REPORT DOCUMENTATION PAGE

DTIC FILE COPY

AD-A230 977	**1b. RESTRICTIVE MARKINGS**
	3 DISTRIBUTION/AVAILABILITY OF REPORT Unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S) Report No. 7	**5. MONITORING ORGANIZATION REPORT NUMBER(S)**

DTIC
ELECTE
JAN 23 1991
S E D

6a. NAME OF PERFORMING ORGANIZATION Department of Information & Computer Science	**6b OFFICE SYMBOL** *(If applicable)*	**7a. NAME OF MONITORING ORGANIZATION**
6c. ADDRESS (City, State, and ZIP Code) University of California Irvine, CA 92717 (PI: Pat Langley)		**7b ADDRESS (City, State, and ZIP Code)**

8a. NAME OF FUNDING/SPONSORING ORGANIZATION Army Research Institute	**8b. OFFICE SYMBOL** *(If applicable)*	**9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER** MDA 903-85-C-0324

8c. ADDRESS (City, State, and ZIP Code) 5001 Eisenhower Avenue Alexandria, Virginia 22333	**10 SOURCE OF FUNDING NUMBERS**			
	PROGRAM ELEMENT NO.	**PROJECT NO.**	**TASK NO.**	**WORK UNIT ACCESSION NO**

11. TITLE (Include Security Classification)

A Unified Framework for Planning and Learning

12. PERSONAL AUTHOR(S) Pat Langley and John A. Allen

13a. TYPE OF REPORT Final Report	**13b. TIME COVERED** FROM 9/1/85 TO 8/31/90	**14 DATE OF REPORT (Year, Month, Day)** November 30, 1990	**15. PAGE COUNT** 28

16. SUPPLEMENTARY NOTATION To appear in S. Minton (Ed.), Computational Approaches to Learning and Planning. San Mateo, CA: Morgan Kaufmann.

17.	**COSATI CODES**		**18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)**
FIELD	**GROUP**	**SUB-GROUP**	concept formation means-ends analysis incremental learning search control knowledge case-based reasoning cognitive models concept hierarchies integrated architectures
			systems,

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

OVER

* Artificial intelligence,

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	**21 ABSTRACT SECURITY CLASSIFICATION**
22a NAME OF RESPONSIBLE INDIVIDUAL Pat Langley	**22b TELEPHONE (include Area Code)** (415) 604-4878	**22c. OFFICE SYMBOL**

DD FORM 1473, 84 MAR   83 APR edition may be used until exhausted.
 All other editions are obsolete.

91 1 22 137

19 ABSTRACT

In this paper we present a computational framework for planning and learning that is constrained by knowledge of human behavior. We first describe DÆDALUS, a planning system that learns from successful problem-solving traces. The model stores plan knowledge in a probabilistic concept hierarchy, retrieves relevant operators through a process of heuristic classification, organizes search using a flexible version of means-ends analysis, and stores plan knowledge through an incremental process of concept formation. We report experimental studies of DÆDALUS' behavior that show learning improves solution quality and reduces search, but that also reveal increased retrieval cost and fewer solved problems. In addition, we find that the model accounts for a variety of qualitative phenomena observed in human problem solving. After this, we present our current designs for ICARUS, an integrated architecture for intelligent agents that extends on the ideas in DÆDALUS. This architecture would store entire problem-solving traces in memory, which should support a number of additional capabilities, including the unification of search control knowledge and macro-operators, the interleaving of planning and execution, and the integration of closed-loop and open-loop processing.

A Unified Framework for Planning and Learning

PAT LANGLEY
JOHN A ALLEN

Department of Computer Science
University of California
Irvine, California 92717

Abstract

In this paper we present a computational framework for planning and learning that is constrained by knowledge of human behavior. We first describe DÆDALUS, a planning system that learns from successful problem-solving traces. The model stores plan knowledge in a probabilistic concept hierarchy, retrieves relevant operators through a process of heuristic classification, organizes search using a flexible version of means-ends analysis, and stores plan knowledge through an incremental process of concept formation. We report experimental studies of DÆDALUS' behavior that show learning improves solution quality and reduces search, but that also reveal increased retrieval cost and fewer solved problems. In addition, we find that the model accounts for a variety of qualitative phenomena observed in human problem solving. After this, we present our current designs for ICARUS, an integrated architecture for intelligent agents that extends on the ideas in DÆDALUS. This architecture would store entire problem-solving traces in memory, which should support a number of additional capabilities, including the unification of search control knowledge and macro-operators, the interleaving of planning and execution, and the integration of closed-loop and open-loop processing.

1. Introduction

A robust intelligent agent must have three general characteristics. First, it should be able to plan, to generate possible action sequences that lead to the achievement of goals. Second, the agent should learn from its problem-solving experience in a domain, improving its ability from previous attempts at plan generation. Finally, the agent should integrate planning and learning with other aspects of behavior, such as execution and perception. These capabilities are central to human behavior, and we believe they are essential to the success of any agent that is situated in a complex physical environment. Our long-term goal is to develop a unified architecture that provides practical abilities of this sort while remaining consistent with knowledge of human cognition.

In this chapter we describe DÆDALUS, a system that addresses two of the above abilities – planning and learning. Our work on DÆDALUS has been influenced by previous work in both artificial intelligence and cognitive psychology. The basic planning algorithm borrows from Newell, Shaw, and Simon's (1960) GPS model of human problem solving, and very similar methods have been used in Minton et al.'s (1989) PRODIGY and Jones' (1989) EUREKA, two systems that learn in planning domains. DÆDALUS' representation and organization of knowledge, and its basic learning method, draws from Fisher's (1987) work on COBWEB, an incremental approach to concept formation intended to account for certain memory phenomena observed in humans. Our approach also makes contact with work in analogical and case-based reasoning (Falkenhainer, Forbus, & Gentner, 1989; Veloso & Carbonell, 1989).

We discuss these historical links in more detail throughout the following section, relating them to distinctions from the literature on planning and learning, and showing that DÆDALUS provides a unified framework that moves beyond these distinctions. We then present a preliminary evaluation of the system, both as a practical learning method and as a psychological model, which reveals some strengths and some limitations. After this, we respond to the limitations by outlining our designs for ICARUS, an integrated architecture that incorporates ideas on planning and learning from DÆDALUS, but that integrates these with mechanisms for perception and execution. As before, we organize our discussion of ICARUS using issues that have recurred in the literature. Finally, we summarize the approach we have taken and its contributions to the study of learning, planning, and intelligent agents.

2. Characteristics of DÆDALUS

Research on learning and planning has led to a number of dichotomies that have divided the field. These range from the algorithms used to generate plans, through the basic representation of acquired knowledge, to the mechanisms used to improve planning ability. In this section, we describe the stance we have taken on four such issues in constructing DÆDALUS. In each case, we find that the system provides an elegant unification of what have often been viewed as antithetical positions.

Table 1. Pseudocode for means-ends analysis, the basic algorithm that DÆDALUS uses to generate plans. This formulation assumes a depth-first ordering on search, with backtracking when one exceeds a depth limit, but other ordering schemes are also possible.

```
Inputs:   STATE is a (partially described) initial state.
          GOAL is a (partially described) desired state.
Outputs:  A final state that matches the description of GOAL.
Variables: DEPTH is the current depth of the search tree.
           MEMORY is the memory containing all known operators.

Procedure MEA(STATE, GOAL)

  If DEPTH does not exceed the depth limit,
  Then if STATE matches GOAL,
       Then return STATE.
       Else let DIFFS be the differences between STATE and GOAL.
            Let OPERATOR-SET be Select(DIFFS, MEMORY).
            For each OPERATOR in OPERATOR-SET,
                Let PRECONDS be the preconditions of OPERATOR.
                If STATE does not match PRECONDS,
                Then let STATE be MEA(STATE, PRECONDS).
                     If STATE is not Failed,
                     Then let NEW be the state that results
                              from applying OPERATOR to STATE.
                          If NEW matches GOAL,
                          Then return NEW.
                          Else let FINAL be MEA(NEW, GOAL).
                               If FINAL is not Failed,
                               Then Return FINAL.
  Return Failed.
```

2.1 Forward Chaining and Means-Ends Analysis

Much of the AI research on problem solving has focused on *forward chaining* or state-space search. In this scheme, one applies an operator to an initial state, another operator to its successor, and so forth, until reaching a state that matches the goal description. At each stage of this process, one considers an operator only if its legal preconditions exactly match the current state. Many of the formal results on heuristic search assume a forward-chaining approach (e.g., Pearl, 1984), and much of the early work on learning in problem solving aimed to find heuristic conditions for operator selection in state-space search (Langley, 1985; Mitchell, Utgoff, & Banerji, 1983; Ohlsson, 1983).

Another important approach to problem solving is known as *means-ends analysis*. In this algorithm, one selects some difference between the current and desired state, selects an operator which reduces that difference, and attempts to apply the operator. If the operator's preconditions are

not met, one recursively calls the method to transform the current state into one that meets these conditions. If the preconditions are met, one generates the state resulting from its application and recursively calls the algorithm to transform the new state into the desired one. Table 1 gives details on this approach to problem solving. The pseudocode assumes a depth-first ordering on search, but one could use breadth-first search, best-first search, or other techniques, just as one can within the forward-chaining framework.

To summarize, means-ends systems selectively retrieve operators that appear relevant to a problem, even if those operators cannot be immediately applied. In some cases this leads to a form of backward-chaining behavior, in that the order of operator selection is the reverse of the application order. In other cases this strategy produces forward chaining, in that selection and application order agree, and in still others it generates mixed behavior. This technique was first used in Newell, Shaw, and Simon's (1960) General Problem Solver (GPS), and then later in Fikes, Hart, and Nilsson's (1971) STRIPS, the precursor of many existing planning systems. Much of the recent work on learning in problem solving has assumed means-ends planners (Minton et al., 1989; Jones, 1989), and Newell and Simon (1972) report evidence that such methods occur in human problem solving.

At first glance, means-ends approaches seem superior to state-space methods, due to their focus on relevant operators and their ability to break problems into useful subproblems. However, traditional means-ends systems examine only one difference at a time, and they ignore relations between states and preconditions. In response, DÆDALUS uses a variation (which we call *flexible means-ends analysis*) that prefers operators which reduce more differences and whose preconditions more closely match the current state. Thus, the retrieval process incorporates ideas from both approaches, biasing the system toward operators that have more effect and that are more nearly applicable. As we will see below, DÆDALUS can also place weights on each difference and state descriptor, giving additional flexibility in its retrieval decisions. However, the basic algorithm is identical to that shown in Table 1, differing from earlier means-ends methods only in its instantiation of the Select procedure.[1]

2.2 Search and Memory

Both forward chaining and means-ends analysis assume that planning requires *search* for compositions of primitive operators that will transform an initial state into a desired one. Although they carry out this search through somewhat different spaces and employ different strategies, both are clear variants of Newell's (1980) problem-space hypothesis. This states that cognition involves search through *problem spaces*, which can be characterized in terms of problem states, goal descriptions, and operators that transform one state into another. Much of the early research on planning took this view (e.g., Fikes et al., 1971), and Newell and Simon (1972) present convincing evidence that it provides a reasonable account of human behavior in novel domains.

---

1. DAEDALUS borrows the notion of flexible means-ends analysis from Jones' (1989) EUREKA system, which used a very similar idea with a quite different retrieval method. Jones' (1990) more recent GIPS system also uses a similar strategy.
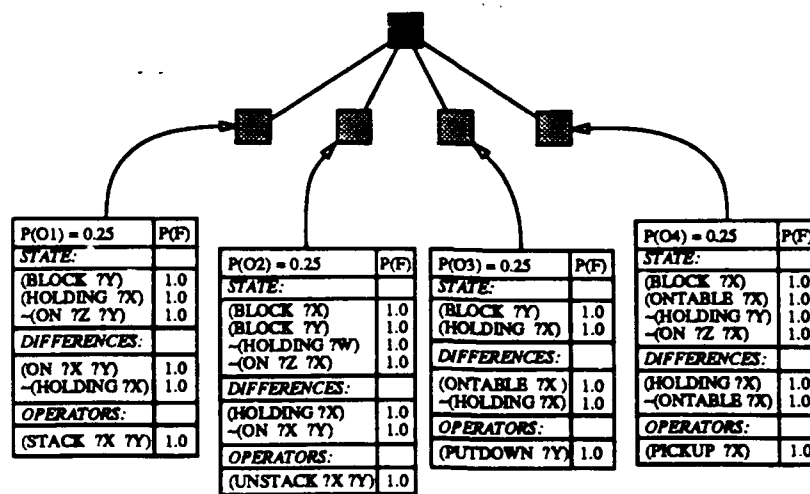
P(O1) = 0.25	P(F)
STATE:	
(BLOCK ?Y)	1.0
(HOLDING ?X)	1.0
-(ON ?Z ?Y)	1.0
DIFFERENCES:	
(ON ?X ?Y)	1.0
-(HOLDING ?X)	1.0
OPERATORS:	
(STACK ?X ?Y)	1.0

P(O2) = 0.25	P(F)
STATE:	
(BLOCK ?X)	1.0
(BLOCK ?Y)	1.0
-(HOLDING ?W)	1.0
-(ON ?Z ?X)	1.0
DIFFERENCES:	
(HOLDING ?X)	1.0
-(ON ?X ?Y)	1.0
OPERATORS:	
(UNSTACK ?X ?Y)	1.0

P(O3) = 0.25	P(F)
STATE:	
(BLOCK ?Y)	1.0
(HOLDING ?X)	1.0
DIFFERENCES:	
(ONTABLE ?X)	1.0
-(HOLDING ?X)	1.0
OPERATORS:	
(PUTDOWN ?Y)	1.0

P(O4) = 0.25	P(F)
STATE:	
(BLOCK ?X)	1.0
(ONTABLE ?X)	1.0
-(HOLDING ?Y)	1.0
-(ON ?Z ?X)	1.0
DIFFERENCES:	
(HOLDING ?X)	1.0
-(ONTABLE ?X)	1.0
OPERATORS:	
(PICKUP ?X)	1.0

Figure 1. Initial concept hierarchy provided to DÆDALUS for the blocks world domain. Terminal nodes (shown with their descriptions) correspond to generic operator schemas.

A separate research tradition posits that planning requires the *retrieval* of relevant plans or plan components from long-term memory. Such knowledge-intensive approaches emphasize the encoding of domain-specific heuristics for decomposing problems into simpler ones, heuristics for selecting states and operators, or combinations of operators that directly solve problems or subproblems. This view of planning provides a plausible explanation of human behavior in highly familiar domains.

DÆDALUS unifies these two views of planning, as does much of the recent work on learning in problem solving (Yoo, Yang, & Fisher, in press; Jones, 1989; Minton et al., 1989; Laird, Hucka, Yager, & Tuck, 1990; Veloso & Carbonell, 1989). The system operates within a problem-space framework, generating sequences of operators to transform an initial state into one that matches a goal description; however, it uses domain-specific knowledge to constrain and direct this search. DÆDALUS stores this knowledge in a *probabilistic concept hierarchy*. Initially, this contains only abstract descriptions of operator schemas, but over time the system uses the same data structure to organize its experience in a domain and to retrieve relevant knowledge during planning.

Figure 1 presents the initial concept hierarchy given to the system for the blocks world domain. This hierarchy plays the same role for DÆDALUS as does the table of connections for Newell et al.'s GPS (1960). Each terminal node corresponds to a generic operator schema, which is summarized in terms of its legal preconditions, the differences it reduces upon application, and its name and arguments. The root of the hierarchy contains a probabilistic summary of all nodes below it; terminal nodes are described in the same language, but all their probabilities are one. In more complex domains, one might also include internal nodes that index and summarize the operators below them in the hierarchy. The description for such a nonterminal node contains four parts: the probability of occurrence relative to its parent, the conditional probability of each precondition given membership in the concept, the conditional probability of each reduced difference given membership, and the probability that one should select each operator in this situation.

The retrieval of operators involves sorting a problem – described as a set of state descriptors and differences – through this concept hierarchy. To do this, DÆDALUS invokes COBWEB$_R$, a variant of Fisher's (1987) COBWEB algorithm that handles relational descriptions. This routine is responsible for selecting a plausible analogical match; the latter is necessary because a problem may partially match a given description in many ways. At each level, COBWEB$_R$ selects the node that best matches the problem and recurs to the next level. Upon reaching a terminal node, the routine returns the associated operator to DÆDALUS for use in extending its plan. If an operator leads to a loop or dead end, the system re-sorts the problem through the hierarchy to find another operator.

As we will see shortly, the learning process alters the structure of DÆDALUS' concept hierarchy and the probabilities stored therein. However, the form of the hierarchy, the retrieval mechanism, and the overall planning algorithm remain unchanged throughout the course of learning, providing a unified view of memory and search in planning.

2.3 Cases and Abstractions

One common approach to encoding plan knowledge involves the use of abstract rules or schemas. For instance, Minton et al.'s (1989) PRODIGY uses abstract selection, preference, and rejection rules, Mooney's (1990) EGGS employs general plan schemas, and G. Iba's (1989) MACLEARN stores abstract macro-operators. Each rule or schema covers many specific situations, allowing these systems to use a simple matching or unification algorithm to determine their applicability. Learning in this framework often uses some variation of explanation-based methods, as in the above systems, but inductive approaches are also possible (Langley, 1985; Mitchell et al., 1983; Ohlsson, 1983).

Another approach encodes knowledge as specific cases from the domain, including particular problems or subproblems, desirable and undesirable approaches to these problems, and possibly the reasons for their desirability. Researchers in this *case-based* paradigm have proposed a variety of methods (Hammond, 1990; Jones, 1989; Kolodner, Simpson, & Sycara, 1985; Veloso & Carbonell, 1989), many of them with direct mapping to techniques that assume abstractions. This approach has close ties with work on analogical problem solving (e.g., Carbonell, 1983), although the focus in 'case-based' methods is on transfer to problems within a domain rather than across domains. However, they share a reliance on more sophisticated matching schemes than needed for abstract knowledge structures, often requiring relational partial matching (i.e., structural analogy).

DÆDALUS unifies these two frameworks by storing both cases and abstractions in a single probabilistic concept hierarchy. Figure 2 shows a blocks world problem that the system cannot solve without search given the initial hierarchy in Figure 1, along with the structure of an optimal derivational trace provided to the system by an expert (the programmer). Each node in this trace can be viewed as a miniature case, which corresponds to a problem or subproblem that is described as a set of state predicates, a set of differences, and the operator used to solve it. DÆDALUS stores each of these cases as terminal nodes in its concept hierarchy, organizing them via internal nodes that index the cases that occur below them in the hierarchy. Given a problem with similar structure, the system uses these stored cases or the resulting internal nodes to direct search on future problems.
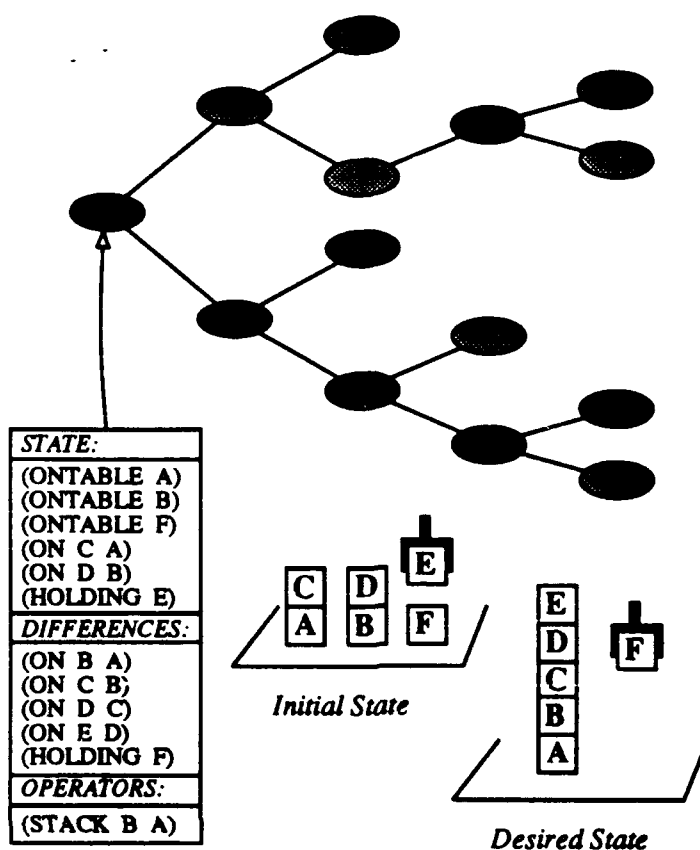
STATE:
(ONTABLE A)
(ONTABLE B)
(ONTABLE F)
(ON C A)
(ON D B)
(HOLDING E)

DIFFERENCES:
(ON B A)
(ON C B)
(ON D C)
(ON E D)
(HOLDING F)

OPERATORS:
(STACK B A)

Initial State

Desired State

Figure 2. A problem from the blocks world, along with an optimal derivational trace given to DÆDALUS by a domain expert. Each node in the trace consists of a state description, a set of differences, and the selected operator. Black nodes correspond to problems on which the system selected the incorrect operator; white nodes specify problems on which it made the right selection.

Yoo et al. (in press) describe a closely related approach to combining cases and abstractions for planning.

Figure 3 shows the modified hierarchy after DÆDALUS has incorporated its experience with the problem in Figure 2. Each new case (in gray) represents one of the problems or subproblems in the derivational trace, described as a set of differences, a set of state predicates, and the operator that led to its solution. The figure includes full descriptions for two of these cases (nodes N2 and N3). The additional terminal nodes (in white) represent the original operator schemas that were already present in memory. The extended hierarchy also contains some abstractions (in black) that DÆDALUS created during the process of storing the trace components. The figure also shows the full description of one abstraction (node N1), which reveals that this concept provides a probabilistic summary of the nodes (N2 and N3) below it. Each such description includes an overall probability of occurrence, together with a conditional probability for each difference, state descriptor, and operator.

Because DÆDALUS attempts to sort new problems to terminal nodes in its concept hierarchy, abstractions act primarily as indices for the retrieval of cases and the initial operator schemas.
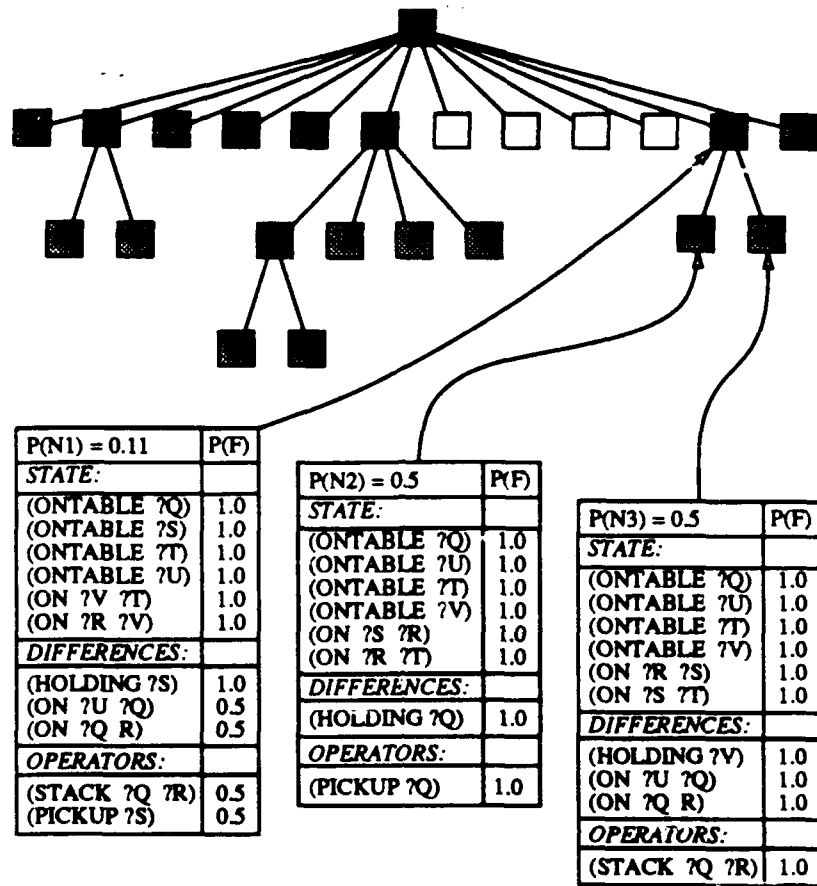
P(N1) = 0.11	P(F)
STATE:	
(ONTABLE ?Q)	1.0
(ONTABLE ?S)	1.0
(ONTABLE ?T)	1.0
(ONTABLE ?U)	1.0
(ON ?V ?T)	1.0
(ON ?R ?V)	1.0
DIFFERENCES:	
(HOLDING ?S)	1.0
(ON ?U ?Q)	0.5
(ON ?Q R)	0.5
OPERATORS:	
(STACK ?Q ?R)	0.5
(PICKUP ?S)	0.5

P(N2) = 0.5	P(F)
STATE:	
(ONTABLE ?Q)	1.0
(ONTABLE ?U)	1.0
(ONTABLE ?T)	1.0
(ONTABLE ?V)	1.0
(ON ?S ?R)	1.0
(ON ?R ?T)	1.0
DIFFERENCES:	
(HOLDING ?Q)	1.0
OPERATORS:	
(PICKUP ?Q)	1.0

P(N3) = 0.5	P(F)
STATE:	
(ONTABLE ?Q)	1.0
(ONTABLE ?U)	1.0
(ONTABLE ?T)	1.0
(ONTABLE ?V)	1.0
(ON ?R ?S)	1.0
(ON ?S ?T)	1.0
DIFFERENCES:	
(HOLDING ?V)	1.0
(ON ?U ?Q)	1.0
(ON ?Q R)	1.0
OPERATORS:	
(STACK ?Q ?R)	1.0

Figure 3. Revised DÆDALUS concept hierarchy that incorporates cases (gray) and abstractions (black) resulting from storage of components from the derivational trace in Figure 2, along with the original operator schemas (white) for this domain.

However, if a new problem is sufficiently different from all children of an abstract node N, the CoBWEB$_R$ routine will halt at that level of the hierarchy, returning the internal node N instead of a terminal node. In such a situation, DÆDALUS simply selects the operator with the highest conditional probability. This strategy should minimize the negative transfer that could result from analogies with cases that bear only limited resemblance to the new problem.

2.4 Data-Driven and Knowledge-Driven Learning

One major paradigm in machine learning emphasizes the detection of regularities in training data. This *data-driven* view includes most work on decision-tree construction (Quinlan, 1986), rule induction (e.g., Langley, 1985; Clark & Niblett, 1989), and conceptual clustering (e.g., Fisher, 1987), along with many other approaches to learning. The majority of work taking this perspective has been applied to classification or diagnostic tasks, though some has been used in problem-solving domains (Langley, 1985; Mitchell et al., 1983; Ohlsson, 1983).

Another major paradigm emphasizes the role of background knowledge in learning. This *knowledge-driven* view includes work on explanation-based learning (e.g., Minton et al., 1989; Mooney, 1990) and other approaches that involve compiling existing knowledge into new forms. The paradigm also includes work on constructive induction, in which background knowledge biases the creation of knowledge structures that summarize observations (e.g., Drastal, Raatz, & Czako, 1989; Elio & Watanabe, in press). The former has been applied primarily in domains like planning and design, in which a combination of rules can be compiled from traces. The latter has focused on classification problems, like the data-driven work on induction.

Although the data-driven and knowledge-driven paradigms differ in their emphases, both data and knowledge play a role – to differing degrees – in each framework. The work on constructive induction provides the clearest case of the interaction between background knowledge and experience. More important, in this work the initial knowledge is typically stated in a form that could plausibly be acquired by data-driven methods themselves, suggesting an approach to unifying these two perspectives on learning.

The learning scheme used in DÆDALUS provides one example of such a unified view, as does Yoo et al.'s (in press) related work. Figure 4 illustrates the four learning operations that lead to changes in the structure of memory. These operations include:

- *extending downward*, which occurs when a case reaches a terminal node in memory; under these circumstances, COBWEB$_R$ creates a new node N that is a probabilistic summary of the case and the terminal node, making both children of N;

- *creating a disjunct*, which occurs if a case is sufficiently different from all children of a node N; in this situation, COBWEB$_R$ creates a new child of N based on the case;

- *merging two concepts*, which occurs if a case is similar enough to two children of node N that COBWEB$_R$ judges all three should be combined into a single child;

- *splitting a concept*, which occurs when a case is different enough from a child C of node N that COBWEB$_R$ decides C should be removed and its children moved up to become children of N.

The last three of these actions are considered at each level of the hierarchy, as the system sorts the new case (taken from a successful trace) downward through memory. If none of these are deemed appropriate, COBWEB$_R$ simply averages the case into the probabilistic description of the best-matching node. Fisher (1987) describes category utility, the evaluation function used in making these decisions. For our present purposes, the important point is that DÆDALUS incorporates each case into its hierarchy *incrementally*, with the very act of classification modifying the structure of long-term memory.

Recall that the system begins with background knowledge in the form of an initial concept hierarchy that summarizes and indexes legal domain operators. This knowledge structure can bias the sorting of new cases, in that different initial hierarchies represent different indexing schemes. Because learning in DÆDALUS is integrated with classification, initial knowledge directly influences changes to the hierarchy's structure and probabilistic descriptions. Given different background knowledge, the system would acquire different heuristics for directing search. Moreover, once DÆDALUS has incorporated the components of a problem into memory, the structural changes
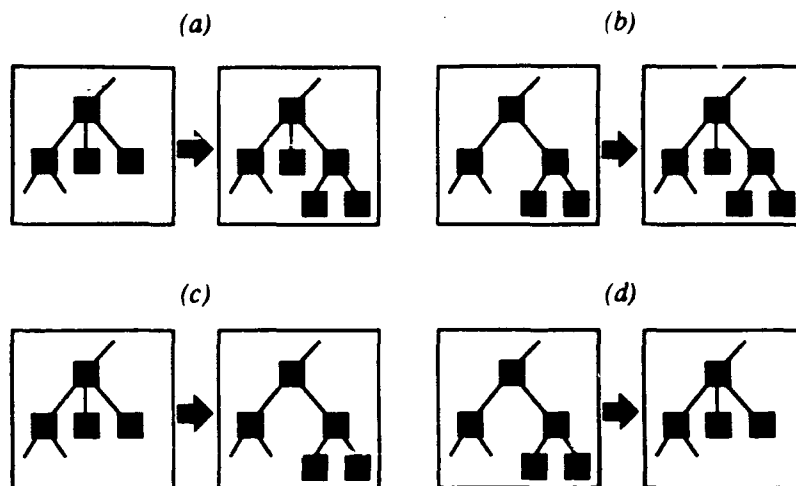
Figure 4. Operations used by the COBWEB_R routine to alter the structure of DÆDALUS' hierarchy: (a) extending the tree downward; (b) creating a new disjunct; (c) merging two concepts; and (d) splitting an existing concept.

introduced by this process bias future learning, while still letting the system respond to the nature of later observations. In this sense, DÆDALUS provides a unification of the data-driven and knowledge-driven views on learning. However, the current approach does not take full advantage of the knowledge available to a planning system, and we will return to this issue in Section 4.

3. Evaluation of DÆDALUS

In the previous section, we argued that DÆDALUS provides an elegant approach to learning and planning that eliminates four dichotomies that have appeared in the literature. However, science requires more than elegance – one must show that a framework or theory actually produces some desirable behavior. In this section we evaluate DÆDALUS as both a practical learning algorithm and as a psychological model. We then summarize the overall strengths and weaknesses of the current system.

3.1 Improvement in Performance

The goal of learning is some improvement in performance, and one can run experiments with any learning system to determine whether it achieves this goal (Kibler & Langley, 1988). To this end, we have carried out preliminary studies with DÆDALUS using the blocks world domain described in the previous section. In each case, we ran the system on ten training problems, measuring its performance after every two problems on a separate set of nine test problems.[2] We selected both training and test problems that could be solved before learning, but not without some search.

---

2. This approach generates true learning curves, which are quite different from the cumulative curves reported by Minton (1990a) and others. Here we present only preliminary results based on a single run, using one division
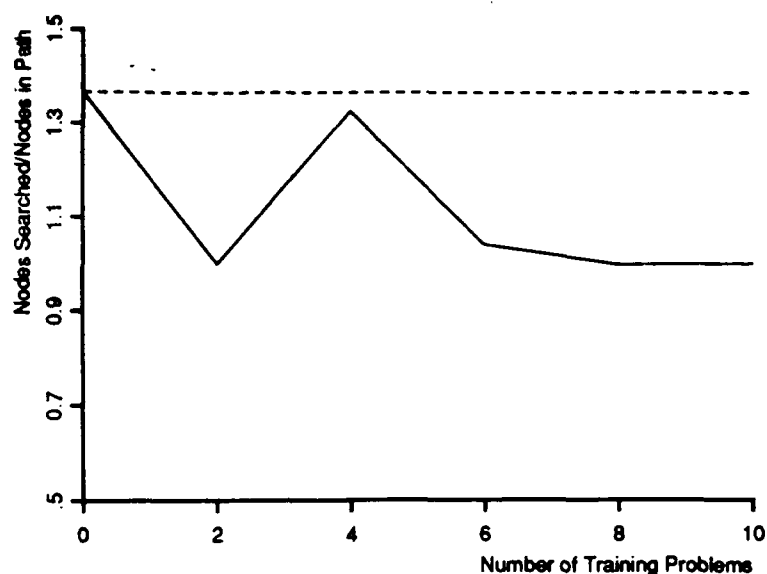
Figure 5. A learning curve showing the reduction in DÆDALUS' search as a function of experience, using sample problems from the blocks world domain. The solid line shows the behavior of the learning system; the dotted line shows behavior without learning.

During training, we operated DÆDALUS in 'learning apprentice' mode, providing it with the optimal derivational trace for each problem; we did this primarily to avoid the variation that would result from nonoptimal traces the system might find through search.

We used a number of dependent variables in these studi~, corresponding to different aspects of planning performance. Figure 5 presents the basic resul; which maps the amount of search against the number of training problems DÆDALUS has experienced. In this case, our measure was the number of nodes in the search tree divided by the length of the final solution path, which represents the amount of extraneous effort carried out on the test problems. The learning curve shows that, as the system gains experience in the blocks wor!d domain, its search becomes more directed. In fact, after working on eight training problems, DÆDALUS appears able to solve the test problems with almost no search (i.e., the node to length ratio approaches one). Similar results hold for another dependent variable, the length of the solution path. Before learning, the average solution length was about 13, whereas after four training problems it dropped to just above six, the optimal length for our test set, and remained at this level thereafter.

However, these dependent measures do not tell the entire story. The results in Figure 5 are based only on test problems that the system solved successfully within the computational limits we set (500 search nodes). In fact, although DÆDALUS solved all nine of the test problems (with some search) before learning, it solved only seven of these problems after processing ten training problems. Moreover, after only two training problems, the system could solve only two of the nine test cases, although on both it produced optimal solutions with no search. After additional training,

---

into training and test problems, and one order of the training problems. Our future experiments will average over different divisions and orders.
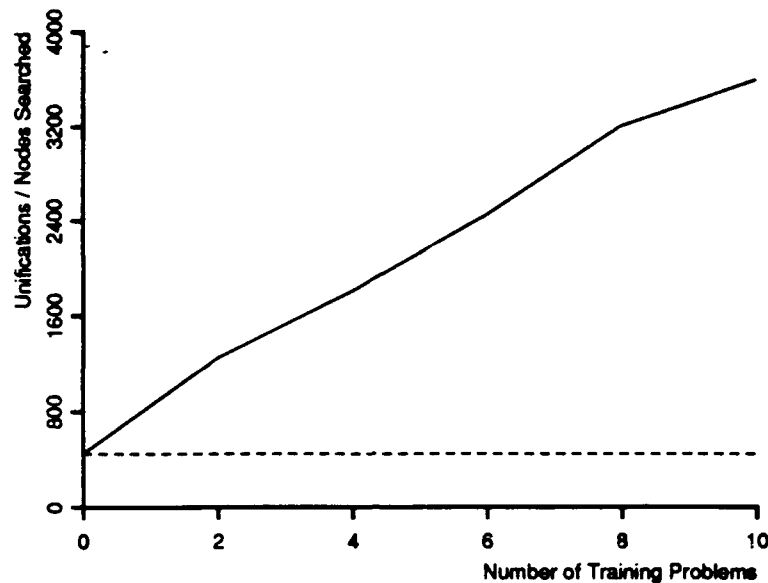
Figure 6. The increase in DÆDALUS' retrieval cost as a function of learning (solid line), compared to the same system without learning (dotted line), using training and test problems from the blocks world domain.

DÆDALUS solved more of the problems successfully, but there were always problems it could not complete. In general, the acquired knowledge reduced search and produced better solutions on most test problems, but performance was worse than before learning on a few tasks. Naturally, we would like to understand the source of this behavior, so we can eliminate it in future systems.

The most obvious hypothesis was that DÆDALUS had encountered a bottleneck similar to Minton's (1990a) *utility* problem or Tambe, Rosenbloom, and Newell's (1990) *expensive chunk* problem. In some cases, the cost of retrieving acquired knowledge can more than offset the savings due to reduced search and solution length. In DÆDALUS, nodes high in the concept hierarchy can come to incorporate many features, some with little information content. Perhaps the cost of partial matching against these abstract concepts was overwhelming the reduction in problem-space search. To check on this possibility, we measured the number of unifications per node in the search tree. As hypothesized, this measure of match cost systematically increased as a function of learning. Figure 6 suggests that this cost grows linearly with the number of training problems encountered, even on test problems that DÆDALUS successfully solved. In this run at least, the increase offset the reduced search costs, leading to greater total effort even on solved problems. Clearly, improving the system on this dimension should have high priority.

However, closer examination reveals that the utility problem cannot explain DÆDALUS' failure to solve all the test problems, because we based its computational limits not on overall cost, but on the number of nodes searched. Thus, the failures must result from errors in selecting operators, rather than from the cost of retrieving them. One version of this hypothesis is that, although DÆDALUS acquired cases and abstractions generally improve operator selection, they occasionally lead it astray. If this occurs early in a problem, the system can spend its entire allocation of search
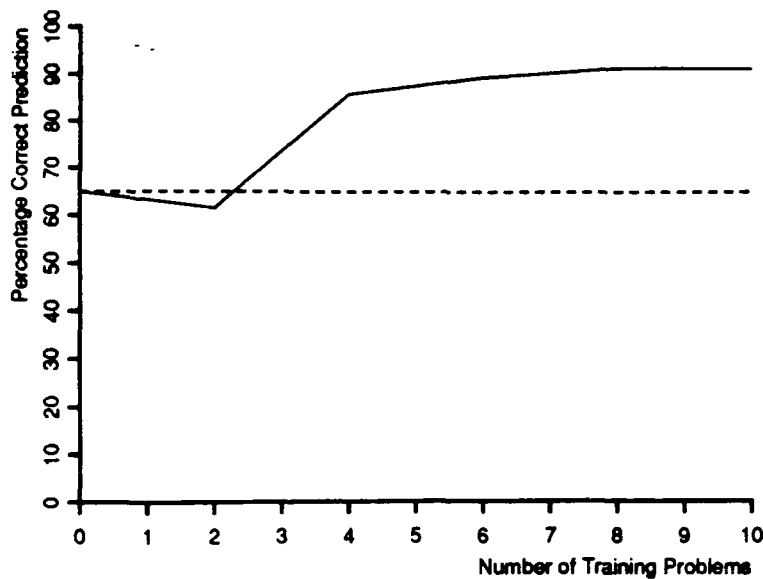
Figure 7. The effect of learning (solid line) on DÆDALUS' ability to retrieve correct operators for problems in the blocks world domain, compated to accuracy without learning (dotted line).

nodes in a 'wild goose chase', even though its acquired search heuristics are generally accurate. This suggests a novel approach to studying learning in planning domains, in which the interesting behavioral measure is the *accuracy* of learned control knowledge, rather than the overall efficiency of the planning system.

We used this insight to design another experiment that would provide information about DÆDALUS' ability to retrieve correct operators at each step along its solution path. This study used the same training and test problems as the previous one, but used in a rather different way. As before, we trained DÆDALUS in 'learning apprentice' mode, providing the correct operator at each stage in its search process. However, this time we used the same scheme on *test* problems, placing the system back on the right track whenever it made a selection error. We also recorded the percentage of correct decisions made during the generation of each plan. Figure 7 presents the results, which indicate that DÆDALUS clearly improves its ability to select the right operators. Before learning, the system makes the right choice in about 65% of the cases, whereas by the tenth training problem, it has reached the 90% level. These occasional errors, when not corrected by a tutor, can send DÆDALUS down fruitless paths and keep it from solving certain problems. However, there is no clear way to avoid the resulting search, and additional experience would presumably decrease their likelihood further, until the system can solve all the problems in a domain with no search.

3.2 Psychological Adequacy

Earlier we mentioned our concern that DÆDALUS be consistent with knowledge of human behavior, giving us a second dimension along which to evaluate the system. VanLehn (1989) gives an excellent review of the major findings with respect to human problem solving, including those related to

Table 2. Psychological adequacy of DÆDALUS and other models of learning in problem-solving domains. The symbol ⊕ indicates that a model accounts for a given phenomenon, ⊖ specifies that it provides no explanation, and ⊙ denotes that the model gives a partial account.

	ACT	SOAR	EUREKA	DÆDALUS
MEANS-ENDS ANALYSIS	⊙	⊕	⊕	⊕
NONSYSTEMATIC SEARCH	⊖	⊖	⊕	⊖
PROBLEM ISOMORPHS	⊙	⊙	⊙	⊙
GRADUAL IMPROVEMENT	⊕	⊕	⊕	⊕
ASSYMETRIC TRANSFER	⊕	⊕	⊕	⊕
EINSTELLUNG	⊕	⊕	⊕	⊕
REDUCED VERBALIZATION	⊕	⊕	⊖	⊖
AUTOMIZATION	⊙	⊙	⊖	⊖
RARITY OF ANALOGY	⊖	⊖	⊕	⊕
SUPERFICIAL ANALOGY	⊖	⊖	⊕	⊕

learning. These phenomena are qualitative in nature, but they still provide constraints on the operation of cognitive simulations. Table 2 lists most of the behaviors that VanLehn reports, along with some items we have added. The table also shows how DÆDALUS fares in comparison with three other models of problem solving and learning: Anderson's (1983) ACT, Laird, Rosenbloom, and Newell's (1986) SOAR, and Jones' (1989) EUREKA.

The first three phenomena address issues about basic problem-solving strategies rather than learning. In Section 2 we noted that Newell and Simon (1972) report evidence that humans appear to use means-ends analysis in novel domains, and other studies have buttressed this hypothesis. We have also seen that, like EUREKA, our system includes a flexible version of this process as one of its central components. SOAR differs somewhat on this issue; the system can simulate means-ends behavior using preference rules, but the architecture itself takes no stance on the centrality of this strategy. Finally, the ACT framework provides support for backward chaining but not true means-ends analysis, in the sense that it cannot select operators with unmatched conditions.

A second characteristic of human problem solving is its nonsystematic nature (Jones, 1989). Short-term memory limitations appear to prevent use of search-control schemes like depth-first, breadth-first, and best-first search, which must keep track of many problem states. Of the four systems, three rely on one of these strategies, with only Jones' EUREKA attempting to model humans' tendency to explore a search path in depth, then return to the initial state if unsuccessful to consider an alternative path (Newell & Simon, 1972). Another model that attempts to explain this behavior is Ohlsson's (1983) UPL.

A third phenomenon involves the relative difficulty of tasks. In some cases, even problems that are formally isomorphic – in that their operators and problem spaces are equivalent – can have quite different levels of difficulty (Kotovsky, Hayes, & Simon, 1985). This situation tends to occur when isomorphic problems have different physical manifestations, suggesting different representations for operators and/or states. Given alternative representations, each of the four systems could probably

model the observed differences on problem isomorphs. However, none provides an account of the origin of these representations.

Some additional behaviors concern changes in performance as humans gain experience in a problem-solving domain. One is so basic that it might easily be overlooked – in general, learning leads to reduced search on a class of problems. As we showed in the previous subsection, DÆDALUS generally improves its performance along this dimension with experience, as do ACT, SOAR, and EUREKA. However, this is no great feat for systems that were designed with this goal in mind.

A related phenomenon involves the asymmetry of *transfer* across problems. The transfer from a class of problems A to another class B is simply the reduction in training time on class B due to training on A. The asymmetry effect relates to situations in which the components of one problem class, say X, are subsumed by another (more difficult) class, say Y. In such cases, the transfer from class Y to class X is greater than that from X to Y. The standard explanation for this result is that transfer results from carrying over learned memory structures to the new task, and since the structures needed for the simpler task are subsumed by the more difficult one, training on the latter generates all the structures needed by the former. Because all four models decompose problems into subproblems, then learn methods for solving these subproblems, they should all produce this effect.

Human learning does not always lead to improvements in performance, and a computational model should have the same flaws as humans, even though they may be undesirable from an engineering perspective. One well-established type of performance decrement is called the *Einstellung* effect (Luchins, 1942). This occurs when one is trained on a set of difficult problems, learns a strategy for solving them, and then is given a similar problem set that can either be solved in the same manner or in a more efficient way. Under such circumstances, subjects typically find solutions analogous to the original ones, even though they find ones with fewer steps if they receive no prior training. Thus, although learning reduces search, it actually increases the length of solution paths. Neves and Anderson (1981) have shown that ACT produces this behavior, and Jones (1889) has produced similar results with EUREKA using quite different mechanisms. The SOAR and DÆDALUS models have not been explicitly tested on this front, but their reuse of structures acquired in earlier problems should generate the same effect.

In addition, experienced problem solvers show a variety of other differences from novices. Experts typically solve problems much more rapidly, even when their solutions involve the same number of steps in the problem space. Also, they tend to verbalize much less than people with less experience, suggesting that they have lost access to intermediate subproblems. Such skills are sometimes referred to as *automatized*, in that one can carry them out with little attention. Both DÆDALUS and EUREKA have difficulty explaining these phenomena, in that they never change the steps taken in generating a solution; learning may eliminate poor choices, but each node in the derivational trace must still be constructed one step at a time. In contrast, the other two systems actually

eliminate subproblems through learning, ACT through a mechanism similar to macro-operator formation and SOAR through a chunking process. These model the reduction in verbalization, but they only partially explain the observed speedup effect, which continues long after search has been eliminated.

A final set of empirical results concern problem solving by analogy. In principle, this could occur when a human is given the answer to one problem, and then later is asked to solve a problem with an analogous solution. However, experiments reveal that such behavior is quite rare, even when the two problems occur near each other in time (e.g., Gick & Holyoak, 1980). People are able to solve problems by analogy when given an explicit mapping between source and target problems, but they seldom find such a mapping on their own. Moreover, in those cases where they do manage to retrieve a relevant problem, the reminding is usually based on some superficial, surface similarity that may produce a misleading analogy (e.g., Ross, 1984). Both DÆDALUS and EUREKA fare well on these phenomena, since both rely on a form of analogical retrieval that operates on surface-level descriptions of problems. The ACT and SOAR models have more difficulty, since neither has any architectural mechanism for analogy. One could implement forms of analogy using explicit rules, but this seems unsatisfactory for a mechanism that (we hypothesize) is so basic.

3.3 Comments on DÆDALUS

In this section, we evaluated DÆDALUS along two dimensions – its ability to improve performance with experience and its adequacy as a psychological model. As a practical learner, preliminary experiments suggested that the system's learning mechanisms lead to improvement on two measures of performance, producing a reduction in search and shorted solution paths. However, we also found that these results held only for problems that DÆDALUS successfully solved, and that learning actually led it to solve fewer problems overall. An additional experiment revealed that retrieval accuracy does increase over time, suggesting that the failures result from acquired heuristics that occasionally lead the system down paths from which it cannot recover. We also noted that DÆDALUS suffers from a clear utility problem in that, even on solved problems, its retrieval cost per operator and overall planning cost increase with experience, rather than decreasing as desired.

As a psychological model, DÆDALUS accounts for a variety of robust phenomena that have been observed in human problem solving. However, three previous models also explain roughly the same behaviors. DÆDALUS differs from Laird et al.'s SOAR and Anderson's ACT in its coverage of analogical reasoning, an area it shares with Jones' EUREKA system. However, it fails to explain the reduction of verbalization and the automatization observed in highly-skilled problem solvers, which the other systems at least partially model. Moreover, DÆDALUS' search organization does not mimic the nonsystematic behavior found in humans', which only EUREKA has attempted to handle. The system also lacks in the broader sense that humans are physical agents that interleave planning with other processes. A fuller model of human behavior would explicitly link cognition with action and perception.

4. Extending the Unified Framework

Our research has been driven by a variety of concerns that DÆDALUS only partially addresses. We are interested in learning within the context of planning, but there are aspects of this domain that the current system simplifies or ignores. We are concerned with modeling the basic features of human problem solving, but DÆDALUS accounts for only some of the known psychological phenomena. Finally, our long-term aim is the construction of an intelligent agent that interacts with a physical environment, yet the existing system can neither represent nor execute physical actions.

In this section we present our designs for ICARUS, a unified architecture that would draw on techniques developed in DÆDALUS, but that would also integrate planning and learning with other behaviors. We envision this architecture as supporting a broader range of learning abilities, providing a better account of human cognition, and serving as the basis for a physical agent that interacts with its environment. Elsewhere (Langley, Thompson, Iba, Gennari, & Allen, in press), we have described our designs for ICARUS in terms of separate components for recognition, planning, and execution. Here we organize the discussion around five distinctions that recur in the literature, as in Section 2.

4.1 Induction and Explanation

In Section 2 we argued that DÆDALUS unified traditional notions of data-driven and knowledge-driven learning, but that further knowledge was available for use in learning. In particular, DÆDALUS constructs a derivational trace that specifies relations among problems and subproblems, then ignores this structure during the learning process, which deals only with the problems and subproblems in isolation. Flann and Dietterich's (1989) IOE system suggests the possibility of a fuller unification of the data-driven and knowledge-driven views. Their method uses background knowledge to construct explanations for each training instance, then carries out induction over these explanations in search of common structures. If one interprets derivational traces as explanations constructed from domain operators, then a system that stored abstracted traces in memory would provide another example of such a system.

The storage and use of derivational traces plays a central role in our designs for the ICARUS architecture. However, these are more complex knowledge structures than search heuristics and plan components, requiring more powerful approaches to representation, organization, retrieval, and learning. In response, we intend to replace the current COBWEB$_R$ routine with Thompson and Langley's (in press) LABYRINTH, a system that classifies and learns about objects with componential structure. For instance, one can view a person as composed of body, head, and limbs, and one can further decompose an arm into an upper arm, a forearm, and a hand. The derivational traces generated during means-ends analysis also have a clear componential structure, in this case a recursive one. In addition to a set of differences and state descriptors, each problem is decomposed into a desired operator application and two subproblems, one before the operator and the other afterward.

Like Fisher's (1987) COBWEB, the LABYRINTH system represents knowledge in a probabilistic concept hierarchy, with cases at terminal nodes and abstractions at internal markers. The main representational difference is that some attributes, rather than pointing to primitive values, point to other nodes in the concept hierarchy. These features of composite concepts can be viewed as separate *roles*. For instance, in the PIRATE concept, one of the roles might point to two possible values – the LEG concept (with a 0.9 probability) and the PEG concept (with a 0.1 probability). Similar alternatives might exist for the INITIAL-STATE role, the OPERATOR role, or either of the subproblem slots at a given level of an abstract derivational trace. However, such 'internal disjuncts' need not always occur; in some cases a role will point to a single component concept (with probability one) that summarizes all component cases that have filled that role.

ICARUS would use LABYRINTH in the same way that DÆDALUS employed COBWEB$_R$, both for retrieving similar problems and for storing problem solutions. Initial retrieval must be based on differences and states in the top-level problem, as in DÆDALUS, since at this point the system has no other information. The use of derivational traces does allow more sophisticated planning strategies, but we delay their discussion until the next subsection. As in COBWEB, the storage process is interleaved with classification, but LABYRINTH takes a recursive approach to handling instances with multiple levels of structure. Briefly, the system first classifies all component objects at the lowest level, then those at the next level, and so forth, until it has classified the top-level object. In ICARUS, this means the system would first incorporate the lowest-level subplans into memory, then higher-level ones, and finally the entire derivational trace.

The above account simplifies the problems that face LABYRINTH along several dimensions. For example, in some domains, the roles of components must be determined during the match process, which uses a greedy method similar to that used in COBWEB$_R$. This would not be an issue for the subproblems and operators that occur in derivational traces, but it would for the objects that appear in differences and state descriptions. In addition, LABYRINTH incorporates another learning operation beyond those shown in Figure 4, called *attribute generalization*, which replaces internal disjuncts with their common parents in the concept hierarchy. This is necessary if composite concepts such as derivational traces are to achieve any generality; without this operation, each role would point to a large set of alternative values, each with very low probability. Finally, we will need to extend LABYRINTH to allow tests for object identity across embedded components, since the same states will occur in many subproblems in a derivational trace.

In summary, our designs for ICARUS call for the storage of entire derivational traces in an extended probabilistic concept hierarchy. The abstractions that summarize these traces can be viewed as resulting from a process of induction over explanations, further unifying the notions of data-driven and knowledge-driven learning. Moreover, these extended knowledge structures provide scaffolding for the storage of additional information that supports even more interesting forms of planning and learning, as we describe in the remainder of this section.

4.2 Search Heuristics and Macro-Operators

Learning improves the ability to generate plans by reducing search, and researchers have explored two main variants on this idea. The first focuses on the acquisition of heuristics that constrain or direct search, thus reducing the effective branching factor. For instance, Minton et al.'s (1989) PRODIGY learns abstract rules that specify some operators, states, or differences as preferable to others, which it then uses to control means-ends analysis. Laird et al.'s (1986) SOAR acquires similar rules for a different class of problem-solving strategies. Some work on case-based reasoning takes an analogous approach. For example, Jones' (1989) EUREKA retrieves components of stored derivational traces, which it uses as miniature cases to bias selection of operators in a means-ends framework.

A second approach deals with the acquisition of composite structures that reduce search by lessening the effective length of solution paths. For instance, G. Iba's (1989) MACLEARN defines macro-operators as compositions of primitive operators; it then uses these macros to solve future problems in fewer steps. Similarly, Mooney's (1990) EGGS constructs abstract schemas that let it solve some problems in a single leap. Many case-based systems take a related approach, retrieving entire derivational traces to problems similar to the one at hand. However, in much of this work, the 'macro' does not apply exactly and thus must be adapted to the new situation (e.g., Veloso & Carbonell, 1989). This approach holds the potential for more transfer than the use of opaque operators, but it requires that one check each step of the derivational trace to determine applicability.

DÆDALUS clearly follows the first of these paths, storing cases and probabilistic abstractions that serve the same role as selection rules in PRODIGY and SOAR. Although the system constructs a derivational trace during the planning process, it does not store the trace itself in memory, as described by Veloso and Carbonell. Rather, it stores the components of this trace in its concept hierarchy, using each of them when it seems appropriate. Laird et al. (1986) have noted that SOAR can effectively simulate the use of macro-operators with search-control rules, in that the latter can reproduce specific sequences of operators; DÆDALUS' knowledge structures can produce macro-like behavior in a similar manner. Nevertheless, one can imagine efficiency reasons for preferring actual composite structures (Minton, 1990b), and timing studies with humans suggest that they store some structures of this sort (Rosenbaum, Kenny, & Derr, 1983).

In contrast, the design for ICARUS supports both forms of plan knowledge. Recall that this system would store entire derivational traces in memory, described as problems, subproblems, and connections among them. The concept hierarchy would also contain probabilistic abstractions of these traces. Upon encountering a new problem, ICARUS would retrieve a similar problem or abstraction, as in the current system. However, this node would point not only to the operator that proved useful in solving it, but also to the subproblems that occurred in the solution. At this point, ICARUS would have two obvious choices:

- select the retrieved operator, create new subproblems, and sort them through the concept hierarchy to select further operators;
- examine the stored subproblems and simply select the operators stored there.

The first approach is identical to the strategy employed by DÆDALUS, and makes no use of the stored substructures. The second takes advantage of this knowledge to avoid sorting subproblems through memory, giving the effect of invoking a macro-operator.

However, there is also a third option, which holds the key to determining an appropriate response and also suggests an approach to learning this knowledge. As in derivational analogy, one can operate in the second of the above modes, but comparing each subproblem to the analogous subproblem in the retrieved trace. If the two are sufficiently similar, the subtrace is used as a further guide; if not, the subproblem is sorted through memory to select an operator. Note that in highly regular domains (where one often encounters equivalent subproblems), these comparisons will typically be successful; in less regular domains, the same subproblems will seldom occur.

To encode this knowledge, ICARUS' abstract derivational traces will store the probability of each subtrace being reused, given that its parent problem has been retrieved. The system will consult these scores in solving a new problem. If the probability for a subproblem's reuse is especially high, ICARUS will simply reuse its operator without bothering to match it against the current subproblem. If this holds for the entire abstract derivational trace, it will be treated like an opaque macro-operator. On the other hand, if the probability for reuse is very low, the system will simply sort the problem through memory to retrieve a plausible operator in this manner. If this holds for each retrieved subtrace, the overall behavior will equivalent to using search heuristics. Finally, if the reuse probability is neither high nor low, ICARUS would base its decision on a comparison of the new subproblem and the stored one, simulating a form of derivational analogy. This would be the default early in the system's experience with a domain, before it had acquired reliable probabilities. Of course, arbitrary mixtures of these strategies could occur as well.

4.3 Planning and Execution

A physical agent must do more than plan, which involves the generation of possible action sequences. It must also be able to *execute* its plans, which involves the enactment of those sequences. A complete intelligent agent requires both capabilities, as well as some way to interleave them. A growing number of researchers have started to examine this problem (e.g., Drummond & Bresina, 1990; Laird et al., 1990; Mitchell et al., in press), but if we want ICARUS to serve as the basis for a robust agent, we must address it within our developing framework. Two basic issues arise with respect to interleaving planning and execution.

First, plans must somehow be grounded in executable actions. Most planning research, especially within machine learning, focuses on abstract, logical formalisms like that used in Section 2. In contrast, most work on robotics and control assumes sensori-motor descriptions that specify locations and velocities of physical objects and limbs. Some work has attempted to combine planning and execution while retaining separate languages for each (e.g., Fikes, Hart, & Nilsson, 1972), but this seems far from satisfactory. Second, one must take some stance on *when* to execute a plan or fragment of a plan. In some cases one can safely carry out some actions even before constructing a complete plan. The agent must determine when it has generated enough of a plan to begin execution.

Our response to the first issue is to modify the representation of states, operators, and plans to ground them in sensori-motor descriptions. Physical agents (including humans) exist not only in space but in time, and they must deal with objects, situations, and actions that have duration. In contrast to most AI work on planning, which assumes that states last indefinitely and that operators are instantaneous, ICARUS will represent both as *qualitative states* (Forbus, 1985; Williams, 1985), which are intervals of time during which the qualitative structure of a situation remains unchanged.[3] This does not mean the environment is static, but that changes occur in a constant direction (i.e., the signs of derivatives remain the same).

Thus, each state and each operator in a plan will be described in terms of the changes that occur while it is active, augmented by numeric information about positions, angles, rates of change, and duration. In a complete plan or subplan, each state specifies an expected observation and each operator indicates an executable action. In this view, motor skills such as throwing a ball or swinging a bat (W. Iba & Gennari, in press) are simply very detailed plans. Upon deciding to execute a plan or plan fragment, ICARUS would simply 'run' those aspects of the plan under its direct control (e.g., the rate and direction of a limb's movement) in the specified manner for the indicated duration. Moreover, it should be easy to adapt means-ends analysis to this representational scheme; one need simply replace the reduction of discrete differences with the reduction of continuous derivatives.

The representation of plans as derivational traces suggests a response to the second issue – when to execute a plan or plan fragment. Note that the means-ends algorithm summarized in Table 1 is ambiguous. The pseudocode states that one should 'apply' a selected operator as soon as its preconditions are met. However, this does not indicate whether one tests the preconditions against an imagined or an actual state; or whether one applies the operator in the mind or in the world. Thus, if backtracking were not an issue, ICARUS could simply execute each operator in its developing plan as soon as its preconditions were satisfied.[4]

Thus, the remaining concern revolves around whether backtracking is likely. As an example, suppose a person drives to work along Highway 1 regularly, usually crossing a bridge to reach this freeway. Further suppose that one day he learns the bridge is washed out and that he must find another route to work. He generates another path to Highway 1, then immediately gets in his car to drive away, without waiting to form a complete plan for the entire journey. The reason is that, once he has reached the highway, the chance of backtracking is slim. To let ICARUS take advantage of such knowledge, we plan to augment abstract derivational traces with information about backtracking probability. If the system retrieves a derivational trace for a given problem, and if experience shows that the second subproblem is very likely to be solved, then the agent would initiate execution as soon as it finds a complete plan for the first subproblem.

According to this scheme, ICARUS would begin its planning career in a particular domain by always forming complete plans, unless it was prevented from this by time demands or memory

---

3. This framework downplays the distinction between states and operators, which has been central to nearly all AI work on planning and problem solving. However, the distinction is not lost entirely, in that nodes will be distinguished by the roles they play in a given plan.

4. This sidesteps an important issue. In continuous domains, an operator's preconditions will never match perfectly. Thus, one must specify the degree of match required for execution. One approach would be to sort a state description through the concept hierarchy to see if it passes through the node for the selected operator's conditions.

limitations. As it gained experience with problems in the domain, it would collect estimates for the probability of backtracking on certain classes of problems, storing this information with abstract derivational traces in the concept hierarchy. On problem classes that seldom require backtracking, the system would gradually come to realize this fact and start to execute subplans before it had constructed an entire derivational trace. In domains where early execution is unjustified by history, ICARUS would remain conservative, continuing to generate a complete plan before execution whenever possible.

4.4 Closed-Loop and Open-Loop Processing

The planning community's growing focus on execution has raised concern about a related issue – the monitoring of changing environments. In some cases, an agent's actions may not have the desired effects; in others, external forces may alter the agent's surroundings. A robust agent must be able to handle both sources of uncertainty, and monitoring is the obvious response. In this extended framework, the agent compares predicted states to observed ones and, if the two disagree, it responds by modifying its plan and thus its actions. Humans can clearly behave in such a closed-loop, reactive fashion.

However, there is also psychological evidence for highly automatized behavior (e.g., Shiffrin & Dumais, 1981). Humans appear able to execute some motor skills in an open-loop mode, running a 'motor program' without external feedback, as though it were an opaque macro-operator. In well-behaved domains, there are clear advantages to such a strategy; monitoring requires attention, which in humans is a limited resource. Of course, the dichotomy between reactive, closed-loop execution and automatized, open-loop behavior is really a continuum. A unified theory of execution should support differing degrees of monitoring during enactment of a plan or motor program.

Our early work in this area (W. Iba & Langley, 1987) modeled the control of jointed limb movements. Our MAGGIE system monitored the positions and velocities of a simulated arm at a constant rate, noting divergences from desired behavior and responding to detected problems by applying local corrections. Such errors led a learning mechanism to produce modified motor schemas that were more accurate on future trials. The system could execute skills at differing speeds, which effectively reduced the rate of monitoring and thus the ability to detect and correct errors. Initially, the model was forced to trade off speed against accuracy; however, learning reduced this effect, letting it carry out highly automatized skills at high rates with little error despite reduced monitoring. There is some evidence for such a transition from closed-loop to open-loop mode in human motor behavior (Keele, 1982; Schneider & Eberts, 1980).

One drawback of our earlier work was that monitoring was *nonselective*, in that it occurred at regular intervals and examined all objects in the environment. In ICARUS we plan to take a more flexible approach, by retaining probabilities on whether specific expectations have been violated in the past. If stored probabilities indicate that the result of applying a given operator is highly predictable, the agent will not bother to monitor the new state (by sorting it through memory) and will continue execution unabated. In uncertain cases, ICARUS will classify the new situation and interrupt execution if it diverges from the expected state, then modify its plan in an attempt

to recover. The first type of knowledge structure will produce automatized, open-loop behavior for a given component of a plan or motor skill; the second will generate closed-loop behavior with respect to a plan fragment.

This unified framework should also let ICARUS learn to distinguish between these two situations. Like MAGGIE, the system would begin in closed-loop mode, monitoring its execution as much as possible and comparing expected states to its observations. Along the way, it would accumulate statistics about whether particular actions (in the context of a given class of plans) produce reliable results. In nonreactive domains, the stored plans will make accurate predictions, and ICARUS will develop automatized skills that require little monitoring and can be executed rapidly. In domains with uncertain operators or external forces, the acquired probabilities will encourage the system to remain in closed-loop mode, telling it where and when to look for potential problems. In many domains, the overall behavior will be some mixture of these two execution styles.

4.5 Directed and Distractable Behavior

In the previous subsection, we examined a notion of reactivity that detected when a plan to achieve some goal went astray, so that repairs could be made. Another type of reactivity involves interruptions in the pursuit of one goal by another (presumably more important) one. This introduces another dichotomy in systems that interleave planning with execution, based on the issue of *interruptibility*. At one extreme, there are highly directed systems, which pursue their original goal in a single-minded manner, independent of other developments in the environment.[5] At the other, there are highly distractable systems, in that they are driven entirely by the current situation and have no long-range view (at least not explicitly).

Clearly, both directed and distractable modes have roles to play in a general intelligent agent. Reflex-like techniques are often viewed as central to real-time behavior, since they generally require much less computation than deliberative schemes, and thus can respond in reasonable time to environmental changes. For example, one does not want to think about an upcoming highway exit when another car has come into the lane going the wrong direction. In such cases, distraction from one's current explicit goal has clear survival value. However, directed deliberation is essential in domains that require a more global perspective. For instance, local decisions can easily lead to dead ends in many navigation problems.

The obvious response is for an agent to incorporate both forms of control. In a system of this sort, directed reasoning leads to the systematic generation of action sequences, as in traditional planning methods. However, the agent also monitors the environment, and reflexes or similar mechanisms can interrupt the deliberation process when the need arises (e.g., Payton, 1990). This shifts the system's attention to a new problem, which becomes the focus of additional deliberation.[6] After this goal has been achieved, the agent may attempt to return to the original problem, though memory

---

5. Such a system can still be reactive, in the sense that it may change its plan and thus its subgoals in response to violated expectations, provided they are relevant to the top-level problem.

6. This account presents an overly negative picture, in that interruptions need not interfere with achieving a current goal. In some cases, they can actually suggest better ways of solving the initial problem. Hayes-Roth and Hayes-Roth (1979) report evidence of such 'opportunistic' planning in humans.

limitations can make this difficult for humans. In some domains, rapid and unexpected changes in the environment can lead to frequent distractions, with constant interruptions and responses to them produces long chains of behavior.

DÆDALUS clearly falls at the directed end of the spectrum, making interruptions the real challenge in developing ICARUS. To allow for distractions, we plan to associate priorities with each problem in memory. On each cycle, ICARUS would attend to that problem with the highest priority, basically using an agenda to focus problem-solving attention. Rather than using an explicit depth-first search regimen, control would pass from problem to subproblem through propagated priority scores, causing the parent to become suspended until the child had been solved or abandoned.

In this framework, classification of a new state – using the LABYRINTH algorithm described in Section 4.1 – could lead to retrieval of a stored problem that differs from the one currently being pursued. This could occur if the state is similar to the initial state of a stored problem, and if the overall match is better than that against the active problem and its predicted state. For instance, an oncoming car would match the initial state of a 'swerve' problem better than the predicted state of driving along an unobstructed highway. If the retrieved problem had higher priority (either positive or negative) than the currently active one, the latter would be suspended and the agent would pursue the more urgent goal. Presumably, this would be the situation in the case of an oncoming vehicle. Once this task has been handled, control would pass back to the original problem, unless another one had taken over in the meantime.

In this view, there is no a priori distinction between plans and reflexes; ICARUS stores a single type of data structure, which can be retrieved in either a goal-driven, directed manner or in a stimulus-driven, distractable manner. This account also explains the origin of top-level problems, which are generated when the agent encounters a familiar state that invokes a stored reflex. Of course, the system must start somewhere, so we assume that it begins with some innate reflexes, which it uses to generate top-level tasks in the early stages of its development.

In Section 4.1 we proposed a learning method for storing new plans and organizing them in memory. Since reflexes would be encoded in the same data structures, ICARUS requires no additional mechanism for their acquisition. However, it does need some means to estimate the priority scores associated with each stored plan. Our response to this issue involves associating priorities with each state in the concept hierarchy, as well as with plans. Upon classification, a new state would inherit the priority associated with its parent. If this state occurs at the end of a plan or subplan, ICARUS would use a variant of Sutton's (1990) temporal difference method to revise the priorities associated with states and plans that led to this situation. These changes would also propagate upward through the concept hierarchy, altering the scores for abstract plans and states. Over time, the score for a plan class would come to predict the scores for its final states. After sufficient experience, a plan would receive high priority to the extent that its likely outcomes have high priority for the agent.

Furthermore, we will assume that states (and thus plans) can have either positive priority (if desirable) or negative priority (if undesirable). In this framework, a plan would not 'succeed' or 'fail', but would simply produce more or less desirable states. As a result, ICARUS would be able to store all experience in a single type of knowledge structure, without need for arbitrary labeling

schemes. Moreover, this should let the system learn from both successes and failures using a single mechanism, Sutton's temporal different method. This is a very different approach than that taken by explanation-based systems like PRODIGY and SOAR, which reason about success and failure as separate types of events.

5. Conclusions

In this chapter we described DÆDALUS, a system that improves its ability to plan with experience. We presented the system in terms of its position on four dichotomies that exist in the literature on planning and learning. We found that to generate plans, DÆDALUS employs a flexible version of means-ends analysis that incorporates aspects of forward chaining approaches. We also saw that both search and memory play cen*ral roles in DÆDALUS behavior, which we cast as problem-space search constrained by knowledge in memory. Moreover, this memory includes both specific cases and abstractions of those cases, which the system organizes in a probabilistic concept hierarchy. Finally, we noted that DÆDALUS unifies notions of data-driven and knowledge-driven learning, in that its initial concept hierarchy biases the concepts it induces, and that knowledge acquired during earlier learning affects later learning.

Our experimental evaluation of DÆDALUS revealed that its learning mechanisms do lead to reduced search during plan generation. In addition, the model accounts for a nu .ber of high-level behaviors observed in human problem solving. However, we also found that the system reduces search only at the expense of increased retrieval cost, and that it fails to model some important psychological phenomena. These limitations suggested some natural directions for further research.

In response, we presented our designs for ICARUS, an integrated architecture for intelligent agents that would subsume its predecessor. Unlike DÆDALUS, the extended system would store entire derivational traces in its concept hierarchy, along with abstractions of these experiences. In addition to synthesizing notions of induction and explanation, this approach would unify a variety of additional forms of behavior. These would include:

- the use and acquisition of structures that support search heuristics, macro-operators, and derivational analogy;
- the interleaving of planning and execution, based on the grounding of plans in temporal sensorimotor descriptions;
- the selective invocation of closed-loop and open-loop processing, and the transition from the former to the latter through a mechanism of automization;
- the interruption of ongoing problems through a mixture of directed and distractable behavior, and the acquisition of problem priorities based on plan results.

Thus, ICARUS promises to cover a much wider range of behaviors than its predecessor, unifying many aspects of planning, execution, perception, and learning in a single integrated framework. Although we have only started on the path from design to implementation, we are confident that it leads to a robust architecture that will be consistent with knowledge of human behavior while providing robust control for a physical agent.

Acknowledgements

References

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

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

Chi, M. T. H., Glaser, R., & Rees, E. (1982). Expertise in problem solving. In R. J. Sternberg (Ed.), *Advances in the psychology of human intelligence*. Hillsdale, NJ: Erlbaum.

Clark, P. & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, *3*, 261–284.

Drastal, G., Czako, G., & Raatz, S. (1989). Induction in an abstraction space: A form of constructive induction. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 708–712). Detroit, MI: Morgan Kaufmann.

Drummond, M., & Bresina, J. (1990). Planning for control. *Proceedings of the Fifth IEEE International Symposium on Intelligent Control* (pp. 657–662). Philadelphia, PA: IEEE Computer Society Press.

Elio, R., & Watanabe, L. (in press). An incremental deductive strategy for controlling constructive induction in learning from examples. *Machine Learning*.

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

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

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

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, *2*, 139–172.

Flann, N. S., & Dietterich, T. G. (1989). A study of explanation-based methods for inductive learning. *Machine Learning*, *4*, 187–226.

Forbus, K. D. (1985). Qualitative process theory. In D. G. Bobrow (Ed.), *Qualitative reasoning about physical systems*. Cambridge, MA: MIT Press.

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

Hammond, K. J. (1990). Case-based planning: A framework for planning from experience. *Cognitive Science*, *14*, 385–443.

Hayes-Roth, B., & Hayes-Roth, F. (1979). A cognitive model of planning. *Cognitive Science*, *3*, 275–310.

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

Iba, W., & Langley, P. (1987). A computational theory of human motor learning. *Computational Intelligence*, *3*, 338–350.

Iba, W. & Gennari, J. H. (in press). Learning to recognize movements. In D. H. Fisher & M. Pazzani (Eds.), *Computational approaches to concept formation*. San Mateo, CA: Morgan Kaufmann.

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

Jones, R. (1990). *A probabilistic approach to learning in planning*. Unpublished manuscript, Department of Computer Science, University of Pittsburgh, Pittsburgh, PA.

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

Kolodner, J. L., Simpson, R. L., & Sycara, K. (1985). A process model of case-based reasoning in problem solving. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 284–290). Los Angeles, CA: Morgan Kaufmann.

Kotovsky, K., Hayes, J. R., & Simon, H. A. (1985). Why are some problems hard? Evidence from tower of Hanoi. *Cognitive Psychology*, *17*, 248–294.

Keele, S. W. (1982). Learning and control of coordinated motor patterns: The programming perspective. In J. A. S. Kelso (Ed.), *Human motor behavior: An introduction*. Hillsdale, NJ: Erlbaum.

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

Laird, J. E., Hucka, M., Yager, E. S., & Tuck, C. M. (1990). Correcting and extending domain knowledge using outside guidance. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 270–283). Austin, TX: Morgan Kaufmann.

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

Langley, P., Thompson, K., Iba, W., Gennari, J. H., & Allen, J. A. (in press). An integrated cognitive architecture for autonomous agents. In W. Van De Velde (Ed.), *Representation and learning in autonomous agents*. Amsterdam: North Holland.

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

Minton, S. N. (1990a). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, *42*, 363–391.

Minton, S. N. (1990b). Issues in the design of operator composition systems. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 304–312). Austin, TX: Morgan Kaufmann.

Minton, S., Carbonell, J. G., Knoblock, C. A., Kuokka, D. R., Etzioni, O., & Gil, Y. (1989). Explanation-based learning: A problem solving perspective. *Artificial Intelligence*, *40*, 63–118.

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

Mitchell, T. M., Allen, J., Chalasani, P., Cheng, J., Etzioni, O., Ringuette, M., & Schlimmer, J. C. (in press). THEO: A framework for self-improving systems. In K. VanLehn (Ed)., *Architectures for intelligence*. Hillsdale, NJ: Lawrence Erlbaum.

Mooney, R. (1990). *A general explanation-based learning mechanism and its application to narrative understanding*. San Mateo, CA: Morgan Kaufmann.

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

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

Newell, A., Shaw, J. C., & Simon, H. A. (1960). Report on a general problem-solving program for a computer. *Proceedings of the International Conference on Information Processing* (pp. 256–264).

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

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

Payton, D. (1990). Exploiting plans as resources for action. *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control* (pp. 175–180). San Diego, CA: Morgan Kaufmann.

Pearl, J. (1984). *Heuristics*. Reading, MA: Addison-Wesley.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81–106.

Rosenbaum, D. A., Kenny, S., & Derr, M. A. (1983). Hierarchical control of rapid movement sequences. *Journal of Experimental Psychology: Human Perception and Performance*, *9*, 86–102.

Ross, B. H. (1984). Remindings and their effects in learning a cognitive skill. *Cognitive Psychology*, *16*, 371–416.

Schneider, W., & Eberts,, R. (1980). *Consistency at multiple levels in sequential motor output processing* (Technical Report 80–4). Urbana: University of Illinois, Human Attention Research Laboratory.

Shiffrin, R. M., & Dumais, S. T. (1981). The development of automatism. In J. R. Anderson (Ed.) *Cognitive skills and their acquisition*. Hillsdale, NJ: Erlbaum.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on ap-
proximating dynamic programming. *Proceedings of the Seventh International Conference on
Machine Learning* (pp. 216–224). Austin, TX: Morgan Kaufmann.

Tambe, M., Newell, A., & Rosenbloom, P. S. (1990). The problem of expensive chunks and its
solution by restricting expressiveness. *Machine Learning*, *5*, 299–348.

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. (1989). Learning analogies by analogy – the closed loop of
memory organization and problem solving. *Proceedings of the DARPA Workshop on Case-
based Reasoning* (pp. 153–158). Pensacola Beach, FL: Morgan Kaufmann.

Williams, B. C. (1985). Qualitative analysis of MOS circuits. In D. G. Bobrow (Ed.), *Qualitative
reasoning about physical systems*. Cambridge, MA: MIT Press.

Yoo, J., Yang, H., & Fisher, D. H. (in press). Concept formation over explanations, plans, and
problem-solving experience. In D. H. Fisher & M. Pazzani (Eds.), *Computational approaches
to concept formation*. San Mateo, CA: Morgan Kaufmann.