CHAPTER 10

A Unified Framework for Planning and Learning

PAT LANGLEY
JOHN A. ALLEN

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) General Problem Solver (GPS) model of human problem solving, and very similar methods have been used in Minton et al.'s (1989) PRODIGY and Jones's (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 is related to 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, that reveals some strengths and some limitations. Next, 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 with our discussion of DÆDALUS, we base 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.

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

Table 10–1. Pseudocode for Means-Ends Analysis

```
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.
  Else return Failed.
```

Note: Means-ends analysis is 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 possible.

current and desired states, selects an operator that reduces the 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 10–1 gives details on this approach to problem solving. Typically, the

method also returns any successful plan it generates, though we do not show this in the table. 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 process leads to a form of backward chaining, in that the order of operator selection is the reverse of the application order. In other cases, this strategy produces forward chaining, in that the 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 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, because of their focus on relevant operators and their ability to break problems into useful subproblems. However, traditional means-ends systems examined only one difference at a time, and they did not consider the relation between operators' preconditions and the current state. In contrast, Dædalus (like some other recent means-ends systems) uses a variation (which we call *flexible means-ends analysis*) that prefers operators that 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 10–1, differing from early means-ends methods only in its instantiation of the `Select` procedure.[1]

1. DÆDALUS borrows the notion of flexible means-ends analysis from Jones's (1989) EUREKA system, which used a very similar idea with a quite different retrieval method. Jones's (1990) more recent GIPS system also uses a similar strategy. Minton et al.'s (1989) PRODIGY incorporates multiple differences and state information into its learned rules, but it requires that they match completely before using them.

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 hypothesis 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.

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 & Fisher, 1991; 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*, a tree of concepts that summarize experience at different levels of abstraction using probabilistic descriptions. Initially, this hierarchy contains only 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 10–1 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

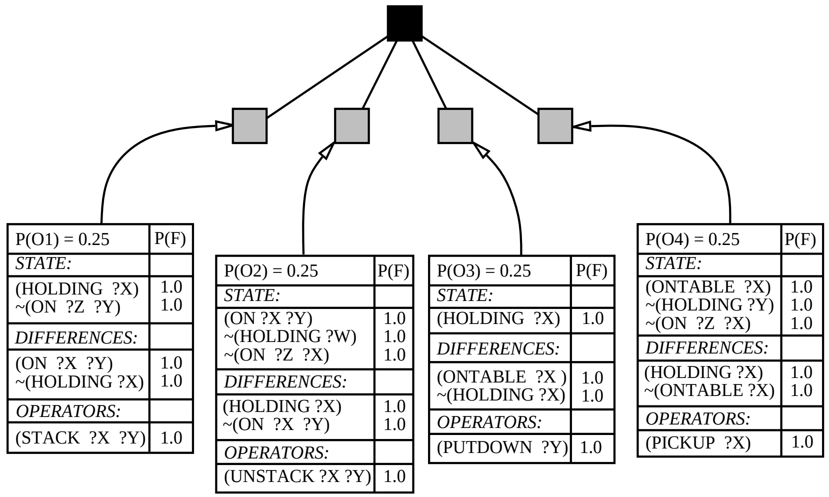

Figure 10–1. Initial concept hierarchy provided to Dædalus for the blocks world domain. Terminal nodes (shown with their descriptions) correspond to generic operator schemas.

upon application, and its name and arguments. The root of the hierarchy contains a probabilistic average of all nodes below it; terminal nodes are described in the same language, but all their probabilities are 1. 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 conditional probability of each operator being useful given the class of situations.

The retrieval of operators involves sorting a problem—described as a set of state descriptors and differences—through this concept hierarchy. To do this sorting, 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; this type of match 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 10–2 shows a blocks world problem that the system cannot solve without search, given the initial hierarchy in Figure 10–1, along with the structure of an optimal problem-solving trace provided to the system by an expert (the

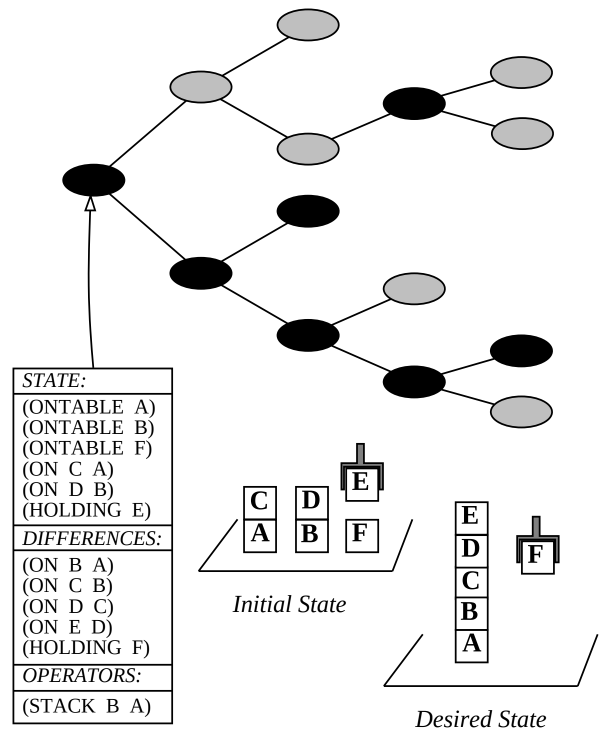

Figure 10–2. A problem from the blocks world, along with an optimal problem-solving trace given to Dædalus. Each node consists of a state description, a set of differences, and the selected operator. Black nodes indicate problems on which the system selected the incorrect operator during training; white nodes specify problems on which it made the right selection.

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. Yoo and Fisher (1991) describe a closely related approach to combining cases and abstractions for problem solving.

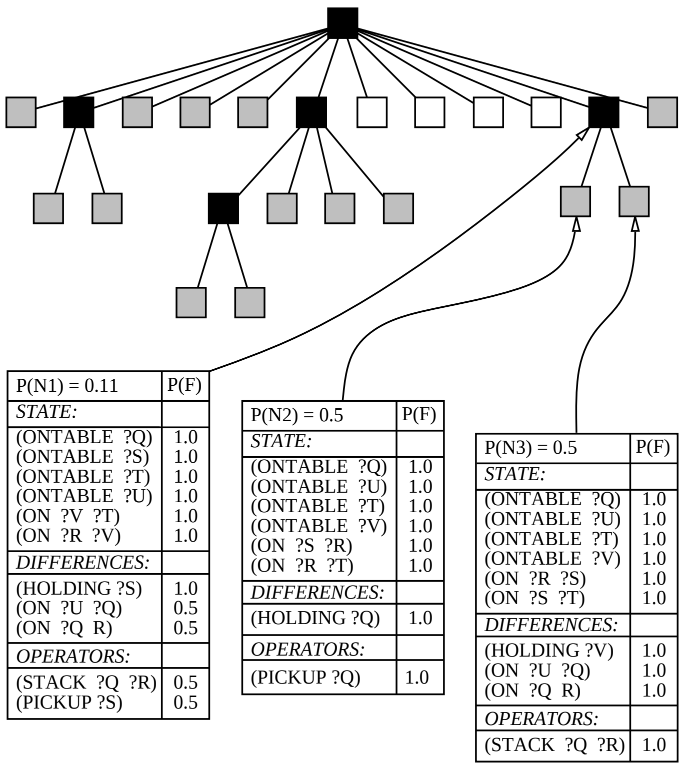

Figure 10–3. Revised DÆDALUS concept hierarchy. This hierarchy incorporates cases (gray) and abstractions (black) resulting from storage of components from the problem-solving trace in Figure 10–2, along with the original operator schemas (white) for this domain.

Figure 10–3 shows the modified hierarchy after DÆDALUS has incorporated its experience with the problem in Figure 10–2. Each new case (in gray) represents one of the problems or subproblems in the problem-solving 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. 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, Czako, & Raatz, 1989; Elio & Watanabe, 1991). 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 and Fisher's (1991) related work. Figure 10-4 illustrates the four learning operations that lead to changes in the structure of memory. These operations include the following:

- *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 is 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 that is 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 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.

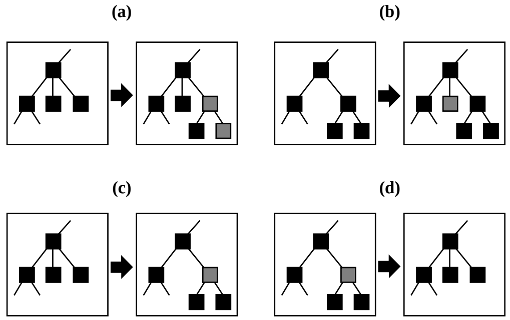

Figure 10–4. Operations used by the COBWEB$_R$ routine to alter the structure of DÆDALUS' hierarchy: (a) extending the tree downward, (b) creating a new disjunct, (c) merging two concepts, and (d) splitting an existing concept.

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 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 to improve performance, and one can run experiments with any learning system to determine whether it achieves this goal (Kibler & Langley, 1988). To this end, we designed studies with Dædalus to determine the effect of experience on its ability to plan. Five measures of performance suggested themselves: (1) the number of solved problems (within a computational limit), (2) the amount of search on solved problems (the number of expanded nodes divided by the number of solution steps), (3) the quality of solutions (the number of steps), (4) the overall planning time (the total number of unifications), and (5) the accuracy of the learned heuristics. These measure different aspects of planning behavior, and they served as the dependent variables in our experiments.

Our first domain involved navigation through a space of qualitative regions (Levitt, Lawton, Chelberg, & Nelson, 1987). Figure 10–5 depicts the space used for this study, in which there are five unique landmarks. These objects generate a number of distinct "places," which are defined as regions in which the pairwise spatial relations of all objects are constant. For instance, in a domain containing objects A, B, and C, one place would be defined as {`(left-of A B)`, `(left-of A C)`, `(left-of B C)`}, and another would be defined as {`(left-of A B)`, `(left-of C A)`, `(left-of B C)`}. An operator takes an agent from its current place to an adjacent place. In the five-object domain, each operator includes ten preconditions, one for each pairwise relation. When arranged in a regular pentagon, as shown in the figure, the five objects generate 35 distinct places (excluding the center), with 90 operators connecting them to each other. Navigation tasks in this space can be readily solved by a means-ends approach, but difficulty can be increased by removing some of the operators to simulate walls.

To study Dædalus' behavior on qualitative navigation, we generated two sets of training and test problems, at two levels of difficulty. For the first level, shown in Figure 10–5(a), we included 82 of the 90 possible operators for moving between adjacent places, making it relatively easy to find paths with little search. For the more difficult level, in Figure 10–5(b), we included only 57 operators. For each condition, we ran the system on ten randomly selected training problems, measuring its performance after every two problems on a separate set of ten test problems, and averaged the results over ten different training orders.[2] Furthermore, we constrained the number of branches searched at any choice

2. This approach generates learning curves, which are quite different from the cumulative curves reported by Minton (1990a) and others.

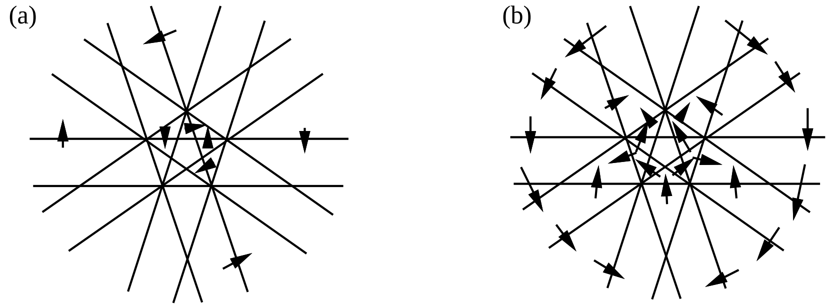

Figure 10–5. A navigation domain based on qualitative relations between five landmarks, with: (a) 82 adjacent move operators, and (b) 57 operators. (Arrows denote one-way passage.)

point to three. During training, we operated Dædalus in "learning apprentice" mode, providing it with the optimal problem-solving trace for each problem; we did so primarily to avoid the variation that would result from nonoptimal traces the system might find through search.

Figure 10–6(a) displays the amount of search as a function of Dædalus' training experience for the two levels of difficulty. The learning curves show that initially the system's search increases, but as it gains experience it becomes more directed. Moreover, this result becomes stronger as the domain becomes more difficult. When there are fewer operators (and thus fewer paths between places), the novice Dædalus is forced to carry out more search to solve the test problems. However, learning systematically reduces the search required, until it nearly always selects a useful operator independent of the difficulty level. The results for overall planning cost follow a similar trend, suggesting that retrieval time was constant or grew slowly.

However, these dependent measures do not tell the entire story. The results in Figure 10–6(a) are based only on test problems that the system solved successfully within the computational limits we set (200 search nodes). Figure 10–6(b) shows corresponding learning curves in which the number of solved problems is the performance measure. From the figure, we see that learning increases the number of problems solved in both domains. Also note that the increase in the number of solved problems corresponds to the increase in the amount of search depicted in Figure 10–6(a). The learning done in the first few training problems

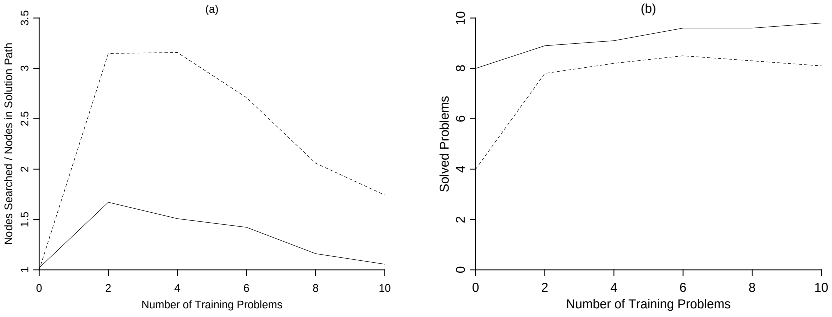

Figure 10–6. Learning curves for DÆDALUS on problems from the navigation domain, measuring (a) amount of search, and (b) number of solved problems as a function of experience. The solid line shows the behavior on easy problems; the dotted line shows behavior on harder problems.

helps DÆDALUS increase the number of problems it can solve but does not provide enough information to solve these new problems without search. As the system gains expertise, the amount of search needed to solve new problems gradually decreases. Despite these changes in search and coverage, we found no improvement in solution quality; the number of solution steps was approximately the same before and after learning. Nevertheless, our results with DÆDALUS in this domain were generally positive, indicating that its stored cases and abstractions aid the system in generating plans.

Our second domain was a version of the blocks world that involved four operator schemas: (1) picking up a block from the table, (2) putting a block on the table, (3) unstacking a block from another block, and (4) stacking a block on another block. Problems required the transformation of one arbitrary configuration of blocks into another configuration. We included this domain because it has been used in other studies of learning in planning (e.g., Minton, 1990a). Also, the presence of variables in its operators raises additional issues of retrieval efficiency. In this case, we used problems that contained between three and five blocks and goal states that included between one and five conditions. As before, we ran the system in learning apprentice mode on ten training problems, measuring its performance after every two problems on a separate set of nine test problems. In this case, we did not select

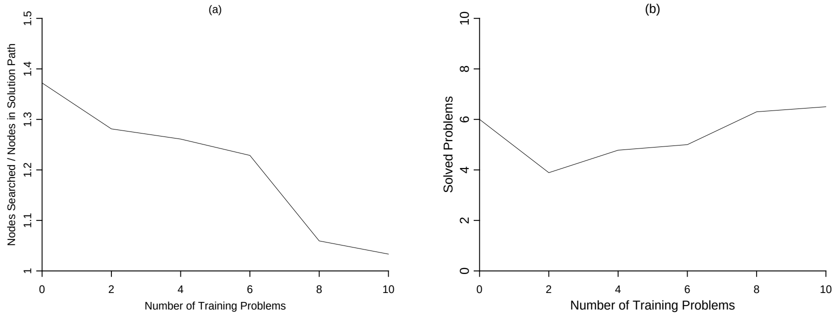

Figure 10–7. The change in DÆDALUS' performance on problems from the blocks world, measuring (a) amount of search, and (b) number of solved problems as a function of learning compared to the same system without learning.

them randomly; rather, we selected both training and test problems that required some search before learning.

Figure 10–7(a) shows the effect of learning on search in this domain. As DÆDALUS gains experience in the blocks world, it expands fewer nodes. In fact, by the tenth training problem, the number of nodes considered nearly equals the number in the solution, indicating an absence of search. Again, this figure reflects only problems that DÆDALUS successfully solved, so we augment it with Figure 10–7(b), which reports the number of problems solved at different stages in learning. In this domain, the results for this second measure are more ambiguous. DÆDALUS begins by solving an average of six problems out of nine, drops to only four after two training problems, and then gradually rises to more than six by the eighth training case.

To determine the source of this temporary decrease, we designed another experiment that would measure 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 we ran the system in learning apprentice mode during both training *and* testing, 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. The results showed that DÆDALUS clearly improved its ability to select the right operators, starting with a 64% chance of making the correct retrieval and steadily moving up to the 91% level. Apparently, early learning did not affect overall retrieval accuracy, but it did introduce occasional retrieval errors that, when not corrected by a tutor, could send Dædalus down fruitless paths on which it exhausted its allocation of search nodes. An alternative search organization like iterative broadening (Ginsberg & Harvey, 1990) might alleviate this problem.

Unlike the navigation domain, learning in the blocks world led to a clear improvement in solution quality. Dædalus' initial solution length in this domain was 8.6 steps on solved problems; by the tenth training problem, this had dropped to 5.8 steps. However, this decrease was offset by an actual increase in the overall planning time, as measured by the number of unifications. Inspection revealed that this increase resulted from an increase in retrieval cost; although retrieval time increased only linearly with the number of training problems, its constant was enough to offset the savings that resulted from reduced search and improved solutions. Thus, Dædalus appears to suffer from its own version of the "expensive chunk" problem (Tambe, Newell, & Rosenbloom, 1990), and we should address this issue in future work. In summary, the results in the blocks world were more ambiguous than for the navigation domain, but again we found that Dædalus did improve on some measures of planning performance.

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 learning. These phenomena are qualitative in nature, but they still provide constraints on the operation of cognitive simulations. Table 10–2 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's (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

Table 10–2. Psychological Adequacy of Dædalus and Other Models of Learning in Problem-Solving Domains.

	Act	Soar	Eureka	Dædalus
Means-ends analysis	⊙	⊕	⊕	⊕
Nonsystematic search	⊖	⊖	⊕	⊖
Problem isomorphs	⊙	⊙	⊙	⊙
Gradual improvement	⊕	⊕	⊕	⊕
Asymmetric transfer	⊕	⊕	⊕	⊕
Einstellung	⊕	⊕	⊕	⊕
Reduced verbalization	⊕	⊕	⊖	⊖
Automization	⊙	⊙	⊖	⊖
Rarity of analogy	⊖	⊖	⊕	⊕
Superficial analogy	⊖	⊖	⊕	⊕

Note: 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.

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 improvement 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 resulting from 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 and 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 decrement 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 (1989) has produced similar results with EUREKA using quite different mechanisms. Runs with DÆDALUS have likewise shown this behavior, and Soar should as well since it reuses structures acquired in earlier problems.

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 problem-solving 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 systems 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 strategy 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 in which they do manage to retrieve a relevant problem, the reminding is usually based on some 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 doing so 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. In evaluating Dædalus as a practical learner, we measured improvement in terms of solved problems, amount of search, solution quality, and overall planning cost. We found that learning systematically reduced search but that improvement on the other measures was less robust. In particular, 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 (1986) Soar and Anderson's Act in its coverage of analogical reasoning, an area it shares with Jones's Eureka system. However, it fails to explain the reduction of verbalization and the automatization observed in highly skilled human problem solvers, which the other systems at least partially model. Moreover, Dædalus' search organization does not mimic the nonsystematic behavior found in that of 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 briefly 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, McKusick, Allen, Iba, & Thompson, 1991), 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 trace of the means-ends process that specifies relations among problems and subproblems, then ignores this structure during the learning process. In contrast, research on methods that carry out induction over entire explanation structures (e.g., Flann & Dietterich, 1989; Yoo & Fisher, 1991) suggests a fuller unification of the data-driven and knowledge-driven views. Our designs for Icarus include a similar scheme in which entire problem-solving traces are the input to the learning process. However, these more complex structures require 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 (1991) Labyrinth, a system that classifies and learns about objects with componential structure. In this case, each object would correspond to a means-ends plan that specifies an initial state, a desired state, an operator, and two subproblems with similar structure.

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. Such features of composite concepts can be viewed as separate *roles*. For instance, the First-Subproblem role for an abstract plan would point to one or more subplans (themselves nonterminal nodes in the hierarchy), as well as specifying the probability of each. Icarus would use Labyrinth both for retrieving similar problems and for storing traces of problem-solving solutions. Initial retrieval would be based on differences and states in the top-level problem, as in Dædalus, since at this point the system would have no other information. The use of problem-solving traces does allow more sophisticated planning strategies, but we delay their discussion until the next subsection. As in Cobweb, the storage process would be interleaved with classification, with Labyrinth incorporating each plan component into memory in turn. Elsewhere (Thompson & Langley, 1991), we describe the system's response to other important issues, including the binding of variables during matching and the replacement of alternative components with their generalizations.

In summary, our designs for Icarus call for the storage of entire problem-solving 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 Control, Analogy, and Macro-Operators

Researchers have explored three basic approaches to the acquisition of planning expertise. Like Dædalus, many systems acquire control knowledge to direct search at each step in plan generation (e.g., Jones, 1989; Laird et al., 1986; Minton et al., 1989). A second approach stores composite structures that are used as macro-operators to lessen the effective length of solution paths (e.g., G. Iba, 1989; Mooney, 1989). Finally, research on analogical planning and case-based reasoning uses similar structures but adapts them to the task at hand (e.g., Veloso & Carbonell, 1989; Hammond, 1990). Although, in principle, search control knowledge can produce overall behavior similar to that of analogy and macro-operators, one can imagine efficiency reasons for using composite structures (Minton, 1990b), and timing studies with humans suggest they store such structures (Rosenbaum, Kenny, & Derr, 1983).

However, rather than view these three methods as competitors, we prefer to cast them as lying along a continuum that varies in the conditions of reuse. Derivational analogy and case-based reasoning fall toward the center of this spectrum, in that they check each component of a retrieved plan for its appropriateness to the current subproblem. If a plan component is deemed appropriate, they use it to solve the subproblem; otherwise, they retrieve some other knowledge that handles the subproblem. In contrast, methods that treat plan knowledge as macro-operators do not bother to test for relevance; they simply assume that all components are appropriate for use on the current problem. Search control methods also dispense with such tests, but they do so because they assume that subplans are never relevant, and so they always search memory for knowledge they can apply to the current subproblem. Thus, the continuum runs from automatic reuse of components, through conditional reuse (which requires testing), to automatic nonreuse.

Recall that our designs for Icarus include the storage of complete problem-solving traces, but storing them does not mean the architecture *must* use the entire structure in all situations. Upon retrieving a plan from memory, Icarus would have the option of using each stored subplan automatically (as though the plan were a macro-operator), upon the condition that its subplans match the current subproblems well enough (as in analogy), or ignoring the subplans and sorting the subproblems through memory (as done in Dædalus). In determining which mode to pursue, Icarus would consult statistics stored with each subplan that specify the percentage of the subplan's structure that it can expect to reuse. If the score for a subplan is high, Icarus would simply reuse its operator without bothering to test the subplan for relevance. If the score is high for the entire abstract problem-solving trace, it would be treated like a macro-operator. On the other hand, if a subplan's score is low, the system would sort the subproblem through memory to retrieve a promising operator. If the score is low for each retrieved subtrace, the overall behavior would be equivalent to using search control knowledge. Finally, given an intermediate score, Icarus would base its decision on a comparison of the current subproblem and the subplan, simulating a form of derivational analogy. An intermediate score would be the default early in the system's experience with a domain, before it had acquired reliable estimates on reuse. Of course, arbitrary mixtures of these strategies could occur as well, if this were appropriate for the current planning domain. Thus, Icarus would combine notions of search control, analogy, and macro-operators within a single framework.

4.3 Planning and Execution

A physical agent must do more than generate possible action sequences; it must also be able to *execute* or enact 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., 1991), and if we want Icarus to serve as the basis for a robust agent, we must address it within our framework.

The above means that plans must somehow be grounded in executable actions. The examples in Section 2 relied on an abstract, logical formalism, as has most planning research. In contrast, most work on robotics and control assumes sensorimotor descriptions that specify locations and velocities of physical objects and limbs. We hope to combine these schemes by grounding states, operators, and plans in sensorimotor descriptions. Physical agents (including humans) exist not only in space but also in time, and they must deal with objects, situations, and actions that have duration. Unlike most AI work on planning, which assumes that states last indefinitely and that operators are instantaneous, Icarus would represent both as *qualitative states* (Forbus, 1985; Williams, 1985), which are intervals of time during which the qualitative structure of a situation remains unchanged. This constraint 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 would 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, each state would specify an expected observation and each operator would indicate an executable action. In this view, motor skills such as throwing a ball or swinging a bat (W. Iba & Gennari, 1991) are simply very detailed plans.

The architecture must also take some stance on *when* to execute a plan or plan fragment. In some cases, an agent can safely carry out some actions even before constructing a complete plan, and it must determine when it has generated enough to begin execution. Icarus' storage of abstract problem-solving traces suggests a response to this issue. The means-ends pseudocode in Table 10–1 states that one should "apply" a selected operator as soon as its preconditions are met. Although normally interpreted as "mental application," one could actually execute each operator in the environment as soon as its preconditions were satisfied, if only backtracking were unlikely. Following a suggestion by J. Bresina (personal communication, 1990), we plan to augment Icarus' stored plans with information about the probability of backtracking. Suppose the system retrieves a plan for a given problem and experience shows that, if it generates a successful plan for the first subproblem, then the same is likely to happen for the second subproblem. In this case, Icarus 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 doing so by time demands or memory 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 problem-solving 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 problem-solving trace. In domains in which 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 also raised concern about the monitoring of changing environments. In some cases, an agent's actions may not have the desired effects or external forces may alter the agent's surroundings, and monitoring is the obvious response. A careful agent would compare predicted states to observed ones and, if the two disagree, respond 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 can execute some motor skills in an open-loop mode, running a "motor program" without external feedback. 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, and a unified theory of execution should support differing degrees of monitoring.

Our early work in this area (W. Iba & Langley, 1987) modeled the control of jointed limb movements but assumed that monitoring of limb's positions and velocities occurred at a constant rate. Our designs for Icarus support 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 would not bother to monitor the new state (by sorting it through memory) and would continue execution unabated. In uncertain cases, Icarus would classify the new situation and, if it diverged from the expected state, interrupt execution and modify its plan in an attempt to recover. The first type of knowledge structure would produce automatized, open-loop behavior for a given component of a plan or motor skill; the second would generate closed-loop behavior with respect to a plan fragment.

This unified framework should also let Icarus learn to distinguish between these two situations. Beginning in closed-loop mode, the system would monitor its execution as much as possible, compare expected states to its observations, and accumulate statistics about whether particular actions produced reliable results in a given plan. In nonreactive domains, the stored plans would make accurate predictions, and Icarus would develop automatized skills that require little monitoring and that can be executed rapidly. There is some evidence for such a transition from closed-loop to open-loop mode in human motor behavior (Keele, 1982; Schneider & Eberts, 1980). In domains with uncertain operators or external forces, the acquired probabilities would 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 would be some mixture of these two execution styles.

4.5 Directed and Distractable Behavior

Another dichotomy in systems that interleave planning with execution, related to monitoring and reactivity, involves *interruption*. At one extreme are highly directed systems, which pursue their original goal

in a single-minded manner independent of other developments in the environment.[3] At the other extreme are systems that are highly distractable, in that they are driven entirely by the current situation and have no explicit long-range view. Clearly, both directed and distractable modes have a role to play in a general intelligent agent. Reflex-like techniques are often viewed as central to real-time behavior since they require much less computation than deliberative schemes, but directed deliberation is essential in domains that require a more global perspective.

Our plans for Icarus support both directed and distractable behavior. To cover this continuum, the architecture would associate priorities with each problem in memory. On each cycle, Icarus would attend to the problem with the highest priority, using an agenda to focus its problem-solving attention. Rather than using an explicit depth-first search regimen, control would normally pass from problem to subproblem through propagated priority scores, causing the parent to become suspended until the child had been solved or abandoned. Events in the environment, which would be classified using the Labyrinth algorithm described in Section 4.1, could retrieve a stored problem that differs from the one being pursued. If the retrieved problem had higher priority than the currently active one, Icarus would suspend the latter and pursue the more urgent goal. Once this task had 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 would store a single type of data structure, which it could retrieve in either a goal-driven, directed manner or in a stimulus-driven, distractable manner.

Since the architecture would encode reflexes in the same data structures as plans, it would require no additional mechanism for their acquisition. However, it would 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 occurred 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

3. 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.

plans that led to this situation. These changes would 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 and thus receive priority to the extent that its likely outcomes are desirable for the agent.

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 central 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 number of high-level behaviors observed in human problem solving. However, we also found that the system sometimes reduces search at the expense of increased cost of retrieval 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 problem-solving 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, including the following:

- 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 framework. Although we have only started on the path from design to implementation, we are confident that it will lead to a robust architecture that will be consistent with knowledge of human behavior while providing robust control for a physical agent.

Acknowledgements

We thank other members of the ICARUS group—Wayne Iba, Deepak Kulkarni, Kate McKusick, and Kevin Thompson—for lengthy discussions that led to many of the ideas in this chapter. John Bresina, Mark Drummond, and Steve Minton also influenced our thinking. All of the above provided useful comments on an earlier draft.

References

Anderson, J. R. (1983). *The architecture of cognition*. Cambridge, MA: 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.

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). IJCAI. San Mateo, CA: Morgan Kaufmann (distributor).

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

Elio, R., & Watanabe, L. (1991). An incremental deductive strategy for controlling constructive induction in learning from examples. *Machine Learning*, *7*, 7–44.

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.

Ginsberg, M. L., & Harvey, W. D. (1990). Iterative broadening. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 216–220). Menlo Park, CA: AAAI Press.

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

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

Iba, W., & Gennari, J. H. (1991). Learning to recognize movements. In D. H. Fisher, M. J. Pazzani, & P. Langley (Eds.), *Concept formation: Knowledge and experience in unsupervised learning*. San Mateo, CA: Morgan Kaufmann.

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

Jones, R. (1989). *A model of retrieval in problem solving*. Doctoral dissertation, Department of Information and 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.

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: Lawrence Erlbaum.

Kibler, D., & Langley, P. (1988). Machine learning as an experimental science. *Proceedings of the Third European Working Session on Learning* (pp. 81–92). London: 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). IJCAI. Los Altos, CA: Morgan Kaufmann (distributor).

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

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). San Mateo, CA: Morgan Kaufmann (distributor).

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

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

Langley, P., McKusick, K. B., Allen, J. A., Iba, W. F., & Thompson, K. (1991). A design for the Icarus architecture. *SIGART Bulletin* *2*(4), 104–109. Stanford, CA: ACM Press.

Levitt, T. S., Lawton, D. T., Chelberg, D. M., & Nelson, P. C. (1987). Qualitative landmark-based path planning and following. *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 689–694). AAAI. Los Altos, CA: Morgan Kaufmann (distributor).

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). San Mateo, CA: 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., Allen, J., Chalasani, P., Cheng, J., Etzioni, O., Ringuette, M., & Schlimmer, J. C. (1991). Theo: A framework for self-improving systems. In K. VanLehn (Ed.), *Architectures for intelligence*. Hillsdale, NJ: Lawrence Erlbaum.

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.

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). Paris: UNESCO.

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

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

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: Lawrence Erlbaum.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 216–224). San Mateo, CA: 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.

Thompson, K., & Langley, P. (1991). Concept formation in structured domains. In D. H. Fisher, M. J. Pazzani, & P. Langley (Eds.), *Concept formation: Knowledge and experience in unsupervised learning*. San Mateo, CA: Morgan Kaufmann.

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). San Mateo, CA: 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., & Fisher, D. H. (1991). Concept formation over problem-solving experience. In D. H. Fisher, M. J. Pazzani, & P. Langley (Eds.), *Concept formation: Knowledge and experience in unsupervised learning*. San Mateo, CA: Morgan Kaufmann.