Improving Efficiency by Learning Intermediate Concepts

James Wogulis (WOGULIS@ICS.UCI.EDU)
Pat Langley (LANGLEY@ICS.UCI.EDU)
Department of Information & Computer Science
University of California, Irvine, CA 92717 USA

Abstract

One goal of explanation-based learning is to transform knowledge into an operational form for efficient use. Typically, this involves rewriting concept descriptions in terms of the predicates used to describe examples. In this paper we present RINCON, a system that extends domain theories from examples with the goal of maximizing classification efficiency. RINCON'S basic learning operator involves the introduction of new intermediate concepts into a domain theory, which can be viewed as the inverse of the operationalization process. We discuss the system's learning algorithm and its relation to work on explanation-based learning, incremental concept formation, representation change, and pattern matching. We also present experimental evidence from two natural domains that indicates the addition of intermediate concepts can improve classification efficiency.

1 Introduction

Knowledge is necessary but not sufficient for intelligent behavior. In addition, knowledge must be stored in some form that lets it be used effectively. One of the central goals of machine learning is to devise mechanisms that transform knowledge from inefficient forms into more efficient ones. Most research on this topic has focused on explanation-based learning [Mitchell *et al* ., 1986, DeJong and Mooney, 1986], which augments a domain theory with rules that are more 'operational' than the original ones. Such operational rules let one bypass intermediate concepts, producing shallower proofs on future cases with the same structure.

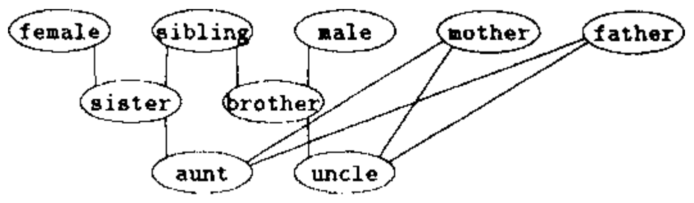
In this paper, we show that more operational knowledge does not always lead to more efficient behavior. In addition, we describe an alternative approach that involves the introduction of new intermediate concepts into the domain theory - effectively the inverse of operationalization. We show that, at least in some domains, this form of learning leads to more efficient forms of knowledge than do explanation-based methods.

In the following section we describe RINCON (Retaining INtermediate CONcepts), a learning system that implements our approach to the transformation of domain knowledge. After this, we report experiments with the system on two natural domains. Finally, we show how RINCON provides a framework for integrating explanation-based learning, incremental concept formation, representation change, and pattern matching.

2 Overview of RINCON

2.1 Representation and organization

RINCON is a system that forms domain theories from examples with the goal of maximizing classification efficiency. Instances are represented as conjunctions of n-ary predicates, allowing one to represent not only attributes, but also relations [Vere, 1975]. For example, father(A,B) A female(B) expresses a father-daughter relationship. Instances also contain a class label that is used for supervised learning.



```
aunt(X,Y)     <- sister(X,Z) ∧ mother(Z,Y)
aunt(X,Y)     <- sister(X,Z) ∧ father(Z,Y)
uncle(X,Y)    <- brother(X,Z) ∧ mother(Z,Y)
uncle(X,Y)    <- brother(X,Z) ∧ father(Z,Y)
sister(X,Y)   <- female(X) ∧ sibling(X,Y)
brother(X,Y)  <- male(X) ∧ sibling(X,Y)
```

Figure 1. A domain theory/hierarchy for family relationships.

Instances and concepts are stored hierarchically in a domain theory that is partially ordered according to the generality of the concepts. Figure 1 shows a simple hierarchy of concepts from a domain theory for family relationships. The highest-level concepts in the domain theory are the primitive features (predicates) used to represent instances. The lowest-level concepts correspond to the classes found in the training examples and may be disjunctive. The learned internal concepts must be conjunctive, appearing in the head of only one rewrite rule. All concepts are expressed in terms of higher-level concepts in the domain theory. For example, Figure 1 shows primitive features used to describe the concept brother, which is used to describe the concept uncle.

2.2 The performance system

The domain theory is used to classify instances. Given an instance and a concept, RINCON determines if the instance is described by the concept. If the concept is relational (conjunctions of n-ary predicates), then the system also determines all of the ways (different bindings) in which the instance is a member of the concept. The matching process is goal directed, starting with the concept to be determined and recursively finding all matches for each subconcept composing the concept.[1] Each time a concept node is matched, the resulting bindings are stored with that concept's node. By storing all matches for all relevant sub concepts, time may be saved if the bindings are needed again. The match algorithm is shown in Table 1.

Table 1. The Match Algorithm used by RiNCON

```
function match(concept, instance)
  if concept has not already been matched
    then if concept is a primitive feature
           then store all matches of concept with
                features from instance
           else for each disjunct in concept do
                  match-disjunct(disjunct, instance)
                store matches from all disjuncts in
                  the node concept
  return stored bindings

function match-disjunct(disjunct, instance)
 let bindings be the list containing the empty list
 for each concept used in disjunct
  while bindings is not empty do
     let bindings be the join of each
       bindings list in bindings with
       each binding in match(concept, instance)
 return bindings
```

As an example of how internal concepts can improve overall match efficiency, consider the following simple domain theory for the concept uncle:[2]

uncle(X,Y) <- male(X) A sibling(X,Z) A mother(Z,Y)
uncle(X,Y) <- male(X) A sibling(X,Z) A father(Z,Y).

Now suppose this domain theory is used to determine all of the uncle relations in the instance male(pat) A sibling(pat,John) A father(John,jean) A male(frank) A sibling(frank,marie) A mother(marie,jean). Since there are two uncles in the instance, the matcher would have to re-join the bindings from the male and sibling concepts. Instead, suppose the domain theory included the concept brother:

```
uncle(X,Y)    <- brother(X,Z) ∧ mother(Z,Y)
uncle(X,Y)    <- brother(X,Z) ∧ father(Z,Y)
brother(X,Y)  <- male(X) ∧ sibling(X,Y).
```

This domain theory would be more efficient to use since the work of matching the brother concept would only be done once when matching against the two definitions for uncle. The next section describes how one can acquire such internal concepts.

2.3 The RiNCON learning algorithm

The RiNCON system begins with an initial domain theory and incrementally extends it to incorporate new instances. At present, the learned theory does not go beyond the data; it simply organizes the instances according to the existing domain theory and any learned intermediate concepts. RINCON'S goal is to produce domain theories that maximize the classification efficiency for both seen and unseen instances. Table 2 presents the algorithm for learning new intermediate concepts.

Table 2. Algorithm for Learning Intermediate Concepts

```
function incorporate(instance, concept, theory)
 match(concept, instance)
 if no bindings exist for concept
  then
   let matched be the most specific concepts
         in theory that do match instance
   let notmatched be the most general concepts
         in theory that do not match instance
   rewrite instance in terms of concepts in matched
   add instance to theory
   let G be all maximally specific generalizations
         of instance with concepts in notmatched
   let NewG be the generalization in G that can
         be used to re-express the most number of
         concepts in theory
   add NewG to theory
   re-express all concepts in theory
         in terms of NewG
   return theory
```

RINCON's learning algorithm carries out incremental hill climbing [Gennari *et a/.*, 1989] through the space of domain theories. The system starts by matching the new instance against the concept with the same label. If the instance is described by the domain theory, then no learning occurs and the existing theory is retained. Otherwise, it collects the most specific concepts that match the instance and the most general concepts that do not match the instance. The system then re-expresses the instance in terms of the concepts it does match and adds it to the domain theory as a new disjunct for its concept class. The re-expressed instance is then generalized [Vere, 1975] with each concept in the set of most general concepts it does not match. Each of these generalizations is a candidate for a new internal concept. RINCON's evaluation function selects the generalization that can be used to re-express the most concepts in the domain theory. The selected generalization is then added to the theory and used to re-express all of the concepts in the domain theory that it can.

As an example, assume the following domain theory, which contains only one instance:

```
uncle(walter,jim) <- male(walter) ∧
                     sibling(walter,carol) ∧
                     mother(carol,jim).
```

If RINCON is presented with the new instance

[1]This differs from logic programming. Instances in RINCON may contain variables but are treated as constants by the matcher. Hence, it does not perform unification.
[2]Another type of uncle is the husband of an aunt.

```
uncle(pat,,jean) <- male(pat) A sibling(pat,John) A
                    tather(j ohn,j ean)
```

it finds that the concept uncle in the domain theory does not match this instance. The system then finds the most specific concepts in the theory that do match (male, sibling, and father), and the most general concepts that do not match (uncle). IIINCON then rewrites the instance using the highest-level concepts matched. Since these are simply the primitive features, the instance description remains unchanged. The instance is then added to the domain theory and is generalized with all of the lowest-level concepts that do not match, in this case uncle. The only maximally specific generalization is male(X) A sibling(X,Y), which is added to the domain theory. This generalization is used to rewrite both of the uncle definitions to produce the following domain theory:[3]

```
uncle(walter,jim) <- brother(walter,carol) ∧
                     mother(carol,jim)
uncle(pat,jean)   <- brother(pat,john) ∧
                     father(john,jean)
brother(X,Y)      <- male(X) ∧ sibling(X,Y).
```

RINCON continues processing new instances, extending the domain theory to incorporate each new instance.

3 Experimental evaluation of RINCON

The goal of RINCON is to improve the efficiency of matching instances. Since the system currently does no induction, classification accuracy is irrelevant. Instead, the natural unit of measure is the amount of work required to match or reject an instance. We measure work in terms of the number ofjoin operations performed in the match process. A join occurs when two lists of bindings are combined to form a new consistent bindings list (which might be empty if the bindings are inconsistent). For attribute-value representations the join of *N* attributes is N — 1, since multiple bindings are never produced. The number of joins provides a reasonable measure of work since at least one join occurs whenever a concept node in the hierarchy is matched (see match-disjunct in Table 1). Also, the time required to perform a join is bounded by a constant for any given domain.

As a baseline for comparison in all of our experiments we measured the work performed by a corresponding domain theory with no intermediate concepts.[4] This 'flat' domain theory is simply an extensional description of all the observed instances.

Our first experiment involved building a domain theory from instances of mushrooms [Schlimmer, 1987] in which each instance was described as a conjunction of 23 attribute-value pairs. A total of 3,078 instances were available. The experiment began with an empty domain theory, to which RINCON incrementally added randomly chosen instances. After every ten instances were incorporated into the domain theory, we computed the average amount of work required for matching each of the previously seen instances. We also measured the average amount of work for matching the same number of mushroom instances not described by the domain theory. Figure 2 presents the learning curves for the average work of matching an instance as a function of the number of instances stored in the domain theory. Each curve shows the average over 25 different runs.

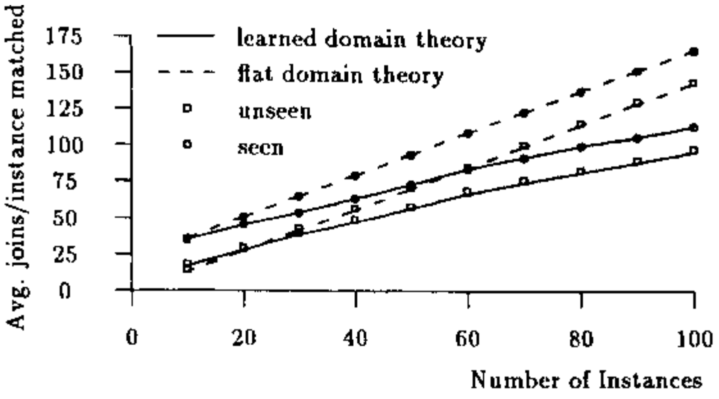


Figure 2. Average work to match previously seen and unseen mushroom instances.

The figure shows that the domain theory containing intermediate concepts was on average more efficient at matching previously seen instances than was the corresponding flat domain theory. Surprisingly, the flat theory also required more match time to reject previously unseen instances than did the learned domain theory. This suggests that the learned theory contains intermediate concepts shared among all mushroom instances. Such intermediate concepts would save on the overall match time for unseen instances, since they would store bindings often needed in the match process.

The results presented in Figure 2 seem to run counter to the notion that operational domain theories are more efficient to use than those containing intermediate concepts. However, for some instances the flat domain theory is more efficient. At the end of each of the 25 experiments, for each 100 mushroom instances processed, we computed the percentage of work saved by using the learned domain theory over the flat one. Figure 3 shows the distribution of instances as a function of the percentage of work saved. Although work is saved on average, intermediate concepts sometimes do reduce efficiency. This suggests a trade-off between retaining intermediate concepts and operationalizing concepts.

The mushroom experiments measured the efficiency of learned domain theories as a function of the number of instances processed. The size of each mushroom instance was constant. Our second experiment measured the efficiency of learned domain theories as a function of the size of the instances matched while holding the number of instances in the domain theory constant. This experiment involved using RINCON to organize the rules of a production system. In this case, the instances' used to build the domain theory were the condition sides of production rules. Unlike the mushroom domain, these instances were relational and contained variables. The production system solved multi-column subtraction problems [Langley and Ohlsson, 1984] such as 128 - 39 using a set of nine

[3]We have named the new concept brother only for clarity.

[4]This is equivalent to a domain theory containing only 'operational' definitions.

rules. The rule set included such operators as subtracting two numbers in a column, shifting attention from one column to another, and borrowing ten from a column. The production rules were written such that only one rule with one set of bindings ever matched against working memory.

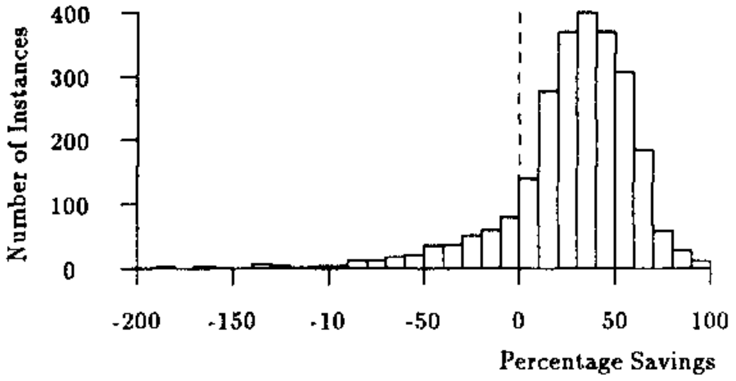


Figure 3. Distribution of instances as a function of work saved by using intermediate concepts.

The experiment consisted of running the production system on sets of subtraction problems of varying complexity, measured as the maximum number of columns in the problem. Each problem was solved using the domain theory of rules built by RINCON and the corresponding flat theory to find which rules matched against working memory. We computed the average work (number of joins) per production system cycle for both of the domain theories when solving each problem. Each cycle of the production system requires matching the rules in the domain theory against working memory.

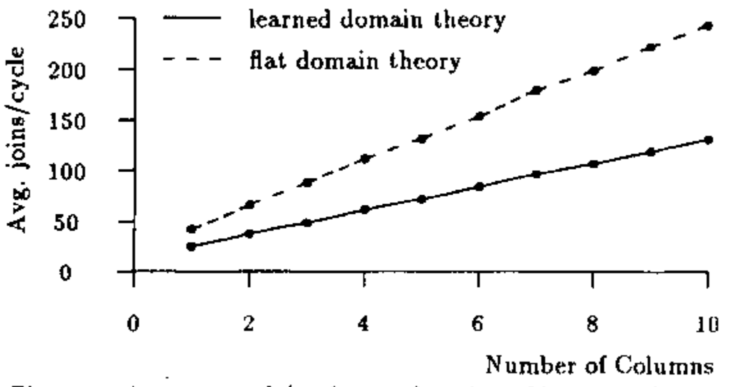


Figure 4. Average work/cycle as a function of instance size.

The graph in Figure 4 shows the average amount of work per cycle as a function of instance size for both of the domain theories. Each point in the graph is the average over 25 different subtraction problems at a given level of problem complexity. The curves for the flat domain theory and for the domain theory built by RINCON suggest that the average work per cycle is a linear function of the number of columns in the subtraction problem. This reflects the fact that the working memory increases linearly in the number of columns. Overall, the domain theory built by RINCON required about half as much work as the flat domain theory.

4 Discussion

The learning mechanism used in RINCON is closely related to methods used in four AI paradigms that have traditionally been viewed as quite diverse - explanation-based learning, incremental concept formation, representation change, and pattern matching. Below we expand on these relations noting some directions for future research.

4.1 Relation to explanation-based learning

Our approach to learning has much in common with work on explanation-based learning [Mitchell *et al.*, 1986, DeJong and Mooney, 1986]. In both cases, domain knowledge is organized as a set of inference rules, recognition involves constructing a proof tree by chaining off those rules, and learning alters the structure of the domain theory by adding new inference rules. Moreover, in both cases this process may affect the efficiency of recognition, but no induction is involved.[5]

However, the basic operations used in the two frameworks differ radically. Explanation-based learning modifies the knowledge base through a 'knowledge compilation* mechanism. The structure of an explanation is compiled into a new inference rule; this lets the performance system *bypass* intermediate terms on future cases with the same structure, giving shallower explanations. In contrast, our approach creates *new* intermediate terms, leading to deeper explanation structures on future cases. One can view RINCON's mechanism for creating new terms as a 'decompilation' process - the inverse operation of that in explanation-based systems.

Our experimental results indicate it is sometimes better to operationalize than to introduce intermediate concepts. An obvious extension to RINCON would be to include a mechanism for knowledge compilation in addition to that for new term creation. Upon encountering a previously unseen situation, the system would extend the knowledge base, generating new terms in the process. Upon recognizing a previously seen case, it would construct a compiled rule for matching the instance in a single inference step. To determine whether the compiled or uncompiled knowledge was more efficient, the system would keep statistics on each rule, eventually eliminating ones with low utility [Minton, 1988]. Such an extension would constitute an important step towards unifying inductive and analytic approaches to learning.

4.2 Relation to incremental concept formation

Gennari, Langley, and Fisher [1989] have reviewed work on incremental concept formation. In this framework one incrementally induces a taxonomy of concepts, which can then be used in classifying new instances and in making predictions. Each instance is sorted through the taxonomy, altering the knowledge base in passing.

Such learning can be characterized as an incremental form of hill climbing, in that only a single concept hier-

[5] In incremental mode, one can view RINCON as changing the deductive closure of its knowledge base, since it accepts new instances as input. However, the system does not move beyond the instances it is given.

archy is retained in memory. Examples of concept formation systems include Levinson's [1985] self-organizing system, Lebowitz's [1987] UNIMEM, Fisher's [1987] COBWEB, and Gennari *et al.*'s [1989] CLASSIT.

The learning method in RINCON can be viewed as a form of incremental concept formation. The domain theory constitutes a taxonomy, with primitive predicates as the most general concepts, instances as the most specific concepts, and defined terms as concepts of intermediate generality. New instances are 'sorted' down this concept hierarchy, and new concepts are introduced in the process. RINCON'S search control is an incremental form of hill climbing, preferring new terms that will be used by more existing concepts.

However, there are also some important differences between the two approaches. Research on concept formation has typically focused on attribute-value representations, whereas RiNCON employs a relational formalism. Most concept formation methods construct disjoint taxonomies, whereas RiNCON forms a nondisjoint hierarchy in which a concept may have multiple parents. Finally, most earlier methods have employed partial matching techniques in the classification process, which let them make predictions about, unseen data. In contrast, our approach uses complete matching and thus only summarizes the observed instances.

The last difference suggests extensions to RiNCON that would let it move beyond the data to make predictions about unseen instances (i.e., to do *induction).* The current system allows disjunctions only at the final level of the concept hierarchy, but the basic learning operator can be extended to create disjuncts at any level. The introduction of multiple disjuncts into a concept definition leads to coverage of unseen instances. A more radical approach involves deleting these structures entirely, so one need not match against them at all. In either case, the system would need to collect statistics to estimate the desirability of such drastic actions.

4.3 Relation to representation change

Another active area of machine learning research focuses on changing representations by introducing new terms into the language of concept descriptions. For instance, given a primitive set of features, a learning system might define new terms as conjunctions or disjunctions of these features, and then attempt to induce a concept description over this extended language. A variety of researchers have taken this general approach to representation change in induction [Fu and Buchanan, 1984, Schlimmer, 1987, Muggleton, 1987, Pagallo and Haussler, 1988, Rendell, 1988].

RINCON's learning method involves a variety of rep resentation change. When the system introduces a new concept into its domain theory, it redefines existing concepts using this term. Also, it uses these intermediate terms during the matching process to redescribe new instances. The more concepts in which an intermediate term is used, the more efficiently the system matches or rejects new instances. Thus, the change in representation has a definite impact on performance.

Muggleton's [1987] DUCE system employs constructive induction in much the same way as RiNCON, but has more operators for introducing new concepts. However, before a new concept is actually retained, the user is required to either accept or reject the concept. DUCE'S main goal is to maximize the symbol reduction of the rule base while creating meaningful intermediate concepts. On the other hand, RINCON'S main goal is to improve the domain theory's efficiency of recognizing instances. Also, RiNCON processes instances incrementally and handles relational input whereas DUCE is non-incremental and is limited to propositional calculus.

With the exception of Fu and Buchanan [1984], most earlier research on representation change has emphasized classification accuracy rather than efficiency. Another difference between RiNCON and other approaches involves its use of a relational formalism rather than a feature-based language. However, our work to date has dealt only with introducing new conjunctive terms. Future versions of RINCON should introduce disjunctive relational terms as well, as do most other methods for representation change.

4.4 Relation to pattern matching

Research on production-system architectures has led to algorithms and data structures for efficient pattern matching. One of the best-known schemes involves *rete networks* [Forgy, 1982], a memory organization that allows sharing of redundant conditions and storage of partial matches. This technique leads to significant reductions in the match time required for certain large production systems.[6]

The rete network approach to matching has many similarities to RINCON'S scheme. In both cases, the performance element stores partial matches at nodes in the network. More important, both methods construct internal nodes for this purpose, based on shared structures in the inputs. Finally, in both cases the resulting 'domain theory' is purely conjunctive, in that internal nodes have only one definition.

However, RINCON also differs in some significant ways from systems based on rete networks. First, Forgy's framework assumes a binary network, in which each internal node is defined as the conjunction of two other nodes. In contrast, our system can use an arbitrary number of nodes in its definitions. Second, methods for constructing rete networks typically detect shared structures only if they occupy the same positions in the condition sides of productions, and they automatically create nodes when they are found. RiNCON carries out a more sophisticated search for shared structures, and it employs an evaluation function to select among alternative concepts that it might construct. Thus, our scheme can be viewed as a heuristic approach to constructing generalized rete networks, and future work should compare the two methods empirically.

Levinson's [1985] work on self-organizing retrieval for graphs also extends Forgy's idea of improving retrieval

[6]Miranker [1987] has presented evidence that, in some cases, using intermediate nodes leads to slower matching. This corresponds to the 'flat' domain theory we used in our experiments; thus, our initial results side with rete networks.

efficiency by creating intermediate concepts. As in RINCON, intermediate concepts correspond to common structures found among the relational examples stored in the database. They may be added or deleted according to a heuristic information-theoretic measure of retrieval efficiency. Levinson's experiments in the retrieval of chemical structures show that introducing intermediate concepts results in only a fraction of the database (on the order of the log of the number of elements in the database) being compared to the query structure during retrieval. He also provides theoretical justification for this increase in efficiency. This reduction in search is critical in structured domains, in which the cost of comparison is potentially exponential in the size of the objects being compared.

5 Conclusion

RiNCON incrementally learns domain theories from examples with the goal of maximizing classification efficiency. The version described in this paper is only an initial step toward our goal of integrating inductive and explanation-based learning. We have focused here on aspects of the efficient use of knowledge, but future work should also address induction and the associated goal of maximizing classification accuracy.

Our preliminary results indicate that introducing intermediate concepts into a domain theory can increase overall match efficiency. This result seems counter to the work on explanation-based learning, which holds that operationalization is the key to efficiency. However, our results suggest that both views are correct. By adding an operationalization component to RiNCON, we will be able to explore the efficiency tradeoff between operationalization and introducing new intermediate concepts.

Finally, the RINCON framework is also closely related to research in the areas of incremental concept formation, representation change, and pattern matching. Our work impacts each of these areas and provides a framework for integrating these diverse fields.

Acknowledgements

We have benefited from discussions with Robert Levinson at the University of California, Santa Cruz. We would also like to thank Wayne Iba, John Gennari, and Mike Pazzani for their discussions on this work.

References

[DeJong and Mooney, 1986] Gerald F. DeJong and Raymond J. Mooney. Explanation-based learning: An alternate view. *Machine Learning,* 1:145-176, 1986.

[Fisher, 1987] Douglas H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning,* 2:139-172, 1987.

[Forgy, 1982] Charles L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence,* 19:17-37, 1982.

[Fu and Buchanan, 1984] Li-Min Fu and Bruce G. Buchanan. Enhancing performance of expert systems by automated discovery of meta-rules. In *Proceedings of the First Conference on Artificial Intelligence Applications,* pages 107-115, Denver, Colorado, 1984. IEEE Computer Society Press.

[Gennari *et al.,* 1989] John H. Gennari, Pat Langley, and Doug Fisher. Models of incremental concept formation. *Artificial Intelligence,* 40, 1989.

[Langley and Ohlsson, 1984] Pat Langley and Stellan Ohlsson. Automated cognitive modeling. In *Proceedings of the Fourth National Conference on Artificial Intelligence,* pages 193-197, Austin, Texas, 1984. Morgan Kaufmann.

[Lebowitz, 1987] Michael Lebowitz. Experiments with incremental concept formation: UNIMEM. *Machine Learning,* 2:103-138, 1987.

[Levinson, 1985] Robert A. Levinson. *A self organizing retrieval system for graphs.* PhD thesis, University of Texas, Austin, TX, 1985.

[Minton, 1988] Steven Minton. Quantitative results concerning the utility of explanation-based learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence,* pages 564-569, Saint Paul, Minnesota, 1988. Morgan Kaufmann.

[Miranker, 1987] Daniel P. Miranker. TREAT: A better match algorithm for AI production systems. In *Proceedings of the Sixth National Conference on Artificial Intelligence,* pages 42-47, Seattle, Washington, 1987. Morgan Kaufmann.

[Mitchell *et al.*, 1986] Tom M. Mitchell, Richard M. Keller, and Smadar T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning,* 1:47-80, 1986.

[Muggleton, 1987] Stephen Muggleton. DUCE, an oracle based approach to constructive induction. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence,* pages 287-292, Milan, Italy, 1987. Morgan Kaufmann.

[Pagallo and Haussler, 1988] Giulia Pagallo and David Haussler. Feature discovery in empirical learning. Technical Report UCSC-CRL-88-08, Board of Studies in Computer and Information Sciences, University of California at Santa Cruz, 1988.

[Rendell, 1988] Larry Rendell. Learning hard concepts. In *Proceedings of the Third European Working Session on Learning,* pages 177-200, Glasgow, Scotland, 1988. Pitman Publishing.

[Schlimmer, 1987] Jeffrey C. Schlimmer. *Concept acquisition through representation adjustment.* PhD thesis, University of California at Irvine, 1987.

[Vere, 1975] Steven A. Vere. Induction of concepts in the predicate calculus. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence,* pages 281-287, Tbilisi, USSR, 1975. Morgan Kaufmann.