Models of Incremental Concept Formation

John H. Gennari, Pat Langley and Doug Fisher*
Irvine Computational Intelligence Project,
Department of Information and Computer Science,
University of California, Irvine, CA 92717, U.S.A.

ABSTRACT

Given a set of observations, humans acquire concepts that organize those observations and use them in classifying future experiences. This type of concept formation can occur in the absence of a tutor and it can take place despite irrelevant and incomplete information. A reasonable model of such human concept learning should be both incremental and capable of handling the type of complex experiences that people encounter in the real world. In this paper, we review three previous models of incremental concept formation and then present CLASSIT, a model that extends these earlier systems. All of the models integrate the process of recognition and learning, and all can be viewed as carrying out search through the space of possible concept hierarchies. In an attempt to show that CLASSIT is a robust concept formation system, we also present some empirical studies of its behavior under a variety of conditions.

1. Introduction

Much of human learning can be viewed as a gradual process of *concept formation*. In this view, the agent observes a succession of objects or events from which he induces a hierarchy of concepts that summarize and organize his experience. This task is very similar to the problem of *conceptual clustering* as defined by Michalski and Stepp [30], with the added constraint that learning be incremental. More formally:

- *Given*: a sequential presentation of instances and their associated descriptions;
- *Find*: clusterings that group those instances in categories;
- *Find*: an intensional definition for each category that summarizes its instances;
- *Find*: a hierarchical organization for those categories.

* Current address: Department of Computer Science, Vanderbilt University, Nashville, TN, U.S.A.

The goals of conceptual clustering are straightforward: to help one better understand the world and to make predictions about its future behavior. Concept formation has essentially the same goals, and differs mainly in the constraints it places on achieving them.

In this paper, we focus on the concept formation task and examine some methods for incrementally forming clusters, concept descriptions, and concept hierarchies. We begin by attempting to abstract the features that are common to the existing work on concept formation and that set it apart from other approaches. After this, we review in some detail three models of the concept formation process—Feigenbaum's [9] EPAM, Lebowitz's [24, 26] UNIMEM, and Fisher's [12] COBWEB. Next we describe CLASSIT, an extension of Fisher's system, and report some experimental studies of the program's learning behavior. We close with some suggestions for future research and a summary of our main observations.

2. Methods for Concept Formation

The majority of machine learning research has focused on the broad area of concept learning. To many readers, the work on concept formation may seem a minor variation on better-known approaches, and it certainly has close ties to other work. However, methods for concept formation share a number of important features that, taken together, distinguish them from other efforts. In this section we identify those features that are common to the approach and that serve to separate it from alternative paradigms, particularly other methods for conceptual clustering. In some sense, one can also view these features as "defining" the term *concept formation.*

2.1. Representing knowledge in a concept hierarchy

The most obvious common feature of concept formation methods is their organization of knowledge into a *concept hierarchy*. This type of data structure contains a set of nodes partially ordered by generality, and thus is similar to the *is-a* hierarchies used by some machine learning systems (Michalski [29], Mitchell, Utgoff and Banerji [32]). Each node in a concept hierarchy represents a concept, but unlike most *is-a* hierarchies, each node also contains an intensional description of that concept.

The hierarchical organization of acquired concepts is one distinctive feature of methods for concept formation (and conceptual clustering). In contrast, most work on learning from examples (Mitchell [31], Michalski [29]) focuses on learning one or a few concepts at a single level of abstraction. Methods for constructing decision trees (Quinlan [34]) are closer in spirit, but lack any explicit descriptions on the nodes themselves.

2.2. Top-down classification of instances

The presence of a concept hierarchy suggests a natural approach for classifying new instances that is shared by all concept formation systems. One simply begins at the most general (top) node and sorts the instance down through the hierarchy. This classification method is very similar to that used by decision-tree systems. However, the scheme for determining which branch to follow need not be based on the result of a single attribute's value, and some concept formation systems allow the instance to follow more than one branch. Nor must the instance always be sorted to a terminal node; in principle, the sorting process may stop at a node higher in the hierarchy.

Once the instance has finished its descent, one can use the concept description at the selected node to make predictions about unseen aspects of the instance. Decision-tree systems typically make predictions about the class of the instance, but concept formation systems can make predictions about a wider range of features. This suggests measuring the performance of an acquired hierarchy in terms of its ability to make predictions about unseen attributes.[1] In principle, other methods for conceptual clustering could be evaluated along the same dimension, but few researchers have taken this approach.

2.3. Unsupervised nature of the learning task

The unsupervised nature of the learning task leads to another common feature of concept formation systems—they must cluster instances without advice from a teacher. In other words, they must decide not only which instances each class should contain, but also the number of such classes. This is the most important feature separating work on concept formation (and conceptual clustering) from research on learning conjunctive concepts from examples (Winston [44], Mitchell [31]).

Techniques for inducing decision trees (e.g., Quinlan [34]) come much closer to concept formation methods on this dimension. Although supervised in the sense that they are given teacher-specified class information, these systems must determine their own subclasses, which equates to forming instance clusters. Rendell, Seshu and Tcheng's [35] work on probabilistic concept learning has a similar flavor.

2.4. Integrating learning and performance

We have defined the concept formation task to be incremental in nature. By *incremental*, we mean not only that the agent accepts instances one at a time,

[1] Although all of the concept formation systems we will examine assume attribute-value representations, the framework we outline can handle relational or structural descriptions as well. See Levinson [27] for some initial work along these lines.

but also that it does not extensively reprocess previously encountered instances while incorporating the new one. Without this constraint, one could make any nonincremental method "incremental" simply by adding the new instance to an existing set and reapplying the nonincremental method to the extended set. Note that our definition of incremental does *not* forbid retaining all instances in memory, only the extensive reprocessing of those instances. In fact, most existing methods for concept formation retain at least some instances as terminal nodes in the concept hierarchy.

This focus on incremental learning leads naturally to the integration of learning with performance. In any incremental system (Winston [44], Mitchell [31], Schlimmer and Fisher [38]), action by the performance component (e.g., classifying an instance) drives the learning element (e.g., modifying a concept hierarchy). In contrast, nonincremental schemes (Michalski [29], Quinlan [34]) isolate the processes of learning and performance. Most research on both numerical taxonomy (Everitt [8]) and conceptual clustering (Michalski and Stepp [30], Fisher [11]) has taken a nonincremental approach. Thus, this dimension constitutes one major distinction between earlier approaches to clustering and concept formation as we have defined it.

The role of classification in concept formation systems exerts a strong influence on the nature of learning. We noted above that the performance component of these methods sort instances down through a concept hierarchy. As a result, it seems natural to acquire the concept hierarchies in a top-down fashion as well. Thus, concept formation methods typically construct their hierarchies in a *divisive* manner, rather than using the *agglomerative* approach more common within the statistical clustering community.[2]

2.5. Learning as incremental hill climbing

The features described above seem almost to follow from the task of concept formation itself, but the final commonality has a different flavor. The models we describe in the following pages can all be characterized as *incremental hill-climbing learners*. We have elaborated on this notion elsewhere (Langley, Gennari and Iba [21], Fisher [12]), and Schlimmer and Fisher [38] described the basic idea (without using this term) even earlier. One can view concept formation as a search through a space of concept hierarchies, and hill climbing is one possible method for controlling that search.

Hill climbing is a classic AI search method in which one applies all operator instantiations, compares the resulting states using an evaluation function, selects the best state, and iterates until no more progress can be made. There are many variants on the basic algorithm, but these do not concern us here. The main advantage of hill climbing is its low memory requirement; since there

[2] For one exception, see Hanson and Bauer's work on WITT [16], an agglomerative clustering system that can operate incrementally.

are never more than a few states in memory, it sidesteps the combinatorial memory requirements associated with search-intensive methods. However, it also suffers from well-known drawbacks, such as the tendency to halt at local optima and a dependence on step size.

We are using the term *hill climbing* in a nontraditional sense, focusing on some features and ignoring others. For instance, we do not require an incremental hill-climbing learner to have an explicit evaluation function, or even that it carry out a one-step lookahead. One can replace this approach with a strong generator that computes the successor state from new input, such as an observed instance. For our purposes, the main feature of a hill-climbing system is its limited memory. At each point in learning, the system may retain only one knowledge structure, even though this structure may itself be quite complex. Thus, hill-climbing learners cannot carry out a breadth-first search (Mitchell [3]) or a beam search (Michalski [29]) through the space of hypotheses, nor can they carry out explicit backtracking (Winston [44]). They can only move "forward," revising their single knowledge structure in the light of new experience.[3]

The most important difference between incremental hill-climbing learners and their traditional cousins lies in the role of input. As we have seen, incremental learning methods are driven by new instances, and in the case of incremental hill-climbing systems, this means that each step through the hypothesis space occurs in response to (and takes into account) some new experience. More generally, each instance may lead to a number of learning steps (e.g., one for each level in the concept hierarchy). In other words, the learner does not move through the space of hypotheses until it obtains a new datum, and this alters the nature of the hill-climbing task.

Recall that hill-climbing methods search an n-dimensional space over which some function f is defined. This function determines the shape of an n-dimensional surface, and the agent attempts to find that point with the highest f score. In traditional hill-climbing approaches, the shape of the surface is constant. In contrast, for systems that learn through incremental hill climbing, each new instance modifies the contours of the surface. Like Simon's [41] wandering ant, the learner's behavior is controlled by the shape of its world. However, the hills and valleys of the hill-climbing learner's space are constantly changing as it gathers more information, altering the path it follows.[4] This

[3] Some "strength-based" methods retain competing hypotheses in memory, gradually deleting some and adding others on the basis of their performance. Genetic algorithms (Holland [17], Grefenstette [15]) follow this approach, as do Anderson and Kline's [2] and Langley's [20] work on production system learning. One could view these methods as incremental hill-climbing learners, provided one treats the entire set of rules as a single "state." However, we believe this violates the spirit of our limited memory assumption.

[4] Note that this feature does *not* hold for nonincremental learners that use hill-climbing methods (Michalski and Stepp [30]) or greedy algorithms (Quinlan [34]); the shape of the surface over which these systems travel remains constant throughout the learning process.

feature of incremental hill climbing is novel enough that it becomes unclear whether the limitations of traditional hill-climbing methods still hold. It also gives the potential for dealing with *concept drift* (Schlimmer and Granger [39]), in which the environment actually changes over time.

However, this dependence on new instances to control the search process can make memory-limited incremental learning methods sensitive to the order of instance presentation. Initial nonrepresentative data may lead a learning system astray, and one would like it to recover when later data point the way to the correct knowledge structure. Thus, Schlimmer and Fisher [38] have argued for including *bidirectional* learning operators that can reverse the effects of previous learning should new instances suggest the need. In the context of concept formation, one might include an operator not only for creating new subcategories, but also for deleting them should they not prove useful. Similarly, one might desire an operator not only for creating new disjunctive classes, but also one for combining classes if the distinction fares poorly. Such bidirectional operators can give incremental hill-climbing learners the *effect* of backtracking search without the memory required by true backtracking. Whether this approach works or not is an empirical question, but in Section 5 we will see evidence that it can help significantly.

2.6. Summary

In this section we identified some common threads that run through a number of research efforts, and we borrowed the term *concept formation* to refer to this research area. The basic approach can be viewed as a form of conceptual clustering, but it also differs from "traditional" work in this area. The common features of concept formation methods include the hierarchical organization of concepts, top-down classification, and an unsupervised, incremental, hill-climbing approach to learning.

We should emphasize that none of these features by itself makes work on concept formation unique. It shares many of these features with other methods for conceptual clustering, and there exist many supervised learning methods that process instances incrementally. Even the incremental hill-climbing approach has been widely used within the machine learning community, though it has not been labeled as such.[5] However, when one takes all these features together, what emerges is a distinctive and promising approach to concept learning.

[5] For example, recent work on supervised concept learning (Schlimmer and Fisher [38], Iba, Wogulis and Langley [18]) has been within this paradigm, as has recent work on theory formation (Shrager [40], Rose and Langley [36]). Much of the work on grammar acquisition (Anderson [1], Berwick [3]) has also occurred within the incremental hill-climbing framework. Even such diverse paradigms as neural networks and explanation-based learning share incremental hill climbing as an unstated assumption.

3. Earlier Research on Concept Formation

Before describing our own research on concept formation, we should review previous work on the problem. In this section we review three models of this process—Feigenbaum's EPAM, Lebowitz's UNIMEM, and Fisher's COBWEB. We will see that, with minor exceptions, each system operates within the common framework described in the previous section. We will also see that each system addresses issues that its predecessor ignored. This does not mean later systems are superior to earlier ones, since they also ignore some issues addressed by their precursors. However, there has been clear progress on certain fronts, and we will focus on these. We describe each model in terms of its representation and organization of knowledge, its classification and learning methods, and its metric for evaluating the resulting concepts and hierarchies.

3.1. Feigenbaum's EPAM

Feigenbaum's EPAM [9] can be viewed as an early model of incremental concept formation.[6] The system was intended as a psychological model of human learning on verbal memorization tasks, and it successfully explained a variety of well-established learning phenomena. These included the serial position effect, the conditions for multi-trial versus one-trial learning, forgetting through oscillation and retroactive inhibition, and a number of other empirical generalizations.

3.1.1. *Representation and organization in* EPAM

EPAM represents each instance as a conjunction of attribute-value pairs, along with an optional ordered list of component objects. Each component is in turn described as a conjunction of attribute-value pairs, with its own optional components, and so forth. For instance, the system might represent the nonsense syllable GAK as a list of three component objects—the first letter, the second letter, and the third letter. Each letter might itself be described in terms of lower-level components (e.g., the lines making it up), or it might be viewed as a primitive object having only attributes and no components. For simplicity, we will avoid examples that involve components and focus on single-level instances that can be described purely in terms of attribute-value pairs.

EPAM represents and organizes its acquired knowledge in a *discrimination network*. Each nonterminal node in this network specifies some *test*, and each link emanating from this node corresponds to one possible result of that test. Some tests involve examining the value of an attribute, whereas others involve examining the category of a subobject, which can itself be learned. Each

[6] For a more comprehensive treatment of EPAM and its extensions, see Feigenbaum and Simon [10].

nonterminal node also includes a branch marked OTHER, which lets EPAM avoid specifying all possible results of the test at the outset. Each terminal node contains an *image*—a partial set of attribute values (and component categories) expected to hold for instances sorted to that node.

Consider the example discrimination network in Fig. 1, which includes only attribute tests. This domain assumes instances composed of a single cell with three attributes—surface color, number of nuclei, and number of tails. The root node in Fig. 1(a) contains a test on the attribute NUCLEI, and the two links emanating from this node are labeled ONE and OTHER. The leftmost successor is a terminal node and thus has an associated image; this contains the partial description NUCLEI = ONE ∧ TAILS = ONE. (Note that color is unspecified.) The rightmost successor is nonterminal and thus has an associated test, this one involving the attribute COLOR. One link (labeled LIGHT) points to a successor node with image COLOR = LIGHT ∧ NUCLEI = TWO. The other (labeled OTHER) leads to a successor node with image COLOR = DARK.

3.1.2. *Classification and learning in EPAM*

As with all the concept formation systems we will examine, EPAM's classification process is completely integrated with its learning method. Table 1 presents the top-level EPAM algorithm, which focuses on performance. As the system encounters each instance, it sorts that instance through the discrimination network, starting at the top (root) node and proceeding until it reaches a

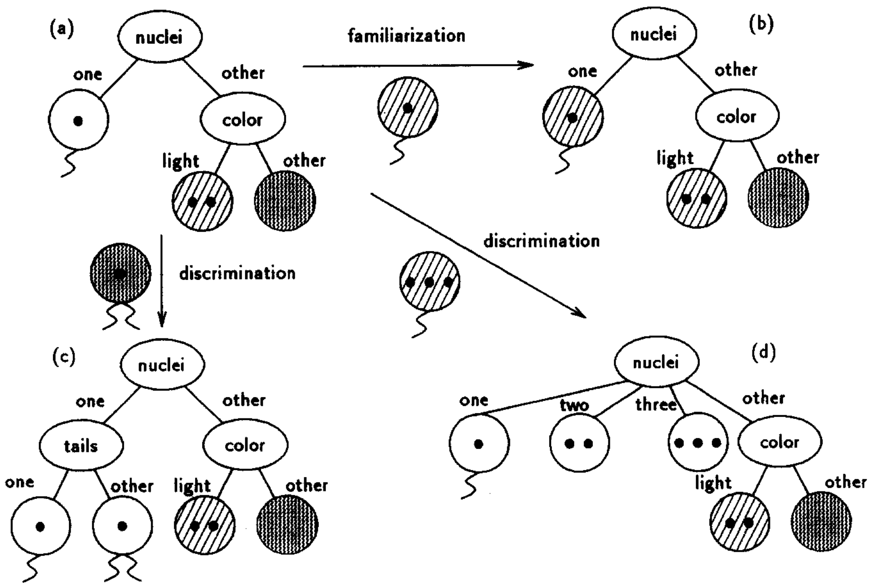


Fig. 1. Examples of EPAM's learning methods.

Table 1
The top-level EPAM algorithm

```
Input: The current node $N$ in the discrimination network.
       An unclassified (attribute-value) instance $I$.
Results: A discrimination net that classifies the instance.
Top-level call: EPAM(Top-node, $I$).
Variables: $N$ and $S$ are nodes in the hierarchy.
           $M$ is an image associated with a terminal node.
           $A$ is an attribute test.
           $V$ is the value of an attribute.
           $D$ is a set of attributes.

EPAM($N$, $I$)
  If $N$ is a terminal node,
    Then let $M$ be the image associated with $N$.
         Let $D$ be the set of tests on which $I$ and $M$ differ.
         If $D$ is the empty set,
           Then Familiarize($M$, $I$).
         Else Discriminate(Top-node, $I$, $M$, $D$, empty set).
  Else let $A$ be the test associated with $N$.
       Let $V$ be the value of instance $I$ on test $A$.
       If $N$ has a branch labeled $V$,
         Then let $S$ be the successor of $N$ by branch $V$.
       Else let $S$ be the successor of $N$ by branch OTHER.
       EPAM($S$, $I$)
```

terminal node. At each node, EPAM examines the instance's value on the test specified for that node. In the case of tests examining the category of a subobject, the model calls on itself recursively to determine the appropriate category; we have omitted this option from the table for the sake of clarity. If the category or attribute value equals that on one of the emanating branches, EPAM sends the instance down that branch; otherwise it goes down the OTHER branch. Eventually, the instance reaches a terminal node. For example, in Fig. 1(a) a dark cell with one nucleus and two tails would reach the leftmost terminal node, whereas a dark cell with two nuclei and two tails would reach the rightmost one.

Once EPAM has "recognized" an object as an instance of a terminal node, it "recalls" the image associated with that node. At this point, the algorithm invokes one of two learning mechanisms. If the image matches the instance (i.e., if no attribute-value pairs differ), then *familiarization* occurs. As summarized in Table 2, this process selects an attribute that occurs in the instance but not in the image, and then adds the attribute (along with the instance's value) to the image. In this way, EPAM gradually makes its images more specific as it encounters more instances. Eventually, a given image may become so detailed that it effectively becomes equivalent to a particular instance. Given the

Table 2
Familiarization and discrimination in EPAM

```
Variables: $I$ is an (attribute-value) instance.
           $N$ and $S$ are nodes in the hierarchy.
           $M$ is an image associated with a terminal node.
           $A$ is an attribute test.
           $U$ and $V$ are the values of attributes.
           $D$ and $L$ are sets of attributes.
           $T$ is a set of attribute-values $((A, V), \ldots)$.

Familiarize($M$, $I$)
  Let $L$ be those attributes in instance $I$ not in image $M$.
  Select an attribute $A$ from $L$.
  Let $V$ be the value of $A$ for $I$.
  Add the attribute-value pair $(A, V)$ to the image $M$.

Discriminate($N$, $I$, $M$, $D$, $T$)
  If $N$ is a terminal node,
    Then Deepen($N$, $I$, $M$, $D$, $T$).
  Else let $A$ be the attribute associated with node $N$.
      Let $U$ be the value of $A$ for instance $I$.
      Let $V$ be the value of $A$ for image $M$.
      If $U$ does not equal $V$,
        Then Add-branch($N$, $U$, Union($T$, $(A, U)$)).
             Add-branch($N$, $V$, Union($T$, $(A, V)$)).
      Else if $N$ has a branch labeled $V$,
             Then let $S$ be the successor of $N$ by branch $V$.
           Else let $S$ be the successor of $N$ by branch OTHER.
           Discriminate($S$, $I$, $M$, $D$, Union($T$, $(A, U)$)).

Deepen($N$, $I$, $M$, $D$, $T$)
  Select an attribute $A$ from $D$.
  Remove the image $M$ from node $N$.
  Associate the attribute $A$ with node $N$.
  Let $U$ be the value of $A$ for instance $I$.
  Let $V$ be the value of $A$ for image $M$.
  Add-branch($N$, $U$, Union($T$, $(A, U)$)).
  Add-branch($N$, OTHER, Union($M$, $(A, U)$)).

Add-branch($N$, $V$, $I$)
  Create a successor node of $N$ called $S$.
  Connect $N$ to $S$ with a branch having value $V$.
  Store the image $I$ on $S$.
```

network in Fig. 1(a) and the instance COLOR = DARK ∧ NUCLEI = ONE ∧ TAILS = ONE, familiarization would produce the network shown in Fig. 1(b).

If the image fails to match the instance (i.e., if any attribute-value pairs differ), then *discrimination* occurs instead. This process sorts the instance through the discrimination network a second time, looking for the first node at which the image and instance differ on a stored test. This can occur at a nonterminal node only if the instance was sorted down the OTHER branch leading from that node. If EPAM finds such a node, it creates two new branches, one based on the instance's value for the test and the other based on the image's value.[7] Each branch points to a new terminal node, and each image consists of the results of tests that lead to the node. In this way, EPAM gradually increases the breadth of its discrimination network. The transition between Fig. 1(a) and (d) gives an example of this type of discrimination, in this case invoked by the instance COLOR = LIGHT ∧ NUCLEI = THREE ∧ TAILS = ONE.

If no such node exists, the system eventually sorts the instance back down to the terminal node where the mismatch originally occurred. EPAM creates two new branches in this case as well, along with corresponding terminal nodes. The discrimination process selects a test on which the image and instance differ and which has not yet been examined. This test's value for the instance becomes the label on one branch and OTHER becomes the label for the other. The image for the instance-based node contains the results of all tests leading to that node; the image for the image-based node contains the original image plus the value for the discriminating test. In this way, EPAM gradually increases the depth of its discrimination network. The transition between Fig. 1(a) and (c) shows this type of learning in action, this time produced by the instance COLOR = DARK ∧ NUCLEI = ONE ∧ TAILS = TWO. Table 2 summarizes the overall discrimination process.

3.1.3. *Search control in EPAM*

In line with our discussion in Section 2, we can summarize EPAM's learning method in terms of search through a space of discrimination networks. Three basic operators make up this search:

- adding features to an image through familiarization;
- creating new disjunctive branches through discrimination;
- extending the network downward through discrimination.

Although the search-based view has its advantages, it provides little insight when one examines EPAM's control scheme. The classification method is completely deterministic, and the learning algorithm has only two choice points. One of these occurs during familiarization, when EPAM must decide which attribute to add to the image. The other occurs when discrimination must deepen the network to avoid a mismatch, when it must decide which attribute to select. One version of EPAM [9] preferred tests that had proven useful in previous discriminations. Other versions simply selected tests in a prespecified order. However, these decisions are minor in comparison to the choice between familiarization and discrimination, and between the branching

[7] The reason for this second branch is not clear, since the branch based on the instance's value is enough to avoid repeating the misclassification. However, we have attempted to faithfully reconstruct Feigenbaum's model as he describes it.

and deepening variants of discrimination. These are completely determined by the data and the existing network.

3.1.4. *Comments on* EPAM

The EPAM model introduced some very important ideas into the machine learning literature. First, it set forth the notion of a discrimination network, and it specified an incremental method that integrated classification and learning. Second, it introduced the distinction between tests (for use in sorting) and images (for use in making predictions). One can view discrimination networks as precursors of the concept hierarchies used in later work, and images as the precursors of concept descriptions. EPAM's distinction between the process of recognition (classification) and recall (prediction) was also an important insight. Finally, it introduced the two learning mechanisms of discrimination and familiarization, which it successfully used to explain aspects of human learning and memory.

Despite its successes, EPAM also had some significant shortcomings. For instance, the system's method for selecting among attributes during discrimination and familiarization was somewhat ad hoc. Moreover, the model retained concept descriptions (images) only at terminal nodes, and so lacked a true concept hierarchy. Finally, it assumed that concepts (images) were "all or none" entities, rather than the more fluid structures suggested by recent psychological studies (Rosch [37]). The last two criticisms are not really appropriate, since EPAM's goal was to model human memorization and not the broader area of concept formation. However, our concern here is with models of the latter process, and so we have evaluated Feigenbaum's work in those terms.

3.2. Lebowitz's UNIMEM

One can view Lebowitz's UNIMEM [24, 25] as a successor to EPAM,[8] since it shares many features with the earlier model, but also introduces some novel ideas. The motivation behind the two systems was also quite different. EPAM modeled empirical results from verbal learning experiments, whereas Lebowitz focused on the acquisition and use of concepts for more complex tasks such as natural language understanding and inference. In addition, UNIMEM was cast within a broader framework called *generalization-based memory*. Another system that independently incorporated many of the same advances as UNIMEM, is Kolodner's [19] CYRUS. We will highlight similarities and differences between these systems as they become relevant. Our stress on UNIMEM is due primarily to Lebowitz's [26] treatment of his system as conceptual clustering, a topic of primary interest for this paper.

[8] Actually, UNIMEM is a direct descendant of Lebowitz's [22, 23] IPP system. For a discussion of the differences between these two models, see Lebowitz [26].

3.2.1. *Representation and organization in UNIMEM*

UNIMEM represents instances in the same manner as EPAM—as a conjunction of features or attribute-value pairs. In one sense, it is less general than the earlier model, since it cannot handle objects with components, though Wasserman [43] has addressed this issue within the UNIMEM framewok. However, Lebowitz's system is more general than Feigenbaum's in that it can handle numeric attributes in addition to nominal (symbolic) ones. Thus, an instance that describes a university would have some nominal attributes (e.g., location, academic-emphasis) and some numeric attributes (e.g., male/female ratio, average SAT score). In addition, nominal attributes can take on more than one value, letting the system represent sets.

Lebowitz's approach diverges even more from Feigenbaum's in its representation and organization of concepts. In EPAM's network, only terminal nodes have associated images, but in UNIMEM both terminal and nonterminal nodes have concept descriptions. Each description consists of a conjunction of attribute-value pairs, with each value having an associated integer. This number measures what Lebowitz refers to as the *confidence* in that feature. Later, we will see that this corresponds to the idea of *predictability*, i.e., how well the feature can be predicted given an instance of the concept. In order to use consistent terminology, we refer to this count as the "predictability score" for a feature.[9]

Like its precursor, UNIMEM organizes knowledge into a concept hierarchy through which it sorts new instances. However, the details of this hierarchy differ from EPAM's discrimination network. We have mentioned that Lebowitz's system stores concept descriptions with each node in the hierarchy. Nodes high in the hierarchy represent general concepts, with their children representing more specific variants, their children still more specific concepts, and so on. Each concept has an associated set of instances stored with it; these can be viewed as terminal nodes in the hierarchy, though Lebowitz does not describe them in this fashion. Thus UNIMEM's terminal nodes are quite specific from the outset;[10] this contrasts with EPAM's images, which converge on completely specified instances only after considerable learning. Another difference is that, unlike EPAM, each instance may be stored with multiple nodes, so that categories need not be disjoint.

As in Feigenbaum's system, UNIMEM's network consists of nodes and links, with each of a node's links leading to a different child. However, in EPAM each link was labeled with the result of a single test. In contrast, UNIMEM allows

[9] Kolodner's [19] CYRUS uses a similar concept representation scheme, but maintains a probability rather than an integer with each attribute value. We argue in the context of our COBWEB discussion that this is an important distinction.

[10] Actually, the system stores only those features not inherited from nodes higher in the hierarchy, but the effect is the same as storing completely specified instances.

each link to specify the results of multiple tests (i.e., to specify multiple features). This redundant indexing lets the system handle instances with missing attributes and, as we describe below, it allows a very flexible sorting strategy. In addition, each feature on a link has an associated integer score, specifying the number of links on which that feature occurs. This second score measures the *predictiveness* of the feature, i.e., how well it can be used to predict instances of the various children.

Figure 2 presents a simple UNIMEM hierarchy after the system has created three concept nodes from six instances. For each node, we have shown its feature list and associated predictability scores. (For simplicity, we have omitted the predictiveness scores.) These scores represent the number of times a feature has been reinforced by successive instances. Note that one instance is indexed into both top-level nodes. This instance affects the predictability scores for both level-one nodes, although it is only incorporated into one of them.

3.2.2. *Classification and learning in UNIMEM*

Like other concept formation systems, UNIMEM integrates the processes of classification and learning. It sorts each instance through its concept hierarchy, modifying this hierarchy in the process. Table 3 summarizes the main steps in the algorithm.

As UNIMEM descends through its hierarchy, it uses the features (i.e., the attribute-value pairs) on each node and its emanating links to sort the instance. If the instance matches the description on the node closely enough, then it sends the instance down those links that contain features in the instance, and it continues the process with the relevant children. Both the number of features

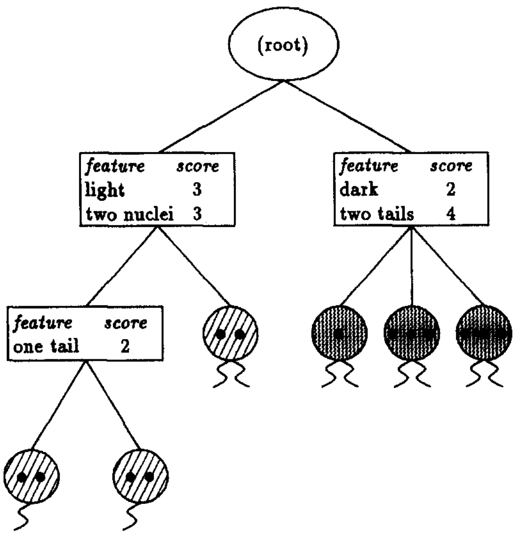


Fig. 2. A sample UNIMEM hierarchy.

Table 3
The basic UNIMEM algorithm

```
Input: The current node $N$ of the concept hierarchy.
       The name of an unclassified instance $I$.
       The set of $I$'s unaccounted features $F$.
Results: The concept hierarchy that classifies the instance.
Top-level call: Unimem(Top-node, $I$, $F$).
Variables: $N$ and $C$ are nodes in the hierarchy.
           $G$, $H$, and $K$ are sets of features (attribute values).
           $J$ is an instance stored on a node.
           $S$ is a list of nodes.

Unimem($N$, $I$, $F$).
  Let $G$ be the set of features stored on $N$.
  Let $H$ be the features in $F$ that match features in $G$.
  Let $K$ be the features in $F$ that do not match features in $G$.
  If $N$ is not the root node,
    Then If Evaluate($N$, $H$, $K$) returns TRUE or there are too few features in $H$,
           Then return the empty list.
  Let $S$ be the empty list.
  For each child $C$ of node $N$,
      If $C$ is indexed by a feature in $K$,
         Then let $S$ be Union($S$, Unimem($C$, $I$, $K$)).
  If $S$ is the empty list,
    Then for each instance $J$ of node $N$,
           Let $S$ be Union($S$, Generalize($N$, $J$, $I$, $F$)).
  If $S$ is the empty list,
    Then store $I$ as an instance of node $N$ with features $K$,
           For each feature $J$ in $F$ serving as in index to $N$,
               Increment the predictiveness score $R$ of $J$ by 1.
               If $R$ is high enough,
                  Then remove $J$ as an index leading to $N$.
  Return $N$.
```

necessary for this match and the closeness of each value (for numeric attributes) are system parameters.[11] Whether or not the instance successfully matches, UNIMEM calls on EVALUATE (which we discuss in Section 3.2.3) to modify the node's scores. Note that, in some cases, the system may sort an instance down multiple paths in the hierarchy.

Eventually UNIMEM reaches a node that matches the instance but none of whose children match. In this case, the system examines all instances currently stored with the node, comparing each of them in turn to the new instance. If an old instance shares enough features with a new one (another system parameter), the model creates a new, more general node based on these features and stores both instances as its children. When this occurs, the system increments the predictiveness count for each feature indexing the new node.

[11] UNIMEM uses a distance metric to determine the degree of match between two numeric values. This is an important issue, to which we will return in Section 4.

Table 4
UNIMEM's update and evaluation processes

```
Variables: $N$ and $C$ are nodes in the hierarchy.
           $F$, $G$, $H$, and $K$ are sets of features (attribute values).
           $I$ and $J$ are the names of instances.
           $S$ and $T$ are predictability scores of nodes' features.
           $R$ is the predictiveness score of a node's feature.

Generalize($N, J, I, F$)
  Let $G$ be the features in instance $J$.
  Let $H$ be the features in $F$ that match features in $G$.
  If $H$ contains enough features,
    Then create a new child $C$ of node $N$.
         Index and describe $C$ by the features in $H$.
         For each feature $K$ serving as index to $C$,
             Increment the predictiveness score $R$ of $K$ by 1.
             If $R$ is high enough,
               Then remove $K$ as an index.
         Remove $J$ as an instance of $N$.
         Let $G'$ be the features in $G$ that are not in $H$.
         Let $F'$ be the features in $F$ that are not in $H$.
         Store $J$ as an instance of $C$ with features $G'$.
         Store $I$ as an instance of $C$ with features $F'$.
         Return $C$.

Evaluate($N, H, K$)
  For each nonpermanent feature $F$ in $H$,
      Raise the predictability score $S$ for $F$ on $N$.
      If $S$ is high enough,
        Then make $F$ a permanent feature of $N$.
  For each nonpermanent feature $G$ in $K$,
      Lower the predictability score $T$ for $G$ on $N$.
      If $T$ is low enough,
        Then remove the feature $G$ from $N$.
             If $N$ has too few features,
               Then remove $N$ from its parent's list of children.
                    Remove all indices serving $N$.
                    Return TRUE
  Return FALSE
```

Since UNIMEM compares the new instance to each of the stored instances, it can form multiple nodes in this manner. Table 4 summarizes the steps in this GENERALIZE process.[12] If none of the existing instances are similar enough to the new one, the system simply stores it with the current node, effectively creating a new disjunct.

[12] Our description of the UNIMEM algorithm (Tables 3 and 4) differs syntactically from that given by Lebowitz [24, 26]. Our somewhat different view of his algorithm produced a different organization to the specification. We believe that our description is clearer and functionally equivalent to Lebowitz's.

Note that when UNIMEM places an instance into more than one category, these categories overlap: they do not form disjoint partitions over the instances. In the literature on cluster analysis (Everitt [8]), this approach has been called *clumping*. Lebowitz [26] has argued that in some domains, overlapping concepts may describe the data more accurately than disjoint partitions. In addition, clumping introduces flexibility into the search for useful categories. UNIMEM may initially decide to retain multiple categories and later decide to remove one or more of them. This gives the effect of a beam search while still working within the hill-climbing metaphor described in Section 2. The clumping strategy and its associated advantages are shared by CYRUS.

3.2.3. *Evaluation and pruning in UNIMEM*

We have noted that UNIMEM retains two counts on nodes' features. The EVALUATE procedure shown in Table 4 updates these scores each time the system attempts to match an instance to a node's description. If a given feature in the instance matches a feature on the node, UNIMEM increments the predictability score for that feature. The increment for nominal attributes is one; the increment for numeric attributes is a function of the distance between the stored and observed values. If a given instance feature fails to match a node feature, the system decrements that feature's predictability score in a similar fashion.

When the predictability score for a feature exceeds a (user-specified) threshold, UNIMEM permanently fixes that feature as part of the node's description, so that future instances no longer affect it. More important, when a feature's score drops below another (user-specified) threshold, the system removes that feature from the concept description. In this way, an initially specific concept may gradually become more and more general. However, it may also become so general that it has little usefulness in making predictions. Thus, when the number of features stored on a node becomes low enough (another parameter), UNIMEM removes the node from memory along with all links to its children.

When the predictiveness score for a node's feature becomes too high (i.e., when the feature indexes too many children), UNIMEM removes that feature from links emanating from the node. In this way, concepts that were originally retrieved often may become accessed more selectively. However, if the system removes all indices to a child, that node is effectively forgotten, since there is no longer any way to sort instances to it. This is another way in which UNIMEM prunes its concept hierarchy.

3.2.4. *Comments on UNIMEM*

To summarize, UNIMEM can be viewed as carrying out a hill-climbing search through a space of concept hierarchies. This search process involves six basic operators:

- storing a new instance with a node (creating a new disjunct);
- creating a more general node based on the features shared by two instances;
- permanently fixing a feature in a node's description;
- deleting an unreliable feature from a node's description;
- deleting an overly general node (and its children);
- deleting a nonpredictive index to a node's children.

Lebowitz's approach to concept formation introduces a number of advances over EPAM. Each node in the UNIMEM hierarchy has an associated concept description, rather than just the terminal nodes. Moreover, each feature in these descriptions has associated weights; thus concepts are less "all or none." There is a clear evaluation of concepts and their components, and the notions of predictiveness and predictability further clarify the distinction between recognition (classification) and recall (prediction). The system also introduced the possibility of multiple indices to a given concept, and provided one method for constructing nondisjoint hierarchies. Each of these general advances is also true of CYRUS, although their realization differs in some important respects from UNIMEM.

However, UNIMEM also has significant drawbacks as a model of concept formation. The measures of predictiveness and predictability are informal and have no clear semantics. The system also lacks a principled method for deciding between learning operators, being dependent on user-specified parameters to make such decisions. Lebowitz [26] has carried out initial studies on how these parameters affect the system's behavior, but much work remains before their full impact becomes clear.

3.3. Fisher's COBWEB

Fisher's [12, 13] COBWEB constitutes another algorithm for incremental concept formation. As we will see below, this research builds heavily on Lebowitz's earlier approach, and it also borrows from Kolodner's [19] work on CYRUS. Although Fisher does not present COBWEB itself as a psychological model, it has been heavily influenced by research in cognitive psychology on basic-level and typicality effects (Rosch [37]). Briefly, experiments with humans suggest that some categories are more "basic" than others, being retrieved more rapidly and named more frequently. In addition, there is evidence that for a given category, some members are more "typical" than others, being retrieved more quickly and rated as better examples. Fisher [13] describes COBWEB/2, a related system that models these effects, but we will focus on the simpler COBWEB instead.

3.3.1. *Representation and organization in COBWEB*

Like its predecessors, Fisher's system represents each instance as a set of attribute-value pairs. The mapping is closest to EPAM, since each attribute takes on only one value and since only nominal attributes are allowed.[13] As in UNIMEM, each concept node is described in terms of attributes, values, and associated weights, but here the similarity ends. One difference is that COBWEB stores the probability of each concept's occurrence. Another is that each node, from the most specific to the most general, includes every attribute observed in the instances. Moreover, associated with each attribute is every possible value for that attribute. Each such value has two associated numbers, which roughly correspond to Lebowitz's predictiveness and predictability scores. However, in COBWEB these scores have a formal grounding in probability theory.

Fisher defines the *predictiveness* of a value v for category c as the conditional probability that an instance i will be a member of c, given that i has value v, or $P(c|a = v)$. Similarly, he defines the *predictability* of a value v for category c as the conditional probability that an instance i will have value v, given that i is a member of c, or $P(a = v|c)$. Actually, COBWEB does not explicitly store predictiveness scores, since it can derive them from predictability and node probability using Bayes' rule. Smith and Medin [42] have used the term *probabilistic concepts* to refer to concept representations that incorporate such conditional probabilities.

Figure 3 presents a sample concept hierarchy, including the probabilities associated with each concept and with its attribute values. For instance, the top node (N_1) has an associated probability of 1.0. It also states that its members have an equal chance of having one or two tails and an even chance of being light or dark. Concept N_3 has a 50% chance of occurring, and its members so far have always had one tail and two nuclei, but have been evenly split among light and dark colors. The terminal nodes in the hierarchy—N_2, N_4, N_5, and N_6—have less interesting probabilistic descriptions, since each is based on a single instance. However, note that the probability of each node's occurrence is specified relative to its parent, rather than with respect to the entire distribution.

COBWEB's concept hierarchy is similar to UNIMEM's in that each node has an associated "image," with more general nodes higher in the hierarchy and more specific ones below their parents. However, the system's terminal nodes are always specific instances that it has encountered; unlike UNIMEM, it never deletes instances. In addition, the hierarchy divides instances into disjoint classes. More important, COBWEB links parents to their children only through *is-a* links. The system differs from both EPAM and UNIMEM in that it avoids explicit indices stated as tests on attribute values. Thus, the sample hierarchy shown in Fig. 3 has a different semantics than those we have seen earlier. This assumption leads to a novel method for sorting instances through the concept hierarchy.

[13] In Section 4, we will see how COBWEB can be extended to handle both numeric attributes and instances involving multiple components.

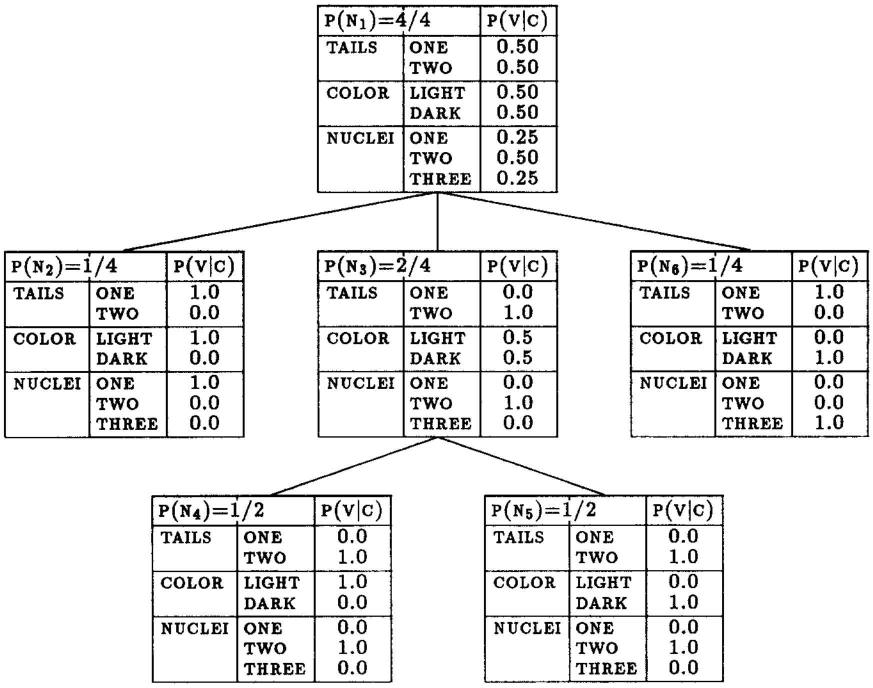


Fig. 3. A sample COBWEB hierarchy with nodes numbered in order of creation.

3.3.2. *Classification and learning in COBWEB*

The basic COBWEB algorithm is quite simple, as can be seen from the summaries in Tables 5 and 6. Again classification and learning are intertwined, with each instance being sorted down through a concept hierarchy and altering that hierarchy in its passage. The system initializes its hierarchy to a single node, basing the values of this concept's attributes on the first instance. Upon encountering a second instance, COBWEB averages its values into those of the concept and creates two children, one based on the first instance and another based on the second.

Unlike EPAM and UNIMEM, Fisher's model does not use explicit tests or indices to retrieve potential categories. Instead, at each node COBWEB retrieves all children and considers placing the instance in each of these categories. Each of these constitutes an alternative *clustering* (a set of clusters with a common parent) that incorporates the new instance. Using an evaluation function that we describe in Section 3.3.3, it then selects the best such clustering. COBWEB also considers creating a new category that contains only the new instance, and compares this clustering to the best clustering that uses only existing categories.

If the clustering based on existing classes wins the competition, COBWEB modifies the probability of the selected category and the conditional prob-

Table 5
The COBWEB algorithm

```
Input: The current node $N$ of the concept hierarchy.
       An unclassified (attribute-value) instance $I$.
Results: A concept hierarchy that classifies the instance.
Top-level call: Cobweb(Top-node, $I$).
Variables: $C$, $P$, $Q$, and $R$ are nodes in the hierarchy.
           $U$, $V$, $W$, and $X$ are clustering (partition) scores.

Cobweb($N$, $I$)
  If $N$ is a terminal node,
    Then Create-new-terminals($N$, $I$)
         Incorporate ($N$, $I$).
  Else Incorporate($N$, $I$).
      For each child $C$ of node $N$,
          Compute the score for placing $I$ in $C$.
      Let $P$ be the node with the highest score $W$.
      Let $R$ be the node with the second highest score.
      Let $X$ be the score for placing $I$ in a new node $Q$.
      Let $Y$ be the score for merging $P$ and $R$ into one node.
      Let $Z$ be the score for splitting $P$ into its children.
      If $W$ is the best score,
        Then Cobweb($P$, $I$) (place $I$ in category $P$).
      Else if $X$ is the best score,
          Then initialize $Q$'s probabilities using $I$'s values
               (place $I$ by itself in the new category $Q$).
      Else if $Y$ is the best score,
          Then let $O$ be Merge($P$, $R$, $N$).
               Cobweb($O$, $I$).
      Else if $Z$ is the best score,
          Then Split($P$, $N$).
               Cobweb($N$, $I$).
```

abilities for its attribute values. Thus, predictability scores for values occurring in the instance will increase, whereas those for values not occurring will decrease. Predictiveness scores change as well, but since the system does not actually store these, it does not update them explicitly. In addition, COBWEB continues to sort the instance down through the hierarchy, recursively considering the children of the selected category. Node N_3 in Fig. 3 shows the result of incorporating a new instance into an existing node. At an earlier stage, this had been a terminal node based on a single instance. However, the act of hosting a new instance has left its COLOR probabilities evenly divided and given it two children.

If the clustering with the singleton class emerges as the winner, COBWEB creates this new category and makes it a child of the current parent node. The system bases the values for this new concept's attributes on those found in the instance, giving them each predictability scores of one. In this case, classification halts at this step, since the new concept is a terminal node. Node N_6 in

Table 6
Auxiliary COBWEB operations

```
Variables: $N$, $O$, $P$, and $R$ are nodes in the hierarchy.
           $I$ is an unclassified instance
           $A$ is a nominal attribute.
           $V$ is a value of an attribute.

Incorporate($N$, $I$)
  Update the probability of category $N$.
  For each attribute $A$ in instance $I$,
      For each value $V$ of $A$,
          Update the probability of $V$ given category $N$.

Create-new-terminals($N$, $I$)
  Create a new child $M$ of node $N$.
  Initialize $M$'s probabilities to those for $N$.
  Create a new child $O$ of node $N$.
  Initialize $O$'s probabilities using $I$'s values.

Merge($P$, $R$, $N$)
  Make $O$ a new child of $N$.
  Set $O$'s probabilities to be $P$ and $R$'s average.
  Remove $P$ and $R$ as children of node $N$.
  Add $P$ and $R$ as children of node $O$.
  Return $O$.

Split($P$, $N$)
  Remove the child $P$ of node $N$.
  Promote the children of $P$ to be children of $N$.
```

Fig. 3 was created in this fashion, since the instance it summarizes was sufficiently different from node N_2 and N_3.

Although in principle the above method provides everything needed to construct hierarchies of probabilistic concepts, it can be sensitive to the order of instance presentation, creating different hierarchies from different orders of the same data. In particular, if the initial instances are nonrepresentative of the entire population, one may get hierarchies with poor predictive ability. For example, if the first instances are all conservative congressmen, the algorithm would create subcategories of these at the top level. When it finally encountered instances of liberal congressmen, it would create one category for them at the top level. However, it would still have all the conservative instances at this same level, when one would prefer them grouped under a separate category.

COBWEB includes two additional operators to help it recover from such nonoptimal hierarchies. At each level of the classification process, the system considers *merging* the two[14] nodes that best classify the new instance. If the

[14] Although one could consider merging all possible node pairs, such a strategy would be costly and unlikely to improve the resulting hierarchy.

resulting clustering is better (according to the function described in Section 3.3.3) than the original, it combines the two nodes into a single category, though still retaining the original nodes as its children. This transforms a clustering of N nodes into one having $N - 1$ nodes, as in the transition shown by Fig. 4.

The system also incorporates the inverse operation of *splitting* nodes. At each level, if COBWEB decides to classify an instance as a member of an existing category, it also considers removing this category and elevating its children. If this action leads to an improved clustering, the system changes the structure of its hierarchy accordingly. Thus, if one of N nodes at a given level has M children, splitting this node would give $N + M - 1$ nodes at this level, as depicted by the transition in Fig. 5.

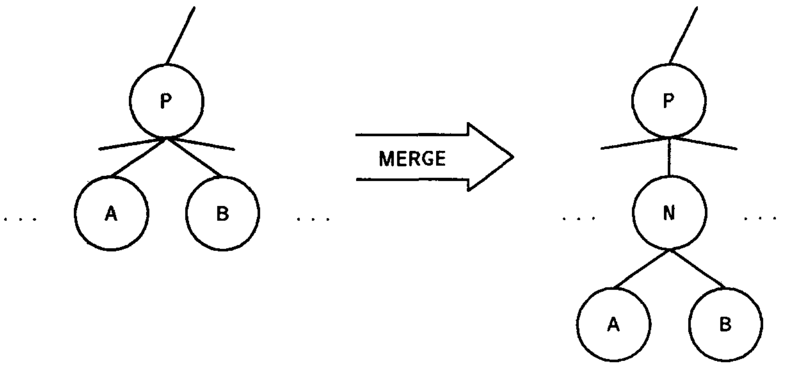


Fig. 4. Merging categories in COBWEB.

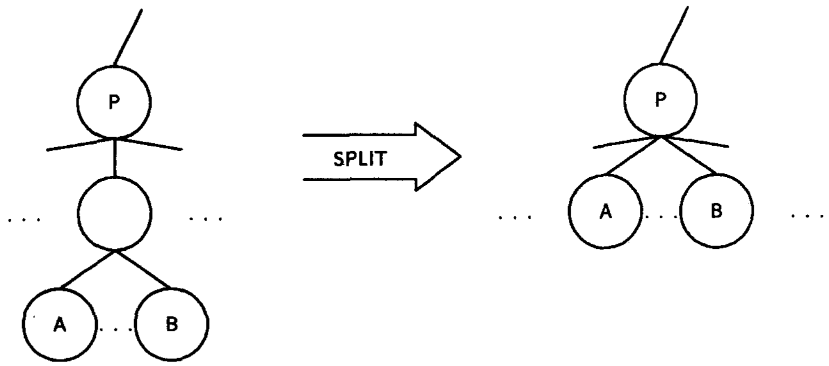


Fig. 5. Splitting categories in COBWEB.

3.3.3. *Evaluation in* COBWEB

We have made numerous references to COBWEB's evaluation function, but we have yet to define this metric. We have also mentioned Fisher's concern with the basic-level phenomena, but we have yet to show how the system has been influenced by these phenomena. The key to both issues involves *category utility*, a measure that Gluck and Corter [14] have shown predicts the basic level found in psychological experiments. They derive this function by two paths, one using information theory and the other using game theory.

COBWEB uses a slightly generalized version of Gluck and Corter's function to control its classification and learning behavior. Category utility favors clusterings that maximize the potential for inferring information (Fisher [13]). In doing this, it attempts to maximize intra-class similarity and inter-class differences, and it also provides a principled tradeoff between predictiveness and predictability. The basic measure assumes that concept descriptions are probabilistic in nature. We do not have space to rederive this metric, but we can consider some of its characteristics.

For any set of instances, any attribute-value pair, $A_i = V_{ij}$, and any class, C_k, one can compute $P(A_i = V_{ij} | C_k)$, the conditional probability of the value given membership in the class, or its predictability. One can also compute $P(C_k | A_i = V_{ij})$, the conditional probability of membership in the class given this value, or its predictiveness. One can combine these measures of individual attributes and values into an overall measure of clustering quality. Specifically,

$$\sum_k \sum_i \sum_j P(A_i = V_{ij}) P(C_k | A_i = V_{ij}) P(A_i = V_{ij} | C_k) \tag{1}$$

represents a tradeoff between predictability $P(A_i = V_{ij} | C_k)$ and predictiveness $P(C_k | A_i = V_{ij})$ that has been summed across all classes (k), attributes (i), and values (j). The probability $P(A_i = V_{ij})$ weights the individual values, so that frequently occurring values play a more important role than those occurring less frequently.

Using Bayes' rule, we have $P(A_i = V_{ij}) P(C_k | A_i = V_{ij}) = P(C_k) P(A_i = V_{ij} | C_k)$, letting us transform expression (1) into the alternative form

$$\sum_k P(C_k) \sum_i \sum_j P(A_i = V_{ij} | C_k)^2 . \tag{2}$$

Gluck and Corter have shown that the subexpression $\Sigma_i \Sigma_j P(A_i = V_{ij} | C_k)^2$ is the *expected* number of attribute values that one can correctly guess for an arbitrary member of class C_k. This expectation assumes a *probability matching* strategy, in which one guesses an attribute value with a probability equal to its probability of occurring. Thus, it assumes that one guesses a value with probability $P(A_i = V_{ij} | C_k)$ and that this guess is correct with the same probability.

Gluck and Corter build on expression (2) in their derivation. They define category utility as the *increase* in the expected number of attribute values that can be correctly guessed, given a set of n categories, over the expected number of correct guesses without such knowledge. The latter term is simply $(\Sigma_i \Sigma_j P(A_i = V_{ij})^2)$, so one must subtract this from expression (2). The complete expression for category utility is thus

$$\frac{\sum_{k=1}^{K} P(C_k) \sum_i \sum_j P(A_i = V_{ij} | C_k)^2 - \sum_i \sum_j P(A_i = V_{ij})^2}{K} . \tag{3}$$

Note that the difference between the two expected numbers is divided by K, the number of categories. This division lets one compare different size clusterings, which must occur whenever one considers merging, splitting, or creating a new category.

Since category utility is based on expected numbers of correct guesses about attribute values, it suggests predictive ability as the natural measure of behavior. Fisher has tested COBWEB on both natural and artificial domains, measuring its performance by asking it to predict missing attribute values on test instances. This approach is similar to Quinlan's [34] methodology for evaluating supervised learning systems, except that one averages across many attributes rather than predicting a single one (the class name). In Section 4, we will extend this notion of prediction (and category utility) to domains involving numeric attributes.

COBWEB is not the first inductive learning system that has employed an evaluation function based on information theory. The best-known work of this type is Quinlan's [34] ID3 method for constructing decision trees. Machine learning researchers have explored many extensions and variations of the basic technique, including incremental versions (Schlimmer and Fisher [38]). Rendell et al.'s [35] PLS system also uses an information-theoretic metric to direct its divisive construction of disjunctive concept descriptions. In addition, Hanson and Bauer [16] have used an information-based function in their WITT clustering system, Cheeseman et al. [6] have used a Bayesian approach in their nonincremental clustering system AUTOCLASS, and Anderson (personal communication) has used conditional probabilities in his recent work on incremental clustering.

3.3.4. *Comments on COBWEB*

Like its predecessors, one can view COBWEB as carrying out a hill-climbing search through a space of concept hierarchies. In this case, there are four main operators:

- classifying the object into an existing class;
- creating a new class (a new disjunct);

- combining two classes into a single class (merging);
- dividing a class into several classes (splitting).

The system employs an evaluation function—category utility—to determine which operator (and which instantiation) to employ at each point in the classification process.

The use of a well-defined evaluation function constitutes an advance over previous work on concept formation, as does Fisher's reformulation of predictiveness and predictability in terms of conditional probabilities. The explicit inclusion of merging and splitting also seems desirable, since they should let COBWEB recover from nonrepresentative samples without losing its incremental, memory-limited flavor.

However, Fisher's work also has some limitations. As implemented, COBWEB can handle only nominal attributes, whereas UNIMEM dealt with both symbolic and numeric data. The system also assumes that each instance consists of a single "object," and thus avoids issues of finding mappings between analogous components. Finally, COBWEB retains all instances ever encountered as terminal nodes in its concept hierarchy. Although this approach works well in noise-free, symbolic domains, it can lead to "overfitting the data" in noisy or numeric domains. In these cases, some form of pruning or cutoff seems in order. These and other concerns led us to carry out the research described in the following section.

4. Modeling the Formation of Object Concepts

With these systems as background, we can turn to CLASSIT, a model of concept formation that attempts to improve upon earlier work. This system has been most strongly influenced by COBWEB, differing mainly in its representation of instances, its representation of concepts, and its evaluation function. However, CLASSIT uses the same basic operators and the same control strategy that Fisher's system employs. Below we describe the new model, stressing its differences from earlier systems, and explaining our motivations for introducing these differences.

4.1. Representation and organization in CLASSIT

Although symbolic or nominal attributes occupy an important role in natural language, they are less useful for describing the physical world. When describing a stick in English, one might say that stick is short or long, but our perceptual system can also distinguish two sticks that differ only slightly in length. This latter capability suggests that humans' representation of real-world objects can include detailed information about the quantitative features of those objects. A variety of real-world attributes can be described using real numbers, including features such as color, which are usually treated symbolically. Since we are concerned with the formation of physical object concepts, CLASSIT currently only accepts real-valued attributes as input.[15] In Section 6, we will discuss combining real-valued and symbolic attributes.

Physical objects can be represented with numeric attributes by describing each object as a set of components, each with a list of attributes such as height and width. Although this approach represents some relation information implicitly (such as the adjacency of components), it does not restrict the types of objects that can be described. Furthermore, this form of numeric representation seems a more plausible output from a perception system.

The introduction of real-valued data requires an analogous extension in one's representation of concepts. There are two obvious approaches to this problem. First, one can divide each numeric attribute into ranges; by "discretizing" the continuous values, one can retain the symbolic concept representation used in COBWEB. Lebowitz [24] has taken this approach in one version of UNIMEM. Alternatively, one can represent concepts directly in terms of real-valued attributes.

CLASSIT takes the second approach, retaining COBWEB's notion of storing a probability distribution with each attribute occurring in a concept. However, instead of storing a probability for each attribute value (e.g., for a given concept C, $P(small|C) = 0.3$; $P(large|C) = 0.7$), our model stores a continuous normal distribution (bell-shaped curve) for each attribute. CLASSIT expresses each distribution in terms of a mean (average) value and a standard deviation.[16] For instance, it might believe that the average length of a dog's tail is 1.1 feet and that its standard deviation is 0.65 feet. Attributes with low standard deviations have narrow, tall distributions, whereas those with high standard deviations have wide, shallow distributions.

CLASSIT organizes concepts into a hierarchy in the same manner as do UNIMEM and COBWEB. General concepts representing many instances are near the top of the tree, with more specific concepts below them. In general, concepts lower in the hierarchy will have attributes with lower standard deviations, since they represent more specific classes with greater within-group regularity.

4.2. Classification and learning in CLASSIT

This new representation scheme requires no modification to COBWEB's learning operators or basic control structure. Thus, CLASSIT includes the same four

[15] Statisticians have developed methods for clustering objects described in terms of real-valued attributes; these are known as *cluster analysis* and *numerical taxonomy* (Everitt [8]). Unfortunately, these methods are usually nonincremental.

[16] Standard deviation is defined as the square root of $\Sigma_{i=1}^{N} (x_i - \bar{x})^2/N$. Note that this equation as written cannot be computed incrementally; all x_i values need to be present. However, one can transform this expression for incremental computation by expanding the squared term and storing the sum of squares. Specifically, each concept contains a count, a sum of values, and a sum of squares. From these, we compute the mean and the standard deviation when needed.

basic operators as its predecessor—one for incorporating an instance into a existing concept, another for creating a new disjunctive concept, a third operator for merging two classes, and a final one for splitting classes. As described in Tables 5 and 6, for every new instance, the algorithm considers all four operators, computes the score of the evaluation function in each case, and selects that choice with the highest score. In Section 4.4, we will step through a detailed example of this procedure.

However, CLASSIT makes a few important additions to the basic algorithm. For example, rather than always descending to the leaves of the hierarchy as it classifies an instance, our system may decide to halt at some higher-level node. When this occurs, the system has decided that the instance is similar enough to an existing concept that further descent is unnecessary and that it should throw away specific information about that instance. We define "similar enough" with a system parameter, *cutoff*, that is based on our evaluation function.

There are two advantages of this modification. First, Quinlan [34] has shown that methods for building exhaustive decision trees tend to "overfit" the data in noisy domains, leading to decreased performance. The same effect should occur with concept formation systems, unless they employ some form of cutoff. Second, a system that retains every instance builds too large a data structure for real applications. Forgetting certain instances should lead to both better performance and to greater efficiency.

The representation of objects that CLASSIT uses requires another addition to the COBWEB algorithm. If instances are described as a set of components, how can the system correctly match instance components to concept components? For example, how can it know that the right front leg in the instance corresponds to the right front leg in the "dog" concept? In general terms, this problem is that of finding an optimal match in a weighted bipartite graph.

The brute force solution to this problem is far too expensive for practical use: to calculate the worth of every possible correspondence for n components has an $O(n!)$ time cost. Instead we have used a cheaper $O(n^2)$ time complexity heuristic algorithm. Using the variances for each attribute in the concept description, CLASSIT finds a match for that component with the least associated variation. Using this as a constraint, the system then finds a match for the next most constrained component and so forth, continuing this process until all components in the concept description have been matched against components in the instance. This "greedy" approach is not assured of finding the best match, but it is likely to find an acceptable one with minimal cost.[17]

We have chosen to retain COBWEB's learning operators because we believe they provide a good framework for concept formation. The hill-climbing search organization provides a robust method for learning while making minimal demands on memory. Rather than formulating new algorithms, our goal has

[17] There also exists an $O(n^3)$ guaranteed algorithm for this problem, which we will describe in Section 6.

been to extend the existing program to work in new domains and with a more general representational scheme.

4.3. CLASSIT's evaluation function

CLASSIT's use of real-valued attributes in both instances and concepts requires a generalization of category utility, COBWEB's evaluation function. In particular, the two innermost summations in category utility (equation (3)) need to be generalized for real-valued attributes:

$$\sum_{j}^{values} P(A_i = V_{ij} | C_k)^2 \quad \text{and} \quad \sum_{j}^{values} P(A_i = V_{ij})^2 \,.$$

Both of these terms are a sum of squares of the probabilities of all values of an attribute. The former uses probabilities given membership in a particular class, C_k, while the latter is without any class information. The second term is equivalent to the probability at the parent, since that node includes all instances for the clustering and therefore has no information about class membership.

In order for these terms to be applied to a continuous domain, summation must be changed to integration, and some assumption must be made about the distribution of values. Without any prior knowledge about the distribution of an attribute, the best assumption is that the distribution of values for each attribute follows a normal curve. Thus, the probability of a particular attribute value is the height of the curve at that value and the summation of the square of all probabilities becomes the integral of the normal distribution squared. For the first summation, the distribution is for a particular class, while the second must use the distribution at the parent. In either case, the integral evaluates to a simple expression:

$$\sum_{j}^{values} P(A_i = V_{ij})^2 \Leftrightarrow \int \frac{1}{\sigma^2 2\pi} \exp\left(\frac{x-\mu}{\sigma}\right)^2 \mathrm{d}x = \frac{1}{\sigma} \frac{1}{2\sqrt{\pi}} \,,$$

where μ is the mean and σ is the standard deviation. Finally, since the expression is used for comparison only (see the COBWEB algorithm), the constant term $1/2\sqrt{\pi}$ can be discarded.

In summary, one can replace the innermost summations from category utility with the term $1/\sigma$. The revised evaluation function used by CLASSIT is:

$$\frac{\sum_{k}^{K} P(C_k) \sum_{i}^{I} 1/\sigma_{ik} - \sum_{i}^{I} 1/\sigma_{ip}}{K} \,,$$

where I is the number of attributes, K is the number of classes in the partition, σ_{ik} is the standard deviation for a given attribute in a given class, and σ_{ip} is the standard deviation for a given attribute in the parent node.[18]

[18] In our implementation, the attribute summations are divided by I. This is necessary because CLASSIT allows instances to have some missing attributes.

This evaluation function is equivalent to the function used by COBWEB; it is a transformation of category utility. Unfortunately, this transformation introduces a problem when the standard deviation is zero for a concept. For any concept based on a single instance, the value of $1/\sigma$ is therefore infinite.

In order to resolve this problem, we have introduced the notion of *acuity*, a system parameter that specifies the minimum value for σ. This limit corresponds to the notion of a "just noticeable difference" in psychophysics—the lower limit on our perception ability. Because acuity strongly affects the score of new disjuncts, it indirectly controls the breadth, or branching factor of the concept hierarchy produced, just as the cutoff parameter controls the depth of the hierarchy.

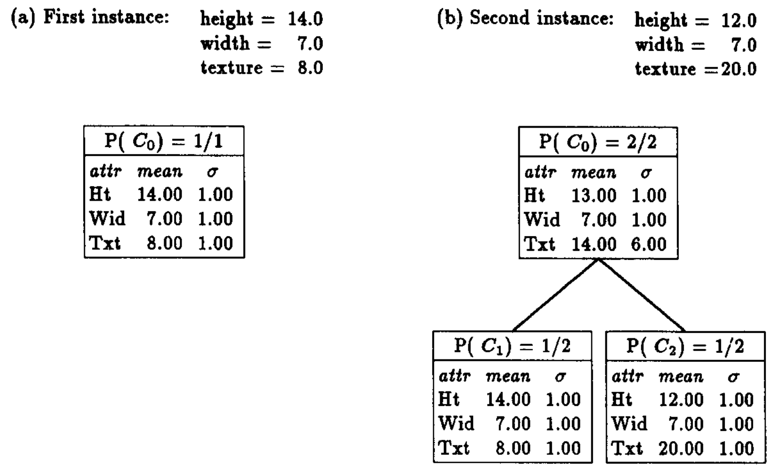
4.4. A detailed example

Now that we have examined CLASSIT's representation, control structure, and evaluation function, we will demonstrate the system's behavior in more detail by stepping through a sample execution. For this example, we have constructed a very simple input domain. Imagine a set of rectangles that naturally divides into three classes: small, medium, and large. Each instance has only one component and is described with only three attributes; height, width, and a texture attribute. For this domain, the texture attribute is irrelevant to classification. Small rectangles have a mean height of 12.5 and width of 6.5; medium rectangles average 30 by 14 and large rectangles average 41 by 35. The texture attribute is allowed to vary over the range 5 to 40, independent of class. Note that the system is not given any class information—it is not told whether a given instance is small, medium, or large. Instead, these concepts must be induced from regularity in the data. This is precisely the task of unsupervised concept formation.

We will now step through an execution as CLASSIT encounters the first six rectangles. The system begins with an empty concept hierarchy. Suppose the first instance is a small rectangle with values of 14 for height, 7 for width and 8 for texture. This instance is used to create the root node of the hierarchy, as shown in Fig. 6(a). Since this initial concept is based on a single instance, it has the minimum value for its σ values. For this execution the acuity parameter specifies this minimum to be 1.0 for all attributes.

For each concept created by the system, we have shown the mean and standard deviation (σ) for all attributes, as well as $P(C_k)$, the probability of that concept within the clustering. As noted earlier, concepts store cumulative sums and sum of squares in order to recompute the standard deviation incrementally. Similarly, $P(C_k)$ is computed on demand by using counts stored at each concept. In order to make clear the semantics of our concepts, we have not shown these computational values in our figures.

Figure 6(b) shows the entire concept hierarchy after the system classifies the


(a) First instance: height = 14.0, width = 7.0, texture = 8.0

P(C_0) = 1/1		
attr	*mean*	σ
Ht	14.00	1.00
Wid	7.00	1.00
Txt	8.00	1.00

(b) Second instance: height = 12.0, width = 7.0, texture = 20.0

P(C_0) = 2/2		
attr	*mean*	σ
Ht	13.00	1.00
Wid	7.00	1.00
Txt	14.00	6.00

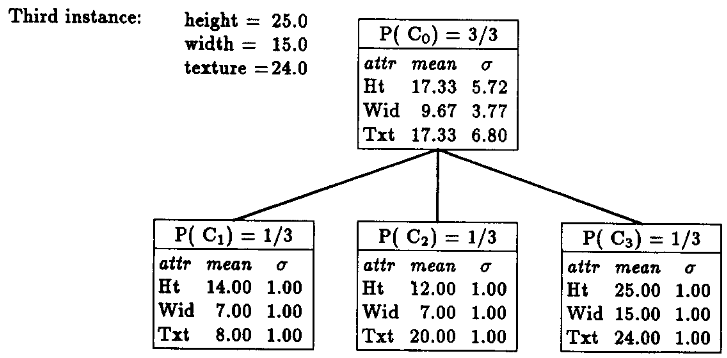
P(C_1) = 1/2		
attr	*mean*	σ
Ht	14.00	1.00
Wid	7.00	1.00
Txt	8.00	1.00

P(C_2) = 1/2		
attr	*mean*	σ
Ht	12.00	1.00
Wid	7.00	1.00
Txt	20.00	1.00

Fig. 6. Extending the CLASSIT hierarchy downward.

second instance. Since every instance encountered is incorporated into the root node, there is only one decision point as the system classifies this instance: is it different enough from the first to warrant extending the hierarchy down a level and creating separate concepts for each instance? In this case, although the second instance is also a "small" rectangle, the texture attribute is different enough from the first instance that CLASSIT creates a new level. Note that the σ scores for height and width at the root node are unchanged; this is because the standard deviations of these attributes remain lower than acuity.

Figure 7 shows the concept hierarchy after the system observes a third instance. After incorporating the instance into the root, the system must decide whether to add the instance into an existing child concept, or to make a new disjunct at level one. In this case, the choice with the highest category quality score is to create a new disjunct. Intuitively, this occurs because the instance is


Third instance: height = 25.0, width = 15.0, texture = 24.0

P(C_0) = 3/3		
attr	*mean*	σ
Ht	17.33	5.72
Wid	9.67	3.77
Txt	17.33	6.80

P(C_1) = 1/3		
attr	*mean*	σ
Ht	14.00	1.00
Wid	7.00	1.00
Txt	8.00	1.00

P(C_2) = 1/3		
attr	*mean*	σ
Ht	12.00	1.00
Wid	7.00	1.00
Txt	20.00	1.00

P(C_3) = 1/3		
attr	*mean*	σ
Ht	25.00	1.00
Wid	15.00	1.00
Txt	24.00	1.00

Fig. 7. Adding a new disjunct to the CLASSIT hierarchy.

a medium-sized rectangle; attributes height and width are sufficiently different from the existing classes to cause the creation of a new concept.

Figure 8 shows the hierarchy after the system classifies a second medium-sized rectangle. In this case, adding to an existing concept has a higher score than creating a new disjunct. This instance is therefore added to the existing "medium rectangle" concept (C_3) at level one. The system also decides that the new instance is different enough from concept C_3 to continue and extend the hierarchy to level two, creating a concept for each instance at that level.

The fifth instance is a large rectangle, and the system chooses to create another disjunct at level one. Figure 9 presents the hierarchy at this stage in the learning process. Remember that CLASSIT does not label this node as "large" nor does it know that the fifth instance belongs to the large class. The system incorporates each instance into its hierarchy without the benefit of class information.

Figure 10 shows the hierarchy after CLASSIT incorporates the final instance, a third "small" rectangle. This instance allows the system to merge two level-one concepts into a more general concept describing all three "small" rectangles. In more detail, the system proceeds as follows: It first considers adding the new instance to each of the four existing classes. In this case, the concept C_1 in Fig. 9 is the best candidate. CLASSIT then compares this score to that of making another level-one disjunct. Finally the system considers merging the best and

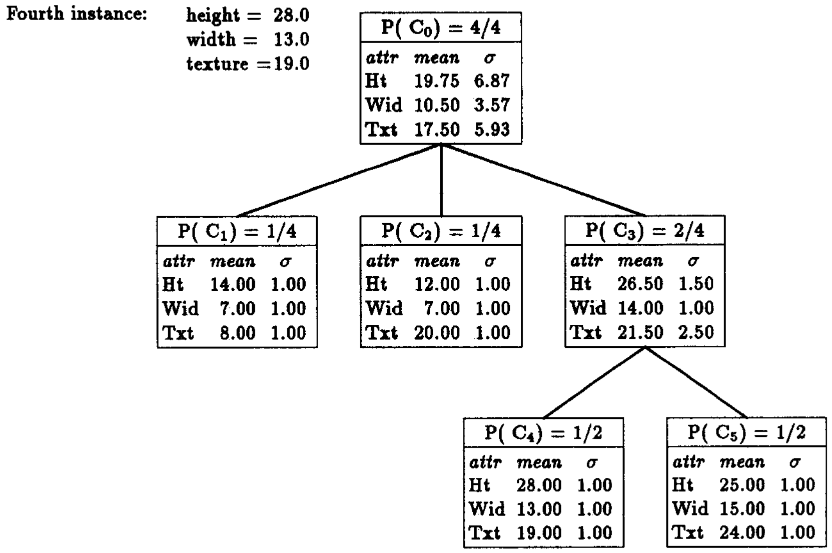


Fig. 8. Adding to an existing concept and extending the CLASSIT hierarchy.

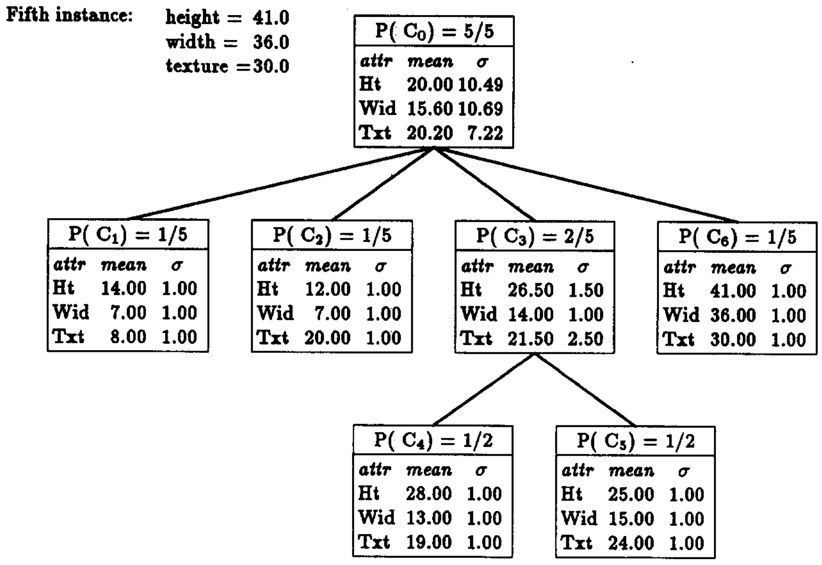

Fifth instance: height = 41.0 width = 36.0 texture =30.0
P(C0) = 5/5 attr mean σ Ht 20.00 10.49 Wid 15.60 10.69 Txt 20.20 7.22
P(C1) = 1/5 attr mean σ Ht 14.00 1.00 Wid 7.00 1.00 Txt 8.00 1.00
P(C2) = 1/5 attr mean σ Ht 12.00 1.00 Wid 7.00 1.00 Txt 20.00 1.00
P(C3) = 2/5 attr mean σ Ht 26.50 1.50 Wid 14.00 1.00 Txt 21.50 2.50
P(C6) = 1/5 attr mean σ Ht 41.00 1.00 Wid 36.00 1.00 Txt 30.00 1.00
P(C4) = 1/2 attr mean σ Ht 28.00 1.00 Wid 13.00 1.00 Txt 19.00 1.00
P(C5) = 1/2 attr mean σ Ht 25.00 1.00 Wid 15.00 1.00 Txt 24.00 1.00

Fig. 9. Creating another disjunct in the CLASSIT hierarchy.

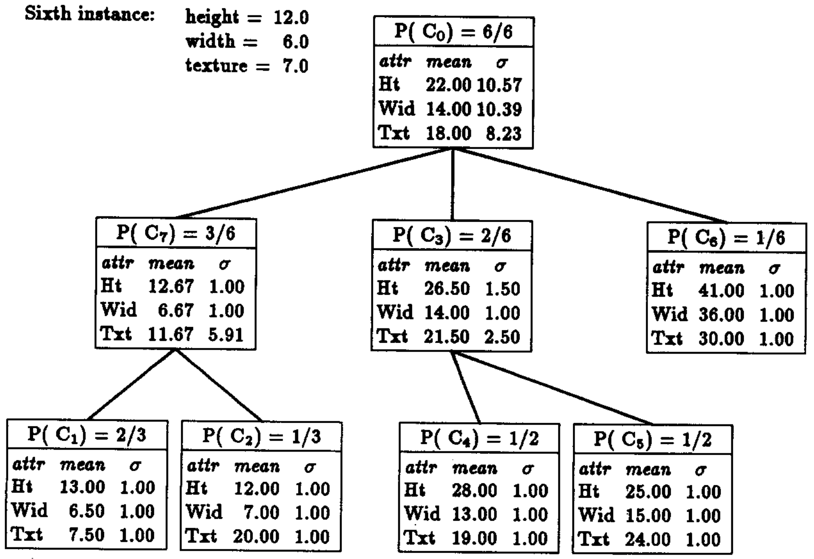

Sixth instance: height = 12.0 width = 6.0 texture = 7.0
P(C0) = 6/6 attr mean σ Ht 22.00 10.57 Wid 14.00 10.39 Txt 18.00 8.23
P(C7) = 3/6 attr mean σ Ht 12.67 1.00 Wid 6.67 1.00 Txt 11.67 5.91
P(C3) = 2/6 attr mean σ Ht 26.50 1.50 Wid 14.00 1.00 Txt 21.50 2.50
P(C6) = 1/6 attr mean σ Ht 41.00 1.00 Wid 36.00 1.00 Txt 30.00 1.00
P(C1) = 2/3 attr mean σ Ht 13.00 1.00 Wid 6.50 1.00 Txt 7.50 1.00
P(C2) = 1/3 attr mean σ Ht 12.00 1.00 Wid 7.00 1.00 Txt 20.00 1.00
P(C4) = 1/2 attr mean σ Ht 28.00 1.00 Wid 13.00 1.00 Txt 19.00 1.00
P(C5) = 1/2 attr mean σ Ht 25.00 1.00 Wid 15.00 1.00 Txt 24.00 1.00

Fig. 10. Merging two concepts in the CLASSIT hierarchy.

the second-best concepts into a new node; in our example, this last option has the best score.[19]

The merge operator merely pushes existing categories down a level. CLASSIT must also consider what to do with the new instance at level two. In this execution, the system decides to incorporate it into an existing child concept, C_1. At this point the cutoff parameter comes into play and the system decides that the new instance does not warrant its own concept at level three. This is hardly surprising, since the new instance is so close to the existing concept description that the standard deviations do not rise above acuity. In fact, the match among the three small rectangles is close enough so that the standard deviations for attributes height and width remain at acuity even for the new level-one concept, C_7.

CLASSIT continues processing new instances in this manner, incrementally modifying both its concept descriptions and the structure of its concept hierarchy as it encounters new data. Unlike some incremental learning systems—such as Mitchell's [31] version space method—CLASSIT never achieves a final knowledge state; the system continues to learn as long as new instances are available. This behavior is the strength of an incremental model. For example, it allows a system to recover from *concept drift*; if the environment changes over time, the learner must continue to modify his conceptual structures in response to new data.

4.5. A summary of CLASSIT

A principle motivation for the CLASSIT system was to model concept formation in the domain of real-valued inputs. This has affected our representation and our evaluation function. As yet, we have worked only with real-valued attributes since we feel that this type of input more closely models the output of the human perceptual system.

Since the same algorithm and four learning operators are used, CLASSIT retains the advantages of COBWEB. Both are incremental systems that integrate learning (concept formation and modification) and performance (classification), while carrying out a hill-climbing search for an optimal concept hierarchy.

5. Experimental Studies of CLASSIT's Behavior

One important approach to evaluating any AI system involves experimentation—studying the system's behavior under a variety of conditions. In this section, we present some experimental results that demonstrate CLASSIT's learning ability. We begin by introducing the domain we have used in most of

[19] The split operator is only considered when CLASSIT is about to add to a concept that already has children.

these studies. After this, we report three experiments in which we vary aspects of CLASSIT, followed by another study in which we vary the regularity in the domain. In each case, we describe the independent and dependent variables used in the experiment, summarizing the results in graphs. We close by reporting the system's behavior on a real-world domain that involves numeric attributes.

5.1. The domain of quadruped mammals

For our initial experiments, we designed an artificial domain involving four-legged mammals, each described as a set of eight cylinders. This approach let us control the environment while still retaining a reasonable approximation of physical objects. One can view our representation of objects as a simplification of Binford's [5] generalized cylinders, which have received wide attention within the machine vision community. Also, Marr [28] has argued that such representations are reasonable approximations of the output of the human visual system.[20]

As discussed earlier, CLASSIT assumes that each instance consists of a set of component objects, each described by a set of real-valued attributes. In the domain of quadruped mammals, each instance consists of eight cylindrical components: a head, a neck, a torso, a tail, and four legs. Each cylinder includes attributes such as height, radius, and location; there are a total of nine attributes per component, hence 72 attribute-value pairs per instance. We believe that real-world objects have at least this order of complexity and that a robust concept formation system should be able to handle instances of this form.[21]

In the runs described below, we assumed four basic categories that differed systematically only in the sizes of their cylinders. We will refer to these classes as *cats*, *dogs*, *horses*, and *giraffes*, since their relative sizes are roughly the same as those occurring for these real-world categories. Figure 11 shows a typical instance for each of these classes. One can view the prototype for a class as the "Platonic form" or ideal for that class. To generate instances from a particular class, we use a template that defines the prototypical value for each attribute

[20] We have developed CLASSIT within the context of the World Modeler's Project, a joint research effort between the University of California, Irvine, and Carnegie Mellon University. This project incorporates a simulated three-dimensional world, representing physical objects in terms of cylinders, spheres, circles, and polygons. Agents that interact with this environment perceive their surroundings directly in terms of such primitive shapes, along with their size, location, and orientation. Of course, CLASSIT need not assume such representations; it can be applied to any domain that one can express using numeric attributes.

[21] A more realistic description would represent physical objects at different levels of aggregation, as Marr [28] has proposed. Thus, an animal might have four legs, with each leg having three components, etc. However, such multi-level representations introduce some difficult problems, which we discuss in Section 6.

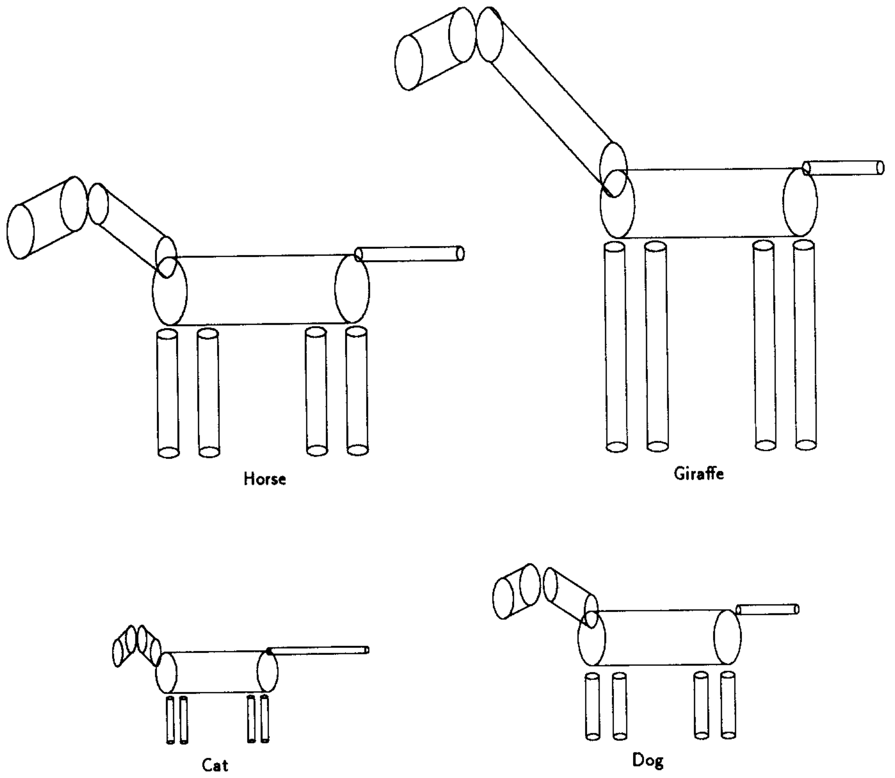


Fig. 11. Typical instances for four categories of quadruped mammals.

and a variance, specifying the degree to which that attribute will vary in the actual distribution of instances. Finally, each category has a probability that it will occur; some classes can be more common than others.

In producing data for our experiments, we used the prototype for each basic category to generate each instance according to the following procedure:

```
Randomly select a template $C$ with probability $P(C)$.
For each component $O$ in the prototype for $C$,
    For each attribute $A$ of component $O$,
        Let $M$ be the typical value of $A$ for $O$ in template $C$.
        Let $S$ be the variance of $A$ for $O$ in template $C$.
        Randomly select a value $V$ for $A$ according to a
          normal distribution with mean $M$ and variance $S$.
```

Thus, every instance is a member of one of the four categories, although CLASSIT is told neither the class name nor the number of classes. Each instance diverges from the ideal for that category, though some diverge more from this ideal than others and some attributes tend to vary more than others. Later we will examine CLASSIT's behavior on another artificial domain, but we will use the same basic method for generating data.

5.2. Learning and component matching

We have claimed that CLASSIT is a learning system, and learning is usually defined as some improvement in performance. Following Fisher [12], our first experiment examined the incremental improvement in the system's ability to make predictions. The dashed line in Fig. 12 presents CLASSIT's learning curve as it incorporates instances from the domain of quadruped mammals into its concept hierarchy.

The independent variable here is simply the number of instances seen. The dependent variable is the system's ability to predict a single missing attribute from all the other attributes in an instance. We measured this variable after every five instances by "turning off" the learning component and presenting CLASSIT with five randomly selected test instances, each missing a single attribute. After classifying each instance, the system uses the selected category to predict the value of the missing attribute. The graph measures the percentage error between the predicted value and the ideal value for the instance's actual class.[22] The percentage error describes the absolute prediction error relative to the other categories present in the hierarchy. One hundred percent indicates that the system has confused the instance with the wrong category.

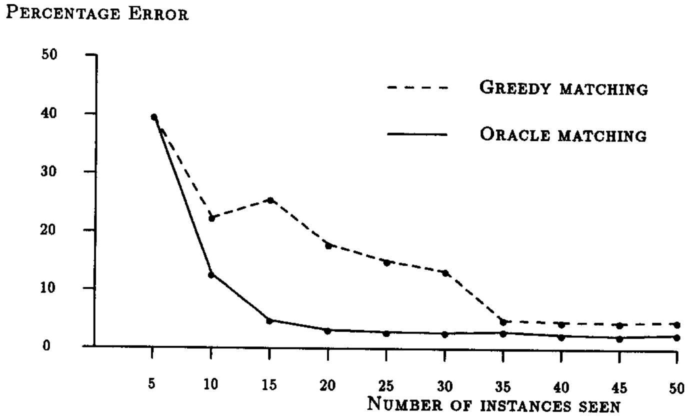


Fig. 12. CLASSIT learning curves with greedy versus oracle matching.

[22] Obviously, this measurement of error only makes sense for attributes that are relevant to classification; those attributes whose values differ across different classes. One cannot expect the system to correctly predict the value of an attribute that is irrelevant with respect to classification. Thus, we omitted only relevant attributes in measuring CLASSIT's improvement in predictive ability.

Clearly, the system's performance improves with experience, starting at 40% error and moving down to less than 5% error after 35 instances.

As described earlier, incremental algorithms tend to be sensitive to instance ordering. Although CLASSIT's split and merge operators allow some recovery from initial nonrepresentative orderings, learning curves still vary with different orderings. In order to minimize this effect, the measures in Fig. 12 have been averaged over 15 runs involving different random orderings. Also, since the data are produced randomly from templates, different instances are used for each ordering. We have followed this procedure in all our experiments.

In Section 4.2 we discussed CLASSIT's use of a greedy algorithm to match components in an instance to components in its concept descriptions, and it is this version that is summarized by the dashed line in Fig. 12. Given the heuristic nature of this matching scheme, we were interested in how it would fare against a version that had the optimal match available. The solid line in the figure shows the learning curve for such a system, in which we supplied CLASSIT with the correct correspondence between concept and instance components. This "oracle"-based variant improves its performance more quickly than the greedy version, reaching an asymptotic level after only 20 instances. However, despite some major errors early on (due to mismatched components), the greedy algorithm gradually narrows the gap, converging on nearly the same performance as the oracle version after 35 instances. This is a fairly impressive result for objects involving eight distinct components. In the remaining experiments, we report results only for the oracle version of CLASSIT, in order to factor out errors due to mismatches.

5.3. The effect of system parameters

We introduce the parameters for acuity and cutoff into CLASSIT only reluctantly, since such parameters encourage fine-tuning to achieve desirable behavior. To determine the effect of such tuning, we carried out the second experiment summarized in Fig. 13. As in the previous study, the horizontal axis specifies the number of instances and the vertical (dependent) axis shows the average percentage error. However, this time there are four learning curves, one for each setting of the acuity and cutoff parameters. We have repeated the oracle curve from Fig. 12, which was based on an acuity setting of 1.0 and cutoff setting of 0.2.

In this experiment we examined two levels of the cutoff parameter—0.2 and zero. The latter is the lowest possible setting, and effectively forces CLASSIT to retain all instances it has ever seen as terminal nodes in the hierarchy. Since the system always sorts a new instance as far down the hierarchy as possible, it will base its predictions on the values for a singleton concept. Unless each instance actually represents a distinct category, this strategy should lead to an

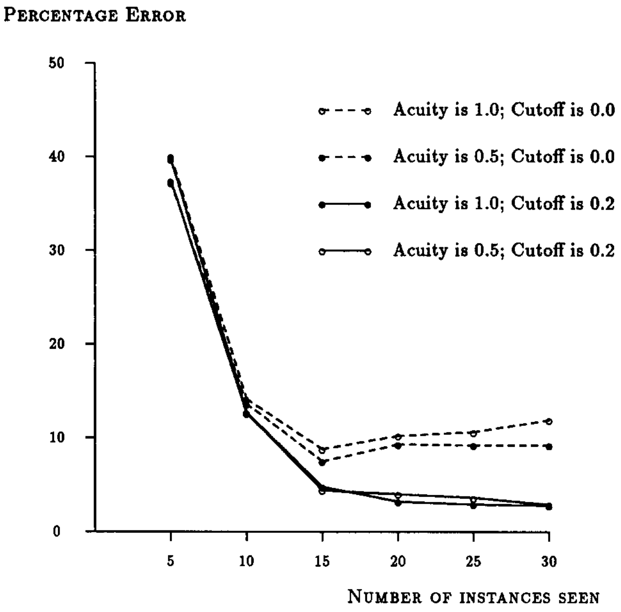


Fig. 13. The effect of acuity and cutoff on learning.

overfitting effect, similar to that Quinlan [34] has observed with decision trees in noisy domains.

Since we designed our quadruped data set to have only four generic categories, we would expect such overfitting on this domain as well. Indeed, the curves in Fig. 13 confirm this prediction. Both learning curves for the no-cutoff condition appear to asymptote at a higher error rate than the curves in the cutoff condition. With a higher setting for this parameter (i.e., with cutoff in operation), the system constructs simpler hierarchies with more general concepts as terminal nodes, and thus is able to make better predictions.

We also examined the effect of acuity, using two settings in this case as well. Unfortunately, the role of acuity is not as clear. In principle, one would expect overfitting to occur for low values of this parameter since this encourages CLASSIT to form many disjuncts. This should lead to a larger number of singleton classes, and thus to idiosyncratic predictions. However, this seems to occur only for extreme settings of the acuity parameter. Modifying the breadth of the hierarchy slightly does not have as strong an effect on prediction as does changing the depth of the tree with the cutoff parameter. Clearly, we need to carry out further studies to clarify the effect of this parameter.

In principle, one can get underfitting as well as overfitting effects. This should result in cases where CLASSIT constructs too shallow a hierarchy or creates too few disjunctive categories. However, the former can occur only if the "true" hierarchy contains multiple levels, and our quadruped data contains only one level of categories. For both parameters, one would expect a U-shaped curve, with high error from overfitting at one end of the spectrum and high error from underfitting at the other end, but we have not yet tested this prediction.

5.4. The effect of merging

We have discussed both COBWEB's and CLASSIT's potential sensitivity to the ordering of instances, and their use of merging and splitting operators to alleviate this effect. Our third experiment verifies that the merge operator has this predicted beneficial effect. Our technique was to "lesion" the system: that is, create a version of CLASSIT that cannot apply the merge operator, and compare its performance to the complete system. Recall that these "backtracking" operators are most useful when the system initially receives nonrepresentative instances. Therefore, for this experiment we arranged the order of instances by hand.

Figure 14 shows the results of an experiment in which two versions of CLASSIT—one with merging and the other without—were given a very skewed ordering of instances from the quadruped domain. First we presented five instances of the "horse" category, then five "giraffes," then five "cats," then five "dogs," then five more "horses," and finally five more "giraffes." Given such data, CLASSIT splits the initial horses into several classes at the top level, then creates new categories upon seeing the giraffes, cats, and dogs. The result is a skewed hierarchy, in which different types of horses are given the same status as the general classes of giraffes, cats, and dogs. The merge operator is designed to restructure such a hierarchy, creating a new category for horses and bringing particular horses down to an appropriate (lower) level.

Since CLASSIT sorts an instance as far down the hierarchy as possible, the internal structure of the hierarchy will have little if any effect on prediction. For this reason, we have used a different dependent measure in Fig. 14—the

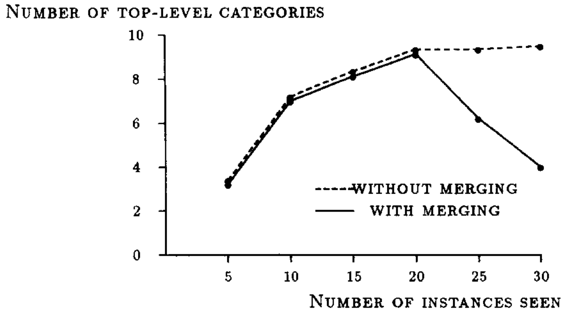


Fig. 14. The effect of merging on hierarchy structure.

number of top-level categories. This measure demonstrated precisely what one would expect. The number of categories at the top level continues to increase through instance 20. At this point, the new instances of "horse" lead the merging version of CLASSIT to combine the horse nodes at the top level into a single category. By instance 25, the number of top-level classes has decreased to around six, and by instance 30 it has reached four, the "correct" number. Note that merging combines only two nodes at a time, so this decrease is due to a sequence of merge operations. In contrast, the nonmerging version of CLASSIT incorporates the new horses into its existing categories, but retains the same top-level classes that the initially skewed data led it to create.

5.5. The effect of overlap and redundancy

Having considered the effect of varying CLASSIT's components on its learning behavior, let us examine the influence of two interesting domain characteristics. The first involves the number of attributes that are *relevant* in the sense that their values vary systematically with category membership. Intuitively, the more relevant attributes, the more *redundant* the data. The second variable involves the degree to which there is overlap between categories' values on an attribute; this corresponds to the percentage area that an attribute's probability distribution shares with the distribution from a neighboring class. Intuitively, the less overlap between two categories' values on an attribute, the more *distinguishable* those classes are on that attribute.

One would expect CLASSIT to have more difficulty in forming useful categories in the presence of highly overlapping attributes. The overlap between two distributions determines the probability that, on any given instance, the attribute value will fall in the region shared by both categories. In such cases, the attribute cannot be effectively used to determine the category to which the instance should be assigned. However, one would also expect highly redundant data to mitigate this effect. The more relevant attributes, the more attributes are likely to have values falling outside the area of overlap. Thus, we can predict an interaction effect, with CLASSIT's learning behavior worsening with increased overlap between categories, but with increased numbers of relevant attributes lessening this effect.

We tested this prediction in a fourth experiment. In this case we used a somewhat simpler artificial domain that let us independently control the two domain variables. Each instance consisted of five components with six attributes each, giving a total of 30 attributes, and instances were generated from only three category templates. (Hence, we assume there should be only three top-level categories). We varied the number of relevant attributes from two to ten. This represents a large amount of irrelevant information; two thirds or more of the attributes are irrelevant to predicting an instance's class. In contrast, an instance from the quadruped domain had two thirds of its attributes relevant. We also varied the amount of overlap between zero and fifty percent.

Figure 15 presents the results of this experiment. For simplicity, we have not reported learning curves in this case. Instead, the dependent variable shows predictive ability (average percentage error) after CLASSIT has viewed 30 instances. In all runs, we set acuity at 1.0 and cutoff at 0.2. As before, we averaged each point over 15 different random orderings.

The results are surprising. For higher numbers of relevant attributes, we see the expected interaction: increasing the number of relevant features helps more for higher levels of overlap, since they are worse to begin with. However, unexpected effects occur for lower redundancy settings, where even data with zero overlap leads to high error rates. Closer inspection suggests an explanation for this phenomenon. When there are only two relevant attributes (only one of which can be used on test instances), there are some 28 irrelevant ones that vary independently of category. Even when the relevant attributes never fall into the overlap areas, the irrelevant ones almost certainly do; despite their small individual contributions to category utility, their numbers overwhelm the small set of relevant features.

Unfortunately, this experiment confounds the total number of relevant attributes with the *percentage* of relevant attributes. To test our explanation, we must carry out further experiments in which we vary these two factors

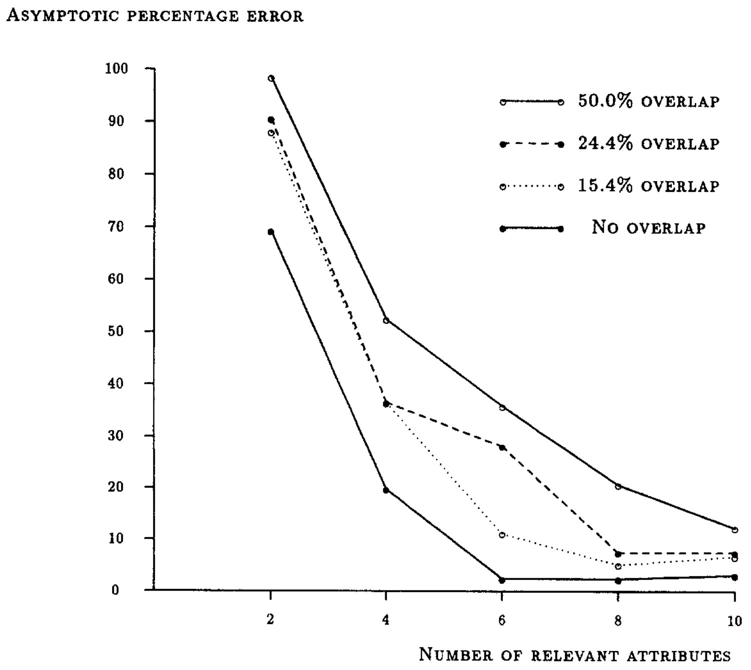


Fig. 15. The effect of overlap and redundancy.

independently. This is an important direction for future work, and it may ultimately let us predict CLASSIT's behavior from domain characteristics.

5.6. CLASSIT in a natural domain

Our final study examined CLASSIT's behavior on a real-world domain, using data on cardiology patients (Detrano [7]). In this data set, each patient has 13 measured or derived numeric attributes, along with a "class" attribute—whether or not the patient has heart disease. Unfortunately, Detrano indicates that this class information does not have a high accuracy; he estimates a 20% error rate.

Since CLASSIT is an unsupervised learning system, we discarded the class name and presented the system only with the numeric attributes for each instance. We then measured performance in terms of whether the system created concepts that corresponded to the prespecified classes. In effect, we asked the system to rediscover the class information from regularity in the numeric data. After seeing only ten of the total 303 instances, CLASSIT created three top-level concepts, and it retained this structure for the entire learning run.

Upon inspection, we found that one of these categories clearly corresponded to patients without heart disease; some 86.1% of its members had this label in the original data. The other two classes corresponded to patients with heart disease, one more consistently than the other; the accuracy was 79.7% and 66.6% for these groups. Overall, this represents a weighted average of 78.9% accuracy, which matches very well with the expected error rate of 20%. This is impressive, given that CLASSIT arrived at these categories without benefit of the class information.

6. Directions for Future Research

We believe that CLASSIT constitutes a promising framework for concept formation, and that it incorporates significant advances of earlier models. However, the existing system has a number of limitations that should be remedied in future efforts, and we discuss these below. We divide our treatment into issues of representation, matching, and learning.

6.1. Extending CLASSIT's representation

CLASSIT is designed to operate on numeric attributes, and we feel this is appropriate for domains based on visual input. However, symbolic or nominal attributes also have their uses, and we need to extend the system to handle this form of data. Recall that Fisher designed COBWEB to operate on nominal representations, and that CLASSIT uses a nearly identical algorithm for classification and learning. Moreover, our system's evaluation function is equivalent to Fisher's category utility metric, though we have modified it to work with numeric attributes. Thus, we hope to use a mixed evaluation function that includes discrete conditional probabilities for symbolic attributes and variances for numeric ones.[23] This should result in an integrated system that supports mixed forms of data.

In our work to date, we have used a simple set of primitives for describing objects, including cylinders and polygons. Clearly, we need to extend our framework to more realistic representations of the physical world. One approach would employ arbitrary polygons, which can be used to describe the surface characteristics on any three-dimensional object in arbitrary detail. However, this approach quickly leads to an unmanageable number of components for moderately complex objects. Alternatively, one might use Binford's [5] generalized cylinders to describe the volumetric aspects of objects. These require fewer components, but they introduce complex functional expressions to describe variations from a simple cylinder, and it would be difficult to extend CLASSIT to handle this scheme.

A more promising approach involves Biederman's [4] theory of *geons*, a set of 36 primitive shapes that can represent a wide range of complex objects. We see no difficulty in replacing our cylinders and polygons with geons, combining them to form more complex structures just as we currently do with simpler shapes. As before, each primitive component would be described in terms of its basic shape, along with numeric parameters specifying its size, location, and orientation. Some geons would require additional attributes to specify relative lengths of edges, but this would not be a problem for CLASSIT. Biederman has presented evidence that humans use geons in recognizing physical objects, and we hope that our revised system would make predictions about the human classification process.

6.2. Improving the matching process

The process of matching components between instance and concept is central to CLASSIT's behavior. Although the "oracle" approach was useful for experimental studies, it is not appropriate for normal operation. The greedy algorithm works reasonably well, but it leads to slower learning than the oracle method. We need additional studies to determine the robustness of the greedy scheme but we should also look for improvements on this method.

One approach involves making the greedy technique more heuristic in nature. The current version selects a component from the concept at random, finds the best matching component from the instance, selects another concept component at random, and so on. However, some components may have more diagnostic attributes than others, and matching against these components first

[23] This means that the $1/2\sqrt{\pi}$ term from Section 4.3 must be retained.

should improve the greedy method's chances for finding the optimal correspondences.

We also plan to examine the Hungarian algorithm (Papademetriou and Steiglitz [33]), a more expensive matching process that is guaranteed to find the optimal match. Given a bipartite graph with $2n$ nodes, along with some function for evaluating the quality of a match, the Hungarian method finds the best match in $O(n^3)$ time, as compared with $O(n^2)$ time for the greedy method. The algorithm works by creating an $n \times n$ cost matrix for all possible pairs of components and then solving an "n rooks" problem over this matrix. In general, we would expect this approach to perform better than the greedy algorithm. However, although it is guaranteed to find the optimal match according to CLASSIT's evaluation function, this need not agree with the "correct" match. Thus, we expect the resulting learning curve for this algorithm to fall somewhere between the two curves of Fig. 12. Whether the n^3 cost is prohibitive is an empirical question, but we guess that it is not, since n (the number of components) should seldom exceed ten for physical objects.

6.3. Handling missing attributes and components

Another aspect of matching involves dealing with instances having missing attributes. The current version of CLASSIT already takes this possibility into account, dividing the summed $1/\sigma$ scores by the number of attributes present. We used this scheme in classifying instances with a single missing attribute in our experiments, but we need further studies of its behavior when many attributes have been omitted.

In addition, entire components may be missing from an instance description. If we assume that CLASSIT's input is generated by a vision system, then components may be omitted because they are not visible. We may be able to use the same evaluation function in this case, simply treating the missing components as a set of missing attributes. However, we must still modify the component matching process to find a *partial* match between components in the instance and the concept. Although we do not have a complete specification, this modification seems feasible for either the greedy matching algorithm or the Hungarian algorithm.

6.4. Multiple levels of aggregation

Another research issue relates to the organization of complex objects with multiple components. Marr [28] has argued that the human visual system can generate descriptions of physical objects at different levels of aggregation. Thus, a dog might be viewed as a single cylinder at one level, as eight connected cylinders for (torso, neck, head, tail, and legs) at a lower level, with each leg described as three cylinders (thigh, calf, and foot) at a still lower level.

One difficulty with such a *part-of* hierarchy of objects lies in specifying the relation between different levels. We need to specify algorithms for moving from lower to higher levels that minimize information loss.

Once we have extended CLASSIT's representation in this direction, we will also need to alter its evaluation function and its matcher. CLASSIT can deal with two levels (a composite object and its components), but it cannot handle the general *n*-level case. Although EPAM was designed to handle composite, multi-level instances, neither UNIMEN nor COBWEB retained this ability. Wasserman [43] has described an extension to UNIMEM that takes a similar approach to EPAM, recursively sorting each component (and its components) through the concept hierarchy. However, EPAM does not address the problem of matching components at all (i.e., each component fills a unique slot), and Wasserman's extension uses a "greedy" matching strategy, the performance characteristics of which are not systematically evaluated.

Adding multiple levels of description to the CLASSIT framework raises a number of questions. Should the system use all levels in classification or only some? EPAM preferred to use attributes of composite objects when these were sufficient for avoiding errors. If we represent different levels in the same language, how can CLASSIT determine analogous levels between an instance and a concept description? How can one adapt the component matching process to work at multiple levels? Finally, how can one match a complex instance to a complex concept when its components are structurally different (e.g., a cylinder versus a block), and how should one alter the concept description in such cases? We must find at least tentative answers to these questions before we can extend CLASSIT in this direction.

6.5. Matching and normalization

We have designed CLASSIT with the domain of physical objects in mind, and this has led to our focus on composite instances and numeric attributes. In our experiments with the system, we have assumed that instances have the same location, orientation, and scale, but we must clearly abandon this simplification in future versions. Upon seeing a cat from a different angle than normal, one still recognizes it as a cat. Similarly, if one sees a cat in a different location, or even a cat of unusually large or small size, there is no recognition problem. Apparently, recognition focuses not on the absolute values of attributes, but on their *relative* values.

One might store in a concept description all pairwise relations between component objects, but this is neither space-efficient nor very plausible. A better approach involves selecting some scale, origin, and set of axes for the overall object concept, and then specify the scale, origin, and axes for each component relative to them. However, this raises a new issue: how can one determine these parameters for a new complex instance before it has been classified? We have not been able to devise a general algorithm that generates a canonical representation regardless of viewing angle, location, and size.

Instead, we hope to solve this *normalization* problem during the act of matching concept to instance. Upon observing an instance with multiple components, an extended CLASSIT would first match one of these components and use it to hypothesize the scale, origin, and axes for the composite object. This will lead to predictions about the locations of other components, which may or may not be correct. Hypothesized coordinate systems would be rejected, and those with better predictive ability would be extended, eventually leading to a completely normalized match. We plan to implement this normalization process in future versions of CLASSIT, though many details must still be specified.

6.6. Abstract descriptions and selective attention

Like Fisher's COBWEB, our system stores all known attributes on every concept description, even when they are neither predictive nor predictable. Earlier models of concept formation were more selective. Feigenbaum's EPAM starts with very general descriptions and gradually makes its images more specific through a process of familiarization. Lebowitz's UNIMEM and Kolodner's CYRUS gradually make their descriptions less specific through a generalization process. We need to explore variants on our basic algorithm that let it generate more abstract concept summaries, though the exact method is an open question.

A closely related problem is that CLASSIT inspects every attribute during the classification process, even if they have no predictive value. An improved system would incorporate the idea of selective attention, in which one focuses only on some features, presumably the useful ones. Earlier models of concept formation have this ability, including EPAM, UNIMEM, and CYRUS, as well as Fisher's COBWEB/2. The latter is encouraging, since it gives one path for incorporating attention into COBWEB, and thus into CLASSIT.

Ideally, the modified system would learn to prefer some attributes over others. In the early stages this selection would be random, since it would not know a priori which features would be diagnostic. However, as the system gained experience, it would come to prefer some attributes to others. Actually, CLASSIT already keeps statistics that would support this process. Using Bayes' rule, one can compute the predictiveness of each attribute from the existing scores. For example, the attribute "height" in Fig. 10 is clearly more predictive than "texture" at the first level. This is reflected by the fact that the difference between the average $1/\sigma$ score and the parent's $1/\sigma$ score is much larger for height than for texture.

In other words, CLASSIT's learning mechanism already supports such a focusing mechanism, and we need modify only the performance algorithm. The revised system would select only those attributes necessary to determine category membership with high probability. We could make this selection a deterministic function of predictiveness scores, but there is danger in this approach. If the initial instances are nonrepresentative or if the environment changes, the system might come to ignore attributes that later proved relevant. For this reason we prefer a probabilistic scheme, with more predictive attributes being selected more often, but even those with very low scores occasionally being sampled. We believe the addition of selective attention will make CLASSIT a more accurate model of human categorization and concept formation.

7. Summary

In this paper, we proposed a unifying framework for concept formation. We identified five features common to work on this task: that knowledge is represented in a concept hierarchy, that classification occurs in a top-down manner, that learning is unsupervised, integrated with performance, and employs an incremental hill-climbing search. We feel the search metaphor is especially important in understanding concept formation; it suggests both operators for learning and heuristics for controlling those operators.

We reviewed three concept formation systems (EPAM, UNIMEM, and COBWEB) that fit within our framework, along with a new system (CLASSIT) that builds on the earlier work. We have tried to emphasize the close relation between the systems, as well as the additions each makes over its predecessor. In particular, CLASSIT extends Fisher's approach to numeric attributes, can handle instances with multiple (unordered) components, and retains only some of the instances it encounters.

Finally, we presented some experimental studies of CLASSIT's behavior. We found that for the artificial domain of quadruped mammals, the system significantly improved its performance with experience, and that the greedy matching algorithm slowed down learning but did not seem to affect asymptotic performance. CLASSIT showed some sensitivity to its parameter settings, with low values for cutoff giving overfitting effects. We also presented evidence that the merge operator leads to more balanced hierarchies when the initial data is nonrepresentative. In examining the effects of domain characteristics, we found that more overlap between categories led to reduced improvement, and that more redundancy alleviated this effect. However, the relationship was more complex than we expected, and we need further experiments along these lines. Finally, we showed that when given real-world data on heart disease, CLASSIT was able to formulate diagnostically useful categories even without class information.

The representation, use, and acquisition of concepts is a complex, interconnected set of problems, and we cannot claim to have solved these problems in any absolute sense. However, we believe the basic approach we have described, and which is reflected in EPAM, UNIMEM, CYRUS, COBWEB, and CLASSIT, constitutes a promising thrust towards the computational understanding of categorization. We encourage other researchers to join in the effort, and to construct incremental models of concept formation that extend the initial results that have been achieved to date.

ACKNOWLEDGMENT

The ideas in this paper have resulted from many fruitful discussions with members of the UCI World Modelers Group. We would like to particularly acknowledge the criticism and encouragement provided by our work with Wayne Iba, Kevin Thompson, Patrick Young, and David Benjamin. Dennis Kibler, David Nicholas, Rick Granger and Jaime Carbonell also contributed important ideas. This research was supported by Contract MDA 903-85-C-0324 from the Army Research Institute.

REFERENCES

1. Anderson, J.R., Induction of augmented transition networks, *Cognitive Sci.* **1** (1977) 125–157.
2. Anderson, J.R. and Kline, P.J., A learning system and its psychological implications, in: *Proceedings IJCAI-79,* Tokyo (1979) 16–21.
3. Berwick, R., Learning structural descriptions of grammar rules from examples, in: *Proceedings IJCAI-79,* Tokyo (1979) 56–58.
4. Biederman, I., Matching image edges to object memory, in: *Proceedings IEEE First International Conference on Computer Vision,* London (1987) 384–392.
5. Binford, T.O., Visual perception by computer, Presented at IEEE Conference on Systems and Control, Miami, FL (1971).
6. Cheeseman, P., Kelly, J., Self, M., Taylor, W. and Freeman, D., Autoclass: A Bayesian classification system, in: *Proceedings Fifth International Conference on Machine Learning,* Ann Arbor, MI (1988) 65–64.
7. Detrano, R., International application of a new probability algorithm for the diagnosis of coronary artery disease, VA Medical Center, Long Beach, CA.
8. Everitt, B., *Cluster Analysis* (Heinemann Educational, London, 1974).
9. Feigenbaum, E.A., The simulation of verbal learning behavior, in: E.A. Feigenbaum and J. Feldman (Eds.), *Computers and Thought* (McGraw-Hill, New York, 1963).
10. Feigenbaum, E.A. and Simon, H., EPAM-like models of recognition and learning, *Cognitive Sci.* **8** (1984) 305–336.
11. Fisher, D., A hierarchical conceptual clustering algorithm, Tech. Rept. No. 85-21, Department of Information and Computer Science, University of California, Irvine, CA (1984).
12. Fisher, D., Knowledge acquisition via incremental conceptual clustering, *Mach. Learning* **2** (1987) 139–172.
13. Fisher, D., Knowledge acquisition via incremental conceptual clustering, Tech. Rept. No. 87-22 (Doctoral Dissertation), Department of Information and Computer Science, University of California, Irvine, CA (1987).
14. Gluck, M. and Corter, J., Information, uncertainty and the utility of categories, in: *Proceedings Seventh Annual Conference of the Cognitive Science Society,* Irvine, CA (1985) 283–287.
15. Grefenstette, J., Multilevel credit assignment in a genetic learning system, in: *Proceedings Second International Conference on Genetic Algorithms* (1987) 202–209.
16. Hanson, S.J. and Bauer, M., Conceptual clustering, categorization, and polymorphy, *Mach. Learning* **3** (1989) 343–372.
17. Holland, J., Escaping brittleness: The possibilities of general purpose algorithms applied to parallel rule-based systems, in: R.S. Michalski, J.G. Carbonell and T.M. Mitchell (Eds.),

Machine Learning: An Artificial Intelligence Approach **2** (Morgan Kaufmann, Los Altos, CA, 1986).

18. Iba, W., Wogulis, J. and Langley, P., Trading off simplicity and coverage in incremental learning, in: *Proceedings Fifth International Conference on Machine Learning*, Ann Arbor, MI (1988) 73–79.
19. Kolodner, J., Maintaining organization in a dynamic long-term memory, *Cognitive Sci.* **7** (1983) 243–280.
20. Langley, P., A general theory of discrimination learning, in: D. Klahr, P. Langley and R. Neches (Eds.), *Production System Models of Learning and Development* (MIT Press, Cambridge, MA, 1987).
21. Langley, P., Gennari, J. and Iba, W., Hill climbing theories of learning, in: *Proceedings Fourth International Workshop on Machine Learning*, Irvine, CA (1987) 312–323.
22. Lebowitz, M., Generalization and memory in an integrated understanding system, Tech. Rept. No. 186 (Doctoral Dissertation), Department of Computer Science, Yale University, New Haven, CT (1980).
23. Lebowitz, M., Generalization from natural language text, *Cognitive Sci.* **7** (1983) 1–40.
24. Lebowitz, M., Categorizing numeric information for generalization, *Cognitive Sci.* **9** (1985) 285–309.
25. Lebowitz, M., Concept learning in a rich input domain: Generalization based memory, in: R.S. Michalski, J.G. Carbonell and T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach* **2** (Morgan Kaufmann, Los Altos, CA, 1986).
26. Lebowitz, M., Experiments with incremental concept formation: UNIMEM, *Mach. Learning* **2** (1987) 103–138.
27. Levinson, R., A self-organizing retrieval system for graphs, in: *Proceedings AAAI-84*, Austin, TX (1984) 203–206.
28. Marr, D., *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information* (Freeman, San Francisco, CA, 1982).
29. Michalski, R.S., A theory and methodology of inductive learning, in: R.S. Michalski, J.G. Carbonell and T.M. Mithcell (Eds.), *Machine Learning: An Artificial Intelligence Approach* (Tioga, Palo Alto, CA, 1983).
30. Michalski, R.S. and Stepp, R., Learning from observation: Conceptual clustering, in: R.S. Michalski, J.G. Carbonell and T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach* (Tioga, Palo Alto, CA, 1983).
31. Mitchell, T.M., Generalization as search, *Artificial Intelligence* **18** (1982) 203–226.
32. Mitchell, T.M., Utgoff, P. and Banerji, R., Learning by experimentation: Acquiring and refining problem-solving heuristics, in: R.S. Michalski, J.G. Carbonell and T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach* (Tioga, Palo Alto, CA, 1983) 163–190.
33. Papademetriou, C. and Steiglitz, K., *Combinatorial Optimization* (Prentice-Hall, Englewood Cliffs, NJ, 1982).
34. Quinlan, J.R., Induction of decision trees, *Mach. Learning* **1** (1986) 81–106.
35. Rendell, L., Seshu, R. and Tcheng, D., More robust concept learning using dynamically-variable bias, in: *Proceedings Fourth International Workshop on Machine Learning*, Irvine, CA (1987) 66–78.
36. Rose, D. and Langley, P., A hill-climbing approach to machine discovery, in: *Proceedings Fifth International Conference on Machine Learning*, Ann Arbor, MI (1988) 367–373.
37. *Rosch, E., The principles of categorization, in: E. Rosch and B.B. Lloyd (Eds.), Cognition and Categorization* (Erlbaum, Hillsdale, NJ, 1978).
38. Schlimmer, J. and Fisher, D., A case study of incremental concept induction, in: *Proceedings AAAI-86*, Philadelphia, PA (1986) 496–501.

39. Schlimmer, J. and Granger, R., Beyond incremental processing: Tracking concept drift, in: *Proceedings AAAI-86*, Philadelphia, PA (1986) 502–507.
40. Shrager, J., Theory change via application in instructionless learning, *Mach. Learning* **2** (1987) 247–276.
41. Simon, H.A., *The Sciences of the Artificial* (MIT Press, Cambridge, MA, 1969).
42. Smith, E.E. and Medin, D.L., *Categories and Concepts* (Harvard University Press, Cambridge, MA, 1981).
43. Wasserman, K., Unifying representation and generalization: Understanding hierarchically structured objects, Doctoral Dissertation, Columbia University, New York (1985).
44. Winston, P.H., Learning structural descriptions from examples, in: P.H. Winston (Ed.), *The Psychology of Computer Vision* (McGraw-Hill, New York, 1975).