFERENCE>, apply a member of <T-OPERATOR-SET>"**. The operators in the applicable set are usually ordered as a function of the heuristic measure of their utility in reducing the given difference. A sample difference table entry would be:

  - If the preconditions to an operator in SOL_1 are not satisfied (i.e. D_A is non-null), try (first) ***subgoal-preserving substitution*** on the inapplicable operator, or try ***solution-sequence splicing*** to satisfy the violated preconditions.

- There are no path constraints in the transform space. Since we are mapping from one solution sequence to another, the intermediate states and T-operators do not necessarily correspond to actual operations performed on an external world, and therefore are not subject to its restrictions. This simplification is offset by the more complex difference metric in T-space.

2.2. An example

Reconsider the monkey-and-bananas and experimenter-and-bananas problem, in light of the analogical problem-solving model.

A monkey watches a behavioral psychologist (i.e., the experimenter) pick up a wooden box and place it under a hook in the ceiling. Next, the experimenter climbs on the box, places some bananas on the hook, climbs off the box, and returns the box to its original location. Then, the experimenter releases the (hungry) monkey and leaves the room. How does the monkey plan to reach the bananas? Can he benefit from having observed the experimenter?

As we mentioned earlier, a "smart monkey" ought to learn from his observations of the experimenter. Let us see how analogical problem solving applies here. For simplicity, assume the monkey does not have prior experience solving similar problems beyond his recent observation of the experimenter. The monkey's problem is: **initial state** = monkey on the floor, bananas on the ceiling, box in the room; **final state** = monkey in possession of the bananas; **path constraints** = physical abilities of the monkey. However, the solution to the experimenter's problem cannot be applied directly. (His problem was **initial state** = possession of the bananas, box in the room, experimenter on the floor; **final state** = Bananas on the ceiling, box *not* under the bananas; **path constraints** = physical abilities of the experimenter.)

Assuming the path constraints match, the differences between the initial states (and the differences between the final states) are so large as to preclude any reasonable attempt at direct analogical transformation. Therefore, the monkey must resort to standard MEA (in the object problem space). He selects the operator GET-OBJECT (applied to bananas). This operator suffers an unsatisfied precondition: The monkey cannot reach the bananas. Therefore, the active subgoal becomes: Reach the ceiling where the bananas are located. How may the monkey proceed at this juncture?

The entire problem can, of course, be solved by standard MEA. However, there is a more direct solution method. If the monkey recalls his observation of the experimenter, he may realize that the problem of reaching the ceiling has already been solved (by the experimenter, as a subgoal to placing the bananas there - although the monkey need not understand the experimenter's higher-level goals). The monkey can apply the ***parameter-substitution*** T-operator (substituting "monkey" for "experimenter"), and optionally the ***solution-sequence truncation*** T-operator (eliminating the need to return the box to its original location after having used it). This problem-solving process in T-space results in a plan that the monkey can apply directly to reach the bananas, and thus achieve his original goal of having (and presumably eating) the bananas.

The significant aspect of the experimenter-monkey-and-bananas example is that standard MEA and T-space MEA were combined into a uniform problem-solving process where standard MEA calls on analogical problem solving to solve a subproblem more directly. The converse process is also possible, and potentially significant. Hence, *Analogical reasoning adds a powerful dimension to standard problem solving when prior experience can be brought to bear, but remains largely unobstrusive when no relevant prior knowledge suggests itself.*

Additionally, it would be useful for the problem solver to remember his observations and problem-solving experiences to use as a basis for future analogical reasoning. These could be remembered directly or abstracted into episodic traces, much like Schank and Abelson's scripts [16, 5], and hierarchically organized as a function of the goals they fulfill.

3. Evaluating the Analogical Reasoning Process

In an informal experiment, not meant to withstand statistical significance tests, I gave the following problem to five undergraduate history-and-art students:

Prove that the product of two even numbers is even.

Somewhat to my surprise, none of the five was able to solve this simple algebraic problem, although all five made serious attempts. Later I explained the solution carefully enough to insure that all five understood it:

First, recall the definition of an even number: a number that is divisible by 2.

Second, write down an expression that represents an even number: You may write "2N" where N is any integer, to represent a number divisible by 2.

Next, multiply two even numbers, writing: 2N x 2M, where M is also any integer. Multiplying we get 4NM.

Now, recall the representation of an even number: 2 x any integer. Therefore you can write 4NM = 2 x 2NM, which by closure of integers under multiplication matches the representation of an even number. Hence, the product of two even numbers is even.

At this point, all five students claimed they understood the proof, and moreover expressed some feeling of embarrassment for not having derived such an "obvious" proof themselves. Then, I suggested they try the following problem:

Prove that the product of two odd numbers is odd.

With a grim determination to redeem their previous poor performance all five attempted the problem and three of them succeeded. Briefly:

Odd numbers can be represented as "even + 1" = 2N + 1 for any integer N.

The product, therefore: (2N + 1) x (2M + 1) = 4NM + 2N + 2M + 1 = 2(2NM + N + M) + 1, which is the representation of an odd number.[6]

This informal experiement strongly indicates that the second problem was solved by analogy from the solution of the first problem. The scratch papers collected from the students suggest direct attempts at transfering and modifying steps of the first solution. The insertion of an extra algebraic step[7] illustrates an application of the ***subsequence splicing*** T-operator. Moreover, the mere fact that three of five students were able to solve a problem more complex that the one where all five failed previously, argues very convincingly for an analogical process exploiting the previous solution (or some abstraction thereof). However, It should be noted that this type of experiment does not in itself demonstrate dominance of analogical reasoning in human problem solving, but rather it provides strong evidence for the existence of analogical processes in cognitive activities. Demonstrating the conjecture that analogy is the central inference mechanism for human problem solving would require a much more thorough (and perhaps more controlled) set of psychological observations.

As a test of the computational feasibility of the analogical problem solving process, a simple version of MEA was programmed to operate on the transform space, and given a subset of the T-operators with a corresponding difference table. It solved the product-of-two-odds problem starting from the solution for two even numbers.[8] The initial program is not too interesting -- it demonstrates that the analogical problem solving process "actually works", but does little else. The truly interesting issues will arise when:

- a much fuller implementation is available allowing comparisons among different problem solving methods over a representative corpus of problems,
- issues of learning from experience are further investigated,
- and the analogical problem solver is integrated with a dynamically-changing long term memory.

[6] Interestingly, one student chose to represent odd numbers as 2N + 3, which is correct but requires a bit of additional algebraic manipulation. Of the two students who did not present an adequate proof, one erred in an algebraic manipulation step, the other was unable to represent odd numbers correctly.

[7] I.e., distributing the product of the two odd numbers is required to fulfill a precondition of the factoring-a-constant operator that factors 2 from three of the four terms in: 4NM + 2N + 2M + 1.

[8] It used N-1 to represent an odd number, since the SUB1 operator was inadvertently listed before ADD1 in the object-space difference table, and therefore had to splice in an additional algebraic step in the solution.

4. Learning as Part of the Problem Solving Process

Let us examine briefly the learning processes inherent in analogical problem solving. My central hypothesis is that human learning occurs as an integral, inalienable aspect of problem solving and comprehension. This does not imply that disembodied concept acquisition and pure grammatical inference are ineffective processes. Such processes are computationally valid but psychologically implausible. Since I am primarily interested in cognitive modeling, I find the creation of a computational model that incorporates both learning and performance as inseparable processes to be an intellectually appealing endeavor. Analogical reasoning provides the basis for such an integrated model. The fact that humans learn in real-world tasks (by practice, observation, trial and error, etc.) with little or no discernible cognitive effort lends credence to my integration-of-learning-and-performance hypothesis. Moreover, there is evidence that for cognitive as well as physical motor-coordination tasks humans *cannot help but learn* in the process of repeated practice [14]. This clearly suggests that at least some forms of human learning cannot be separated from cognitive-performance or motor-coordination systems.

Learning can occur in both phases of analogical problem solving: 1) the reminding process that organizes and searches past experience, and 2) the analogical transformation process itself. The first phase is discussed briefly below, and at greater length by Schank and Lebowitz [9]. The second phase is analyzed in [3]. Environmental feedback (such as success or failure of the planned solution) can trigger solution analysis, where solutions to several similar problems can be considered exemplars for the task of synthesizing a general plan to deal with future problems of the same general type. Simon and others [1, 19] discuss this form of reducing a learning-by-doing problem to a learning-from-examples task.

4.1. Episodic Memory Organization

Memory of solutions to previous problems, whether observed or directly experienced, must be organized by similarities in goal states, initial states, and means available (or path constraints present). Otherwise, there can be no reasonable reminding process when solving future problems of a similar nature. Hence, a hierarchical indexing structure on an episodic memory must be dynamically constructed and extended as the system gradually accumulates new experience. Thus, continuously expanding and structuring episodic memory is a relatively simple, but absolutely essential, aspect of learning that proceeds concurrent with analogical reasoning.

5. Concluding Remark

The objective of this paper has been to lay a uniform framework for analogical problem solving capable of incorporating skill refinement and acquisition processes. Most work in machine learning does not attempt to integrate learning and problem solving into a unified process. (However, Mitchell [11] and Lenat [10] are partial counter-examples.) Past and present investigations of analogical reasoning have focused on disjoint aspects of the problem. No past investigation has resulted in a unified analogical problem solving method. For instance Winston [20], investigated analogy as powerful mechanism for classifying and structuring episodic descriptions. Kling [7] studied analogy as a means of reducing the set of axioms and formulae that a theorem prover must consider when deriving new proofs to

theorems similar to those encountered previously. In his own words, his system "...derives the analogical relationship between two [given] problems and outputs the *kind of information* that can be usefully employed by a problem-solving system to expedite its search." However, analogy takes no direct part in the problem-solving process itself. Hence, the extension of means-ends analysis to an analogy transform space is, in itself, a new, potentially-significant problem-solving method, independent of the learning mechanisms that it can support.

References

1. Anzai, Y. and Simon, H. A., "The Theory of Learning by Doing," *Psychological Review*, Vol. 86, 1979 , pp. 124-140.

2. Carbonell, J. G., "Δ-MIN: A Search-Control Method for Information-Gathering Problems," *Proceedings of the First AAAI Conference*, August 1980 .

3. Carbonell, J. G., "Learning by Analogy: Skill Acquisition in Reactive Environments," in *Machine Learning*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell, eds., Palo Alto, CA: Tioga Pub. Co., 1981.

4. Carbonell, J. G., "Metaphor Comprehension," in *Knowledge Representation for Language Processing Systems*, W. Lehnert and M. Ringle, eds., New Jersey: Erlbaum, 1981.

5. Cullingford, R., *Script Application: Computer Understanding of Newspaper Stories*, PhD dissertation, Yale University, Sept. 1977.

6. Fikes, R. E. and Nilsson, N. J., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, Vol. 2, 1971 , pp. 189-208.

7. Kling, R. E., "A Paradigm for Reasoning by Analogy," *Artificial Intelligence*, Vol. 2, 1971 , pp. 147-178.

8. Korf, R. E., "Toward a Model of Representation Changes," *Artificial Intelligence*, Vol. 14, No. 1, 1980 , pp. 41-78.

9. Lebowitz, M., *Generalization and Memory in an Integrated Understanding System*, PhD dissertation, Yale University, Oct. 1980.

10. Lenat, D., *AM: Discovery in Mathematics as Heuristic Search*, PhD dissertation, Stanford University, 1977.

11. Mitchell, T. M., Utgoff, P. E. and Banerji, R. B., "Learning Problem-Solving Heuristics by Experimentation," in *Machine Learning*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell, eds., Palo Alto, CA: Tioga Pub. Co., 1981.

12. Moore, J. and Newell, A., "How can MERLIN Understand?," in *Knowledge and Cognition*, L. Gregg, ed., Hillsdale, NJ: Erlbaum Assoc., 1974, pp. 253-285.

13. Newell, A. and Simon, H. A., *Human Problem Solving*, New Jersey: Prentice-Hall, 1972.

14. Newell, A. and Rosenbloom, P., "Mechanisms of Skill Acquisition and the Law of Practice," in *Cognitive Skills and Their Acquisition*, J. R. Anderson, ed., Hillsdale, NJ: Erlbaum Assoc., 1981.

15. Sacerdoti, E. D., *A Structure for Plans and Behavior*, Amsterdam: North-Holland, 1977.

16. Schank, R. C. and Abelson, R. P., *Scripts, Goals, Plans and Understanding*, Hillside, NJ: Lawrence Erlbaum, 1977.

17. Schank, R. C., "Reminding and Memory Organization: An Introduction to MOPS," Tech. report 170, Yale University Comp. Sci. Dept., 1979.

18. Schank, R. C., "Language and Memory," *Cognitive Science*, Vol. 4, No. 3, 1980 , pp. 243-284.

19. Simon, H. A., Carbonell, J. and Reddy, R., "Research in Automated Knowledge Acquisition," 1979. Research Proposal to the Office of Naval Research from the Carnegie-Mellon Computer Science Department.

20. Winston, P. H., "Learning and Reasoning by Analogy," *CACM*, Vol. 23, No. 12, 1979 , pp. 689-703.