

# Unsupervised Learning of Probabilistic Concept Hierarchies

WAYNE IBA (IBA@ISLE.ORG)  
PAT LANGLEY (LANGLEY@ISLE.ORG)

Institute for the Study of Learning and Expertise  
2164 Staunton Court, Palo Alto, California 94306

## Abstract

Fisher's COBWEB provided a well-defined framework for research on the unsupervised induction of probabilistic concept hierarchies. The system also sparked the development of many successors that extended this framework along various dimensions. In this paper, we summarize the assumptions that COBWEB embodies about the representation, organization, use, and formation of probabilistic concepts, along with experimental studies that examine its sources of power. After this, we consider three systems – ARACHNE, TWILIX, and OXBOW – that incorporate significant extensions and present empirical evidence that these improve behavior. In closing, we discuss other paradigms for the unsupervised learning of probabilistic knowledge and their relation to the COBWEB framework.

We thank our collaborators, including John Gennari, Kathleen McKusick, Kevin Thompson, and John Allen, for their contributions to the research described in this paper. Grant MDA 903-85-C-0324 from the Army Research Institute supported our early work within the COBWEB framework, whereas later extensions were developed at NASA Ames Research Center.

## 1. Introduction

Since the field’s inception, most research in machine learning has focused on the problem of supervised induction from labeled training cases. If anything, this trend has been strengthened by the creation of data repositories that, typically, include class information. But this emphasis is misguided if we want to understand the nature of learning in intelligent agents like humans. Clearly, children acquire many concepts about the world before they learn names for them, and scientists regularly discover patterns without any clear supervision from an outside source. Even the availability of class labels in public data sets can be misleading; many such domains are medical in nature, and medical researchers first had to discover a disease before they could diagnose it for particular patients.

Naturally, different approaches to induction from unlabeled data are possible, each stemming from different research goals. In this paper, we report on a class of unsupervised methods designed to support learning in intelligent agents, whether human or artificial. This concern suggests some criteria that such methods should satisfy; these include:

- The aim of unsupervised learning should be to acquire *concepts* or *categories*;
- Concept descriptions should represent the *variability* that occurs in the natural world;
- These concepts should describe experience at *different levels of generality*;
- The learning process should be *incremental*, since intelligent agents interact with the environment over time.

Taken together, these criteria place strong constraints on the representation and organization of knowledge, and on the mechanisms that support performance and learning.

Our research has explored a paradigm for unsupervised learning that meets these criteria. The framework assumes that knowledge takes the form of probabilistic concept descriptions and that these concepts are organized into an ‘is-a’ hierarchy that describes different levels of generality. For this reason, we will often refer to the framework as one that assumes *probabilistic concept hierarchies*. However, we also assume that using this knowledge involves sorting new experiences downward through the hierarchy, and that this act of retrieval also produces changes in the concept descriptions and hierarchy structure, resulting in learning. In this paper, we report on a large body of work that falls within this paradigm.

We begin by describing COBWEB, the system that played the founding role in this framework, and some experimental studies that reveal the importance of its various components. After this, we examine three systems that extend the basic approach; these include ARACHNE, which incorporates restructuring operators designed to handle noise and minimize order effects, TWILIX, which constructs more complex concept hierarchies that support overlapping categories, and OXBOW, which forms concepts about temporal phenomena. We also briefly review a number of other systems that incrementally construct hierarchies of probabilistic concepts. In closing, we examine the framework’s relation to two more recent approaches to unsupervised induction, one based on the AUTOCLASS family and the other involving Bayesian networks.

## 2. Incremental formation of probabilistic concept hierarchies

As noted above, our focus in this paper is a framework for unsupervised learning that grew out of Fisher’s COBWEB (1987). We can describe the system’s goals in terms of its performance and learning tasks. For the former, COBWEB aims to infer the values of attributes missing from test cases, a task that Fisher called *flexible prediction*. For learning, the system aims to carry out hierarchical clustering over unlabeled training cases that are presented in an online fashion and thus organize these instances in memory. Incidentally, this also lets it estimate the probability density function over the space of possible instances, which in recent years has become a more popular way to describe unsupervised learning. Now let us consider the manner in which COBWEB achieves these goals.

### 2.1 Representation and organization of knowledge

The central representational notion in COBWEB is the *concept* or *category*. The system generates an arbitrary name  $C$  for each such category and associates with  $C$  a descriptive summary. For each attribute  $A$ , this summary specifies a probability distribution over the values of  $A$ , conditioned on the category  $C$ . For a nominal or discrete attribute, this takes the form of a discrete probability distribution; for a numeric or continuous variable, the system uses a normal distribution, which can be characterized by its mean and variance.<sup>1</sup> Since it stores only marginal probability distributions, COBWEB makes the assumption that the observable attributes are independent given the category.

The key organizational theme in COBWEB is that categories are placed in a *concept hierarchy*. Each node  $C$  in this ‘is-a’ tree specifies a set of children and the conditional probability of  $C$  given its parent category. Moreover, the descriptive summaries between a parent and its children obeys an important relation: the probability distribution for each attribute of a parent is a weighted mixture of those for its children, so that each node constitutes a probabilistic summarization of all its descendants. The terminal nodes or leaves in a COBWEB hierarchy correspond to specific instances, typically training cases observed during learning. Thus, the root node corresponds to the most general category, which summarizes all instances the system has seen, and categories become increasingly specific as one moves down the hierarchy.

Figure 1 shows a small probabilistic concept hierarchy for the domain of household pets, with the descriptive summaries for three nodes. The leaf node  $N8$  describes a single animal (a Mexican hairless dog) that is small, has four legs, barks, and has a body covered with skin. Since this category involves only one example, all attribute probabilities are either 1 or 0, and the node’s conditional probability given the parent is  $\frac{1}{2}$ , as it has only one sibling. The parent node,  $N5$ , is somewhat more abstract, summarizing two instances (both dogs) that differ on some dimensions but not others. Thus, the conditional probability of having four legs and barking is still 1, given an instance of category  $N$ , but the probability distributions for size and body cover are more diffuse. The node next higher in the hierarchy,  $N2$ , is even more abstract, since it also covers instances

---

1. Most implementations of COBWEB store these internally using counts for the category, counts for nominal attribute values, and the sums and sums of squares for numeric attributes.

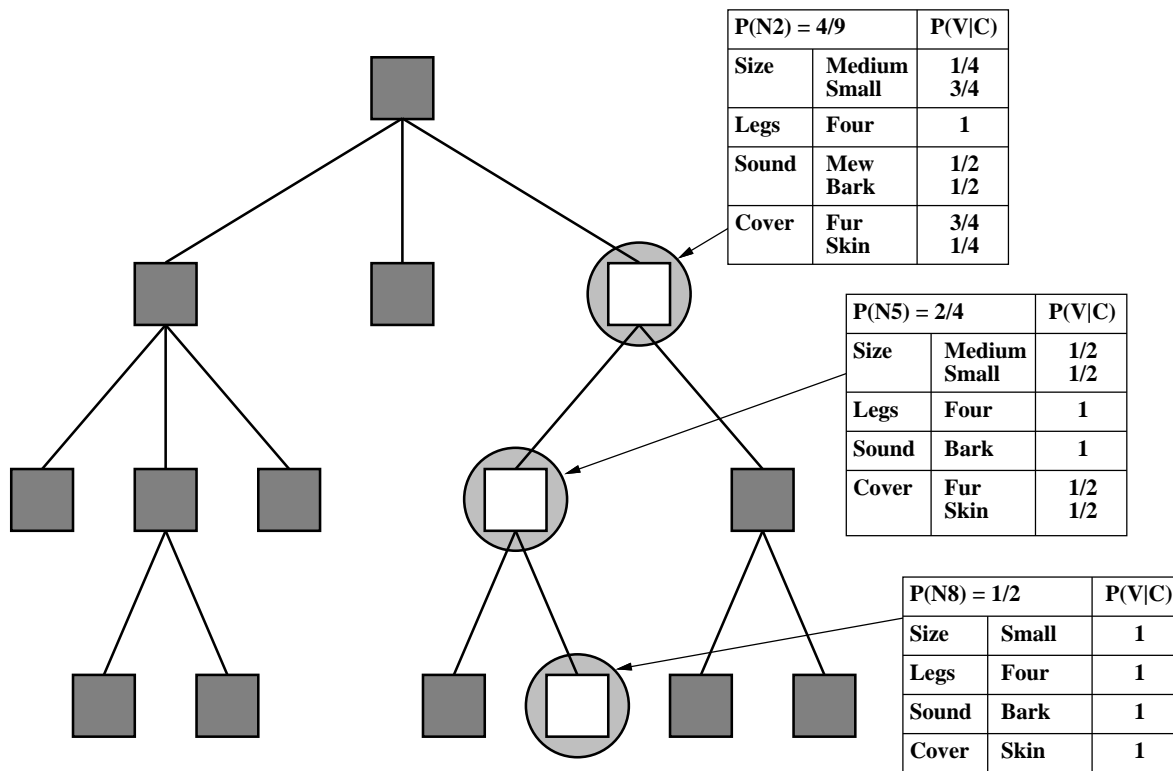


Figure 1. A small hierarchy of probabilistic concepts for the domain of household pets, illustrating COBWEB's representation and organization of knowledge.

(cats) that mew rather than bark, but still has little variation in the legs attribute, since all its children are quadrupeds. This node has a conditional probability of  $\frac{4}{9}$  given its parent, the root, because  $N2$  subsumes that fraction of the leaves in the tree.

## 2.2 Performance and learning mechanisms

Naturally, COBWEB's representation and organization of knowledge figure prominently in its use of that knowledge. Given a new test case, typically with only some attributes specified, the system sorts the experience downward through the concept hierarchy by recursively assigning the instance to the best child category at each level. This process halts when the test case reaches a terminal node or when it is not similar enough to any child to justify sorting downward further. At this point, COBWEB infers the values for attributes missing from the instance, using the modal values from the deepest (most specific) node through which it has passed.

COBWEB integrates classification and learning, so that the latter process takes place as the system sorts training cases through the hierarchy. As this occurs, the algorithm updates the conditional probability distributions for each node through which the instance passes, guaranteeing the property mentioned earlier that each category summarizes all cases below it in the hierarchy.

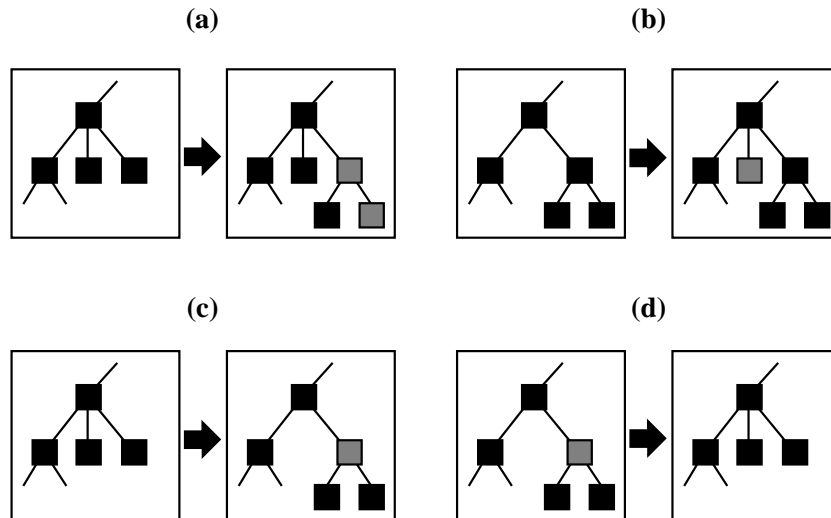


Figure 2. Learning operators used to modify the structure of a hierarchy of probabilistic concepts: (a) extending the hierarchy downward; (b) creating a new disjunct at an existing level; (c) merging two existing classes; and (d) splitting an existing category. Newly created nodes are shown in gray.

However, COBWEB can also invoke four learning operators to alter the structure of its concept hierarchy. As illustrated in Figure 2, these include:

- *extending downward*, which occurs when a training case reaches a terminal node in memory; under these circumstances, the learner creates a new node  $N$  that is a probabilistic summary of the case and the terminal node, making both children of  $N$ ;
- *creating a disjunct*, which occurs if a training case is sufficiently different from all children of a node  $N$ ; in this situation, the learner creates a new child of  $N$  based on the case;
- *merging two categories*, which occurs if a case is similar enough to two children of node  $N$  that the learner judges all three should be combined into a single child;
- *splitting a concept*, which occurs when a child  $C$  of node  $N$  no longer serves as a useful category;  $C$  is removed and its children are promoted to become direct children of  $N$ .

The system considers the last three of these actions at each level of the hierarchy, as it sorts the new training instance downward through memory. Merging and splitting are designed to reduce sensitivity to training order, giving the effect of backtracking in the space of concept hierarchies without requiring the storage of previous hypotheses.

We have not yet described how COBWEB decides which category to select at each level, whether to halt at an internal node, and whether to merge or split categories. To this end, the system uses an evaluation function called *category utility* that measures the quality for a set of probabilistic categories. Given a set of  $K$  categories with discrete attributes, category utility was originally defined by Gluck and Corter (1985) as the *increase* in the expected number of attribute values that can be correctly guessed over the expected number of correct guesses without such knowledge. The modification introduced by Fisher (1987) and used by COBWEB is

$$\frac{1}{K} \left[ \sum_{k=1}^K P(C_k) \sum_i \sum_j P(A_i = V_{ij}|C_k)^2 - \sum_i \sum_j P(A_i = V_{ij}|C)^2 \right] , \quad (1)$$

where  $k$  varies over categories,  $i$  over attributes, and  $j$  over values for each attribute. This function evaluates a *partition* — defined as a parent node  $C$  and its immediate children  $C_k$ . The probability  $P(C_k)$  represents the *base rate* or the prior probability that an instance is a member of the child  $C_k$ , whereas  $P(A_i = V_{ij}|C_k)^2$  is a measure of *within-class similarity* for an attribute  $A_i$ ; that is, how well the instances summarized by  $C_k$  resemble one another. Subtracting the sum over the parent’s within-class similarities,  $P(A_i = V_{ij}|C)^2$  for each attribute, lets category utility measure the information gained by partitioning the parent class into the given set of children. Dividing by  $K$ , the number of  $C$ ’s children, biases the system against proliferation of singleton classes.

For numeric domains, we must modify this evaluation function because the probability that a continuous attribute will take on a particular number is zero. For such attributes, probabilities are estimated by assuming that values conform to a normal distribution with a particular mean and standard deviation. Thus, for a domain with continuous attributes, Gennari, Langley, and Fisher (1989) define category utility as

$$\frac{1}{K} \left[ \sum_k P(C_k) \sum_i \frac{1}{\sigma_{ik}} - \sum_i \frac{1}{\sigma_{ip}} \right] , \quad (2)$$

where  $P(C_k)$  is the probability of class  $C_k$ ,  $K$  is the number of categories,  $\sigma_{ik}$  is the standard deviation for an attribute  $i$  in class  $C_k$ , and  $\sigma_{ip}$  is the standard deviation for attribute  $i$  in the parent node.<sup>2</sup> One can also combine continuous and discrete attributes in the category utility calculation simply by using the appropriate form of the equation for each attribute. That is, for nominal attributes, the inner summation over values uses Equation 1, and for numeric attributes it uses the inverse of the standard deviation.

We should close our review of COBWEB with some remarks about its assumptions and representational power. At each level of the concept hierarchy, the system assumes that attributes are conditionally independent given the category. This condition, which COBWEB shares with the naive Bayesian classifier (Langley, Iba, & Thompson, 1992), will clearly be violated in many domains. But note that the system seldom makes its predictions at the hierarchy’s top level, and that categories lower in the tree describe local portions of the instance space where approximate independence may be satisfied. Indeed, recall that the leaves in a probabilistic concept hierarchy correspond to individual training cases, and that COBWEB sometimes bases its prediction on these nodes. In these situations, the system operates as a nearest neighbor classifier that uses the concept hierarchy to weight attributes and bias retrieval. In summary, COBWEB’s use of a hierarchical memory gives it the ability to represent complex target concepts, at least in principle. But whether its approach works in practice is an empirical issue, to which we now turn.

---

2. As discussed by Gennari et al. (1989), the value of  $1/\sigma$  is undefined for any concept based on a single instance, so an *acuity* parameter is needed. Acuity corresponds to the notion of “just noticeable difference” in psychology.

### 3. Empirical studies of COBWEB

COBWEB appeared on the scene during a period when machine learning researchers were first starting to carry out systematic experiments with their algorithms. As a result, the system has always been under close empirical scrutiny, though some early studies occurred before clear standards developed within the community. Here we review a number of experimental evaluations of COBWEB, concentrating on ones that have not appeared previously in the literature.

#### 3.1 Basic experimental results

Initial experimental studies of COBWEB, as with all learning systems, aimed mainly to show that its performance improved with experience. Fisher (1987) reported results on a number of natural domains for a task he called *flexible prediction*. This involved testing on instances with some attribute omitted and letting the system predict the missing value for each case, then repeating this process for each attribute and averaging the result. The technique extends naturally to continuous attributes, and Iba (1991a) adapted it to measure prediction errors about temporal phenomena.

Another approach, taken by Gennari (1990), used labeled training data but held back the label during hierarchy construction. This scheme added the most likely class values to categories only after the hierarchy was complete, then used them to predict the classes for test cases and produce a standard measure of classification accuracy. Figure 3 shows two learning curves that McKusick and Langley (1991) obtained in this manner for COBWEB and a related algorithm, ARACHNE, that we discuss later. One domain involved predicting the party of U.S. Congressmen from their voting records, whereas the other domain dealt with diagnosing disease in soybean plants. Although the latter task is clearly more difficult, COBWEB shows steady improvement in its ability to predict class labels, even though it could not use them in constructing its concept hierarchy.

Perhaps the most extensive experiments with COBWEB<sup>3</sup> come from Gennari (1990), who wanted to understand the sources of power in the framework. His studies included variation of system parameters and domain characteristics, using predictive accuracy and learning curves to measure COBWEB's sensitivity to particular settings and domain features. In the remainder of this section, we consider three of Gennari's experiments that have not appeared in the literature. These include the significance of category utility as the evaluation metric used for clustering, the importance of COBWEB's algorithm for hierarchy formation, and the effect of missing attributes on behavior.

#### 3.2 Effect of the evaluation function

Let us first consider the importance of category utility, the evaluation function that COBWEB uses when sorting cases through memory and when restructuring its hierarchy. Although Fisher presents convincing arguments for using this metric, its behavior relative to other functions remains an empirical question. Gennari was interested in whether more traditional *distance* metrics, common

---

3. Gennari's studies dealt with a rational reconstruction of Fisher's system, which he called CLASSIT, that also included a number of extensions. Because most of these additions were later subsumed by COBWEB/3 (McKusick & Thompson, 1990), we will also refer to it as COBWEB.

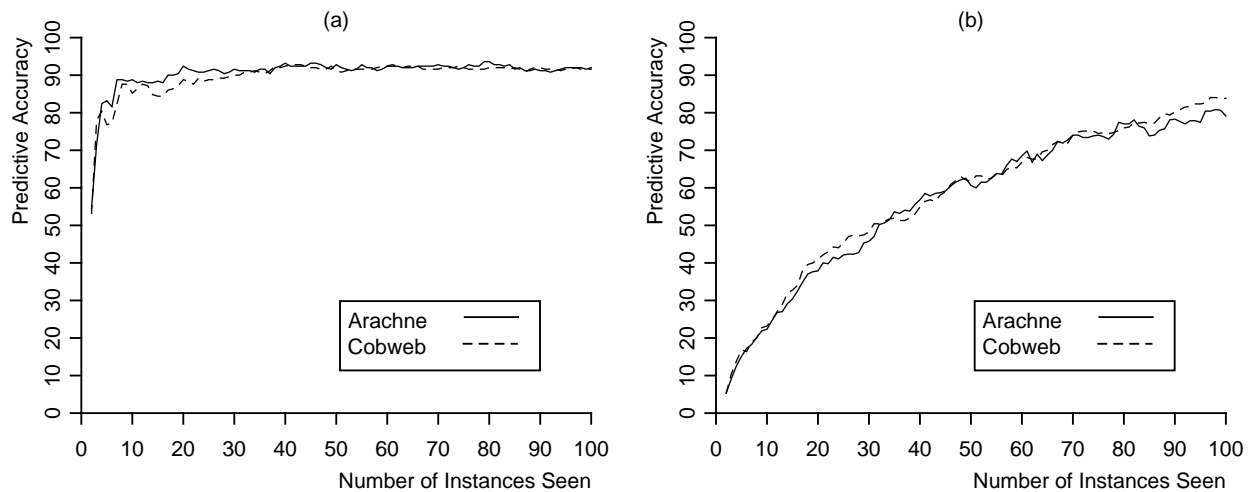


Figure 3. Learning curves for COBWEB and a similar system, ARACHNE, on (a) congressional voting records and (b) soybean diseases, using classification accuracy as a performance measure.

in the clustering literature, would produce similar results. However, to embed such a measure in COBWEB required modifying them to operate over entire partitions rather than between pairs clusters. To this end, he introduced the function

$$\text{trace}(\mathbf{W}) = \sum_k^K \frac{1}{N_j} \sum_i^{N_j} (x_i - \bar{x}_{jk})^2, \quad (3)$$

which gives an overall measure of the difference among the probabilistic summaries for sibling nodes (i.e., a distance between probability distributions). He also notes that this function is similar to another metric, which he calls *category value*, that is simply category utility without information from the parent node subtracted out.

Indeed, he notes that  $\text{trace}(\mathbf{W})/K$  and category value are *isotonic*, in that they produce the same ordering over partitions and thus can be used interchangeably when making decisions. In fact, when the number of classes is held constant, category value is also isotonic to category utility. This means that the two functions differ primarily in that the former prefers fewer children per node than the latter. Because category value ignores the information at the parent node, Gennari expected COBWEB to perform somewhat worse using this metric than it does when relying on category utility.

An experiment with two natural data sets, one on a glass domain and another on heart disease, revealed almost identical learning curves for the two evaluation functions, violating predictions. To gain a better understanding of system behavior, Gennari designed two synthetic domains, each with nine attributes but one having three distinct classes and the other six. In this study, he measured both classification error and hierarchy depth after the system had processed 150 training cases, which were presumably enough to master the concepts in these simple domains.

Table 1 shows the predictive error and average tree depth for both evaluation functions on the three-class and six-class domains. Note that the error rates for the two metrics are very similar,

Table 1. The behavior of COBWEB with category utility vs. category value on two synthetic domains.

	THREE CLASSES		SIX CLASSES	
	ERROR	DEPTH	ERROR	DEPTH
CATEGORY UTILITY	2.15	1.23	2.08	1.35
CATEGORY VALUE	2.05	2.40	1.99	3.77

replicating the result with natural domains. However, category value constructs a deeper concept hierarchy, as one would expect given its bias toward fewer children for each node, with category utility giving trees that more closely reflected the target concepts. Since COBWEB’s performance element often sorts instances to terminal nodes when making predictions, tree structure often has little effect on predictive accuracy. But structural differences in the hierarchy can play a role in other contexts, as we discuss later, and category utility seems to hold advantages in such situations.

### 3.3 The effect of search control

Another key factor in the COBWEB framework concerns the algorithm that controls search through the space of concept hierarchies. However, the literature on clustering abounds with unsupervised algorithms, suggesting that some of these approaches might give similar or even better results. Thus, Gennari carried out a second comparative study between COBWEB’s incremental sorting method and two techniques that are widely used in clustering circles.

The first method also constructs hierarchies, but does this in a nonincremental, agglomerative fashion. Starting with each training case as a separate category, this algorithm finds the two categories that are nearest to each other, creates a parent node that specifies them as children, and replaces the original nodes with the new one in the set of candidates. This process continues, repeatedly combining the most similar pair of categories, until only one node (the root) remains. Because it must calculate all pairwise similarities on each iteration, this agglomerative method is considerably more expensive than COBWEB in computational terms.

The second method, known as *iterative optimization*, clusters training cases into  $k$  categories at a single level, where the user specifies the number of clusters. This process selects  $k$  instances at random as seeds to serve as initial category centroids, then assigns other cases to the cluster with the nearest seed. Next, it computes a new centroid for each cluster based on the cases it contains and reassigns each instance based on its distance to the revised centroids. This process repeats until no changes in the categories occur. This technique is sometimes known as  $k$  means clustering.

To compare the behavior of these algorithms with COBWEB, Gennari transformed the taxonomy constructed by the agglomerative scheme into a probabilistic concept hierarchy on which he could run COBWEB’s performance element. In a similar manner, he transformed the clusters generated through iterative optimization into a one-level probabilistic ‘hierarchy’ on which COBWEB could operate. He studied the three methods’ behavior on two domains. One involved a synthetic data

Table 2. The behavior of different clustering methods on a synthetic and natural domain.

	THREE CLASSES		GLASS DOMAIN	
	ERROR	DEPTH	ERROR	DEPTH
COBWEB	2.40%	2.63	13.9%	2.93
AGGLOMERATIVE	1.71%	7.67	18.2%	11.73
ITERATIVE OPT. ( $k = 6$ )	5.07%	(1)	18.5%	(1)
ITERATIVE OPT. ( $k = 8$ )	2.82%	(1)		

set with nine numeric attributes and four top-level classes, two of which had two subclasses (giving six total leaf categories); the other consisted of the glass data set from the UCI repository.

Table 2 shows the results for both domains. Note that, on the synthetic data, the agglomerative method outperforms the other algorithms in terms of classification error, although it constructs a substantially deeper hierarchy than COBWEB. The latter difference comes as no surprise, since the agglomerative technique always generates binary trees. On the glass data set, COBWEB again creates a relatively shallow tree, but also gives the lowest classification error. Clearly, studies with more domains seem in order, but these preliminary results suggest that COBWEB’s incremental search-control methods can hold their own with more traditional nonincremental schemes from the clustering literature.

### 3.4 The effect of missing information

Many real-world data suffers from the characteristic of missing information, in that instances have no value for some attributes. Although COBWEB was designed to infer missing attributes in test cases, the omission of values during training is another matter entirely. Naturally, as the number of missing attributes increases, we would expect the learning rate to degrade somewhat, but we would also hope that COBWEB’s reliance on probabilistic summaries would make this degradation a graceful one.

COBWEB’s treatment of missing information is extremely simple and straightforward. Given a training case with some attributes marked as “missing”, the calculation of the scores when incorporating the new instance into an existing concept uses only those attributes that are present in the instance.<sup>4</sup> For example, given a domain with seven attributes and a training case with two missing attributes, the system calculates category utility over the five attributes present, rather than over all seven features.

Gennari carried out an experimental study with a synthetic domain designed to evaluate COBWEB’s tolerance to partial training data. In this domain, each instance is described by a class label

4. More complex responses to missing attributes are necessary for other learning frameworks, such as decision-tree induction (e.g., Quinlan, 1986) and Bayesian networks, but this simple scheme is appropriate given COBWEB’s assumption of conditional attribute independence given the category.

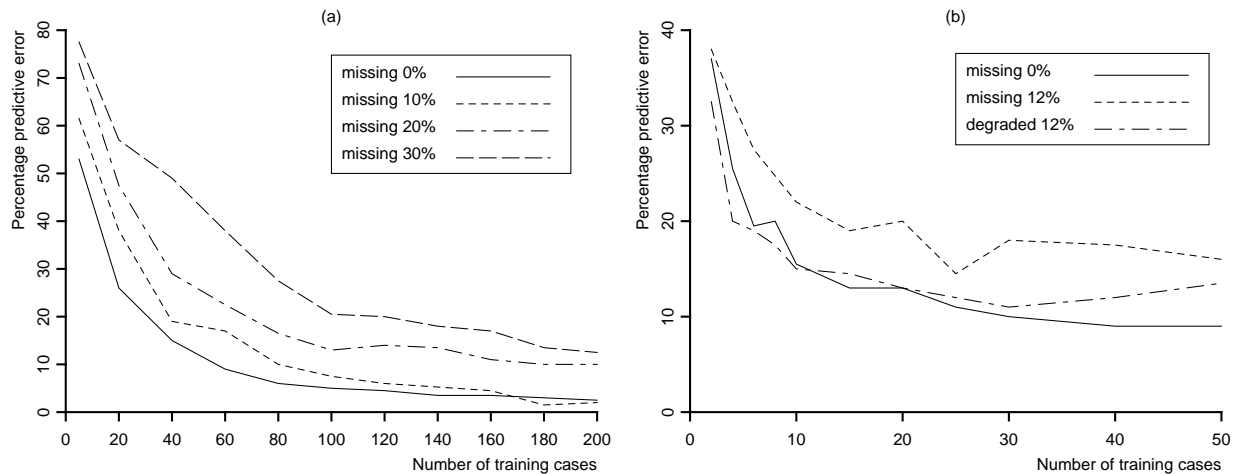


Figure 4. Learning curves for several levels of missing information on (a) a synthetic domain and (b) Congressional voting records (from Gennari, 1990).

and 12 attributes, six numeric and six nominal, with three attributes of each type being irrelevant with respect to the class. For this study, a filter removed attribute values from each training case with a specified probability  $p$ , replacing them with an identifier for ‘missing’. Varying the parameter  $p$  produced a number of domains that were identical except for the number of omitted values. From each domain, Gennari ran the system on ten different training sets and measured classification error on a common separate test set.

Figure 4 (a) shows COBWEB’s learning curves for this domain with 0%, 10%, 20%, and 30% of the attributes omitted. As expected, error increases when fewer attributes are available in the training cases, and the learning curves for higher omission levels are slower, but this degradation occurs in a graceful manner. However, the overall effect is somewhat stronger than Gennari predicted. If one removes 20% of the attribute values from training data, we would expect that 20% additional training would make up the difference. Instead, we see the curves generated with missing information needing several times the number of training cases to reach comparable levels of accuracy. Gennari attributes this effect to a higher chance that COBWEB will find local optima when enough features are absent. The fact that the learning curve for 5% (not shown) and 10% omissions are nearly identical to the curve for no omissions seems consistent with this interpretation.

In a similar study with the congressional voting data, Gennari collected a subset of the instances with missing attributes that, on average, had 12% missing values. He also took a subset of completely specified instances and randomly filtered values such that this “degraded” set also had 12% missing information. Figure 4 (b) shows the learning curves for this study. We see an interesting contrast between training on instances with missing data and training on data that was degraded. The former performs significantly worse than COBWEB with no missing data, while the degraded condition, with the same percentage of missing values, fares about as well as the no missing condition. Certainly, the system is robust with respect to certain forms of data loss, but missing information remains an open question that merits additional attention.

## 4. Extensions to COBWEB

When first developed, COBWEB appeared to offer many advantages over other approaches to learning available at the time. The system combined the hierarchical structure of decision trees with a clean probabilistic semantics, and it constructed its concept hierarchies in an incremental and unsupervised manner. However, experience with COBWEB suggested both representational and algorithmic limitations, which led to many attempts to extend the basic approach. In this section, we report on three such efforts in detail, then briefly review other research within the framework.

### 4.1 Minimizing effects of noise and training order

Despite its many attractions, our direct experience with COBWEB suggested that often it constructed hierarchies that did not reflect the underlying class structure of the domain. This behavior was especially apparent with noisy data and with certain training orders. In response, we developed ARACHNE, an unsupervised system that seeks to construct well-formed hierarchies of probabilistic concepts while maintaining high predictive accuracy. The algorithm bears many similarities to COBWEB, but employs different criteria for tree formation and uses alternative restructuring operators. In this section, we review ARACHNE and present some experimental results on its behavior.

#### 4.1.1 THE ARACHNE SYSTEM

ARACHNE assumes the same representation and organization of knowledge as COBWEB, but differs somewhat in its learning algorithm, classification mechanism, and evaluation function. Like its predecessor, the system includes two operators for restructuring its probabilistic concept hierarchy, one that merges children and another that promotes a child to the same level as its parent. Unlike COBWEB, the system applies these operators according to local constraints that can be tested efficiently but that should produce better structured hierarchies of probabilistic concepts. The basic learning activity involves sorting a training case downward through memory, with its values being used to update the probability distributions for the node subsuming it. However, this sorting process occurs only as the byproduct of the system's merging operator, which can lead to other restructuring along the way.

Each time such restructuring alters the children of some node  $N$ , the system checks two constraints designed to ensure a well-formed hierarchy. ARACHNE first checks each child  $C$  of  $N$  in turn to make sure the child is 'vertically well placed'; that is, it is more similar to its parent than to its grandparent. If  $C$  violates this condition, ARACHNE promotes  $C$ , removing it as a child of  $N$  and making it a child of  $N$ 's parent.<sup>5</sup> This ensures that no children of  $N$  are more similar to their grandparent than to their parent. Thus, storing a new instance as a child of a concept can cause a sibling instance or concept to 'bubble up' to a higher location in memory.

The system's next step involves checking each child of  $N$  to make sure it is 'horizontally well placed', that is, it has equal or greater similarity to its parent than to any sibling. If the most

---

5. Actually, the system must recheck each constraint after applying the promote operator, since this changes the description of the parent node  $N$ .

similar pair of children are more like each other than either is to  $N$ , the system merges this pair by replacing these siblings with a new node that is their probabilistic average and taking the union of their children as its children. ARACHNE then recursively considers merging this new node's children. In some cases, this leads to recreation of the original siblings at a lower level in the hierarchy; in other cases, it produces further reorganizations in the subhierarchy. In particular, if the original instance is merged with an existing concept, recursive calls of the merge operator can effectively sort it down through memory.

Once it has merged two nodes at a given level, ARACHNE checks the remaining nodes for satisfaction of horizontal well placement. If it finds another pair of nodes that violate this constraint, it merges them as well, then repeats this process until all nodes at this level satisfy the constraint. In this way, a single new instance can cause the system to merge successively many of the nodes previously stored at a given level, including pairs of nodes dissimilar from it.

For instance, suppose ARACHNE had stored four instances of cats under a common parent, and a dog instance is added (through merging from above). Here the system would first merge the two most similar cats, then merge a third into the resulting node, and finally add the fourth. The result would be two concepts, one representing the abstraction of four cat instances and the other based on a single dog. This iterative merging process differs from that used in COBWEB, which merges nodes only when they are similar to a new instance. Thus, we expect ARACHNE will create well-structured trees regardless of the order in which instances are presented.

ARACHNE's evaluation function also differs somewhat from that used in COBWEB. The two constraints require some measure of similarity between pairs of nodes and/or instances, rather than a metric over an entire partition. To this end, the system calculates the shared area under the probability distributions for each attribute, averaged over all attributes. This usage is akin to that found in Hadzikadic and Yun's (1989) INC system, which also uses a similarity function to guide the formation of probabilistic concept hierarchies. In addition, ARACHNE uses its similarity measure to determine the depth to which it should sort an instance, halting whenever the best similarity score at the next level is no better than that at the current level.

ARACHNE uses the same metric and essentially the same control structure for prediction that it uses in learning. The system sorts an instance down the hierarchy in accordance with its constraints, except that no promotion is allowed and only merges that involve the instance are executed. Thus an instance sorts to the class at which it would ordinarily become a disjunct, and a prediction is made from the last node to which it sorted. ARACHNE also includes a simple 'recognition' criterion to foster prediction from internal nodes and thus avoid overfitting. As it sorts an instance through memory, the system makes a prediction from an internal node if its modal values perfectly match all the values of the instance.

#### 4.1.2 EXPERIMENTAL EVALUATION OF ARACHNE

Our initial experiments were designed to show that ARACHNE is competitive with COBWEB in terms of predictive ability on natural domains. For this purpose, we followed Gennari's scheme of including the class attribute as an additional feature that the systems used for prediction but not in clustering.

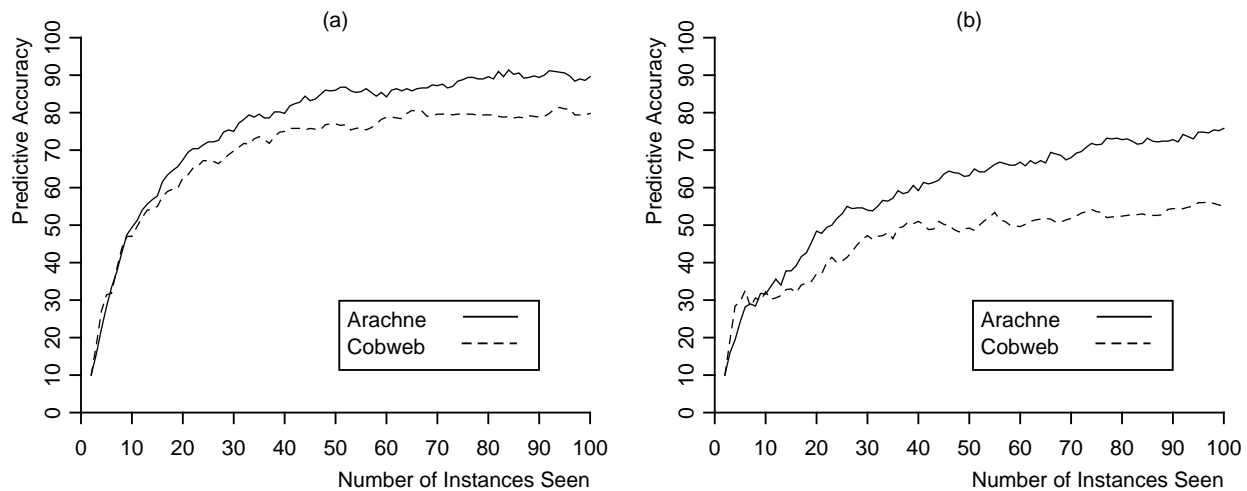


Figure 5. Learning curves for ARACHNE and COBWEB on an artificial domain with (a) low and (b) high attribute noise, using predictive accuracy as a performance measure.

As shown in Figure 1, averaged results on the voting records and soybean domains revealed nearly identical learning curves, with the two systems improving at almost the same rate and reaching the same asymptotic accuracy. Although such studies show relevance to real-world problems, artificial domains provide better understanding of the reasons for an algorithm’s behavior. For example, we predicted that ARACHNE’s performance and learning algorithms would be more robust on noisy data, in terms of both accuracy and tree structure, since its more powerful reorganization operators should make it less subject to constructing overly deep concept hierarchies. We defined tree quality as the number of *well-placed instances*, that is, singleton nodes that are descendants of a target concept and that match 50% or more of the modal attribute values of the target concept. A high percentage of well-placed instances in a tree implies many accurate concepts.

To test this hypothesis, we designed artificial data sets that contained instances with four attributes, each of which could take on ten discrete values, giving ten distinct categories. For each concept, each attribute had a prototypical value but could take on other ‘noise’ values at some specified probability. In the low-noise data set, the modal value for each attribute occurred with probability 0.7, while three ‘noise’ values occurred with probability 0.1. In the noisier data set, the modal value for each attribute occurred with probability 0.5, while five noise values occurred with probability 0.1. Because a noise value appearing in one class was the modal value of some other class, the class descriptions overlapped to some extent. About 24% of the low-noise instances and only about 6% of the high-noise instances should conform perfectly to the modal class description, with the remainder being noisy variants.

We carried out ten runs at both noise levels, presenting COBWEB and ARACHNE with 100 training examples in each case. Figure 5 shows the learning curves for predictive accuracy at each level. As expected, increasing noise produced larger differences between the two systems. For low noise, ARACHNE asymptoted at 90% accuracy and COBWEB reaches 80%, whereas for high noise their asymptotes dropped to 76% and 56%, respectively. We observed a similar interaction regarding tree

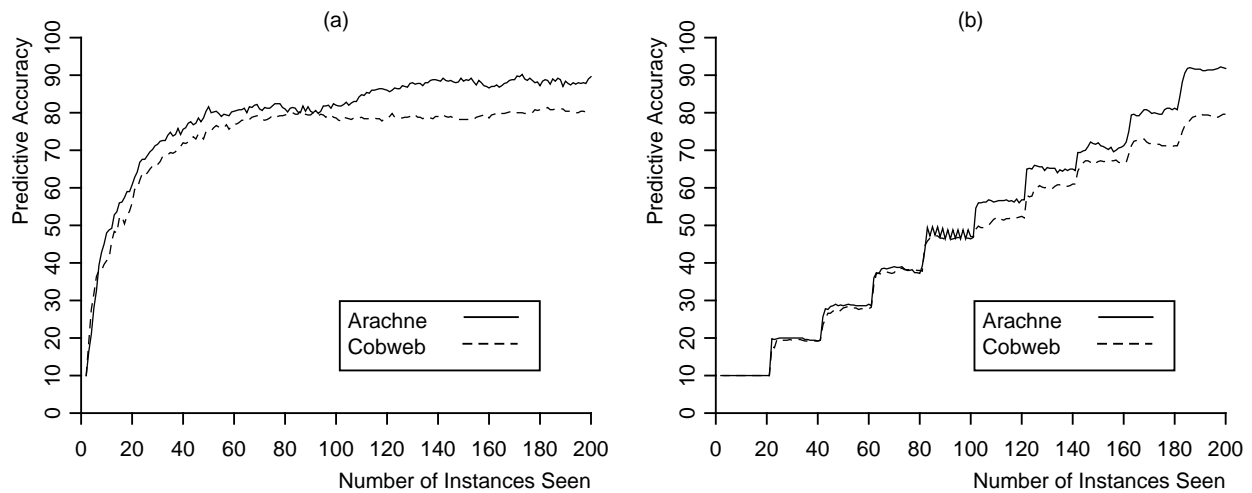


Figure 6. Improvement in predictive accuracy for ARACHNE and COBWEB on artificial data with (a) random instance order and (b) misleading training sequences.

quality, with the older system being more affected by increased noise levels. Surprisingly, COBWEB’s tree quality in the low-noise condition (82% well-placed nodes) was higher than for ARACHNE (74% well-placed nodes), but this order reversed for the high-noise condition, with COBWEB (37% well-placed nodes) faring worse than ARACHNE (52% well-placed nodes). Differences in accuracy were significant at the .001 level, whereas those in tree quality were significant only at the .025 level, but both measures were consistent with the predicted interaction effect.

Another one of our concerns in designing ARACHNE was stability with respect to different orders of training instances. The system’s operators for merging and promotion should enable recovery from nonrepresentative training orders, leading to the prediction that COBWEB will suffer more from order effects than ARACHNE with respect to tree quality but not accuracy. Our experience with COBWEB suggested it has difficulty when every member of a class is presented at once, followed by every member of a new class, and so forth. Thus, we tested our hypothesis by presenting both systems with ten random orderings and ten “bad” orderings of 200 training instances from the low-noise data set described earlier. The bad orderings were strictly ordered by class, so the systems saw 20 examples of each class in turn.

The results, shown in Figure 6, indicate that instance order did not affect asymptotic accuracy for either system, although naturally learning was slower for the bad ordering, since they did not see a representative of the final class until the 181st instance. At this stage, random and bad orderings produce hierarchies capable of analogous predictive accuracy, approximately 90% for ARACHNE and 80% for COBWEB. However, tree quality differs significantly in the two situations. Both systems locate most or all of the concepts at some level, but COBWEB is vulnerable to misplaced instances with the pathological ordering. Whereas ARACHNE arrives at 88% well-placed nodes with the bad ordering, and a similar 86% with random ordering, COBWEB averages only 74% well-placed nodes when learning from the bad ordering, compared to 84% for the random ordering. Differences in predictive accuracy were significant for both conditions at the .001 level. Differences in tree quality

between the two systems were not significant for random orderings of training cases, but were significant at the .001 level for bad instance orderings.

In summary, we found that ARACHNE's alternative control structure produced the same predictive accuracy as COBWEB on two natural domains, but we observed significant differences in both accuracy and tree quality using synthetic data. In particular, we found that COBWEB was more sensitive than ARACHNE to noisy training and test cases, and that the quality of COBWEB's trees suffers from misleading orders of training instances, while new system's tree structure is relatively unaffected. We also saw additional evidence that predictive accuracy is poorly correlated with tree quality. These results suggest that ARACHNE embodies a promising approach to the construction of probabilistic concept hierarchies and deserves additional study.

## 4.2 Learning overlapping concepts

Another limitation of COBWEB lies in its assumption that regularities in a domain can be summarized by a single hierarchy of probabilistic concepts. One can easily imagine domains in which two or more orthogonal organizations of instances, involving different sets of attributes, are possible. If COBWEB uses one of these taxonomies to structure its experience for one attribute set, it cannot form generalizations and make useful predictions about the other attributes. Enough training cases should let the system deal with such domains, but an excessive reliance on data is a drawback of any inductive system. In response, Martin and Billman (1994) developed TWILIX, a system that extends COBWEB to deal with domains that have overlapping category structure.

### 4.2.1 THE TWILIX SYSTEM

Like the ARACHNE system, TWILIX shares many features with its predecessor in terms of representation, organization, performance, and learning. As in COBWEB, each concept  $C$  is encoded using a conditional probability distribution for each attribute given  $C$ , along with a base rate  $P(C)$  for the concept itself.<sup>6</sup> The system also organizes these probabilistic categories in a concept hierarchy, with higher nodes subsuming those below them. However, the immediate children of each node in the TWILIX hierarchy are not individual categories, but rather *sets* of categories. Each such 'conflict set' represents a distinct way of partitioning instances, typically emphasizing different attributes and thus supporting overlapping concepts. Each such set contains nodes for more specific probabilistic categories, which can themselves have sets of children, and so forth.

Naturally, TWILIX uses this extended memory organization to process an instance  $I$  somewhat differently than COBWEB. Within a conflict set, the system always assigns  $I$  to exactly one of the mutually exclusive categories, just as in COBWEB, updating the probability distributions that describe that concept. Across conflict sets, TWILIX finds the best set by tentatively assigning  $I$  to the concepts within each set, then repeats this process to find the next best set, and so forth. The result is that the system can assign the case to more than one conflict set, but never to more than one concept within each set. Moreover, just as COBWEB sometimes decides to create a new

---

6. TWILIX differs slightly from COBWEB in incorporating uniform prior probabilities for each distribution, rather than estimating distributions solely from the training data.

category at the current level based on a distinctive training case, so TWILIX sometimes decides to create an entirely new conflict set. The system does not include COBWEB’s split and merge operators, since it should be less sensitive to training order than its predecessor.

Like the earlier system, TWILIX relies on evaluation functions over instance partitions to direct the classification and learning process. In fact, the function for assigning cases to a category within a conflict set, which Martin and Billman call *set utility*, differs only in minor ways from COBWEB’s category utility. However, the system also includes a function  $U$  that evaluates different sets  $\theta$  of conflict sets. This can be stated as

$$U(\theta) = \left[ \prod_{l=1}^M SU(S_l) \right]^{\frac{1}{M}}, \quad (4)$$

where  $SU$  is the set utility for cluster  $S_l$  and  $M$  is the number of conflict sets in  $\theta$ . TWILIX applies this function to candidates sets of sets, selecting the one that gives the highest score. However, note that the exponent  $1/M$  gives a strong bias against unnecessary conflict sets, which should lead to a single hierarchy in domains where that is appropriate.

#### 4.2.2 EXPERIMENTAL STUDIES OF TWILIX

Since TWILIX was specifically designed to learn overlapping concepts, Martin (1992) tested the system using a data set on Pittsburgh bridges that he felt had this characteristic. This domain includes descriptions for 108 bridges that were built in the Pittsburgh area between 1818 and 1986. Each bridge is described by twelve nominal attributes, seven representing design specifications or requirements, and the other five capturing the design descriptors. Martin also tested a version of COBWEB that omitted the merge and split operators to provide a closer basis for comparison.

The experiment involved running each system five separate times, training on nine successive training blocks and testing on one test block. Each block consisted of ten instances, made up from 100 instances selected randomly from the 108 total cases available. After each block of ten training instances, both systems were run without learning on the test block to obtain learning curves. The performance measure was the average accuracy on predicting each of the twelve attributes given the other eleven.

Figure 7 (a) shows the resulting learning curves (averaged over five runs) for the two systems. The graphs reveal that TWILIX has a significant advantage over COBWEB in this condition ( $p < 0.001$ ). However, Martin conducted a tandem experiment in which the performance task was to predict all five design descriptors given only the specifications. As Figure 7 (b) indicates, there was no difference between the systems in this setting. Furthermore, for this performance task, COBWEB’s overall performance increased (i.e., generally higher accuracy overall), but the overall accuracy for TWILIX was actually lower. Martin speculated that the seven specification attributes in this domain might be too impoverished to make accurate predictions, yet COBWEB fared better in this condition than when only predicting a single attribute. Evidence from other empirical studies using synthetic domains generally indicates that TWILIX is more robust than its ancestor, but there seems room for additional work to identify the relative contributions of its extensions to the COBWEB framework.

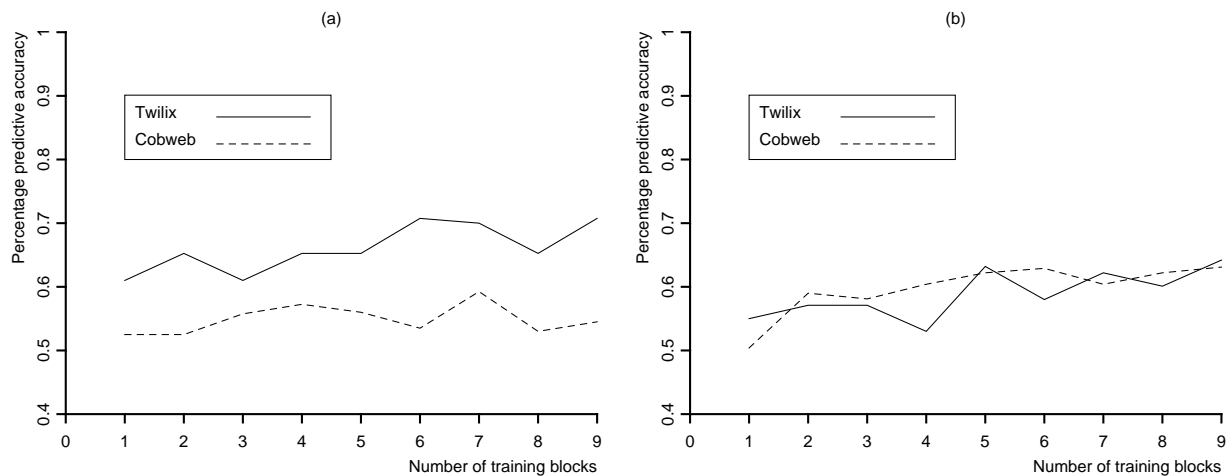


Figure 7. Average proportion of correct predictions (a) for one attribute given the other eleven, and (b) for the five design descriptors given the seven requirement specifications (from Martin, 1992).

### 4.3 Concept formation in temporal domains

Many concepts in the real world describe events that occur over time with certain indicative characteristics. We can contrast such temporal concepts with those addressed to this point, which describe static objects or states of an environment. For example, motor skills correspond to a temporally structured domain in which it seems natural to use a *sequence* to characterize and discriminate among concepts. However, such domains introduce challenges for concept formation systems like COBWEB, which have no direct representation of sequence. In response, Iba (1991a, 1991b) developed OXBOW, an extension to COBWEB that forms concepts in temporal domains. Here we focus on the system's application to learning movement concepts, though the underlying approach is more general.

#### 4.3.1 DESIGN OF OXBOW

OXBOW represents movements as sequences of descriptions with temporal relations among them. The system employs a movement parser, which takes a dense sequence of attribute-value descriptions as input, to create a sparse sequence based on zero crossings in velocity and acceleration. At such events, the parser creates *states* containing the positions, angles, and velocities (e.g., for the joints of a limb), as well as the time (relative to the start of the movement) associated with the zero crossing. This sparse representation is sufficient to capture and summarize the original movement with very low error.

Two issues arise for representing movements in a COBWEB-like concept hierarchy. First, the movement is a sequence of states instead of single set of attribute values. But more significantly, movements have variable numbers of states according to their complexity (number of zero crossings). In response, OXBOW represents a single movement concept using a probabilistic hierarchy of states.

The top-level partition of this hierarchy is organized with respect to time only, and the nodes at this level are ordered by time to yield the state sequence of the movement.

Movement concepts are organized in a traditional probabilistic ‘movement’ hierarchy, each node of which points to an entire ‘state’ hierarchy that describes the temporal structure of the movement category. Thus, a movement hierarchy of baseball pitches might have two top-level concepts corresponding to sidearm and overhand throws, and the overhand concept might have three children: fast-ball, curve-ball, and fork-ball. Each of these concepts points to a state hierarchy in which the top level consists of an ordered AND tree that represents the sequence of states for the given movement. To continue our example, the sequence might include states for the wind-up, initial forward motion, wrist-snap, release, and follow-through.

The recognition process for a newly observed movement is similar to COBWEB’s classification mechanism. The new motion is parsed into a sequence of states, which is sorted down through the movement hierarchy. At each level, OXBOW applies and evaluates the four COBWEB operators from Section 2.2. On termination, the system returns, as the retrieved category, the most recent node into which the new instance was incorporated. This category can be used to evaluate the goodness or accuracy of the classification process.

On the surface, OXBOW’s learning method is quite similar to COBWEB’s, but it introduces a significant variation to deal with structured objects having temporal components. Instead of simply updating attribute-value counts, the system must incorporate a new state sequence into an existing movement concept. This involves sequentially incorporating each state from the movement into the state hierarchy rooted at the movement concept. For each state, OXBOW employs the basic category utility function. However, time is the only attribute considered at the first level; in this way, the partition structure is organized by the temporal aspects of the movement. At subsequent levels, time is ignored and the state features are used in the category utility calculation.

This scheme for incorporating movements carries out a partial match between the sequence of states that comprise the newly observed movement and the sequence in the concept (as defined by the time-ordered top level). OXBOW views the top level of the state hierarchy as a ‘part-of’ structure. Not surprisingly, some modifications of category utility are necessary to accommodate this structured representation.

Since OXBOW works with continuous attributes, it uses the continuous version of category utility from Equation 2. However, this expression assumes that every class consists of a simple set of attributes, so we must extend this to consider classes with a set of components or, in this case, state descriptions. Because the number of states is not the same for all movement instances, the information in each component is weighted by the probability of that component. For OXBOW, the evaluation function over movements is

$$\frac{1}{K} \left[ \sum_k P(C_k) \sum_j P(S_{kj}) \sum_i \frac{1}{\sigma_{kji}} - \sum_m P(S_{pm}) \sum_i \frac{1}{\sigma_{pmi}} \right], \quad (5)$$

where  $P(S_{kj})$  is the probability of the  $j$ th state description in class  $C_k$ , which specifies the proportion of all the state descriptions from schema instances of node  $C_k$  that have been classified at state

description  $S_{kj}$ . The probability  $P(S_{pm})$  is defined in a similar manner for the  $m$ th state description in the parent of the current partition.

We have extensively tested this approach to unsupervised learning over movement categories has been at both the movement and state levels (Iba, 1991) in both natural and artificial domains. We have also evaluated OXBOW on other temporal domains, including the recognition of cursive letters and events in telemetry data from the space shuttle. Here we present one study that demonstrates the temporal extension to COBWEB and highlights a variation on the performance task.

#### 4.3.2 PREDICTING UNSEEN MOVEMENT

Most work in unsupervised learning has resorted to evaluating a method's utility in supervised learning contexts. We could do this with OXBOW, but we also have an opportunity to measure the error between the prototype motion, which is stored at the indexed concept, and the observed motion. We measure the error as the absolute difference in joint positions between the two movements at each corresponding point in time, which we then average over all time steps. An even more challenging performance measure requires OXBOW to predict the continuation of a partially observed movement. That is, given an initial glimpse of a movement, predict its continuation over time. If we ignore issues of learning, then varying the observed percentage of a test movement provides a method for adjusting the difficulty of OXBOW's retrieval task, thereby allowing a more direct assessment of its contribution to error.

For a number of our empirical tests, we constructed an artificial movement domain that consisted of four movement types controlling a two-jointed arm. Instances of each movement type were generated by a distinct schema, the elements of which were perturbed according to a variability (noise) level. The four movement types were equally likely to occur. OXBOW learned from a series of observed movements and was then tested on a set of new movements (with learning turned off). We conducted these tests in a simulated environment with an arm that had a maximum possible error of 300 units.

To explore OXBOW's ability to predict future movement from partial observation, we ran a series of 20 sessions, each one consisting of 30 randomly generated training movements. The test set consisted of the noise-free schemas, one for each movement type. When testing, we presented an initial portion of the prototypical movement and then measured error over the remaining unobserved movement. Note that complete movements were given during training and only when evaluating system performance did we limit the extent of the observed prototype. We can compare errors among different lengths of predicted movements because we average the total error by the number of time slices compared during prediction. Any differences in errors can be attributed to classification problems during retrieval, because the knowledge base is the same for each level of observation at a given point in training.

This formulation of the task suggests a prediction: as less of the movement is observed, classification should become more difficult and mistakes should lead to greater measured error. Simply stated, the more the system sees of a movement, the more it should know about what will happen

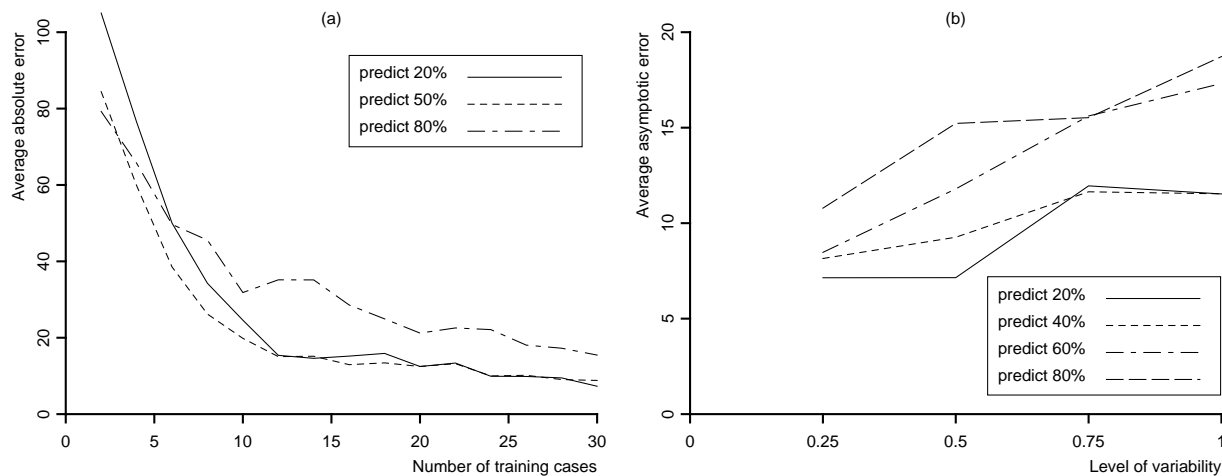


Figure 8. (a) Learning curves showing error for three levels of partial observation, and (b) asymptotic error rates for four levels of domain variability and four levels of partial observation.

next. Figure 8(a) shows the learning curves from an experiment in which we varied the portion of the movement to be predicted, with results averaged over ten runs of 30 training instances each.

The figure shows that the errors are consistently the highest when OXBOW must predict 80% of the movement, except very early in training, when it has not yet seen all the movement types. However, there is little difference between predicting 50% of the movement and only 20%. This result suggests that the system is not severely affected by having less information available for classification, except in extreme cases. However, from previous experiments with OXBOW, we know that increased domain variability leads to higher asymptotic errors, presumably because greater noise makes it harder to construct high-quality generalizations, which in turn hinder classification. This idea suggests a predicted interaction: as training data becomes more variable, OXBOW should require larger portions of the test movement in order to prevent increased error.

To test this prediction, we ran the system in partial prediction mode while training it on data with different levels of variability. In a single experimental run for a given level of noise, we trained OXBOW on 60 observed movements, then tested on four levels (20%, 40%, 60%, and 80%) that must be predicted. As before, the system used the initial segment of the movement for classification. For each condition of noise and observation level, we averaged the results over 20 different training orders, to guard against order effects.

In this experiment, we were only interested in asymptotic error levels, having considered the affects of variability upon learning rate in an earlier study. Figure 8(b) shows the asymptotic error rates for the four noise levels as a function of the portion of each test movement to be predicted. The graph indicates similar asymptote levels for the 0.25 variability condition but a wide range of asymptotes for the 1.0 level. Separate analyses of variance for these two conditions revealed a statistically significant difference for variability equal to 1.0 ( $p < 0.001$ ) but none for the 0.25 condition ( $p > 0.1$ ), which seems to support the predicted interaction. However, an analysis of variance over all the data shows a significant main effect of the portion to be predicted, but no

significant interaction between the two factors.<sup>7</sup> Although the results did not strongly support our prediction, they indicate that OXBOW is somewhat robust with respect to noise; that is, the system is no more adversely affected by incomplete observations when learning from noisy data than when learning from more regular data.

Moreover, the above experiments held the learning system constant while varying the amount of information in the test movement, thus indicating the sensitivity of the classification process. The results suggest that OXBOW is not making misclassifications when given partial structures in the input. This provides supporting evidence that the increase in error, in conjunction with increased variability in the domains, is due to problems in the generalization process during incorporation of a new experience. Understanding and reducing these predictive errors remains an important topic for future research.

#### 4.4 Other variations on the COBWEB framework

In addition to the extensions described above, a number of researchers have developed clustering algorithms that derive either directly from COBWEB or that bear a striking resemblance to it. Each of these systems construct hierarchies of probabilistic concepts in an incremental manner, sorting training cases through memory and updating concept descriptions in the process. Many of these methods are covered in Fisher, Pazzani, and Langley (1991), but they deserve some comment here.

Pazzani (personal communication, 1990) and colleagues developed an extension to COBWEB that takes advantage of knowledge about which features one wants to predict. Their system retained the same control structure but generalized the measure of category utility so that it weights features according to the importance of predicting them. They argued that this approach should lead to better predictive accuracy than methods like COBWEB, which weights all features equally. One can view both unsupervised and supervised learning as special cases of this framework. Martin (1994) has reported a similar technique that also learns the weights on features from sample queries.

Another extension to COBWEB comes from Reich and Fenves (1991), who adapted the approach to problems in parametric design. Their BRIDGER system supports a variety of classification and prediction methods, including the ability to halt when all attributes that encode the design specifications have been met. The program also handles numeric attributes in a different manner than COBWEB and incorporates a mechanism for grouping nominal values into new features. Finally, BRIDGER mitigates order effects with a procedure that removes any node (and its descendents) if it contains ‘characteristic’ values (ones with a sufficiently high probability) that differ from those in its ancestors. Experiments using the system for synthesis of bridge designs, which involves predicting many attributes, gave encouraging results.

Anderson and Matessa (1991) describe a COBWEB-like algorithm that incrementally constructs a binary tree of probabilistic concepts. One difference lies in their performance element, which follows the Bayesian philosophy of using a weighted average over all categories to make predictions about test cases. Their system also invokes Dirichlet priors to initialize the probabilistic descriptions

---

7. If we consider only the high and low variability (i.e., remove the 0.5 and 0.75 noise levels), an analysis of variance indicates a significant interaction with  $p < 0.05$ .

for each category. Moreover, when deciding whether to incorporate a training case into existing children or to create a new category, it computes the probability that the case belongs to each child and that it belongs to a new category, rather than using an evaluation metric like category utility that operates over an entire partition. Their algorithm qualitatively fits numerous results on category learning from the psychological literature.

Thompson and Langley (1991) report on an explicit extension to COBWEB designed to carry out concept formation over structured representations, in which the values of attributes can themselves be component objects with relations among them. Their LABYRINTH algorithm calls on COBWEB recursively, in that acquired component concepts are used to describe and influence the acquisition of composite concepts. The system also introduced a new operator for ‘attribute generalization’ that replaced sets of values in a composite node’s description with a single value that resides above them in the component hierarchy. This required a new evaluation metric, related to category utility, that determined when to take such a restructuring step. Handa (1990) has extended this approach to take context into account during the classification of component objects.

Langley and Allen (1991, 1993) describe DÆDALUS, another extension to COBWEB that organizes plan knowledge in an effort to improve the efficiency of problem solving. Each node in the concept hierarchy contains a probabilistic summary of problems the system has solved previously, described in terms of the relational differences involved and the operators used in their solution. DÆDALUS includes a means-ends problem solver that sorts each new subproblem through memory, using a relational variant on category utility, to retrieve an appropriate operator. After solving a problem, the system stores each subproblem and its operator in the hierarchy, updating probabilities and changing retrieval on future problems. Experimental studies of DÆDALUS’s learning behavior showed reduction in search on both navigation and blocks-world tasks.

Yoo and Fisher (1991) take a different approach to concept formation over problem-solving experience. Their EXOR system associates with each conceptual node the abstract solution (stated as an AND tree) for a class of problems, rather than a single operator, with nodes lower in the hierarchy specifying more detailed solutions. EXOR uses a version of category utility to sort a new problem, based on its surface features, downward through memory. The system falls back on problem-reduction search to complete the solution if this process finds only a partial solution. The learning mechanism incorporates an explanation-based component that generalizes the solution stored at each node. Experiments with EXOR showed that its learning method improves search efficiency on algebra story problems.

Nor does this exhaust the variations and extensions on the COBWEB framework. Hadzikadic and Yun (1989) report a very similar algorithm for the incremental formation of concept hierarchies that differs in details like its evaluation metric. Kilander and Jansson (1993) describe a variant on COBWEB designed explicitly to deal with environments that change over time. Day (1992) has adapted the basic approach to learn constraints that produce more efficient search on complex scheduling tasks. Taken together, these intellectual descendants indicate that Fisher’s original approach to constructing hierarchies of probabilistic concepts lends itself naturally to a wide and interesting range of problems.

## 5. Relation to other research on unsupervised learning

Before closing, we should clarify the relation between the COBWEB framework and two other families of algorithms that learn probabilistic descriptions from unsupervised training data. We begin with another paradigm that also constructs probabilistic concept hierarchies, but in a quite different manner, then examine work within another representational formalism that relies on different assumptions than our own.

### 5.1 The AUTOCLASS family

Soon after the initial publication on COBWEB, Cheeseman et al. (1988) introduced AUTOCLASS, another clustering algorithm that has led to its own distinct family of systems. The two frameworks share some key assumptions, including the notion of describing each category in terms of probability distributions over its component attributes. Although the first system constructed only a one-level clustering, later versions like AUTOCLASS/3 (Cheeseman et al., 1991) generated multi-level descriptions, with more general categories summarizing their more specific children. Thus, in terms of its representation and organization, the AUTOCLASS family constitutes an approach to creating and using a probabilistic concept hierarchy.

Despite this clear similarity between the two frameworks, there are also some significant differences. Unlike COBWEB and its relatives, AUTOCLASS does not store training cases as terminal nodes in its concept hierarchies or, indeed, even assign cases definitively to one category or another. Rather, the system assigns each training case to *every* category with some probability, which in turn means that its values contribute only partially to each category description. Another representational difference is that AUTOCLASS stores not only the conditional probability distribution for each attribute given the category, but also the conditional covariance matrices. For numeric attributes, this means that decision regions are not limited to ellipsoids with axes parallel to those of the instance space, as in COBWEB. More generally, the system can represent relations that violate independence at a given level of the hierarchy, albeit at the price of more parameters.

The AUTOCLASS systems also differ from COBWEB and its kin in their performance and learning mechanisms. Cheeseman et al. take a strong Bayesian position on the classification of new instances, so their algorithms assign a test case to each category at each level with a certain probability, rather than to the most probable one. Predictions about missing attributes are then based on a weighted vote that takes into account each category's prediction and its probability given the instance. In this framework, the hierarchical organization of knowledge provides no computational benefits during the prediction process, but it retains advantages in understandability.

The clustering process in AUTOCLASS also has a quite different flavor. Rather than relying on incremental sorting, as in COBWEB, it incorporates a probabilistic variant of the nonincremental  $k$  means algorithm known as *expectation maximization*. This method initializes the probabilistic descriptions for each of  $k$  clusters randomly and uses these descriptions to compute the probability of each training case belonging to each cluster. It then uses these partial assignments to update the descriptions, reassigns each instance using the new descriptions, and continues the process until

no changes occur. After clustering is complete, AUTOCLASS removes those clusters that have only improbable assignments.<sup>8</sup> Despite their clear relationship, there exist no systematic experimental comparisons of COBWEB and AUTOCLASS or their close relatives.

## 5.2 Induction of Bayesian networks

Another probabilistic approach to unsupervised learning, one that has become quite popular in the years since COBWEB first appeared, involves Bayesian networks (e.g., Heckerman, 1995). This framework also organizes memory using nodes and links, but their meanings are quite different from those in a probabilistic concept hierarchy. Rather than corresponding to concepts, each node represents an observable or unobservable attribute, and a link from one node to another means the values of the former influence those of the latter. Like a probabilistic concept hierarchy, a Bayesian network specifies a probability density function over the instance space, but it achieves this through very different assumptions. Information about probability distributions reside not in category descriptions but in ‘conditional probability tables’ stored with each attribute. These specify the probability for each value of that attribute under each possible combination of values that influence it.

Research on learning in Bayesian networks has focused on two distinct issues. The first involves estimating the conditional probability tables from training data when the structure of the network is already known. When no attribute values are missing, this simply involves counting the number of times each combination of values occurs in the training set, so effort here has focused on the issue of learning from data with omitted values (e.g., Binder et al., 1997). A second body of work deals with learning a network’s structure from training data (e.g., Cooper & Herskovitz, 1992; Provan & Singh, 1995). Most algorithms carry out a greedy search through the space of Bayes net structures, starting with no links and adding the most probable link, given the data, on each step, then halting when reaching a local optimum.

A few researchers have also examined methods that introduce hidden attributes into learned Bayesian networks. One special form of this process introduces a single unobservable node that influences each observable attribute, which are themselves independent given this attribute. In fact, this task is equivalent to creating a set of probabilistic clusters, and the most common approach invokes the expectation maximization algorithm that is called at each level in AUTOCLASS. At first glance, this mapping suggests a close relationship between the induction of Bayesian networks and the formation of probabilistic concept hierarchies that has been our focus in this paper.

However, closer inspection reveals deep differences between the two frameworks. Granted, both paradigms have a clear probabilistic semantics that supports unsupervised induction for the prediction of missing attribute values. But Bayesian networks assume that their graphical structure and associated probability tables hold across the entire instance space, whereas probabilistic hierarchies explicitly partition this space into regions. Bayes nets can create such a partition by using a hidden attribute, but they cannot introduce partitions at different levels of abstraction, which

---

8. This clustering scheme requires the user to specify the number of clusters, but the ability to remove low-probability categories mitigates this reliance.

happens regularly in the hierarchical framework. On the other hand, each category in a probabilistic concept hierarchy assumes that attributes are conditionally independent, whereas Bayes nets can easily represent more complex situations.

Thus, the key difference between probabilistic concept hierarchies and Bayesian networks lies not in their learning algorithms. Indeed, we have seen that AUTOCLASS uses expectation maximization in a recursive manner to construct a probabilistic hierarchy, and one can imagine more gradual, incremental methods for creating Bayes nets. Rather, they differ mainly in their *representational bias*. Both frameworks can represent arbitrary target concepts, at least for discrete attributes, but each one finds it easier to describe some target functions than it does others. This means one can design synthetic domains on which either framework will learn more rapidly, achieving higher accuracy from fewer training cases, than the other. Of course, natural domains may or may not have similar characters, so it remains unclear which approach will fare best in practice. However, we anticipate that many real-world induction tasks will have hierarchical structure, and that methods for learning probabilistic concept hierarchies will prove useful on such problems.

## 6. Closing remarks

In this paper, we set out to address issues related to unsupervised learning in support of intelligent agents. This emphasis constrained the space of solutions in several ways and led us to focus attention on systems that represent concepts probabilistically and incorporate them incrementally into generalization hierarchies. Fisher's (1987) COBWEB, which satisfies these constraints, served as our prototype system.

We reviewed the COBWEB framework as it exists after several rational reconstructions, characterizing the system's representation and organization of knowledge, as well as its processes for recognition and learning. We summarized a selected sample of empirical studies that have established the framework's robustness along several dimensions, including variations in the evaluation function, search strategy, and domain characteristics. Although these studies were generally encouraging, they also suggested some limitations and directions for improvement.

In response, we examined three extensions to the framework in some detail. One system, ARACHNE, incorporated new restructuring operators designed to improve the quality of the learned concept hierarchy, whereas another extension, TWILIX, introduced a more sophisticated memory organization to improve prediction in domains with overlapping categories. A third system, OXBOW, extended the basic framework to handle domains with temporal structure. We also briefly described a variety of other systems that build on COBWEB in some fashion. In closing, we noted links to other approaches to unsupervised learning within a probabilistic framework.

In summary, the formation of probabilistic concept hierarchies has proven a fertile paradigm for the study of unsupervised learning. Although early work on the topic made many simplifying assumptions, researchers have since extended the framework in many directions, producing methods that are both more robust and that apply to a broader class of domains. Nevertheless, each extension has raised intriguing issues that deserve attention, pointing the way for additional research in this promising paradigm.

## References

- Anderson, J. R., & Matessa, M. (1991). An iterative Bayesian algorithm for categorization. In D. H. Fisher, M. J. Pazzani, & P. Langley (Eds.), *Concept formation: Knowledge and experience in unsupervised learning*. San Mateo, CA: Morgan Kaufmann.
- Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., & Freeman, D. (1988). AUTOCLASS: A Bayesian classification system. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 54–64). Ann Arbor, MI: Morgan Kaufmann.
- Cooper, G. F., & Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9, 309–347.
- Day, D. S. (1992). Acquiring search heuristics automatically for constraint-based scheduling and planning. *Proceedings of the First International Conference on AI Planning Systems* (pp. 45–51). College Park, MD: Morgan Kaufmann.
- Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2, 139–172.
- Fisher, D. H., & Langley, P. (1990). The structure and formation of natural categories. In G. H. Bower (Ed.), *The psychology of learning and motivation: Advances in Research and Theory* (Vol. 26). Cambridge, MA: Academic Press.
- Fisher, D. H., Pazzani, M. J., & Langley, P. (Eds.) (1991). *Concept formation: Knowledge and experience in unsupervised learning*. San Mateo, CA: Morgan Kaufmann.
- Gennari, J. H. (1990). *An experimental study of concept formation*. Doctoral dissertation, Department of Information & Computer Science, University of California, Irvine.
- Gennari, J. H., Langley, P., & Fisher, D. H. (1989). Models of incremental concept formation. *Artificial Intelligence*, 40, 11–61.
- Gluck, M., & Corter, J. (1985). Information, uncertainty and the utility of categories. *Proceedings of the Seventh Annual Conference of the Cognitive Science Society* (pp. 283–287). Irvine, CA: Lawrence Erlbaum.
- Hadzikadic, M., & Yun, D. (1989). Concept formation by incremental conceptual clustering. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 831–836). Detroit, MI: Morgan Kaufmann.
- Handa, K. (1990). CFIX: Concept formation by interaction of related objects. *Proceedings of the Pacific Rim International Conference on Artificial Intelligence*. Nagoya, Japan.
- Heckerman, D. (1995). *A tutorial on learning Bayesian networks* (Technical Report MSR-TR-95-06). Redmond, WA: Microsoft Research.
- Iba, W. (1991a). Learning to classify observed motor behavior. *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence* (pp. 732–738). Sydney: Morgan Kaufmann.
- Iba, W. (1991b). *Acquisition and improvement of human motor skills: Learning through observation and practice*. Doctoral dissertation, Department of Information & Computer Science, University of California, Irvine.

- Kilander, F., & Jansson, C. G. (1993). COBBIT: A control procedure for COBWEB in the presence of concept drift. *Proceedings of the 1993 European Conference on Machine Learning* (pp. 244–261). Vienna: Springer-Verlag.
- Langley, P., & Allen, J. A. (1991). Learning, memory, and search in planning. *Proceedings of the Thirteenth Conference of the Cognitive Science Society* (pp. 364–369). Chicago: Lawrence Erlbaum.
- Langley, P., & Allen, J. A. (1993). A unified framework for planning and learning. In S. Minton (Ed.), *Machine learning methods for planning and scheduling*. San Mateo: Morgan Kaufmann.
- Langley, P., Iba, W., & Thompson, K. (1992). An analysis of Bayesian classifiers. *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 223–228). San Jose: AAAI Press.
- Martin, J. D. (1992). *Direct and indirect transfer: Explorations in concept formation*. Doctoral dissertation, Department of Computer Science, Georgia Institute of Technology.
- Martin, J. D. (1994). Goal-directed clustering. *Proceedings of the AAAI Spring Symposium on Goal-Directed Learning*. Stanford, CA.
- Martin, J. D., & Billman, D. O. (1994). Acquiring and combining overlapping concepts. *Machine Learning*, 16, 121–155.
- McKusick, K. B., & Langley, P. (1991). Constraints on tree structure in concept formation. *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence* (pp. 810–816). Sydney: Morgan Kaufmann.
- McKusick, K. B., & Thompson, K. (1990). COBWEB/3: A portable implementation (Tech. Rep. No. FIA-90-6-18-2). Moffett Field, CA: NASA Ames Research Center, Artificial Intelligence Research Branch.
- Provan, G. M., & Singh, M. (1995). Learning Bayesian networks using feature selection. *Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics* (pp. 450–456). Fort Lauderdale, FL.
- Quinlan, J.R. (1986). Induction of decision trees. *Machine Learning*, 1, 81–106
- Reich, Y., & Fenves, S. J. (1991). The formation and use of abstract concepts in design. In D. H. Fisher, M. J. Pazzani, & P. Langley (Eds.), *Concept formation: Knowledge and experience in unsupervised learning*. San Francisco, CA: Morgan Kaufmann.
- Thompson, K., & Langley, P. (1991). Concept formation in structured domains. In D. H. Fisher, M. J. Pazzani, & P. Langley (Eds.), *Concept formation: Knowledge and experience in unsupervised learning*. San Mateo, CA: Morgan Kaufmann.
- Yoo, J., & Fisher, D. H. (1991). Concept formation over problem-solving experience. In D. H. Fisher, M. J. Pazzani, & P. Langley (Eds.), *Concept formation: Knowledge and experience in unsupervised learning*. San Francisco, CA: Morgan Kaufmann.