NASA-CR-192726

GRANT
IN-64-CR
153688

p, 51

Final Technical Report

NCC 2-645

(6/1/89 to 10/31/92)

Improving Performance Through Concept Formation and Conceptual Clustering

Douglas H. Fisher (Principal Investigator)

Box 1679, Station B
Department of Computer Science
Vanderbilt University
Nashville, TN 37235

(615) 343-4111

Vanderbilt University
Division of Sponsored Research
512 Kirkland Hall
Nashville, TN 37240

(NASA-CR-192726) IMPROVING PERFORMANCE THROUGH CONCEPT FORMATION AND CONCEPTUAL CLUSTERING Final Technical Report, 1 Jun. 1989 - 31 Oct. 1992 (Vanderbilt Univ.) 51 p

N93-25262

Unclas

G3/64 0153688

1 Short Summary of Research in Problem-Solving, Planning, and Diagnosis

Research from June 1989 through October 1992 focussed on concept formation, clustering, and supervised learning for purposes of improving the efficiency of problem-solving, planning, and diagnosis. These projects have resulted in two dissertations on clustering, explanation-based learning, and means-ends planning, and publications in conferences and workshops, several book chapters, and journals; a complete Bibliography of NASA Ames supported publications appears in Section 2. This section gives cursory overviews of each research tract, but more detailed descriptions of the various projects follow in subsequent sections.

1.1 Clustering of Explanations and Problem-Solving Experiences

Research by Jungsoon Yoo culminated in a system known as EXOR (EXplanation ORganizer), which clusters problem-solving experiences into similarity classes to facilitate reuse of past experience in new situations (10, 11, 12, 19, 25, 28). The system combines and unifies inductive and explanation-based learning. The system begins with a domain theory which describes the basic rules of inference for a domain. As problems are solved using this domain theory, solutions are incrementally clustered into an abstraction hierarchy of solutions. The hierarchy identifies and segregates distinct solution types along the horizontal dimension of the hierarchy. The vertical dimension of the hierarchy serves to specialize general solutions into more specialized solution categories. As the hierarchy is constructed, subsequent problems are categorized relative to it, and partial solutions uncovered during this categorization are accumulated into a complete solution for the new problem. Problem-solving only turns to the original domain theory when the abstraction hierarchy is incomplete, in the sense that it does not support a complete solution to the problem. With training, the relatively efficient process of categorization and reuse displaces the expensive process of domain theory search, thus improving overall problem-solving efficiency.

The contributions of this work are several fold. First, we have examined explanation-based learning and reuse in the context of multiple solution types. That is, many studies have examined the speedup brought about by explanation-based learning for solving (or proving) single goals, predicates, or problem types. In contrast, EXOR is a complete problem-solving architecture that shows how problem solving over a problem set exhibiting multiple top-level goals can be coordinated. Methodologically, the work breaks up the measurement of problem-solving effort into the effort contributed by domain theory search and the effort contributed by reusing previous, learned knowledge. Generally, research on problem-solving and explanation-based learning collapses these components into a single measure of overall effort. The work also contributes to some specific issues of explanation-based learning; notably, it links the operationality or usability of learned knowledge to its *cost-effectiveness* by a formal ratio of the probability that a solution category is likely to lead to successful problem solving, to the problem-solving cost that is likely to be accrued by reuse of the generalized solution associated with that category (10, 25). The research also has implications for human problem solving (20).

1.2 Clustering and Means-Ends Planning

Research in planning by Hua Yang has resulted in PLOT, a system that clusters macro operators into similarity classes in order to facilitate reuse (4, 13, 15, 18, 27). In spirit, it is very similar to Yoo's work as described above, but it is designed for means-ends planning and it adds to the 'clustering for reuse' paradigm in other significant ways. In particular, PLOT clusters macro-operators obtained through means-ends analysis based on their applicability (i.e. PRE and ADD) conditions. During performance, previously-clustered operators are retrieved based on their applicability to the current situation, and can be used as is or modified to fit the current situation. Thus, the system combines macro-learning ability, with search control learning, and certain case-based abilities to adapt previous plans to an existing situation. The combination of these capabilities leads to significant speedups in planning performance. The ability to modify macro-operators to better fit the current situation is a major conceptual difference between PLOT and EXOR. The modification of macro-operators retrieved during planning is facilitated by storing the plan derivation tree along with each macro. This tree can be efficiently reorganized, and new macros derived, if circumstances warrant modification.

The primary contributions of PLOT lie in its unification of several planning paradigms - namely means-ends and case-based planning. In particular, previous case-based planners rely almost exclusively on a case library of user-coded plans. If this case library is incomplete then planning cannot proceed on certain tasks. By suitably combining case-based abilities with weak (means-ends) methods, the system can circumvent the incompleteness problem, while still using past cases, when relevant, to improve planning efficiency. In addition, PLOT uses a similarity metric (as does EXOR), which effectively serves as search-control knowledge for macro-operator retrieval. The system is relatively novel in its learning of search control knowledge for macro-operator application; in contrast, most 'learning to plan' systems rely either exclusively on search control or macros to improve efficiency. By learning both, the system reduces the effective branching factor of search (through improved search control), and the effective depth of search (through macro-operators).

1.3 Diagnosis of Space Shuttle and Space Station Operating Modes

Accurate and timely fault diagnosis is critical in the life cycle of many physical systems. Methods of *machine induction* can be used to identify system parameters that are good candidates for dynamic probing and continuous sensing over a population of diagnostic episodes. More specifically, we employ clustering to identify similarity classes of system behavior in terms of observable system quantities (2, 5, 6). These similarity classes are exploited to minimize the cost of probing during fault isolation. In addition to building efficient diagnostic procedures, these learning methods inform design activities such as sensor placement and the identification of fault modes.

This research borrows lessons from our research on problem-solving and planning described above. In addition, we have applied these clustering techniques using a model of a life support subsystem from Space Station Freedom, and from telemetry data of the RCS system from the Space Shuttle. In both cases, highly accurate diagnoses are achieved in a relatively efficient manner. This work forms the core of our ongoing research into performance improvement through clustering.

1.4 Summary

These research tracts are linked by the idea that clustering methods can segregate 'observations' (e.g., problem solutions, plans, system behaviors) into similarity classes that can facilitate various kinds of reuse. There are several contributions that we summarize as follows:

- Problem-solving can be rendered more efficient through categorization relative to past experiences. This work contributes in several ways to the literature on explanation-based and speedup learning, notably in representing experiences at appropriate levels of abstraction and using cost-effective features to index and retrieve this knowledge.

- Macro operators learned during planning can be adapted to current situations in a way that facilitates more efficient planning in the future. The effectiveness of this reuse strategy depends on retrieving appropriate macros, and adapting these in a flexible way. This planning work unifies planning strategies of subtasking and case-based planning.

- Ideas in these earlier systems have led to two distinct research directions in diagnosis of NASA systems. We have obtained promising results in discovering operating modes of the Space Shuttle RCS from telemetry data, and we have combined model-based reasoning and clustering to do the same with a model of a Space Station life support subsystem. Currently, this work is our primary focus.

The methods that we have employed are largely inspired by our earlier work with a clustering system called COBWEB. As a convenience to the reader, we summarize this learning technique in the Section 3. We then move to more detailed descriptions of various projects in subsequent sections.

2 Publications supported by NASA Ames Cooperative Agreement NCC 2-645

Journal Articles

[1] Fisher, D., & Hapenyengwi, G. "Database management and analysis tools of machine induction." *Journal of Intelligent Information Systems* (in press).

[2] Fisher, D., Xu, L., Carnes, R., Reich, Y., Fenves, S., Chen, J., Shiavi, R., Biswas, G., & Weinberg, J. "Selected applications of an AI clustering technique to engineering tasks". *IEEE Expert* (accepted).

[3] Fisher, D. "Optimistic and Pessimistic Induction," *Machine Learning* (under review).

Conferences and Workshops

[4] Yang, H., & Fisher, D. "Similarity-based retrieval and partial reuse of macro-operators." *IJCAI-93* (under review).

[5] Manganaris, S., Fisher, D., & Kulkarni, D. "Towards a machine learning framework for acquiring and exploiting monitoring and diagnostic knowledge." *Applications of AI: Knowledge-Based Systems in Aerospace and Industry* (in press).

[6] Carnes, R., & Fisher, D. "Inductive Learning Approaches to Sensor Placement and Diagnosis," *Second International Conference on Diagnosis* (1992).

[7] Fisher, D., Xu, L., & Zard, N. "Ordering Effects in Clustering," *Proceedings of the Eighth International Machine Learning Conference*, Aberdeen, UK: Morgan Kaufmann (1992).

[8] Fisher, D., Manganaris, S., & Yoo, J. "Clustering Approaches to Speedup Learning," *Workshop on Knowledge Compilation and Speedup Learning*, Aberdeen, UK (1992).

[9] Fisher, D., Carnes, R., Yang, H., & Yoo, J. "Basic Levels of Problem Solving and Related Phenomena," *AAAI Workshop on Approximations and Abstractions*, San Jose, CA (1992).

[10] Yoo, J., & Fisher, D. "Concept formation over explanations and problem-solving experiences" *Proceedings of the International Joint Conference on Artificial Intelligence*, Sydney, Australia: Morgan Kaufmann (1991).

[11] Fisher, D., & Yoo, J. "Combining evidence from deep and surface features" *Proceedings of the International Workshop on Machine Learning*, Chicago, IL: Morgan Kaufmann (1991).

[12] Yoo, J., & Fisher, D. "Identifying cost-effective boundaries of operationality" *Proceedings of the International Workshop on Machine Learning*, Chicago, IL: Morgan Kaufmann (1991).

[13] Yang, H., Fisher D., & Franke, H. "Improving Planning Efficiency by Conceptual Clustering" *Third International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, ACM Press (1990).

[14] Fisher D., Yang, H., and Yoo, J. "Case-Based and Abstraction-Based Reasoning" AAAI Symposium on Case-Based Reasoning, Palo Alto, CA: AAAI Press (1990).

[15] Yang, H., Franke, H., and Fisher D. "Planning, Replanning, and Learning with an Abstraction Hierarchy" AAAI Symposium on Planning in Uncertain Environments, Palo Alto, CA: AAAI Press (1990).

[16] Carlson, B., Weinberg, J., & Fisher, D. "Managing Search Using Incremental Conceptual Clustering" *Seventh International Conference on Machine Learning.* Austin, TX: Morgan Kaufmann (1990).

[17] Fisher, D. "Noise-Tolerant Conceptual Clustering" *Proceedings of the International Joint Conference on Artificial Intelligence*, Detroit, MI: Morgan Kaufmann (1989).

[18] Yang, H. and Fisher, D. "Conceptual Clustering of Means-Ends Plans," *Sixth International Machine Learning Workshop*, Ithaca, NY: Morgan Kaufmann (1989).

[19] Yoo, J. and Fisher, D. "Conceptual Clustering of Explanations," *Sixth International Machine Learning Workshop*, Ithaca, NY: Morgan Kaufmann (1989).

Books and Book Chapters

[20] Fisher, D., & Yoo, J. "Categorization, Concept Learning, and Problem Solving: A Unifying View," In Nakamura, G., Taraban, R., & Medin, D. (Eds.) *Categorization in Humans and Machines.* Special volume in the series *The Psychology of Learning and Motivation*, D. Medin (Ed.). San Diego, CA: Academic Press. (in press).

[21] Fisher, D., Pazzani, M., & Langley, P. (eds.) *Concept Formation: Experience and Knowledge in Unsupervised Learning* Morgan Kaufmann (1991).

[22] Fisher, D., & Pazzani, M. "Computational Models of Concept Induction" In Fisher, D. and Pazzani, M. (eds.) *Concept Formation: Experience and Knowledge in Unsupervised Learning* Morgan Kaufmann (1991).

[23] Fisher, D., & Pazzani, M. "Theory-Guided Concept Formation" In Fisher, D. and Pazzani, M. (eds.) *Concept Formation: Experience and Knowledge in Unsupervised Learning* Morgan Kaufmann (1991).

[24] Fisher, D., & Pazzani, M. "Concept Formation in Context" In Fisher, D. and Pazzani, M. (eds.) *Concept Formation: Experience and Knowledge in Unsupervised Learning* Morgan Kaufmann (1991).

[25] Yoo, J., & Fisher, D. "Concept Formation over Problem-Solving Experience" In Fisher, D. and Pazzani, M. (eds.) *Concept Formation: Experience and Knowledge in Unsupervised Learning* Morgan Kaufmann (1991).

[26] Fisher, D. and Langley, P. "The Structure and Formation of Natural Categories," *The Psychology of Learning and Motivation*, Gordon Bower, Ed., Academic Press (1990).

Dissertations

[27] Yang, H. *Learning Abstract and Macro Operators for AI Planning.* Nashville, TN: Department of Computer Science, Vanderbilt University. (1992).

[28] Yoo, J. *Concept Formation over Explanations and Problem-Solving Experiences.* Nashville, TN: Department of Computer Science, Vanderbilt University. (1992).

3 Clustering using COBWEB

Unsupervised tasks require the learner to find categories in data. However, there are a vast number of ways to group observations into contrast categories. Clearly, most of these possibilities are not informative to a learner. To ferret out the informative categorizations an unsupervised system relies on a measure of category quality, and a method of searching through the possibilities for the categorization that optimizes (or is satisfactory by) this measure. Gluck and Corter (1985) suggest that certain categories are preferred because they best facilitate predictions about observations in the environment. If observations are represented as sets of features, V_j, then Corter and Gluck's measure of *category utility* can be partially described as a tradeoff between the *expected* number of features that can be correctly predicted about a member of a category C_k and the proportion of the environment, $P(C_k)$, to which those predictions apply:

$$P(C_k)E(\# \text{ of correctly predicted } V_j|C_k).$$

For example, little can be predicted about a highly-general category like 'animals', but those features that can be predicted (e.g., 'animate') apply to a large population. In contrast, many features can be predicted with near certainty about highly-specific categories like 'robins', but these predictions are true of a relatively small population. Intuitively, a category of intermediate generality such as 'birds' maximizes the tradeoff between the expected number of accurate predictions, and the scope of their application.

The expectation can be further formalized by assuming that predictions will be generated by a *probability matching* strategy: one can predict a feature with probability $P(V_j|C_k)$, and this prediction will be correct with the same probability. Thus,

$$E(\# \text{ of correctly predicted } V_j|C_k) = \textstyle\sum_j P(V_j|C_k)^2.$$

Gluck & Corter (1985) define category utility as the *increase* in the expected number of features that can be correctly predicted, given knowledge of a category, over the expected number of correct predictions without such knowledge:

$$CU(C_k) = P(C_k)[\textstyle\sum_j P(V_j|C_k)^2 - \sum_j P(V_j)^2].$$

Note that by moving the $P(C_k)$ term into the summation and applying Bayes rule

$$P(C_k)\textstyle\sum_j P(V_j|C_k)^2 = \sum_j P(V_j)P(C_k|V_j)P(V_j|C_k).$$

Thus, $CU(C_k)$ is also a tradeoff between standard measures of cue validity, $P(C_k|V_j)$, and category validity, $P(V_j|C_k)$, weighted by the probability of V_j. As Medin (1983) notes, cue validity will tend to be higher for very general categories, since they have fewer contrasts. Inversely, category validity will tend to be higher for highly specific categories. An alternative measure that is based on information-theoretic principles, but which leads to very similar clusters is given by:

$$CU(C_k) = P(C_k)[\textstyle\sum_j P(V_j|C_k)\log_2 P(V_j|C_k) - \sum_j P(V_j)\log_2 P(V_j)^2].$$

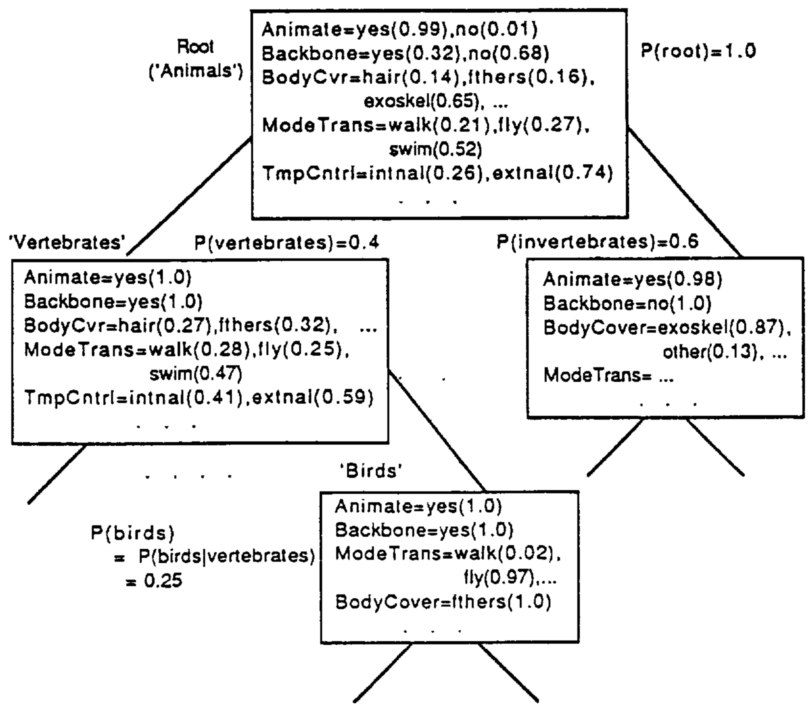

Root ('Animals')
Animate=yes(0.99),no(0.01)
Backbone=yes(0.32),no(0.68)
BodyCvr=hair(0.14),fthers(0.16), exoskel(0.65), ...
ModeTrans=walk(0.21),fly(0.27), swim(0.52)
TmpCntrl=intnal(0.26),extnal(0.74)
P(root)=1.0
'Vertebrates'
P(vertebrates)=0.4
P(invertebrates)=0.6
Animate=yes(1.0)
Backbone=yes(1.0)
BodyCvr=hair(0.27),fthers(0.32), ...
ModeTrans=walk(0.28),fly(0.25), swim(0.47)
TmpCntrl=intnal(0.41),extnal(0.59)
Animate=yes(0.98)
Backbone=no(1.0)
BodyCover=exoskel(0.87), other(0.13), ...
ModeTrans= ...
'Birds'
P(birds) = P(birds|vertebrates) = 0.25
Animate=yes(1.0)
Backbone=yes(1.0)
ModeTrans=walk(0.02), fly(0.97),...
BodyCover=fthers(1.0)

Figure 1: A probabilistic concept tree over 'imaginary' animal descriptions.

A concept formation system called COBWEB (Fisher, 1987) uses category utility to partition a set of observations into contrasting categories, C_k, so as to maximize the average utility of categories in the partition,

$$\frac{\sum_{k=1}^{n} CU(C_k)}{n},$$

where n is the number of categories in the partition. Because category utility requires only information about individual feature distributions within each C_k, one can effectively represent a category with a probabilistic or independent cue representation (Smith & Medin, 1981), where each feature, V_j, is weighted by $P(V_j|C_k)$.

Figure 1 illustrates how COBWEB organizes categories at multiple levels of abstraction. In this example, observations correspond to animal descriptions. Each category is also weighted by the proportion of observations, $P(C_k)$, classified under it.

The method that COBWEB uses to incorporate a new observation into an existing categorization tree is designed for simplicity and efficiency. COBWEB assimilates an observation by evaluating the partitions that result by adding the observation to each existing category, and the partition that results from creating a new singleton category. Intuitively, this latter action occurs if the observation is sufficiently dissimilar from each of the existing categories. It then evaluates each of these alternatives using category utility and retains the best choice. Incorporating a new animal description into the tree of Figure 1 would require three evaluations at the top level of the tree: adding the instance to the 'vertebrate' category, adding it to the 'invertebrate' category, and creating a new category that contained only the new instance. If the instance is incorporated into an existing category, then the observation is assimilated into the respective subtree by the same procedure. Anderson and Matessa (1991) describe an approach in which a Bayesian measure guides object assimilation in the same manner.

We noted that category utility and COBWEB were motivated by a desire to facilitate inference - in this case, the prediction of features that are not directly observed in a new object. As with assimilation, COBWEB uses category utility to guide object categorization: an observation is sorted down a path of 'best matching' categories to a point where the unknown feature is best predicted. Some features may be best predicted at the leaves (specific, past instances) of the categorization tree, while others may be best predicted at categories of intermediate generality (Fisher, 1989). For example, consider a zoological categorization that decomposes 'animals' into 'vertebrates' and 'invertebrates', and then further decomposes these categories into 'mammals', 'birds', etc.. If we observe that an observation has 'feathers', then the path of best matching categories will include 'animals', 'vertebrates', and 'birds'. During top-down categorization we may choose to predict that the object is probably 'animate' when categorization reaches the 'animal' category, that it has a 'backbone' when categorization reaches the 'vertebrate' category, and that it probably 'flies' at the 'bird' category. Thus, to the extent possible, inferences about an incomplete observation are accumulated as one descends the categorization hierarchy. This ability improves with learning. The exact nature of the learning curve and best point of prediction varies between features, and is based on the degree that the features are intercorrelated with other features (Fisher & Langley, 1990).

4 EXOR: Improving Problem-Solving Efficiency through Categorization

This section describes a categorization model of problem solving inspired by research on COBWEB. We open with a view of problem solving as a search-intensive process, then move to issues of learning that render the task more efficient.

4.1 Problem Solving as Search

Traditionally, much of the research on problem solving in AI and cognitive psychology view it as a search task (Simon, 1969). Given a problem description, the task is to search background knowledge for a solution that satisfies all the constraints in the problem's description. As an illustrative example, consider an algebra story problem like those cataloged by Mayer (1981):

> *"A train leaves a station and travels east at 72 km/h. Three hours later a second train leaves and travels east at 120 km/h. How long will it take to overtake the first train?"*

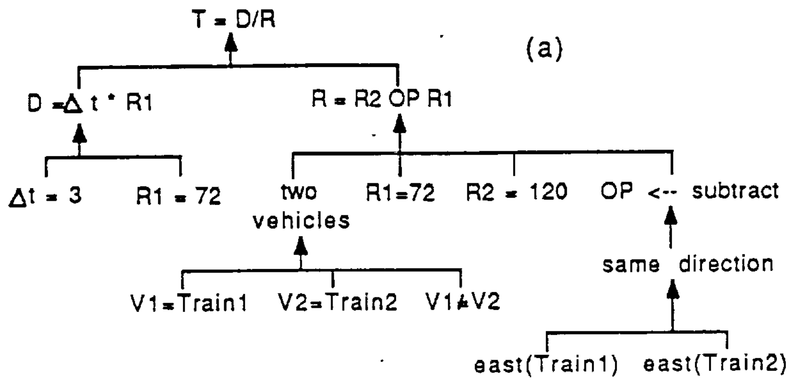
We assume that a (human or machine) problem solver's background knowledge includes domain-specific knowledge about motion problems (e.g., $D = R{\times}T$) and general algebraic knowledge (e.g., $a * b = c \rightarrow a = c/b$). Problem solving requires a search of the possible ways to chain these background rules together in order to derive a complete solution. However, there are many factors that can complicate the process. In some cases the problem solver's knowledge may be incorrect or incomplete, and the problem solver may or may not have access to external assistance. For example, the problem solver may realize that the time, T_2, traveled by the second train equals the distance that it has traveled divided by its rate ($T_2 = D_2/120$), thus triggering an attempt to derive D_2 from the known facts: $D_2 = R_2 \times T_2 = 120 \times T_2$. However, this circularity leads to a 'dead end'. Alternatively, the problem solver might observe that the second train travels three hours less than the first train, or $T_2 = T_1 - 3$. Thus, solving T_1 provides a solution to T_2. In general, many paths may be tried before a solution is found. The line of reasoning above will be successful if the problem solver observes that the distances traveled by the two trains to point of overtake are equal, $D_1 = D_2$. This additional constraint is necessary to solve the problem. If the problem solver fails to encode this knowledge from the problem statement or lacks background knowledge to exploit it, then this line of reasoning will fail.

An alternative solution to the *overtake* problem is abbreviated in Figure 2(a). The solution is expressed in a formal notation required by the computer, but intuitively it encodes that the time until overtake ($T = D/R$) is obtained from the distance that must be made up by the faster train, where $D1$ is the distance traveled by the first train before the second train starts, and the relative rate of travel of the second train ($R = R2 - R1$). Again, if appropriate knowledge is lacking or incorrect, then this line of reasoning will fail as well.

The search for a correct solution to a problem will succeed if the necessary constraints and facts are encoded from the problem statement and the required background knowledge is present. In cases where problem solving is being learned, simultaneous fulfillment of these conditions is often not realizable. Rather, external coaching is required by an instructor. Nonetheless, the computer model that we will describe assumes that any problem presented to the system can be solved correctly and to completion. In this case, learning is exclusively concerned with speeding up problem solving. Initially, even with complete and correct background knowledge, the problem solver may search in a relatively unguided manner through many possible partial solutions, until happening upon a correct one. During this search, however, information about productive paths can be gleaned, and reused to make future problem solving more efficient. This speedup-learning focus is shared by Larkin's (1981) study of learning in *formal domains*, in which subjects know or be able to access (e.g., in textbooks) the requisite principles for problem solving.

4.2 Improving the Efficiency of Problem Solving

'Speedup' learning improves problem-solving efficiency by reusing past experiences. The simplest form of reuse is to use a previous solution as is. Unfortunately, a particular solution trace like the one of Figure 2(a) is tailored to only one problem - in this case, an overtake problems with two trains of 72 and 120 km/hr respectively, leaving three hours apart. However, we can generalize this solution trace by replacing constants (e.g.,



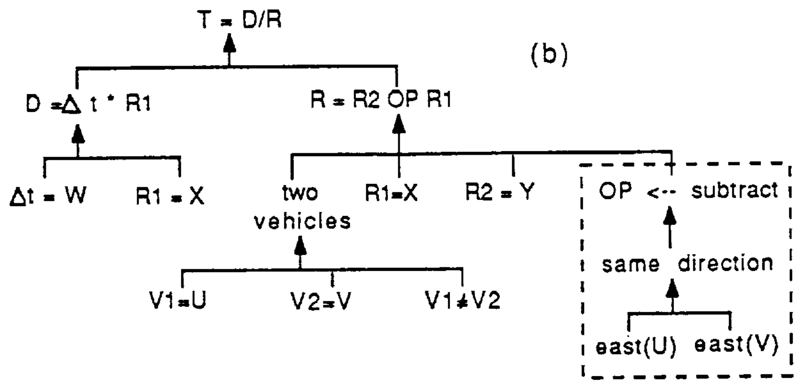


Figure 2: A specific solution and generalized schema for OVERTAKE.

72 km/hr) by variables, so that it matches problems other than the one for which it was first derived. This process of first deriving a solution to a particular problem by searching background knowledge, then generalizing it by appropriately replacing constants by variables is known as *explanation-based generalization* or EBG (Mitchell, Keller, & Kedar-Cabelli, 1986), since a solution trace or proof is regarded as an 'explanation' of the answer. A generalized solution or schema for overtake problems obtained by this process is shown in Figure 2(b). In the future, if the problem solver notices that a problem involves 'two vehicles' moving in the 'same direction' at rates $R_1 = X$ and $R_2 = Y$, $Y > X$, then the overtake schema can be reused to solve it without searching background knowledge from scratch.

Unfortunately, even after variablization, the applicability of this schema will be highly limited. Consider a second *opposite-direction* problem:

> *"Two trains leave the same station at the same time. One train travels 64 km/h to the south and the other travels north at 104 km/h. In how many hours will they be 1008 km apart?"*

This has a solution structure almost identical in form to the schema of Figure 2(b); it differs only in the structure of the boxed subtree; rates must be added, not subtracted, to obtain a relative rate in this latter case. Nonetheless, the earlier solution cannot be used as is to help solve the new problem. In addition, the overtake schema may be more than simply useless - it may actually slow future problem solving. This happens because the solution to the new problem and the schema for the old problem share much in common. Most problem solvers will require some time examining the old schema before abandoning it as a viable option. This adds to the time required to solve the new problem. More generally, if many schemas exist and differ in small ways, then considerable time can be spent searching unsuccessfully for the correct schema, if it exists at all.

In response, Flann and Dietterich (1989) suggest the utility of generalizing over several schemas of the type in Figure 2(b). Figure 3 illustrates the general process; schemas are superimposed, and subtree structure that is not shared by all the schemas is severed. Letters represent propositions, which must match across solutions as well. For example, generalizing the schemas for *opposite-direction* and *overtake* problems yields a tree structure with the boxed rightmost subtree of Figure 2(b) severed.

The superimposition process results in schemas that can be reused in a wider variety of situations. Notice that a generalized schema no longer provides the complete solution structure since some of this has been 'severed', but it ideally provides a sizable chunk that can be completed by doing a small search of background knowledge. In this case a problem involving two vehicles with arbitrary rates would signify the relevance of the generalized schema; background knowledge would only be used to determine how the relative rate should be determined based on whether actual directions of the vehicles implied that they were moving in the same or opposite directions.

It is apparent that something like this schema generalization process is required for effective reuse, but care must be taken in how the procedure is applied. For example, consider a *round-trip* problem in which one vehicle goes from point to point at one rate, and returns at another. In this case, relative rates are not applicable to computing trip

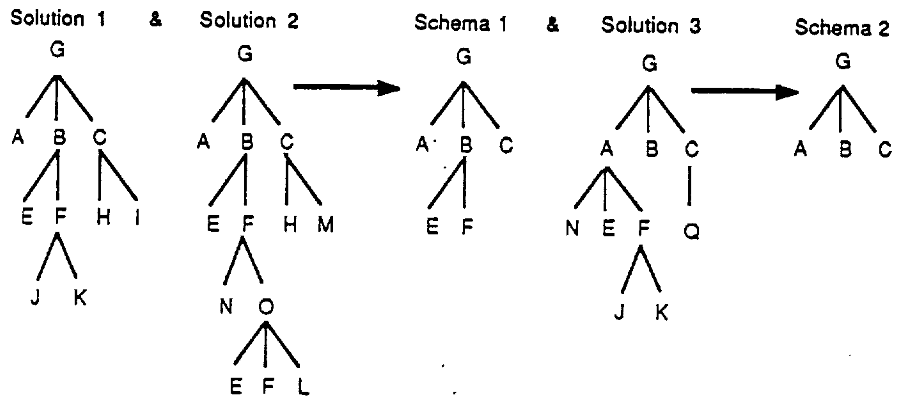


Figure 3: Acquiring generalized schemas by superimposing solutions.

time. Applying the superimposition method to a schema for *round-trip* problems and a schema for either *overtake* or *opposite direction* or both, would result in a trivial schema: $T = D/R$. Little or no problem-solving advantage is gained by its derivation.

Our examples illustrate some important principles: overspecialized schemas can actually detract from problem solving efficiency, since many distinct but similar schemas introduce redundancy into the search for past experiences that are relevant to new situations. In contrast, overgeneralization results in schemas that are underconstrained and provide little or no benefit.

4.3 Improving Problem Solving by Concept Learning

Yoo and Fisher (1991b) defined a system called EXOR (EXplanation ORganizer), which forms an abstraction hierarchy over a stream of problems and their solutions. For instance, in the domain of algebra story problems an abstraction hierarchy over 48 problems drawn from 12 problem types enumerated by Mayer (1981) is formed like the one shown in Figure 4. Associated with each node is a generalized schema like those described previously. The schema at each node is shared by *all* the schemas stored below it; put another way, each schema extends its 'parent' schema in a unique way.

4.4 An Example of Problem Solving by Categorization

The advantage of this abstraction hierarchy is that it constrains problem solving search. Figure 5 illustrates how this process operates on a specific example. In step (1), a problem statement is presented along with a quantity that must be computed. The problem statement is compared against the schemas at the first level of the abstraction hierarchy. In this case, we want to solve for 'time' so a first level node that solves this quantity is selected (as opposed to 'distance' or 'rate'). The general 'schema' that 'time' can be solved by dividing 'distance' by 'rate' is asserted as relevant to the current problem, and this becomes the current hypothesis. In step (2) the schema is specialized further by categorizing the problem relative to one of the children of the current hypothesis. We will describe how a child is selected shortly, but suffice it to say that one is chosen,

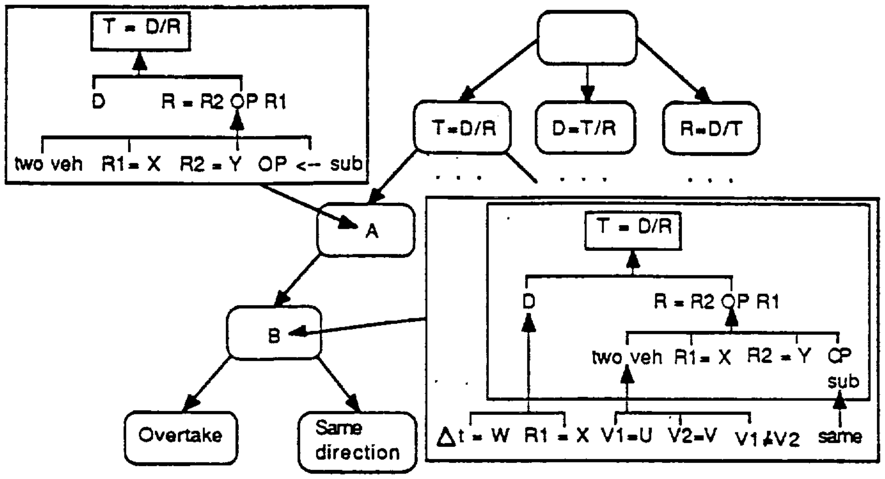


Figure 4: An abstraction hierarchy over problem schemas. Adapted from Yoo and Fisher (1991a, b).

and its schema extends the current hypothesis. This extension to the current hypothesis becomes the new hypothesis. The specialization process continues in steps (3) and (4), but in step (4), a contradiction between the surface features of the new problem and the surface features present in the highly-specific schema is found. In particular, the new problem involves one car going 'east' and the other going 'west', whereas the specialized schema requires two cars going 'west'.

Thus, the new problem is incompatible with the hypothesis. This hypothesis is *retracted*, and one of its siblings is chosen as the current hypothesis in step (5). However, *no* extension of schema B will be applicable for this reason, though our problem solver has no way of knowing this in advance of investigating all of B's specialized schemas. After all extensions of B are exhausted, a final attempt to complete B's schema using background knowledge is made, but this will not work for the reasons stated. Thus, attention returns to B's parent and a sibling of B is chosen as the next hypothesis in step (6). This hypothesis represents the class of problems with vehicles moving in opposite directions; one of its extensions, represented as an existing schema or found by a search through background knowledge if no appropriate extension has been previously encountered, will solve the current problem.

4.5 Categorization and Learning

Figure 6(a) generalizes our example; it illustrates that a new problem will be classified down an 'appropriate' path of the hierarchy. At each node along the categorization path, the schema associated with the node is asserted as participating in the solution of the problem being classified. If at any point, a condition known from the problem statement contradicts one of the conditions asserted at a node, then the partial schema of the node is retracted, and control returns to the node's parent where another child is investigated. If all children of a node fail to complete the node's partial schema then an attempt is

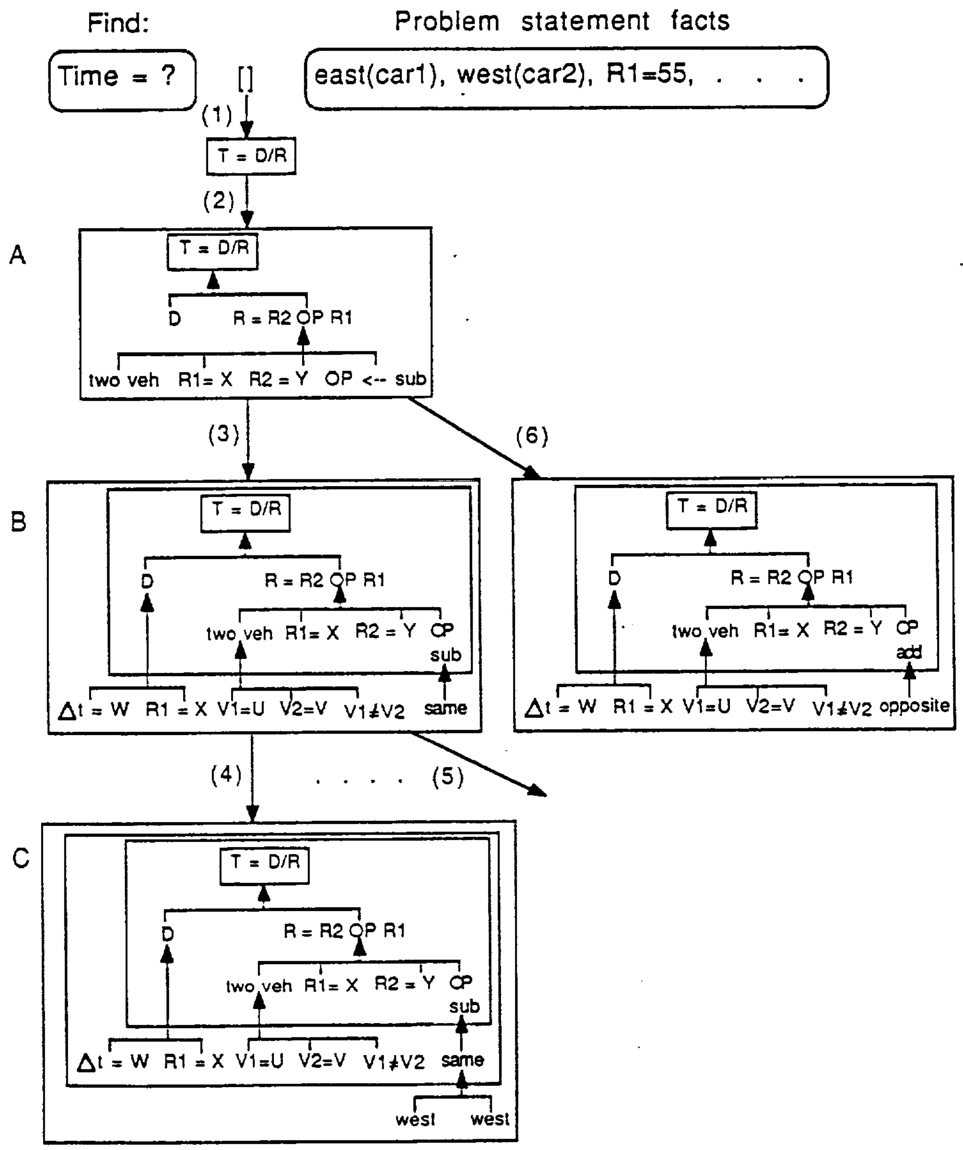


Figure 5: An example of problem solving by categorization.

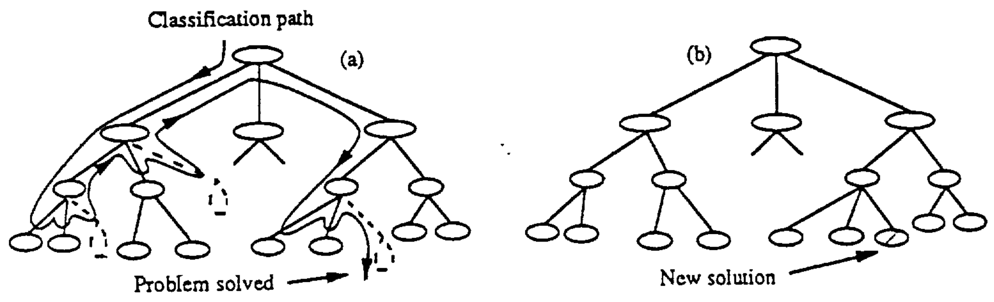


Figure 6: Categorization with and storage of problem-solving schemas.

made to complete it by reverting to background knowledge (i.e., the dashed, triangular nodes of Figure 6). If this fails, then the node fails and control is returned to its parent as described before. The general categorization procedure is quite simple: inferences are accumulated as one descends the hierarchy, and each inference is a complex schema extension that can be retracted and redirected if necessary.

In describing the procedure above, we omitted two important points: how is categorization guided, and how are new schemas added to the categorization tree? To improve efficiency over a simple search of background knowledge, categorization must be well directed. An initial strategy guides a problem's classification using the conditions that are given in the problem statement. These correspond to surface features – e.g., the fact that a problem involves 'two' (versus 'one') 'cars' (versus 'boats') going 'east' and 'west' (versus 'north' and 'south'). In cases where surface similarity is correlated with problem types and solution strategies, some gain in efficiency can be realized. In particular, EXOR computes category utility over the surface conditions, which provides a degree of match between a new problem and each node of the categorization hierarchy. The nodes of a level are ordered by this match score and investigated in order until a complete solution is found. Notice that to compute this score we need to know the distribution of surface features among problems that are stored in the hierarchy. Thus, probability distributions over a subpopulation's surface features are stored at a node, in addition to background structure that is true of *all* of a node's descendents.

If a problem is solved by appealing to background knowledge at a particular node, then the solution is added as a new child (i.e., schema) to the tree at this point. If an existing schema solves the problem, then no change to the categorization structure is made, other than to update the probability distribution of surface features at nodes under which the problem was finally placed. In some cases the new solution is generalized with existing schemas as well, thus creating new schemas. In the tree of Figure 6(a), the solution to a new problem would be added as indicated in Figure 6(b). Thus, the system incrementally clusters problem-solving solutions, and creates generalized schemas in the process.

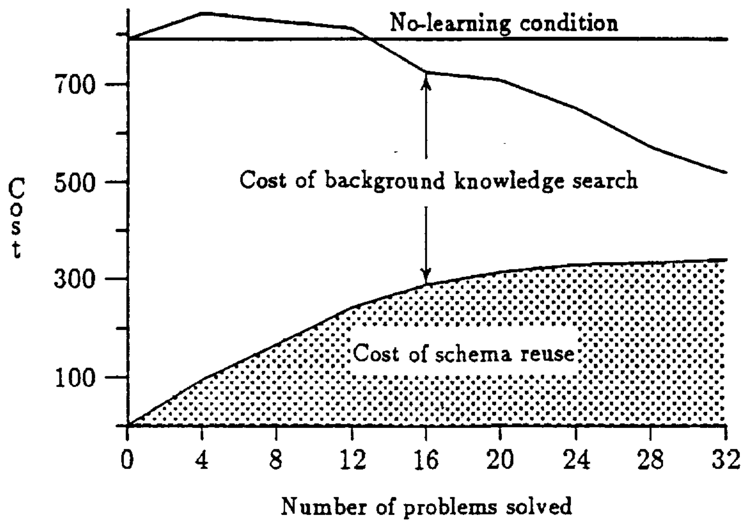


Figure 7: Performance as a function of training. Adapted from Yoo and Fisher (1991a, b).

4.6 Empirical Results

To test the merits of this problem-solving strategy, 32 training problems were incrementally added to the categorization hierarchy. At intermediate points in training, a separate set of 16 test problems were solved via categorization with the hierarchy. The average cost of solving these test problems was measured by the number of features that were instantiated (matched) during categorization. Roughly, this is the sum of the surface and background features in all the schemas successfully and unsuccessfully hypothesized in the course of solving a problem. This measure of cost is well-correlated with the time required to solve the problem, but it does not depend so strongly on the efficiency characteristics of a particular computer implementation.

Figure 7 shows the learning curve averaged over 10 random orderings of the 32 training problems. The declining upper curve illustrates a decrease in the total cost of problem solving; the darker, increasing curve represents the subset of instantiations performed just in matching features that are present in nodes of the hierarchy. The difference in the two curves is the number of instantiations performed using primitive background rules in an attempt to complete generalized schemas. Thus, total problem-solving effort declines with training, but an increasing amount of that remaining effort is placed on the growing categorization hierarchy.

In sum, EXOR builds on the basic COBWEB strategy of top-down categorization and learning. Each node corresponds to a chunk or schema, which is a partial solution common to all category members. These problem solving 'chunks' are not used to guide categorization, but upon making a categorization, are asserted as applicable to a new problem. Rather, categorization relies on probabilistic distributions over the surface features.

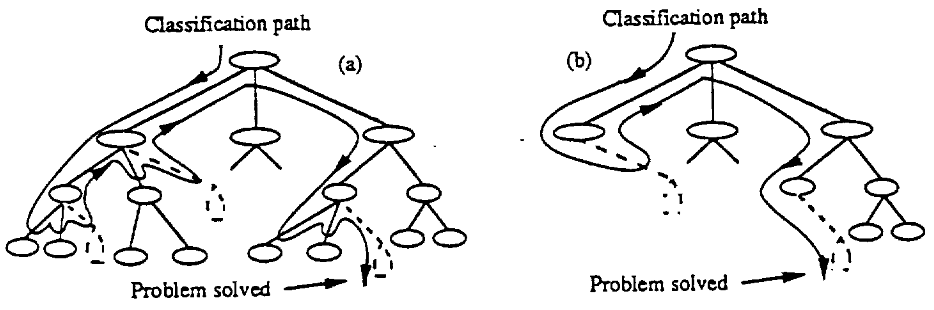


Figure 8: Categorization without and with pruning. Adapted from Yoo and Fisher (1991a, b).

4.7 Inferred Features and Pruning

The gains in EXOR's problem-solving performance can be improved in two ways. As we noted in Section 4.2, overly-specific schemas can detract from problem-solving efficiency. An example of this is illustrated by schema C of Figure 5. In this case it is not important that two vehicles are both moving 'west' (or both move 'east', 'north', or 'south'). It is only important that they are moving in the same direction. Likewise, for *opposite-direction* problems it is only important that two vehicles are moving accordingly; the particular directions do not matter. Such schemas can differ in very small, but nonetheless significant ways from novel problems, thus 'misleading' categorization for a large population of new problems. Therefore, each node of the categorization tree maintains two counts: a count of the number of times that the node was visited during problem solving, and the number of problems successfully solved under the node. If the ratio of successful visits to total visits drops below a threshold (e.g., 0.5), then the node and its descendents are 'pruned' from the categorization tree. Figure 8(b) illustrates the effect of this pruning. If all of a node's children are pruned, then an attempt to complete the node's schema appeals directly to background knowledge.

EXOR's problem-solving efficiency can be further improved by focusing on inferred features. EXOR categorizes problems based on facts presented in a problem statement, but recall that Chi et al. (1981) found that 'experts' categorize problems based on principles that play an important role in a problem's solution (e.g., problems with solutions that depend on Newton's third law are categorized together).

As with surface features, useful inferred features are those that are discriminating of paths that should be followed during categorization. In general, these will be features, F_k, with high $P(C_i|F_k)$, relative to a schema category, C_i. The more predictive a feature is of a particular path or small number of paths, the more efficiently search of the categorization tree will be directed. For example, as we noted above, surface features indicating that one car is going 'east' and one car is going 'west' are of little predictive value, but an abstract feature that is highly predictive of a solution strategy is the fact that the two cars are moving in 'opposite direction's. However, inference of a feature comes with a cost. Thus, we wish to weigh the predictiveness of a feature and cost savings in categorization due to its inference, against the cost that will be required to infer it.

We formalize this tradeoff in terms of the *expected number* of problem-solving steps (or features instantiated, or any of several other measures of *cost*) required to solve a problem with and without knowledge of a feature's truth. Let $EC(C_i)$ be the expected cost of solving an arbitrary problem beginning at node C_i of an EXOR categorization tree, $EC(C_i|F_k)$ is the expected cost required to solve the problem if a (derived or surface) feature F_k is known, and $EC(\text{prove } F_k)$ is the expected cost of proving (deriving) the feature. In the case of surface features, this latter cost is zero. Putting these quantities together, it is useful to verify a feature at a particular node C_i if

$$\begin{aligned} EC(C_i) > {} & P(F_k|C_i)[EC(C_i|F_k) + EC(\text{prove } F_k)] \\ & + [1 - P(F_k|C_i)][EC(C_i|\neg F_k) + EC(\text{prove } \neg F_k)], \end{aligned}$$

where $P(F_k|C_i)$ is the probability that we will be able to prove the truth of F_k (e.g., 'opposite-direction') and $1 - P(F_k|C_i)$ is the probability that its complement (e.g., 'same-direction') is true, which can also be predictive of a particular course of action. Intuitively, the more ways there are to infer a feature in background knowledge, the more possibilities that must be examined, and the greater the cost of inferring it.

EXOR identifies features that are good candidates for inference at each node in the categorization tree. Having made such identifications, statistics on these *cost effective* background features are maintained and exploited in the category utility calculation that was originally limited to surface features. Yoo and Fisher (1991b) report that the additional guidance provided by these inferred features, and the pruning method described above, collectively improves problem-solving efficiency by approximately 12% over the results of Figure 7.

4.8 Summary

The primary lessons of this work are that inferences are facilitated by categorization; schemas composed of both surface and background features are inferred and accumulated into a complete solution as a problem is categorized relative to past experiences. EXOR's categorization appears consistent with the findings and ideas of Chi et al. (1991), Ross and Spalding (1991), and others on human categorization; the system infers background features that guide categorization in problem solving, but surface features are still used if they are predictive.

Furthermore, the evolutionary process underlying EXOR may also account for other data on the transition from novice to expert problem solving. Larkin (1981) found that novices tended to reason 'backward' from the goal, but that experts reasoned forward from the known facts. This forward-reasoning characteristic among experts was also found by Koedinger and Anderson (1990) in geometry problem solving. However, expertise in other domains such as design and troubleshooting (or diagnosis) suggests a dominant backward-reasoning component (Perez, 1991). As Larkin suggests, EXOR begins as a backward reasoner: hypotheses are extended from the goal through categorization and/or background knowledge search until a solution consistent with the known facts is found. However, as cost-effective background features are identified they are inferred at appropriate points in categorization by forward reasoning from the givens. We speculate

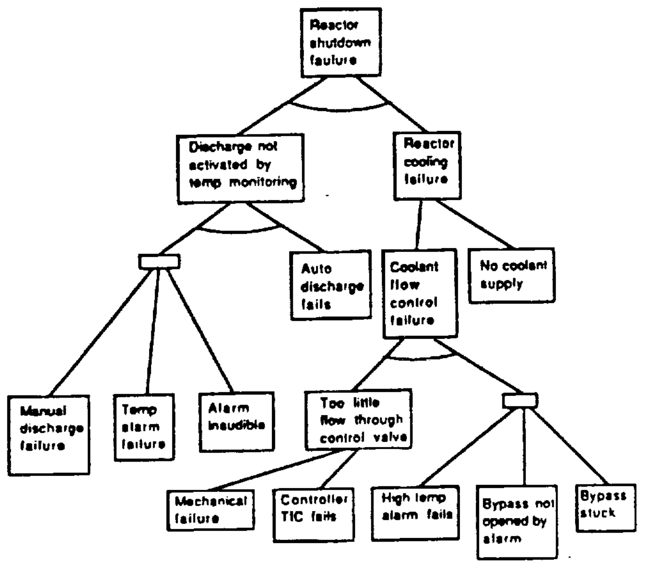


Figure 9: A sample fault tree.

that forward reasoning among experts is dominant in some domains because the top-level goals themselves are deemed cost-effective background features. This may help explain the variance of the forward versus background phenomena across domains, and suggests that in many domains a combination of the two is most appropriate. While psychologists have uncovered plentiful evidence on the importance of inferred features in categorizing problem-solving experiences, we know of no work that identifies the criteria that humans use to select these features. Rather, a starting point in the search for these criteria is suggested on computational grounds: if humans are bounded-rational agents then they may select inferred features that are *cost-effective* – relatively inexpensive to infer and effective at discriminating relevant past experience.

As a cognitive model, EXOR contributes to our understanding of expert problem solvers, in areas such as diagnosis and design, where relatively complete and consistent background knowledge is present. Thus, it is relevant to tasks such as diagnosis of physical systems, where a theory or model of the physical system is present, but the complexities of these models may overwhelm novice users. Additionally, EXOR appears to be a reasonable basis for an automated diagnostic system using *fault trees* (Malasky, 1982), which are a common representation of physical system functionality. An example fault tree is shown in Figure 9. This tree is a domain theory that describes the ways in which a top-level fault `reactor shut-down failure` can occur. For example, `high-temp alarm failure`, `auto-discharge failure`, `controller TIC failure`, and `bypass not opened on alarm` form a minimal set of conditions (i.e., a *cut* set) necessary to cause a top-level fault. Ideally, diagnosis identifies a cut set that explains the top-level fault. EXOR is well suited to clustering over cut sets and determining in subsequent cases the cut set which most likely explains the observed fault. While we have not employed EXOR directly in this task, ideas stemming from its development have led to other work in diagnosis that we describe in Section 6.

5 PLOT: Clustering and Reusing Macro-Operators in Means-Ends Planning

Planning is a critical aspect of intelligence. Traditionally, planning has referred to a deliberative process of projecting one's actions onto the world before acting. Computational complexity stems from coordinating many actions that may interact in nonobvious ways. The usual strategy for mitigating this complexity is subtasking: a problem is decomposed into smaller parts, each is solved, and the solutions are weaved into a global plan. Mean-ends analysis as found in STRIPS (Fikes, Hart, & Nilsson, 1972) is an exemplar of this strategy, in which one uses the goals specified in a problem to delineate subproblems. In particular, operators for achieving a goal are found, and constrain subplans for achieving subsequent goals. Failure to find a global plan for a conjunctive goal leads to backtracking – the exploration of alternative plans with different operators and/or different goal orderings.

Work in case-based planning takes a very different approach to planning (Hammond, 1989). In response to a new problem, plans are retrieved from a case library that succeeded under similar conditions in the past. These plans are modified to fit the conditions of the new situation and executed. Ideally, planning is rendered efficient relative to a subtasking approach of constructing plans 'from scratch'. Case-based approaches, however, often lack the ability to plan when the case library is 'incomplete' or new problems require plans at a considerably different level of complexity as those stored in the case library.

Ideally, we would like to combine the flexibility of subtasking with the efficiency of case-based planning (or planning through retrieval and reuse). Steps in this direction include the identification and reuse of *macro-operators*, which are composites of more primitive operators. The composite operators are typically used in a manner similar to primitive operators, but they take bigger steps towards a solution. Thus, macro-operators are motivated by a desire for increased efficiency as are CBR approaches, but systems that learn and reuse macros are not typically able to modify them in a manner that best exploits their power.

PLOT, combines the benefits of subtasking and case-based approaches by learning, modifying, and reusing macro-operators. The system's *partial* use strategy of exploiting macros is contrasted with more typical *full* and *limited* use strategies, which simply use macros as is. In addition, we show that partial use is most advantageous when it is coupled with an appropriate selection strategy; the PRE and POST conditions of the macro must be 'similar' to the task at hand, else modification is unlikely to improve planning efficiency, and is quite liable to hurt it by expending wasted effort. To promote retrieval of appropriate macros, PLOT organizes operators into similarity classes, that collectively form an abstraction hierarchy over operators. We discuss the general advantages of operator abstraction next, then move to our particular construction of similarity classes, and our strategy for macro reuse.

5.1 Abstraction in Planning

In planning, or any search process generally, we can improve efficiency by reducing the effective depth of search and/or reducing the effective branching factor. As composites, macro operators take large steps in search, thus promising to reduce effective depth, but their introduction also adds considerably to the branching factor. In many cases, a reduction in the effective depth may be outweighed by an increase in effective breadth. These tradeoffs are similar to those encountered in EXOR, though the mechanisms underlying the two systems are somewhat different.

Consider a set of operators defined in terms of their preconditions and projected effects on the environment. Similarity among PRE and POST conditions can confuse a planner's decision about the 'best' operator to apply under current circumstances. This leads to a nonoptimal behavior known as the *problem-solving fan effect* (Shrager, Hogg, & Huberman, 1987): operators can generally be found most quickly (minimal backtracking) for problems that are most idiosyncratic, since the relevant operators are more likely to be idiosyncratic and thus more likely to be deterministically identified as relevant; operators that are highly similar to other operators are more likely to be sources of confusion for the planner.

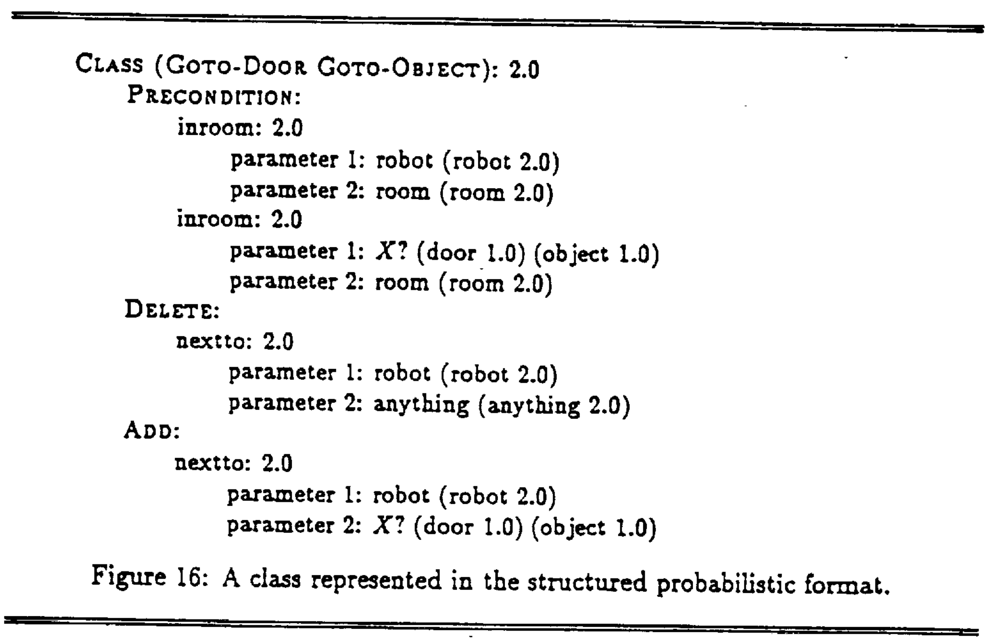
Intuitively, we would like problem solving to be most efficient for the problem's exhibiting the most common patterns! We can achieve this desired result through operator abstraction. In particular, by abstracting over operators we expose the problem solver to 'abstract operators' that are relatively distinct so that an unambiguous decision as to the correct abstract operator can be made at each level of the hierarchy.

If we assume that the hierarchy is such that at each level a 'correct' operator is selected deterministically, and that each node contains n (abstract) operators as children, and that there are T leaves (i.e., executable operators), then we can show formally (Yang, 1992) that the number of operators that need be inspected is reduced from $O(T)$ (with no abstraction) to $O(\ln T)$ for appropriate n at each planning step.

The analysis that leads to this result is related to and inspired by similar analyses (Korf, 1897; Knoblock, 1990) concerned with decreasing effective depth by searching in state abstraction spaces. Here we are not concerned with such a search, but only in the selection of appropriate executable operators at each step in a search through the original state space by exploiting an operator abstraction hierarchy. The effect is to reduce effective branching and to mitigate the problem solving fan effect. The critical assumption if we are to approximate the best case behavior indicated by the analysis is that we can deterministically traverse the abstract operator hierarchy from the root to an appropriate leaf. In general, this is not possible, thus backtracking is still required, though the hierarchy can guide the selection of alternatives, as was the case with EXOR. In any case, operators must be organized by some similarity metric that facilitates near-deterministic traversal.

5.2 Operator Similarity Classes

In traditional means-ends problem solving the problem solver finds operators that most reduce the differences between the current conditions and goal state. In STRIPS, this



```
CLASS (GOTO-DOOR GOTO-OBJECT): 2.0
    PRECONDITION:
        inroom: 2.0
            parameter 1: robot (robot 2.0)
            parameter 2: room (room 2.0)
        inroom: 2.0
            parameter 1: X? (door 1.0) (object 1.0)
            parameter 2: room (room 2.0)
    DELETE:
        nextto: 2.0
            parameter 1: robot (robot 2.0)
            parameter 2: anything (anything 2.0)
    ADD:
        nextto: 2.0
            parameter 1: robot (robot 2.0)
            parameter 2: X? (door 1.0) (object 1.0)
```

Figure 16: A class represented in the structured probabilistic format.

Figure 10: An abstract operator definition.

idea is implemented as a search for operators with ADD conditions that match the goal. Intuitively, we would like more than this: operators should both achieve unsatisfied goals and possess preconditions that best match the current state, thus reducing the need for backward chaining on the preconditions of the selected operator. Operator similarity classes are formed by clustering the operator definitions using a measure of match between operator PRE and ADD conditions. The particular clustering strategy that is used is based on Fisher's (1987) COBWEB, which forms a categorization hierarchy of operator similarity classes.

Each abstract operator discovered through clustering is probabilistically defined. Figure 10 illustrates an abstract operator definition over two primitive operators. The operator definition is represented at three levels: by the PRE, ADD, and DEL lists, by the predicates occurring in each list, and by the arguments to each predicate. Frequency counts of the number of occurrences over class members for each predicate and for each argument are stored. In addition, the number of members (i.e., primitive operators) are stored with the abstract operator. Intuitively, we wish to group operators that agree on many dimensions – their predicates within each list and their arguments to each predicate.

The quality of a class or abstract operator is defined by

$$P(class) \times \sum_{List \in (PRE, ADD)} \sum_{pred \in List(class)} \sum_{par \in pred} [P(par = val|class)^2 - P(par = val)^2]$$

This measure guides the search for meaningful operator classes. Intuitively, an abstract operator is good if it is likely to be applicable (i.e., reflected in P(class)), and it is likely to be useful, which is reflected by the triple sum. This latter term quantifies usefulness as our *certainty* in the truth and appropriateness of predicates and parameters of PRE and ADD conditions, respectively, at the time of operator selection. We will not delve into the measure beyond this, but note that it is a function of both PRE and ADD

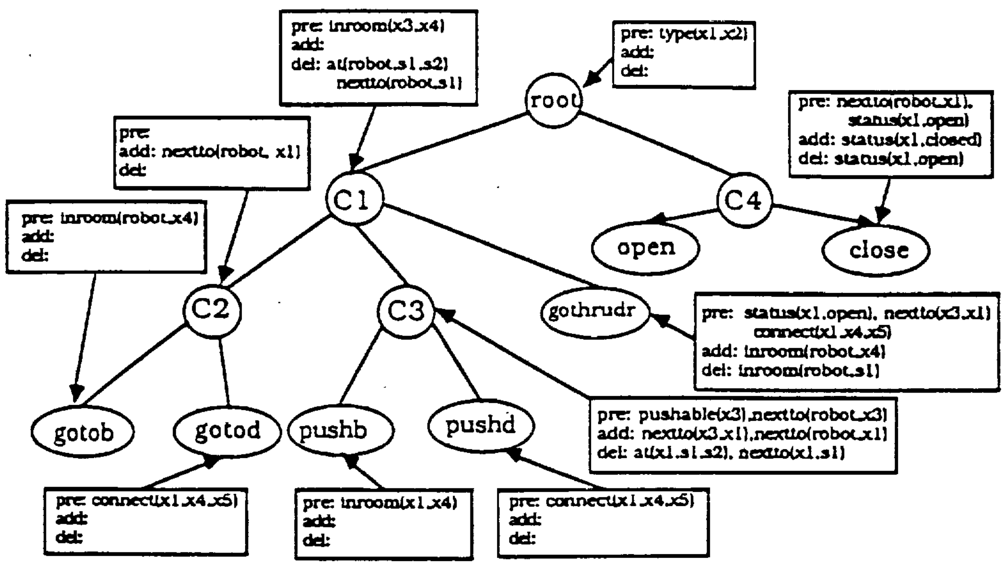


Figure 11: An abstraction hierarchy over STRIPS operators in the robot planning domain.

conditions, and that it separates operators into classes of high within-category similarity and low between-category similarity. The best partitioning of operators into classes is determined, and the procedure is recursively applied to each class, thus generating an abstraction hierarchy with primitive operators at the leaves.

Figure 11 illustrates an operator abstraction hierarchy over the eight STRIPS operators of the robot planning domain. We do not include the full probabilistic definition of each abstract operator, but only those conditions that are common to all members of the class (i.e., those that occur with probability 1.0). A final relevant aspect of this clustering procedure is that it is incremental. The measure used to construct the initial hierarchy can guide categorization and placement of new operators. This is an important feature, if new macro operators are to be learned and stored for reuse.

The operator abstraction hierarchy is used in planning by treating the problem statement as a hypothetical operator definition; the initial state serves as a PRE list, and the conjunctive goal serves as an ADD list. Intuitively, the leaf of the hierarchy found through categorization is an actual operator that best matches the 'hypothetical' operator that will solve the problem. This actual operator is retrieved for means ends planning. The categorization path is remembered, and can guide backtracking, if necessary.

5.3 Learning Macro Operators

As new problems are solved the planner has the opportunity to store macros for future use. In particular, the system analyzes a successful planning tree – a binary tree in which each node is an operator, the left subtree is composed of operators that achieve unsatisfied preconditions of the node, and the right subtree contains operators that achieve the remaining unachieved goals of the node's parent task. PLOT packages together operators that collectively achieve one goal. In particular, every node and its entire left subtree (i.e., its support) are used to construct a macro operator. The rationale for this packaging

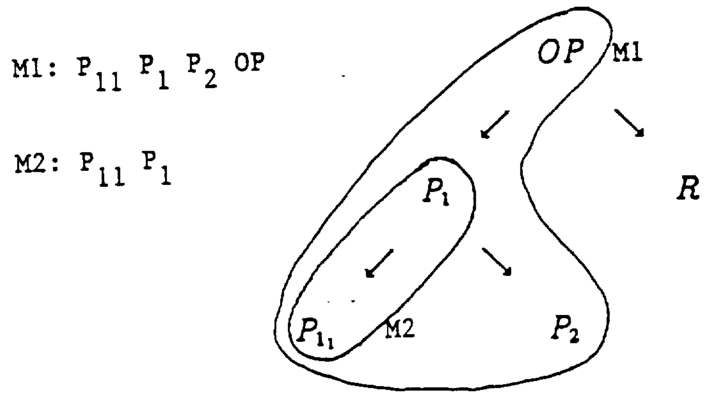


Figure 12: Packaging macros from a planning tree.

strategy is that it yields macros that can be applied independently to achieve a top-level goal, regardless of the conjunctive goal in which this goal appears. Successful operator orderings (i.e., sensitive to goal interactions) for achieving the conjunctive goal which constitutes the preconditions of a node are still retained as the left subtree of that node. Figure 12 illustrates the macros that are formed from a general planning tree.

Having packaged a macro, we wish to place it into the operator hierarchy, so that it can be retrieved and reused under appropriate circumstances by the means ends planner. To do this we must construct PRE, ADD, and DEL lists for the macro. PLOT considers the linear order of the operators obtained by an inorder traversal of the appropriate plan subtree. We then initializes the PRE, ADD, and DEL lists of the macro as follows:

$PRE_1 = PRE(Op_1)$
$ADD_1 = ADD(Op_1)$
$DEL_1 = DEL(Op_1)$

PLOT iteratively analyzes each subsequent operator and updates the macro lists at the mth operator so that

$PRE_m = PRE_{m-1} \cup [PRE(Op_m) - ADD_{m-1}]$
$ADD_m = ADD(Op_m) \cup [ADD_{m-1} - DEL(Op_m)]$
$DEL_m = DEL(Op_m) \cup [DEL_{m-1} - ADD(Op_m)]$

Intuitively, preconditions of a macro are the preconditions of each operator that are not achieved by ADD conditions of preceding operators. Similar intuitions apply for the ADD and DEL lists. The macro and its planning subtree are stored in the operator hierarchy based on their similarity to existing classes along the PRE and ADD lists.

5.4 Planning with Partial Use of Macro Operators

As operators are incorporated into the abstraction hierarchy they can be retrieved and reused in the same manner as primitive operators. This straightforward strategy is known as *full* use of macro-operators. Unfortunately, full use has some problems. For example, if a macro operator of going through 15 consecutive rooms with all doors open is used to solve a very similar problem of going through the same 15 rooms with the second-to-last door closed, it is unacceptable to let the Robot go through these rooms, open the door, and return, just so that we can reuse the macro. This has motivated a *limited* use strategy (Mooney, 1989), in which macros are only used if they completely solve a given subproblem - no backward chaining on the macro's preconditions is allowed. This eliminates the problems associated with full use, but its extreme stance does not exploit macros in many situations where they might be useful with some slight modification.

5.4.1 Partial Use of Macro Operators

We propose a *partial*-use strategy. If all preconditions of a macro are satisfied by the current state then it is used as in limited use. If some preconditions are unsatisfied then instead of initially backward chaining on its unsatisfied preconditions as in full use, or discarding it as in limited use, the planner examines each operator within the macro-operator, and adds and/or deletes certain operators to make the macro workable for the given problem.

This modification process is initiated by restoring the planning tree stored with the macro and considering its root. If the root operator does not achieve any of the desired current goals (i.e., it is *useless*), or it achieves goals that are already satisfied in the current state (i.e., it is *redundant*), then it is removed, otherwise it is kept since it achieves an unsatisfied goal (i.e., it is *useful*). If the root is kept, then analysis moves to its left subtree to see whether operators stored here achieve the necessary preconditions. This analysis process is recursive, in that operators of the left subtree are evaluated relative to their parents preconditions. If an operator at a node is a primitive operator, then examination of its ADD list determines whether it achieves any of its parent's unsatisfied preconditions and it is removed or retained accordingly. However, if a node is a macro operator itself, and it is useful, then it is recursively evaluated by partial use to make it workable under current conditions.

After examining the left subtree of a node, if there are no remaining unsatisfied preconditions of the node's parent task, then the right subtree of the node is truncated, since its operators are no longer needed to achieve any parent preconditions. If parent preconditions do remain unsatisfied, then the node's right subtree is recursively evaluated; if no right subtree exists, then the means-ends planner is called to add additional operators to achieve remaining parent preconditions.

If at any point in recursive analysis, an operator is deleted, then its right-subtree is promoted as the root, since the right subtree was originally included to achieve remaining goals of the parent task; the assumption is that it is more likely to obtain such parent goals in the current context. If a right subtree is empty, as is the case when we are examining the root of an entire macro, then the left subtree is promoted. Figure 13 illustrates this process.

$$\begin{tikzcd}
 & \Re^d \arrow[dl] \arrow[dr] & & & \Longrightarrow & & R \arrow[dl] \arrow[dr] & \\
L & & R \arrow[dl] \arrow[dr] & & & R_L \arrow[dl] & & R_R \\
 & R_L & & R_R & L & & &
\end{tikzcd}$$

Figure 13: Operator deletion.

5.4.2 Related Work

STRIPS can edit a macro to better fit current planning requirements. However, STRIPS particular strategy of using *triangle tables* may only delete operators. STRIPS applies a full use strategy plus deletion. In contrast, we retain the planning tree so that operators may be added by recursive application of the planning module in an appropriate way.

Partial use is also similar to the kinds of modifications that can occur in CBR and its ancestors such as HACKER (Sussman, 1975). Deficiencies in a plan (or macro) can be remedied by searching its case library (or *answer* library in the case of HACKER) for other macros or parts thereof that can patch the deficiency. We view these systems as performing a kind of limited use with the possibility of hard-coded additions. In contrast, partial use exploits the full power of the means-ends planner to patch a macro. Note that the planner can retrieve other macro operators in this process, and thus need not build the patch from scratch.

Allen and Langley (1989) and Anderson and Farley (1988) also form operator hierarchies, but neither stores and reuses macro operators.

5.4.3 Selection by Similarity and Partial Use

Intuitively, it is important that partial use be used in conjunction with an operator selection strategy that returns (macro) operators that are similar to current conditions and goals, else considerable effort may be wasted. As previous sections indicate, selection by similarity is the strategy that we use. However, other selection strategies have been proposed. One strategy is to examine macro operators in the order in which they were constructed, since macros discovered early are more likely to apply to more common classes of problems - an observation that is true, if a problem population is sampled randomly. A second strategy is to examine macros by size; focusing on the largest macros first may bring about the largest gains in problem solving.

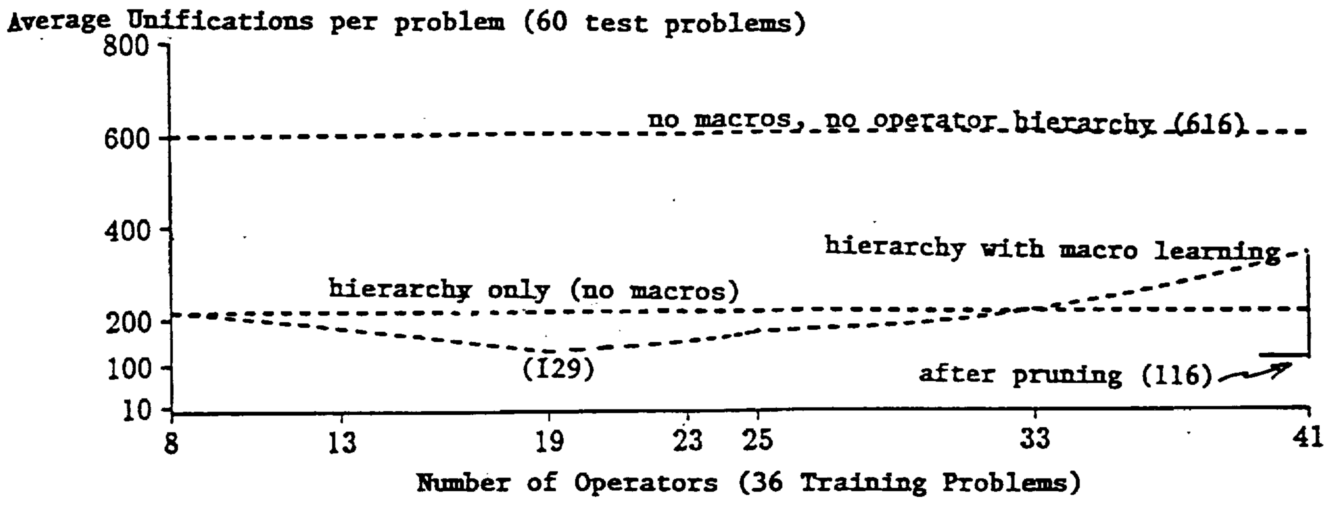


Figure 14: Performance of the learning planner.

5.4.4 Experimental Results

To evaluate similarity-based selection and partial use, we examined 60 artificially-constructed robot planning problems. Problems were ordered by the length of the plan required to solve the problem – i.e., the planner was trained from 'easiest' to 'hardest' problems. The planner accumulated and added macros during training. Intermittently, we tested the planner on the full set of 60 problems with learning 'turned off'. Figure 14 illustrates the performance of our planner, a traditional means-ends planner, a planner endowed with an abstraction hierarchy over the primitive operators but no macro learning ability, and an important variation to our planner. The notable phenomena are that simply employing an abstraction hierarchy over the primitive operators does well ($\approx$ 220 unifications) relative to a traditional planner (616 unifications). In addition, partial use of acquired macro operators improves performance for a time (to 129 unifications), but performance degrades after many macros are added to the hierarchy. The basic reason is that probabilistic class definitions become less distinct as more operators are added, thus confusing similarity-based selection. There are many possible fixes to mitigate this, including other categorization structures, but we have implemented a simple strategy of pruning macro operators that are either infrequently used and/or generally incorrect as measured by a simple product of these factors. Macro operators that are evaluated as falling below a threshold of acceptable performance are removed. This strategy significantly mitigates the *utility* problem (Minton, 1988), and appears fairly robust across a range of thresholds.

Applying this pruning strategy at the point indicated, served to remove 24 macro operators, reducing the set of operators in the hierarchy from 41 to 17 (8 primitives and 9 macros). This led to an increase in performance over the the problem set (116 unifications).

Our real goal here is to evaluate the utility of partial use. In particular, we examine partial use with various operator selection strategies. The bar graph of Figure 15 illustrates partial use performance when used in conjunction with selection strategies based on macro operator operator size, learning order, and similarity. This is a snapshot of

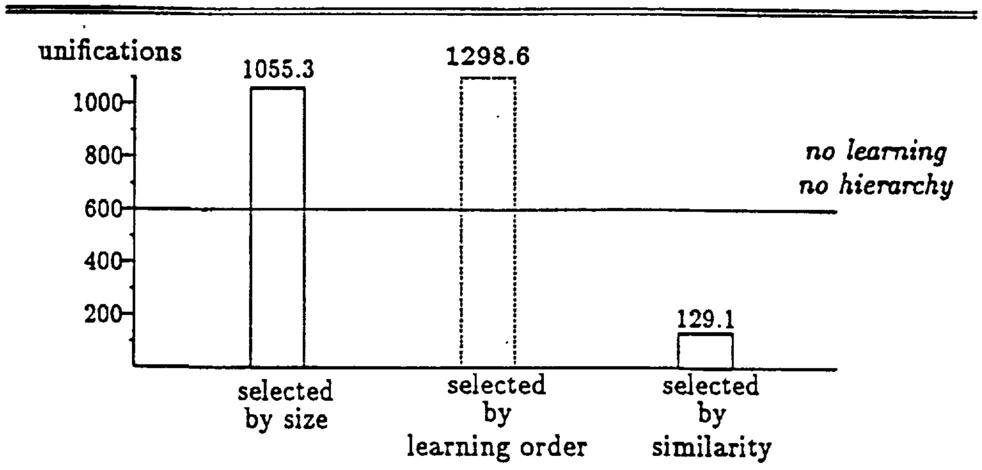


Figure 15: Performance of partial use.

planner performance at the point where 19 operators are stored in the hierarchy. Performance indicates that partial use only makes sense when we select based on similarity.

Figure 15 indicates the relative merits of partial use with various selection strategies. Figure 16 evaluates limited use in the same way. In this case, similarity-based ordering of macros is still preferred, in part, because it prefers operators with preconditions that match current conditions, as well as the desired goal conditions. However, comparisons across graphs indicate that limited use is also advantageous in conjunction with selection based on learning order. Finally, we note that similar experiments were performed with a full use strategy, which does very poorly regardless of operator selection strategy.

5.5 Summary

In sum, PLOT introduces the idea of similarity-based retrieval and partial use of macro operators. These selection and use strategies are compatible, and best applied in conjunction. Both ideas stem from work in CBR, but our goal was to integrate them into a traditional means-ends planner, which retained the flexibility to solve new and novel problems, and to patch deficiences in macros through subtasking, as well as by using portions of other macros. A secondary idea is that operator selection should be guided both by conditions that an operator can achieve so that forward progress can be made, and by the match between the preconditions and current state, thus minimizing backward chaining on the preconditions of the selected operator.

6 Cluster-Analytic Approaches to Diagnosis

Accurate and timely fault diagnosis is critical in the life cycle of many physical systems. Seemingly minor faults can, if unremedied, lead to catastrophic faults that disable a

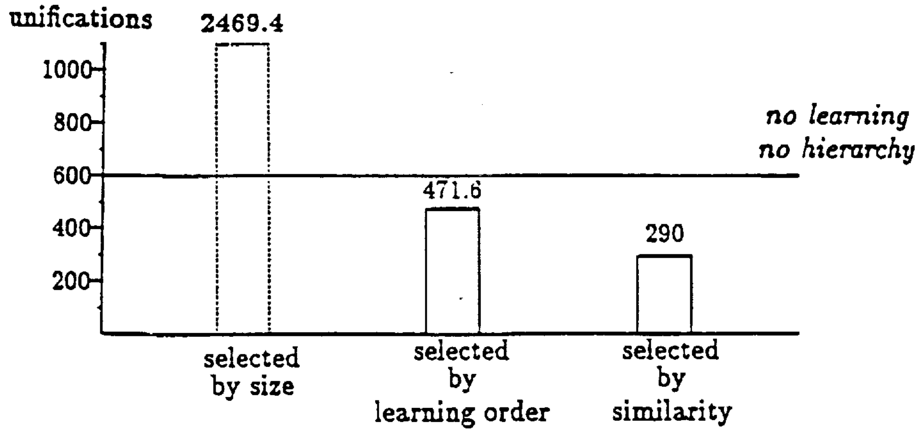


Figure 16: Performance of limited use.

system permanently. To *detect* faults, technicians observe the system's behavior through sensor readings. Sensors should be maximally informative about the state of the system. However, in many cases it is neither feasible nor desirable to monitor all quantities of a system. Thus, the diagnostician interfaces with the system in other ways, including *probing*. One can think of probing as sensing a quantity dynamically to determine its value at a particular point in time. In general, we wish to minimize probing by judiciously selecting the 'most informative' probes throughout diagnosis.

Diagnostic procedures can be judged by their efficiency. This section forwards two ideas.

[1] Methods of *machine induction* can be used to identify system parameters that are good candidates for dynamic probing and continuous sensing over a population of diagnostic episodes.

[2] More specifically, inductive methods of *clustering* can identify similarity classes of system behavior in terms of observable system quantities. These similarity classes can be exploited to minimize the cost of probing during fault isolation.

Section 6.1 discusses the relationship between traditional methods of candidate generation (DeKleer, 1991) and machine induction, notably of decision trees (Quinlan, 1986). Section 6.3 discusses the weaknesses of the decision tree approach, and motivates an alternative that clusters behaviors that are similar in terms of observable parameter values; each behavior cluster is represented by a probabilistic summary of parameter values that are common to and that discriminate the class from contrast clusters. Our particular approach is contrasted with a similarly-motivated approach by Wu (1990, 1991). Section 6.5 discusses the manner in which discovered similarity classes are exploited for diagnosis. In particular, probabilistic summaries of behavior classes efficiently indicate diagnostic probes that best discriminate among members of the class, thus focusing the diagnostic process.

6.1 Model-Based Diagnosis and Induction of Decision Trees

DeKleer (1991) and others describe model-based diagnosis as roughly the following:

Step 1: Generate candidate diagnoses consistent with system outputs.

- **(a):** Note discrepancies between known system outputs, and those anticipated conditioned on system inputs.
- **(b):** Explain how each (anticipated or unanticipated) output might arise in terms of the correct or incorrect functioning of individual system components and combinations of components. This analysis produces 'conflict sets' that offer differing explanations for each output.
- **(c):** Combine the conflict sets into 'candidates' (i.e., the cross product of the conflict sets).
- **(d):** Retain those candidates, which explain all of the known system outputs conditioned on system inputs, and that are internally consistent.

Step 2: Select probes that discriminate candidates until an unambiguous candidate is isolated.

- **(a):** If only one candidate (or 'sufficiently' small number) are consistent with known system values (initially outputs), then STOP and return the (ideally singleton) candidate set.
- **(b):** Make the most discriminating probe according to an information-theoretic measure – i.e., that probe with possible values or outcomes that best discriminates the remaining candidates.
- **(c):** Eliminate those candidates that are inconsistent with the value obtained by the probe.
- **(d):** GOTO Step 2(a).

Step 2 can be viewed as tracing out a 'decision path' that discriminates the true diagnosis from possibilities initially created by candidate generation in Step 1. Initial candidates are conditioned on known system input and output values, and subsequently known values obtained through probing restrict diagnosis to increasingly more specific candidate classes. This methodology presumes that a model of the system can be simulated on particular inputs so that other system-wide values can be computed.

Intuitively, we wish to select probes in an order that minimizes the *expected cost* of diagnosis. DeKleer (1991) approximates the expected cost of isolating one candidate from a set of possibilities, C, conditioned on prior known system values, V, by an information-theoretic measure. In the case where probes are of uniform cost and each probe is binary-valued (e.g., expected, unexpected), the expected number of steps to isolate a candidate conditioned on a probe's selection is:

$$EC(C|V) \approx -\sum_{v_{ij}} P(t_i = v_{ij}|V) \sum_{c_k} P(c_k|t_i = v_{ij} \wedge V) \log_2 P(c_k|t_i = v_{ij} \wedge V).$$

This measure is used to guide the expansion of the decision path through probe selection. Those probes that most reduce uncertainty about the true diagnosis (i.e., candidate) are selected at each step.

Importantly, diagnosis based on candidate generation is a data driven process; each probe selection further specializes a data set of candidate causes (and their projected effects on system parameters) that was initially constructed via simulation to be consistent with initially observed outputs. This strategy can be approximated and rendered more efficient by induction over a population of diagnostic episodes, where the population is assumed to be representative, and the knowledge structures constructed through induction mitigate the need for candidate generation on a problem-specific basis. Decision tree induction is one of the simplest and most straightforward in terms of its relation to the candidate generation strategy outlined above.

In diagnosis we can view faults as labeling classes of observations that need be distinguished. System outputs and other values obtainable through probing form the feature set over which faults can be discriminated. To build a useful decision tree requires that we select discriminating probes in a rational way - that is, by the extent to which the values of the feature *increase* certainty about category membership (Quinlan, 1986). In the case of diagnosis, the form of the evaluation function typically used is a simple extension of that which we gave earlier for probe selection:

$$\sum_{v_{ij}} P(t_i = v_{ij}|V) \sum_{c_k} [P(c_k|t_i = v_{ij} \wedge V) \log_2 P(c_k|t_i = v_{ij} \wedge V) - P(c_k|V) \log_2 P(c_k|V)].$$

Since $P(c_k|V) \log_2 P(c_k|V)$ is constant across all t_i, maximizing the measure above is equivalent to minimizing the expected cost approximation given earlier:

$$-\sum_{v_{ij}} P(t_i = v_{ij}|V) [\sum_{c_k} [P(c_k|t_i = v_{ij} \wedge V) \log_2 P(c_k|t_i = v_{ij} \wedge V).$$

Intuitively, the decision tree represents a composition of decision paths that would be expanded for differing problems using a candidate generation approach. However, like decision paths of the candidate generation algorithm, the decision tree must be constructed from some initially generated 'data' or candidates. The primary differences are that observations across a representative set of I/O vectors must be obtained, but once generated the tree serves as associative knowledge that mitigates (but may not entirely eliminate) the need to generate candidates for problems encountered in the future.

6.2 Decision Tree Induction over Space Station Model Behaviors

Carnes and Fisher (1992) have used a decision tree induction system, C4.5 (Quinlan, 1987), to discover categories of fault conditions over system simulations. This work has

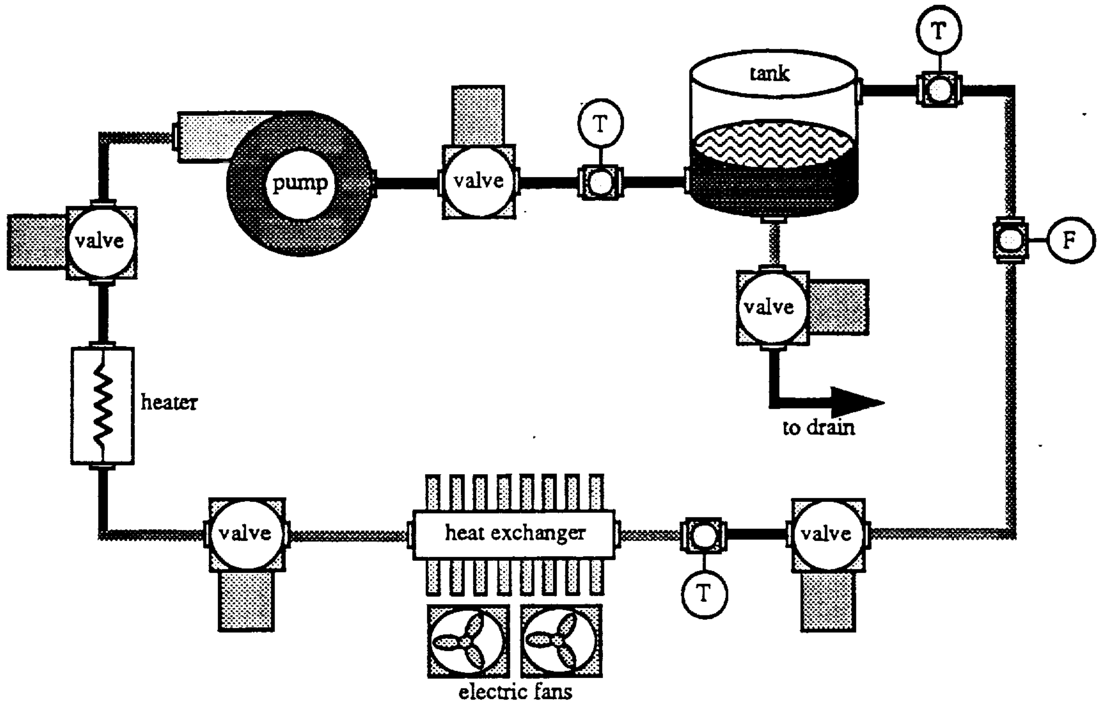


Figure 17: A thermal model.

proceeded with a simplified model of part of the life-support system of Space Station Freedom. In particular, consider a model of a thermal system given in Figure 17. Carnes and Fisher used the following strategy to generate data and to learn rules that distinguish a variety of conditions that can cause anomalous behavior in this system.

[1] System designers specify a simulator that represents each major system component as a function that maps component inputs to outputs. Simulation using a model-based methodology begins with an initial state of system parameter settings and propagates parameter changes through component functions until the simulator converges on a stable state.

[2] Associated with each system component are permissible parameter ranges, within which the component is assumed to operate satisfactorily. Initial simulator parameters are systematically perturbed beyond extreme ends of these ranges for each component, thus yielding conditions under which the system is liable to malfunction.

[3] Each condition set generated in step [2] is propagated through the system until a stable (or some error) condition is reached.

[4] The system state descriptions of all simulations (i.e., composed from observable system parameters) that lead to anomalous behavior are collected together, labeled by the perturbation that initiated the anomaly, and passed to C4.5.

[5] The learning system forms a decision tree that distinguishes anomalous behaviors that were caused by different parameter perturbations.

The decision tree formed in Step 5 implicitly groups fault modes based on similar values along the selected system parameters. In total there were 87 fault types. In addition,

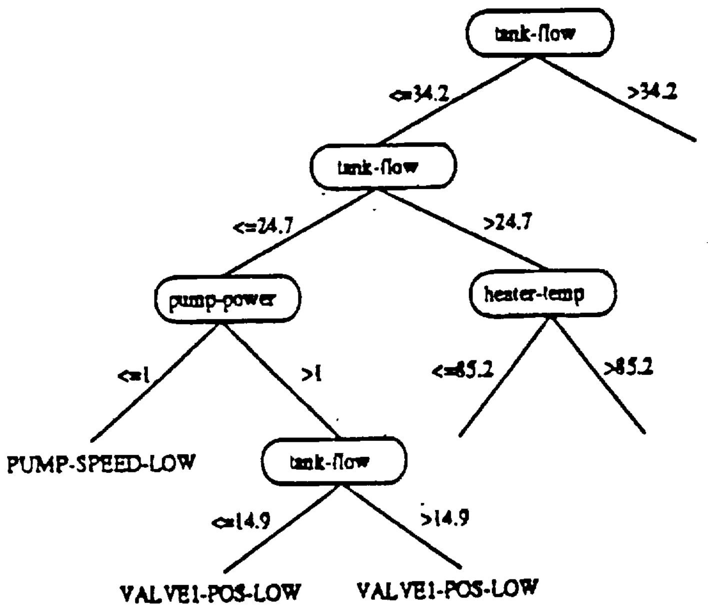


Figure 18: A decision tree over anomalous behaviors.

three versions of each perturbation type were generated, corresponding to cases where the selected parameter value was perturbed just above (or below) acceptable ranges, moderately out of range, and far out of range. Intuitively, these corresponded to conditions of high (low), very high (low), and very, very high (low) values, but each case was labeled by a single fault (e.g., the parameter was 'above acceptable range'). Thus, the decision tree had to distinguish 87 'faults', represented over 261 observations (snapshots). Each snapshot was represented by 23 system parameter values.

C4.5 constructed decision trees much like the one partially shown in Figure 18. The decision tree is a highly efficient and conceptually-simple classifier, which identifies candidate causes accurately to the extent that the training population is representative of behaviors that will be experienced during system operations. However, a decision tree has some notable limitations for diagnostic applications. First, the tree is very sensitive to noise in sensed or probed system values: each test value implicitly defines a candidate class – a single misleading value can lead diagnosis considerably astray by suggesting the wrong candidate class. One implication is that uncertainty in a domain may insist on some redundancy in the probed values to protect against noise. In addition, the tree is designed to yield diagnosis of low expected cost over a population of problems (i.e., observations and underlying faults). However, individual problems may come with differing observed system values initially – values that do not occur near the root of the tree where diagnosis begins; this may necessitate unnecessary redundancy in probing system parameters at the root.

6.3 A Cluster-Analytic Approach to Diagnosis

Traditionally, probe selection is determined by the extent that the probe's possible outcomes minimize uncertainty about which candidate is causing faulty behavior. An alternative view of diagnosis forwarded by Wu (1990, 1991) is that diagnosis should minimize uncertainty about the state of system parameters. Diagnosis is the process of determining a system's parameter vector with some degree of certainty. A reasonable strategy in such cases is to discover behavior categories exhibiting relatively low uncertainty in observable parameters. This section describes an inductive *clustering* approach that naturally fits the view of diagnosis as identifying a (probabilistic) system state vector.

A clustering system constructs a classification scheme over a set of observations. We have used an extension of Fisher's (1987) COBWEB system. As we noted in introducing COBWEB, it is every clustering system relies on a criterion of cluster quality. The basis for a criterion function might be Wu's notion of *explanatory power*. Wu suggests that the expected ability of a target disorder to explain an arbitrary set of observed symptoms can be estimated as the proportion of (total) symptoms that are caused with nonzero probability by the disorder. More generally, we are interested in using some measure of explanatory power of a cluster of 'disorders' or behaviors to explain the observed 'symptoms' or parameter values of a system. This measure can then be used to discover clusters that 'explain' or predict the observed parameter values of a system during operation.

We previously noted that COBWEB uses a measure of cluster explanatory power known as *category utility* (Gluck & Corter, 1985). We repeat this for the reader's convenience.

$$CU(C_k) = P(C_k)[\sum_i \sum_j P(t_i = v_{ij}|C_k) \log_2 P(t_i = v_{ij}|C_k) - P(t_i = v_{ij}) \log_2 P(t_i = v_{ij})],$$

which rewards clusters, C_k, that most *decrease* the uncertainty in system parameter value distributions. An alternative measure (Gluck & Corter, 1985) which we have used is

$$CU(C_k) = P(C_k)[\sum_i \sum_j P(t_i = v_{ij}|C_k)^2 - P(t_i = v_{ij})^2].$$

These measures heuristically rank the quality of clusters in an almost identical manner. In either form, category utility is a more sophisticated measure of explanatory power than Wu's. Notably, category utility considers the absence of a 'symptom' as possibly diagnostic as well as its presence. More generally, a probe or sensor may have one of many possible values, and category utility is a function of the probability distribution over all possible probe values. Whereas, Wu was interested in *demonstrating* the *existence* of connections between target disorders and symptoms, for which his explanatory power measure was adequate, we are interested in discovering and exploiting probabilistic connections between disorders and symptoms in a manner that yields an approximately optimal organization of disorders into categories for purposes of diagnosis.

The category utility expression above is appropriate for nominally-valued (i.e., discrete, unordered, finite) attributes, but several variations on this basic scheme have been adapted to handle observations described over continuously-valued attributes as well. We have adopted COBWEB/3 (McKusick & Thompson, 1990), which handles continuous dimensions by seeking clusters that most reduce the standard deviation over continuous parameter values. In particular, terms over continuous attributes are computed as

$$\frac{1}{\sigma_{ik}} - \frac{1}{\sigma_i},$$

where σ_{ik} is the standard deviation over the values of parameter value i at cluster C_k, and σ_i is the standard deviation of the same parameter value at its parent.

Category utility is used recursively, first to build a partition over the entire population of observations, and then to subpartition each of these initially-constructed clusters, thus yielding a categorization hierarchy. Each category in this hierarchy is represented by the proportion of behaviors classified in the category (i.e., which approximates the probability of classifying future observations into this category), and the probabilistic distribution of test values exhibited among members of the category. Our particular interest in this process is its ability to discover clusters over snapshots or instantaneous descriptions of system simulations.

6.4 Clustering over Space Station Model Behaviors

We have used COBWEB to discover categories of fault conditions over system simulations. This proceeds in much the same way as the simulation/induction procedure of Section 6.1, except that in Step (4), the snapshots are passed to COBWEB/3 rather than C4.5. An example of a categorization tree of discovered fault modes for the thermal system is partially shown in Figure 19. Each datum consists of inputs and outputs, for all components, including the single perturbed value (as described in step 2); that is, each datum is a snapshot of the system. We do not show the probability distributions over all attribute values for clusters, but simply label each low-level node by a descriptor that conveys the fault-mode meaning. These labels correspond to the perturbation that initiated each simulation and resulted in an anomalous behavior, as described in Section 6.1.

The benefits of clustering are at least two-fold. First, it is difficult for engineers to completely design against system faults in advance. Collectively, simulation and clustering identify fault models that benefit design decision making by identifying 'new' faults for the designer and by indicating that known faults may appear quite similar, and thus easily confused if appropriate discriminating sensors are not used. For example, low flow through the radiator and a malfunction to the heater both result in high water temperatures, despite the fact that this behavior emerges for very different reasons. Similarly, high flow through the pump appears somewhat similar to a second heater malfunction: both result in low water temperatures.

Second, like C4.5, a COBWEB classification tree can facilitate fault diagnosis. In particular, categories discovered through clustering associate observable/sensor/probe values with component faults that lead to the observed anomalies. We can classify an observable set of parameter values down a path of 'best matching' nodes to a point that corresponds to an identifiable operating mode. Initially, an entire system snapshot consisting of all parameter values is tentatively placed in each child (cluster) of the root node of the hierarchical clustering. The average category utility, $\frac{\sum_k^N CU(C_k)}{N}$, is computed for the partition that would result from this placement. The child which yields the best category utility score after this tentative placement is selected to classify the snapshot. Intuitively, categorization proceeds by computing the 'match' between the parameter values of a particular observation and the probability distribution over parameter values

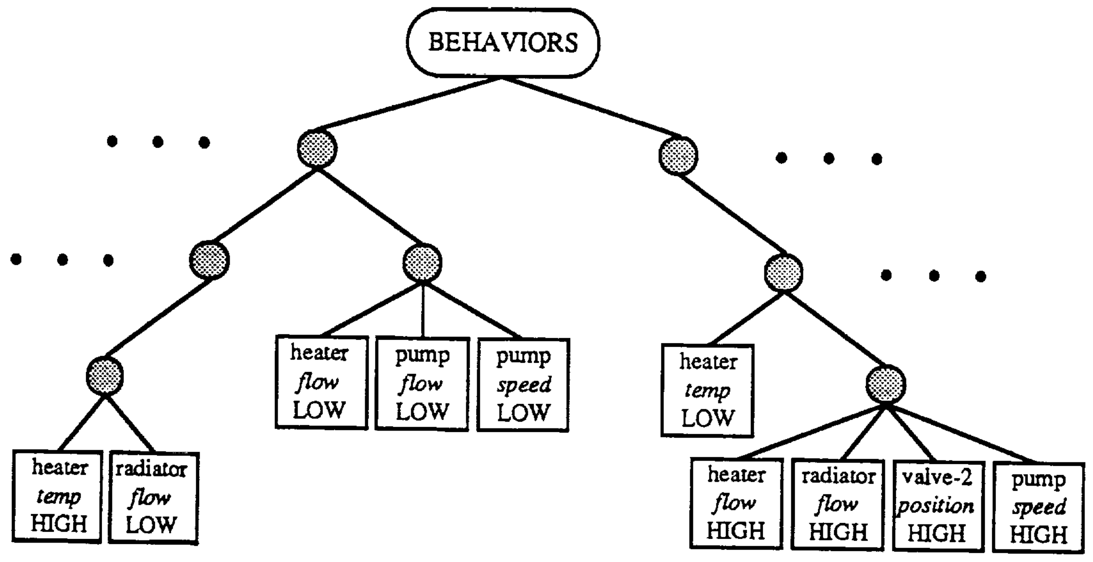


Figure 19: A partial classification tree of fault modes for the thermal model.

Table 1: Salience scores for a behavior cluster.

```
USER(14): (print-class-info 'n49)
Node - N49
  Parent: N30   Children: (HEATER-1-TEMP-NOM-45.67 N53)
  Attribute List (partial order):
       RAD-TEMP      V1-TEMP       PUMP-TEMP
       TANK-TEMP     V2-TEMP       HEAT-TEMP
       V2-FLOW       RAD-FLOW      PUMP-FLOW      HEAT-FLOW
       TANK-FLOW     V1-FLOW       V2-PRES        RAD-PRES
       V1-PRES       PUMP-PRES     HEAT-PRES      TANK-PRES
       V2-POS        COMMENT       COMPONENT      HEAT-POWER
       RAD-POWER     TIME          PUMP-POWER     V1-POS
  Salience List (partial order):
       1.025914      0.988109      0.988109
       0.988109      0.915097      0.900949
       0.596295      0.596295      0.542278       0.542278
       0.405532      0.405532      0.222549       0.222549
       0.107379      0.107379      0.107379       0.107378
       0.000000      0.000000      0.000000       0.000000
       0.000000      0.000000      0.000000       0.000000
```

at each node. The node with parameter value distributions that are best 'reinforced' by the new observation is selected to classify it. This categorization decision is then applied recursively at this 'best' matching node, until a leaf in the hierarchy is reached. When a leaf node is reached, the label (identifying the perturbed parameter value, and direction of the perturbation), which is attached to the behavior at the leaf is used as the predicted 'cause' of the behavior being categorized.

6.5 Attention Focusing in Space Station Model Diagnosis

To classify an observation in COBWEB requires knowledge of all parameter values. In diagnosis, this amounts to probing each value and obtaining a result. In some cases this could lead to a significant number of probes (e.g., in our domain example from Section 2). In contrast, we would like to classify an object or event in as few probes as possible. One way in which we accomplish this is to examine the *salience* of each feature, calculating what amounts to a category utility for each feature within the scope of its parent node. Thus, a continuously-valued parameter has a salience of:

$$\frac{\sum_k^N P(C_k)[1/\sigma_{ik} - 1/\sigma_i]}{N}$$

Intuitively, parameters with larger salience contribute more to the category determination and more greatly influence the placement of an observation. Highly salient parameters serve to focus attention, and limit the number of parameters that actually need be probed during categorization. Table 1 illustrates the salience scores for the parameters used to describe behaviors at one node in the behavior tree.

The general philosophy behind a focusing method (Gennari, 1989) is that we wish to properly categorize a behavior by probing a minimal number of parameters. The general method is shown in Table 2. Categorization/diagnosis begins at the root of the categorization tree. The salience of all parameters is computed, and the parameter with highest salience is selected. The category utility measure is computed over the known

Table 2: Algorithm for Developing Salience List.

```
Algorithm: Attention[ ]
           [step 1] Compute the salience of all parameters;
                    store these scores at the root.
           [step 2] Select an unprobed parameter with greatest salience.
           [step 3] Find the best matching cluster by computing
                    category utility only over the probed parameter values.
           [step 4] Consider the probed parameters at the root;
                    does the sum of salience over these parameters surpass
                    a threshold?
           [step 5] If no, then goto [step 2],
                    else ignore remaining parameters and categorize the
                    behavior into the 'best' matching host over the parameters
                    sampled thus far. Go to step 1.
```

parameters, and the cluster that best matches the behavior is identified. If the total salience of selected parameters surpasses some threshold then the best cluster identified is selected to further categorize the behavior. In our experiments, we required that this sum of saliences was at least 1/3 the total salience sum over all parameters at the root. This heuristic strategy is prone to error, but intuitively captures the idea that a sufficient body of evidence has been accumulated on which to categorize the behavior. Once selected, the best cluster is treated as the root, and the salience scores are recomputed within this cluster's subpopulation of behaviors; subsequent parameter selection and categorization continues at this level of the tree. This process continues until a leaf is reached, at which point the parameter perturbation labeling this leaf is predicted as the perturbation that led to the behavior being classified.

To test the cluster-analytic approach to diagnosis, we performed 5 learning trials, in which a categorization tree was constructed over 400 training behaviors generated as described in Section 6.1. In addition, 100 randomly generated behaviors were categorized relative to the learned behavior tree. As Table 3 illustrates, testing with focusing yields slightly less average accuracy, but there is a large reduction in the number of parameters that need be probed. In general, the focusing algorithm as currently implemented with threshold, limits the number of probes that can be made, but in this domain, the small number appears sufficient for accurate diagnosis in this domain.

Table 3: Test results for COBWEB without and with focusing.

clustering system	*accuracy* (average)	*parameters evaluated* (average)
without Focusing	92.5	25.0
with Focusing	90.4	6.1

In model-based strategies such as candidate generation, the data is rendered explicit through candidate generation on a problem-specific basis. In this case, the reasonableness of an inductive, data-driven process over a problem population to mitigate the need for repeated candidate generation is clear. In addition, inductive learning methods can be applied when no explicit model of the system exists at all, but observations of system behavior can be used as data for an inductive engine (Manganaris, Fisher, & Kulkarni, 1993). We describe this work next.

6.6 Discovering and Exploiting Operating Modes in the Space Shuttle RCS

The Reaction Control System (RCS) of the Space Shuttle provides propulsive forces from a set of jets to control the motion of the Shuttle (pitch, yaw, roll). It replaces the aerodynamic surfaces, which become ineffective in the upper atmosphere. The RCS is located in three different areas of the orbiter. The forward RCS module is in the forward fuselage nose area. The aft (right and left) RCS modules are located in the right and left RCS pods, which are attached to the aft fuselage. Each RCS has two subsystems. One for each propellant: Oxidizer (OX) and Fuel (FU). The OX and FU subsystems are very similar in construction. Each consists of a Helium system, a propellant system (OX or FU), cross-feed and interconnect capabilities, and a jet thruster system. The helium system is used to pressurize the propellant and drive it to the jets. It consists of a Helium tank storing helium under high pressure, two legs in parallel, of two pressure regulators in series, each controlled by a valve. The propellant system consists of a propellant tank and an isolation valve at its output. The jet system consists of a manifold (pipes and valves), and jets. A FU and an OX pipe goes to each jet. A jet fires when OX and FU are allowed contact in its chamber.

Many quantities of the system are transmitted back to earth via a telemetry link. Each RCS has two He pressure sensors at the He tank, one temperature sensor at the He tank, one pressure sensor for the ullage pressure in the propellant tank, one temperature sensor at the propellant tank, and one pressure sensor at the output of the propellant tank. In addition, every valve's position is transmitted as sensed (talk-back) and every command affecting a valve is also transmitted.

Induction for diagnosis of the RCS was initiated by Stefanos Manganaris while visiting NASA Ames in Summer, 1992. This project also involves clustering and other forms of inductive learning, but requires attention to time-dependent, telemetry data. In particular, incoming data must be incrementally interpreted to verify expected changes in the operating mode of the RCS (e.g., jets on a predetermined point) or unexpected changes due to faults. The primary hurdle in developing such an automated system is the identification of operating modes from time series data. For example, proper functioning of jets cannot be determined from a single snapshot of RCS parameter (e.g., pressure) values, but only from trends of these parameters observed over time (e.g., decreasing pressure).

Over a set of training data, our method (Manganaris, Fisher, & Kulkarni, 1993) segments the values of each continuous variable, then fits an equation to each segment (i.e., the variable's values over a time interval). Each equation or variable model summarizes the variable's behavior over the time interval by features such as its mean, slope, and

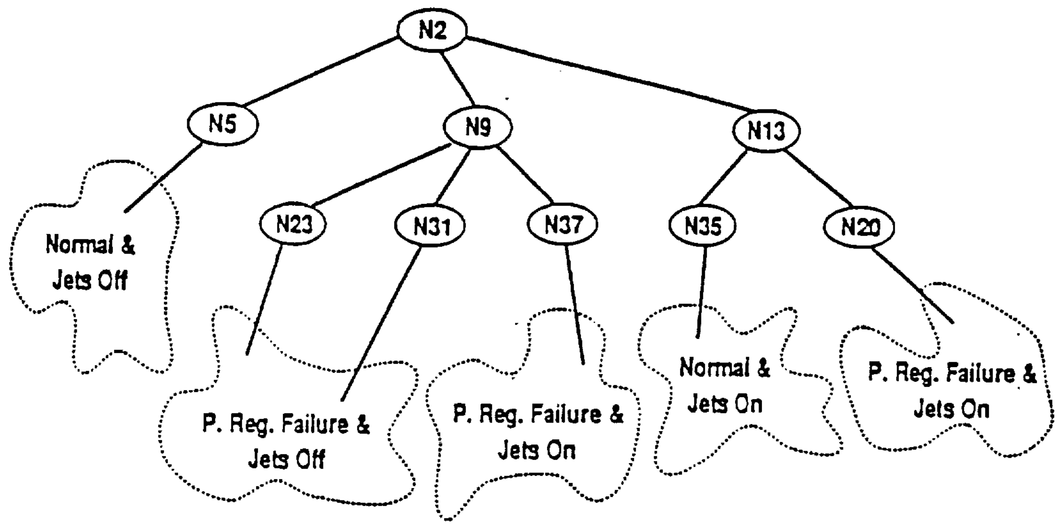

N2
N5
N9
N13
N23
N31
N37
N35
N20
Normal & Jets Off
P. Reg. Failure & Jets Off
P. Reg. Failure & Jets On
Normal & Jets On
P. Reg. Failure & Jets On

Figure 20: A Hierarchy of Behavior Classes Corresponding to RCS's Operating Modes

acceleration. These trend summaries are then passed to COBWEB/3 for analysis.

Initial results using COBWEB/3 indicate that trends are well partitioned into categories that reflect known (faulty or correct) operating modes. In particular, we have run several experiments using RCS data. Twelve quantities were monitored in addition to the discrete commands and talk-back for detecting configuration changes. Behavior summaries in this experiment consist of the two parameters of the best fitting *linear* approximation models (i.e., slope and intercept) and the squares of the correlation coefficient for *each* of the twelve quantities. RCS behavior in each interval is thus characterized by thirty-six (3×12) numerical attributes. Noisy data, from several minutes of RCS operation and under various conditions, were processed to generate fifty-six behavior summaries.

Figure 20 shows a typical hierarchy of classes formed by COBWEB/3. The leaf nodes correspond to individual behavior summaries. They were manually labeled according to the configuration of the RCS in the corresponding interval, for testing the accuracy of predictions. The operating modes of the RCS we have studied can be roughly classified into four categories: normal, with the jets on or off, and abnormal, with a failed pressure regulator and the jets either on or off. The labels were not used for inducing the classes. Node N9 corresponds to the class of behaviors when a pressure regulator has failed, and nodes N35 and N5 correspond to the class of normal behaviors.

To test the accuracy of the acquired knowledge we ran experiments, where we predicted the class for behaviors that COBWEB/3 was not trained on. We focused on behaviors where a pressure regulator has failed closed and on nominal behaviors, under different operating conditions: when different jets fire, for different periods, under different temperatures, etc. For each experiment we trained COBWEB/3 on fifty behavior summaries, presented in random order, and tested prediction accuracy on six behavior summaries, which were not used for training. The accuracy achieved was 85.5%, averaged over 30 runs. Given the limited amount of data for training, this level of accuracy is promising.

In this case, categorization is performed as we noted earlier - a behavior summary is matched against all categories at the top level of the tree. The behavior is categorized relative to the 'best matching' behavior category, and categorization recursively continues downward to descendents of this best matching node. In this project, where the system parameters on which categorization is based are all being monitored and transmitted to Earth, this categorization strategy is reasonable. However, if sensed data is not sufficient for categorization and further probing is necessary, then we cannot assume all features can be accessed simultaneously. Rather, we wish to select and execute individual probes in some order. This was the motivation for the attention-focusing approach that we described earlier.

6.7 Current Research

Our work with the Space Shuttle RCS, as described thus far, involves several stages of data processing and analysis: segmentation, summarization, and clustering. Each of these can be improved. Currently, we segment into uniform ranges fit this with a linear equation. However, we plan to investigate other methods which we briefly outline. In general, the characteristics of a signal vary depending on the operating mode of the system. The Data Acquisition module partitions the continuous stream of data points into intervals of homogeneous behavior characteristics. Depending on the signal's class, different techniques can be used. For example, one can detect abrupt changes in the spectral behavior of a signal, or abrupt changes in the mean level. For deterministic signals for which approximate models are known one can use the Minimum Message Length principle to decide when a break would yield a "better" description (Rao & Lu, 1992). Naturally, information regarding mode changes can also be gained by observing known commands to and talk-back from the system.

After an interval has been identified, the behavior of all signals is characterized. Depending on the signal, one or more of various methods can be used. One can characterize a signal's amplitude. A simple way is to compute its mean value, its mean square value (or power), and its variance. A more informative technique, which captures all the above information and more, estimates the probability density function (*pdf*) of the amplitude. The *pdf*, however, does not contain any information about the shape of a signals plot over time. To capture this, one can either characterize the signals spectrum or estimate its autocorrelation function. A promising approach to segmentation and subsequent fitting of multiple models to the data are the use of MDL principles as reported by Rao and Lu (1992).

Thus far, we have discussed cluster analysis for discovering operating modes. This involves segmenting and summarizing temporal data streams in preparation for clustering. A second aspect of this work is the discovery of transitions between operating modes. Transitions can be identified, in part, by talkback with the system However, we wish to independently verify that transitions are as expected. Our tentative approach is re-

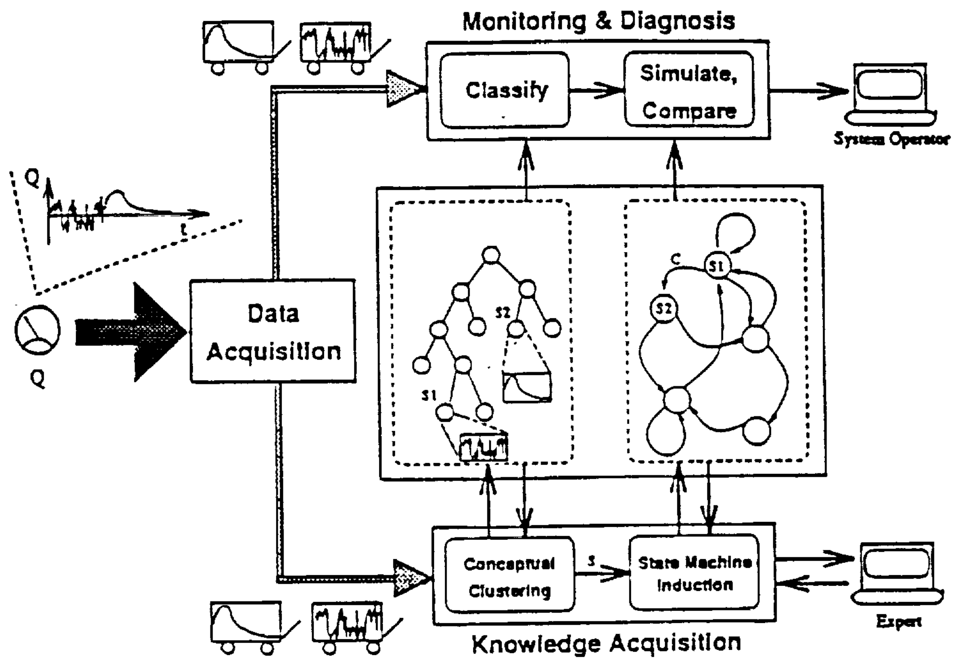


Figure 21: Diagnostic system for the RCS.

gard operating mode categories discovered through clustering as a reduced 'alphabet' of behaviors. Telemetry data is reexpressed in terms of this alphabet and methods of state-machine or sequence induction (Dietterich & Michalslki, 1986) are applied to discover a network of transitions that are to be expected from the data sampled or, in the case of Space Station simulations, from the data generated through model-based simulation.

The complete architecture that we propose is illustrated in Figure 21. Behavior summaries, generated by the initial 'data acquisition' component, are first classified using the classifier constructed by COBWEB. If a behavior summary is very different from all known classes, a new singleton class is formed (via standard COBWEB methods that we have not elaborated here). Novel behaviors are thus detected and the system operator is warned, even when the system has not been trained on them. When a behavior summary is classified to a known class annotated as faulty, again the system operator is warned and information from the annotation is also presented. When the classification is to a known class annotated as normal, that class is compared to the one predicted by the state machine from the last known state. If the transition at hand is novel, and thus unexpected, or is known to be associated with a malfunction, an appropriate warning is generated.

6.8 Summary

Our work in diagnosis renders diagnosis more efficient than model-based approaches when these models exist and are used, and it is perhaps the most viable approach when no model exists, as is the case in our work with the Space Shuttle. We proposed a cluster-analytic approach that is quite dissimilar from earlier work by Wu (1990, 1991) in implementation, but similar in intent. Like Wu, we define a measure of explanatory power known as *category utility*, and use it to organize observations into similarity classes. These probabilistically-described behavior classes are efficiently examined for purposes of probe selection. Probe selection with probabilistic class definitions is not as efficient as say a decision tree approach, but it incorporates redundancies that make it robust in the face of noise and flexible in the face of differing initially known conditions.

References

Ahn, W. (1990). Effects of background knowledge on family resemblance sorting. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society* (pp. 149–156). Cambridge, MA: Lawrence Erlbaum.

Allen, J., & Langley, P. (1989) Using concept hierarchies to organize plan knowledge. In *Proceedings of the Sixth International Workshop on Machine Learning*, Ithaca, NY: Morgan Kaufmann.

Anderson, J. R. (1974). Retrieval of propositional information from long term memory. *Cognitive Psychology, 6*, 451–474.

Anderson, J. R. (1983). *The Architecture of Cognition.* Cambridge, MA: Harvard University Press.

Anderson, J. R. (1990). *The adaptive character of thought.* Hillsdale, NJ: Lawrence Erlbaum.

Anderson, J. R., & Kline, P. J. (1979). A learning system and its psychological implications. *Proceedings of the Sixth International Joint Conference on Artificial Intelligence* (pp. 16–21). Tokyo, Japan: Morgan Kaufmann.

Anderson, J. R., & Matessa, M. (1991). An incremental Bayesian algorithm for categorization. In D. Fisher, M. Pazzani, & P. Langley (Eds.), *Concept formation: Knowledge and experience in unsupervised learning.* San Mateo, CA: Morgan Kaufmann.

Anderson, J. S., & Farley, A. (1988). Plan abstraction based on operator generalization. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, St Paul, MN: Morgan Kaufmann.

Barsalou, L. W. (1983). Ad hoc categories. *Memory & Cognition*, *11*, 211–227.

Barsalou, L. W. (1985). Ideals, central tendency, and frequency of instantiation as determinants of graded structure in categories. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *11*, 629–654.

Bransford, J. D., Sherwood, R. D., Hasselbring, T. S., Kinzer, C. K., & Williams, S. M. (1990). Anchored instruction: Why we need it and how technology can help. In D. Nix and R. Spiro (Ed.), *Cognition, education, multimedia: Exploring ideas in high technology.*

Carnes, R., & Fisher, D. (1992). Inductive Learning Approaches to Sensor Placement and Diagnosis, *Second International Conference on Diagnosis.* Rosario, WA.

Chi, M., Feltovich, P., & Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive Science*, *5*, 121–152.

Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., & Freeman, D. (1988). AUTOCLASS: A Bayesian learning system. *MLC-88.* Ann Arbor, MI.

Corter, J. E., & Gluck, M. A. (1992). Explaining basic categories: feature predictability and information. *Psychological Bulletin, 111*, 291–303.

DeKleer, J. (1991). Focusing on probable diagnoses. *AAAI-91.* Anaheim, CA.

DeGroot, A. D. (1966). Perception and memory versus thought: Some ideas and recent findings. In B. Kleinmuntz (Ed.), *Problem solving: Research, methods, and theory.* New York: John Wiley.

Dietterich, T., & Michalski, R. (1986). Learning to predict sequences. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.) *Machine Learning II.* Morgan Kaufmann.

Doyle, R., Chien, S., Fayyad, U., & Porta, H. (1992). Attention focusing and anomaly detection in real-time systems monitoring. *Second International Conference on Diagnosis.* Rosario, WA.

Feigenbaum, E. (1961). The simulation of verbal learning behavior. In J. W. Shavlik and T. G. Dietterich (Eds.), *Readings in Machine Learning*, San Mateo, CA: Morgan Kaufmann, 1990. (Reprinted from *Proceedings of the Western Joint Computer Conference*, 121–132.)

Fikes, R., Hart, P., & Nilsson, N. (1972). Learning and executing generalized robot plans. *Artificial Intelligence, 3*, 151–188.

Fisher, D. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning.*

Fisher, D. H. (1989). Noise-tolerant conceptual clustering. *Proceedings of the International Joint Conference Artificial Intelligence* (pp. 825–830). Detroit, MI: Morgan Kaufmann.

Fisher, D., & Langley, P. (1990). The structure and formation of natural categories. In G. H. Bower (Ed.), *The Psychology of Learning and Motivation, 26.* San Diego, CA: Academic Press.

Fisher, D., & Pazzani, M. (1991). Computational models of concept learning. In D. Fisher & M. Pazzani (Eds.), *Concept formation: Knowledge and experience in unsupervised learning.* San Mateo, CA: Morgan Kaufmann.

Fisher, D., Xu, L., Carnes, R., Reich, Y., Fenves, S., Chen, J., Shiavi, R., Biswas, G., & Weinberg, J. (1991). *Selected applications of an AI clustering technique to engineering tasks* (Technical Report CS-91-08). Nashville, TN: Department of Computer Science, Vanderbilt University.

Fisher, D., Xu, L., & Zard, N. (1992). Ordering effects in clustering. In *Proceedings of the Ninth International Conference on Machine Learning* (in press). San Mateo, CA: Morgan Kaufmann.

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

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

Gluck, M. A., & Corter, J. E. (1985). Information, uncertainty, and the utility of categories. *Proceedings of the Seventh Annual Conference of the Cognitive Science Society* (pp. 283–287). Irvine, CA: Lawrence Erlbaum.

Hammond, K. (1989). Case-based planning: viewing planning as a memory task. *Perspectives in Artificial Intelligence, I.* Academic Press.

Iba, G. (1989). A heuristic approach to the discovery of macro-operators. *Machine Learning, 3*, 285–318.

Jolicoeur, P., Gluck, M., & Kosslyn, S., (1984). Pictures and names: Making the connection. *Cognitive Psychology, 16*, 243–275.

Kline, P. J. (1983). *Computing the similarity of structured objects by means of heuristic search for correspondences.* (Doctoral Dissertation). Ann Arbor, MI: University of Michigan.

Knoblock, C. (1990). Learning abstraction hierarchies for problem solving. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA: Morgan Kaufmann.

Koedinger, K. R., & Anderson, J. R. (1990). Abstract planning and perceptual chunks: Elements of expertise in geometry. *Cognitive Science, 14*, 511–550.

Kolodner, J. L. (1983). Reconstructive memory: A computer model. *Cognitive Science, 7*, 281–328.

Kolodner, J. L. (1987). Extending problem solver capabilities through case-based reasoning. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 167–178). Irvine, CA: Morgan Kaufmann.

Korf, R. E. (1987). Planning as search: A quantitative approach. *Artificial Intelligence, 33*, 65–88.

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.

Larkin, J. H. (1981). Enriching formal knowledge: A model for learning to solve textbook physics problems. In J. R. Anderson (Ed.), *Cognitive Skills and Their Acquisition.* Hillsdale, NJ: Lawrence Erlbaum.

Lassaline, M. E., Wisniewski, E. J., & Medin, D. L. (in press). Basic levels in artificial and natural categories: Are all basic levels created equal? In B. Burns (Ed.), *Percepts, Concepts, and Categories: The Representation and Processing of Information.* Amsterdam: North Holland.

Manganaris, S., Fisher, D., & Kulkarni, D. (in press). Towards a machine learning framework for acquiring and exploiting monitoring and diagnostic knowledge. *Applications of AI: Knowledge-Based Systems in Aerospace and Industry.* Orlando, FL.

Marr, D. (1982). *Vision.* San Fransico: Freeman.

Mayer, R. (1981). Frequency norms and structural analysis of algebra story problems into families, categories, and templates. *Instructional Science*, *10*, 135–175.

McKusick, K., & Thompson, K. (1990). COBWEB/3: A portable implementation. Technical Report FIA-90-6-18-2. NASA Ames Research Center, CA.

Medin, D. (1983). Structural principles of categorization. In T. Tighe & B. Shepp (Eds.), *Perception, cognition, and development.* Hillsdale, NJ: Lawrence Erlbaum.

Medin, D. L. (1989). Concepts and conceptual structure. *American Psychologist*, *44*, 1469–1481.

Minton, S. (1988). Quantitative results concerning the utility of explanation-based learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, St Paul, MN: Morgan Kaufmann.

Mitchell, T., Keller, R., & Kedar-Cabelli, S. (1986). Explanation-based learning: a unifying view. *Machine Learning, 1*, 47–80.

Mooney, R. (1989). The effect of rule use on the utility of explanation-based learning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI: Morgan Kaufmann.

Morris, M. W., & Murphy, G. L. (1990). Converging operations on a basic level in event taxonomies. *Memory & Cognition*, *18*, 407–418.

Murphy, G., & Smith, E. (1982). Basic level superiority in picture categorization. *Journal of Verbal Learning and Verbal Behavior*, *21*, 1–20.

Quinlan, J. R. (1987). Simplifying decision trees. *International Journal of Man-Machine Studies.*

Pazzani, M. (1990). *Creating a memory of causal relationships: An integration of empirical and explanation-based learning methods.* Hillsdale, NJ: Lawrence Ehrlbaum.

Perez, R. S. (1991). A view from troubleshooting. In M. U. Smith (Ed.) *Toward a Unified View of Problem Solving: Views from the Content Domains.* Hillsdale, NJ: Lawrence Erlbaum.

Porter, B., Bareiss, R., & Holte, R. (1990). Concept learning and heuristic classification in weak-theory domains. *Artificial Intelligence, 45*, 229–263.

Rao, R., & Lu, S. (1992). Learning engineering models with the minimum description length principle. *AAAI-92.* San Jose, CA.

Reder, L. M., & Ross, B. H. (1983). Integrated knowledge in different tasks: The role of retrieval on fan effects. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 9*, 55–72.

Reed, S. K. (1989). Constraints on the Abstraction of Solutions. *Journal of Educational Psychology, 81*, 532–540.

Reed, S. K., Dempster, A., & Ettinger, M. (1985). Usefulness of analogous solutions for solving algebra word problems. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 11*, 106–125.

Richman, H. (1991). Discrimination net models of concept formation. In D. Fisher, M. Pazzani, & P. Langley (Eds.), *Concept formation: Knowledge and experience in unsupervised learning.* San Mateo, CA: Morgan Kaufmann.

Rifkin, A. (1985). Evidence for a basic level in event taxonomies. *Memory & Cognition, 13*, 538–556.

Rosch, E., & Mervis, C. (1975). Family resemblances: studies in the internal structure of categories. *Cognitive Psychology, 7*, 573–605.

Rosch, E., Mervis, C., Gray, W., Johnson, D., & Boyes-Braem, P. (1976). Basic objects in natural categories. *Cognitive Psychology, 18*, 382–439.

Ross, B. H., & Spalding, T. (1991). Some influences of instance comparisons on concept formation. In D. Fisher, M. Pazzani, & P. Langley (Eds.), *Concept formation: Knowledge and experience in unsupervised learning.* San Mateo, CA: Morgan Kaufmann.

Ruby, D., & Kibler, D. (1991). SteppingStone: An empirical and analytical Evaluation. In *Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 527–532). Anaheim, CA: AAAI Press.

Shrager, J., Hogg, T., & Huberman, B. A. (1988). A graph-dynamic model of the power law of practice and the problem-solving fan effect. *Science, 242*, 414–416.

Silber, J., & Fisher, D. (1989). A model of natural category structure and its behavioral implications. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 884–891). Ann Arbor, MI: Lawrence Erlbaum.

Simon, H. A. (1969). *The sciences of the artificial.* Cambridge, MA: MIT Press.

Simon, H. A., & Lea, G. (1974). Problem solving and rule induction: A unified view. In L. W. Gregg (Ed.), *Knowledge and cognition.* Hillsdale, NJ: Lawrence Erlbaum.

Sleeman, D., Hirsh, H., Ellery, I., & Kim, I. (1990). Extending domain theories: Two case studies in student modeling. *Machine Learning, 5,* 11–38.

Sussman, G. (1975). *A Computer Model of Skill Acquisition.* New York: American Elsevier.

Smith, E. E., & Medin, D. L. (1981). *Categories and concepts.* Cambridge, MA: Harvard University Press.

Tan, M., & Schlimmer, J. (1990). Two case studies in cost-sensitive concept acquisition. *AAAI-90.* Boston, MA.

Vere, S. A. (1978). Inductive learning of relational productions. In D. A. Waterman & F. Hayes-Roth (Eds.), *Pattern-Directed Inference Systems.* New York: Academic Press.

Weld, D., & Addanki, S. (1990). Task-driven model abstraction. University of Washington.

Wisniewski, E., & Medin, D. L. (1991). Harpoons and long sticks: The interaction of theory and similarity in rule induction. In D. Fisher & M. Pazzani (Eds.), *Concept formation: Knowledge and experience in unsupervised learning.* san Mateo, CA: Morgan Kaufmann.

Yang, H. (1992). *Learning Abstract and Macro Operators in AI Planning,* Ph.D. Dissertation. Department of Computer Science, Vanderbilt University, Nashville, TN.

Yoo, J., & Fisher, D. (1991a). Concept formation over explanations and problem solving experience. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence* (pp. 630–636). Sydney, Australia: Morgan Kaufmann.

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