# Early Extensions to Cobweb

**Pat Langley**

Institute for the Study of
Learning and Expertise
Palo Alto, California

*Thirty-Eighth AAAI Conference on Artificial Intelligence*
*Vancouver, BC, February 20–27, 2024*

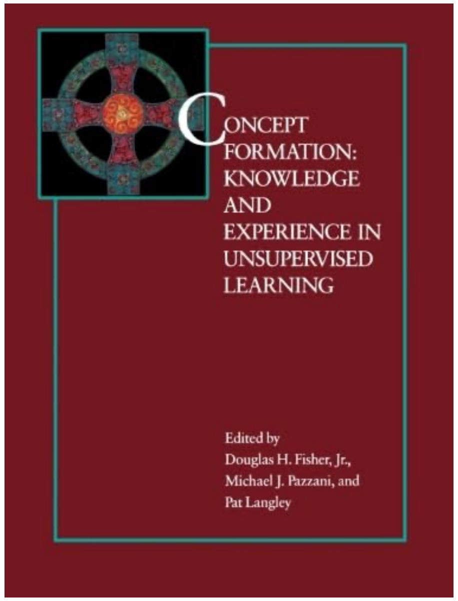
## Early Excitement about Cobweb

The late 1980s / early 1990s saw enthusiasm for Cobweb in the machine learning community, as it:

- Acquired conceptual knowledge in an incremental, unsupervised, and human-like manner;
- Combined ideas from decision trees, naive Bayes, and nearest neighbor in a unified framework;
- Rested on strong theoretical foundations but also came with solid empirical support.

Researchers built on Cobweb both to extend its abilities and apply it to AI problems.

Some early results appeared in an edited volume (Fisher, Pazzani, & Langley, 1991).


CONCEPT FORMATION: KNOWLEDGE AND EXPERIENCE IN UNSUPERVISED LEARNING
Edited by Douglas H. Fisher, Jr., Michael J. Pazzani, and Pat Langley

# Learning Continuous Concepts

Cobweb described cases and concepts using attributes in terms of discrete values, which works for some domains.

This is sufficient in many cases, but others are more naturally handled with continuous attributes:

- *E.g., measurements taken from medical devices*
- *E.g., dimensions of animals and physical objects*

Early decision-tree learners assumed symbolic features but were enhanced to handle numeric ones.

Gennari (1990) added similar abilities to his CLASSIT system, one of the first Cobweb extensions.

## CLASSIT: Representation / Organization

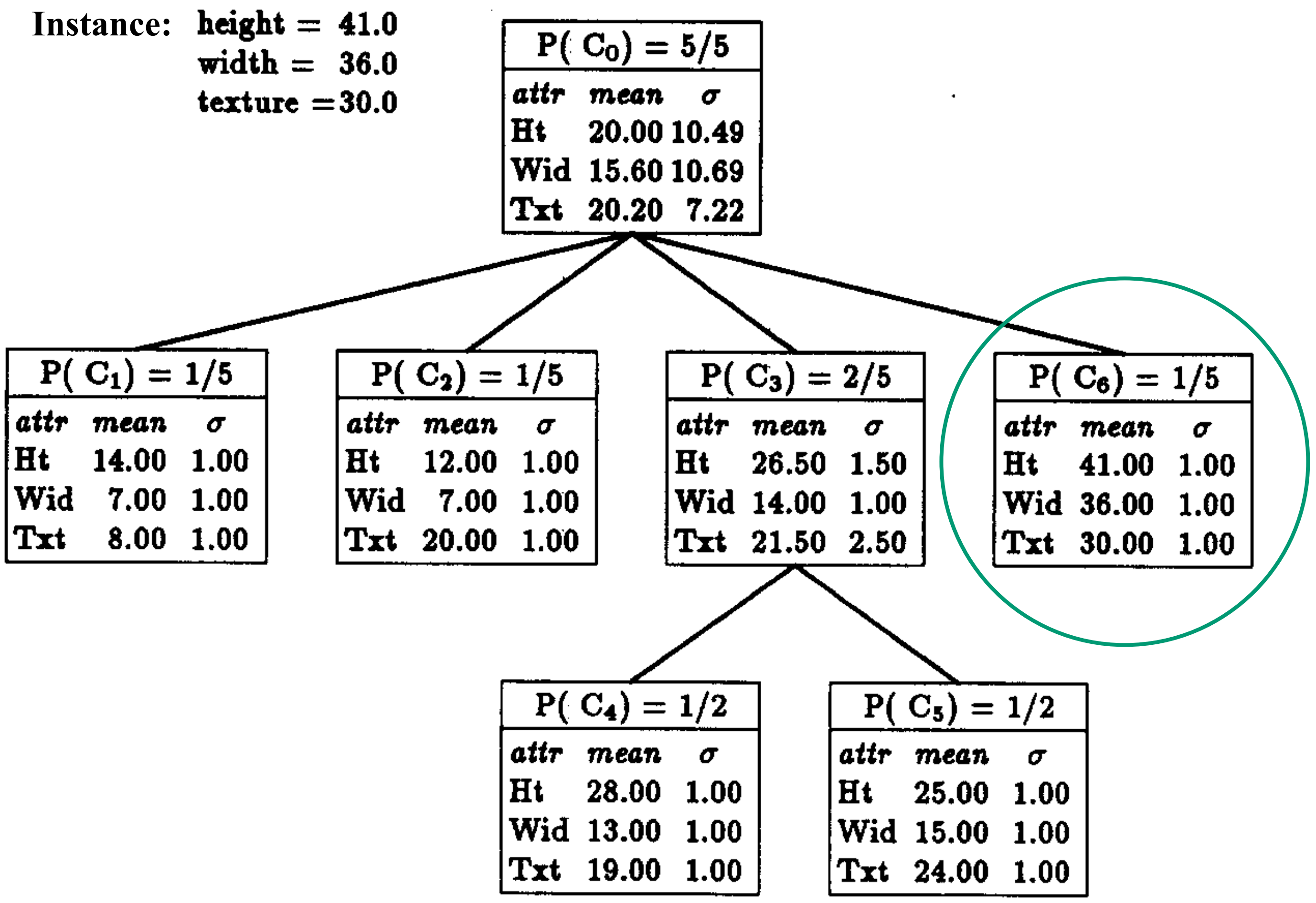
Like Cobweb, CLASSIT organizes concepts in a hierarchy, with each node summarizing its descendants.

However, its representation supports continuous attributes in addition to discrete ones:

- *Each instance specifies attributes with numeric values*
- *Each concept has a probabilistic description with:*
  - *Probability of occurrence given its parent*
  - *Continuous attributes with means and standard deviations*

The latter assumes values follow Gaussian distributions that are independent given the parent.

# CLASSIT: Representation / Organization



*CLASSIT also handles discrete attributes but emphasizes continuous ones.*

## CLASSIT: Performance / Learning

Like Cobweb, CLASSIT sorts a new case through the hierarchy, updating it along the way.

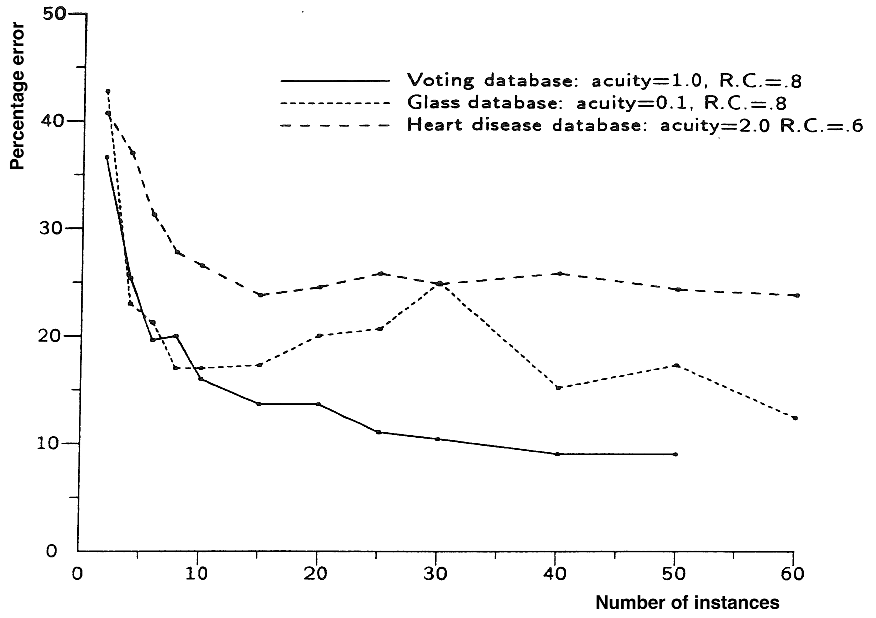
The key differences from its predecessor are related to the:

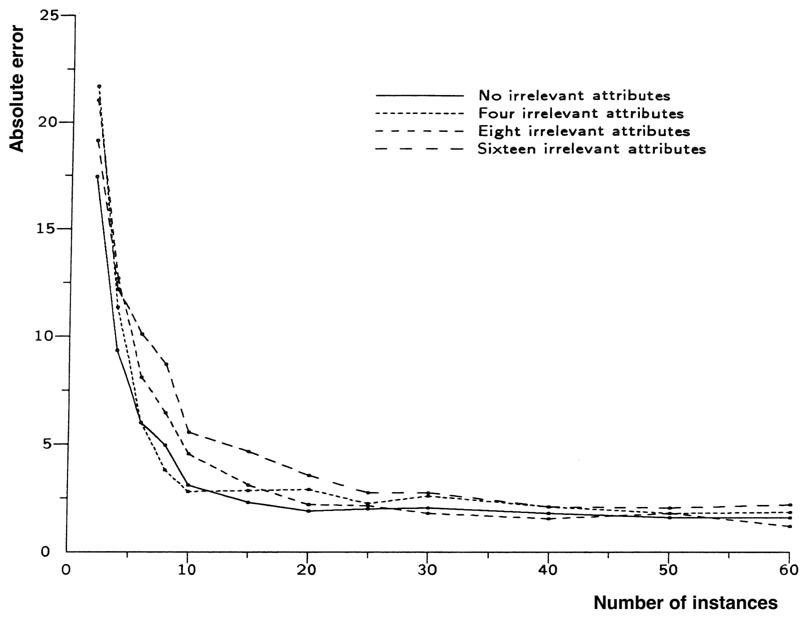
- Definition of category utility, which is $\dfrac{\sum_{k}^{K} P(C_k) \sum_{i}^{I} 1/\sigma_{ik} - \sum_{i}^{I} 1/\sigma_{ip}}{K}$ ,
  - *Where $\sigma_{ik}$ is the standard deviation of attribute i for node k*
- Updating scheme, which alters sums and sums of squares
  - *These are used to compute means and standard deviations*

CLASSIT also includes an ***acuity*** parameter in order to compute a minimum standard deviation for single-case concepts.

# CLASSIT: Experimental Results

Gennari (1990) reports empirical studies of CLASSIT's behavior on both natural and synthetic data sets.





Learning curves were rapid on natural domains and the system was robust to increases in noise and irrelevant attributes.

## Learning Overlapping Concepts

Cobweb assumed that a single hierarchy will suffice to describe regularities in a domain.

But we can identify settings in which multiple organizations of concepts are useful:

- *E.g., animals: mammals vs. birds, herbivores vs. carnivores*
- *E.g., meals: meat vs. vegetables, organic vs. nonorganic*

Using only one taxonomy to make predictions may not suffice in such situations.

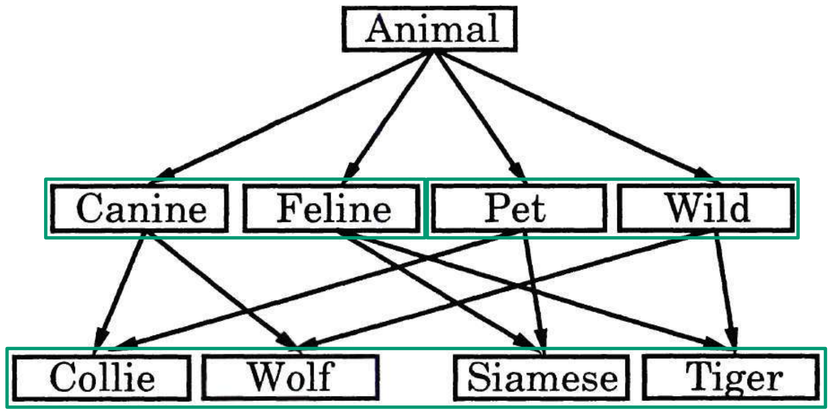
In response, Martin (1992) developed TWILIX, a system that forms ***overlapping*** concepts.

# TWILIX: Representation / Organization

Like Cobweb, TWILIX organizes conceptual knowledge in a hierarchy but it differs in that:

- Children of nodes are grouped into ***contrast sets***
  - Set members are ***mutually exclusive***
  - But not concepts in different sets
- Nodes may have ***multiple parents***
  - Memory is structured as a DAG
  - Not organized as a simple tree

From Martin (1992)



TWILIX concepts have the same probabilistic representation as found in Cobweb.

## TWILIX: Performance / Learning

As before, TWILIX sorts an instance through the hierarchy, updating probabilities and structure along the way.

The primary differences from Cobweb are that it can:

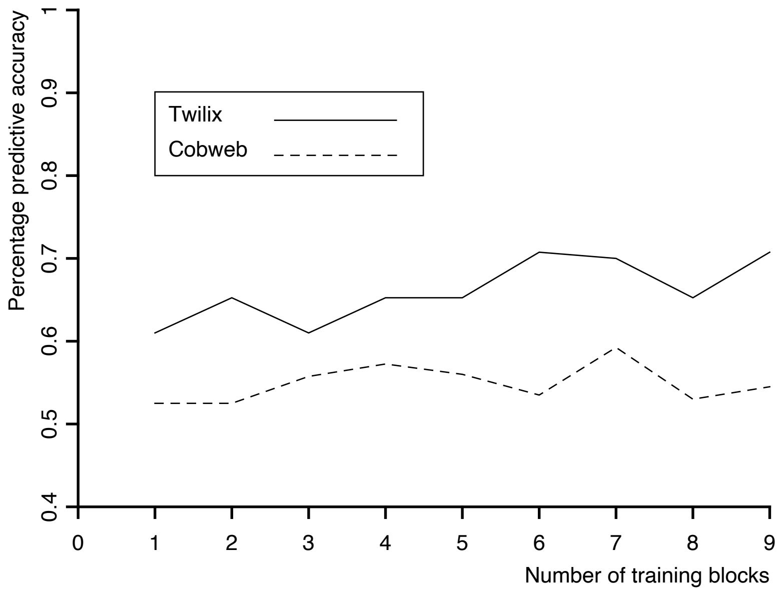
- Assign the case to more than one of a node's children
  - *But only to* ***one*** *child within each contrast set*
- For instances that are sufficiently novel:
  - *Create a new child in an* ***existing*** *contrast set*
  - *Create a child in an* ***entirely new*** *contrast set*

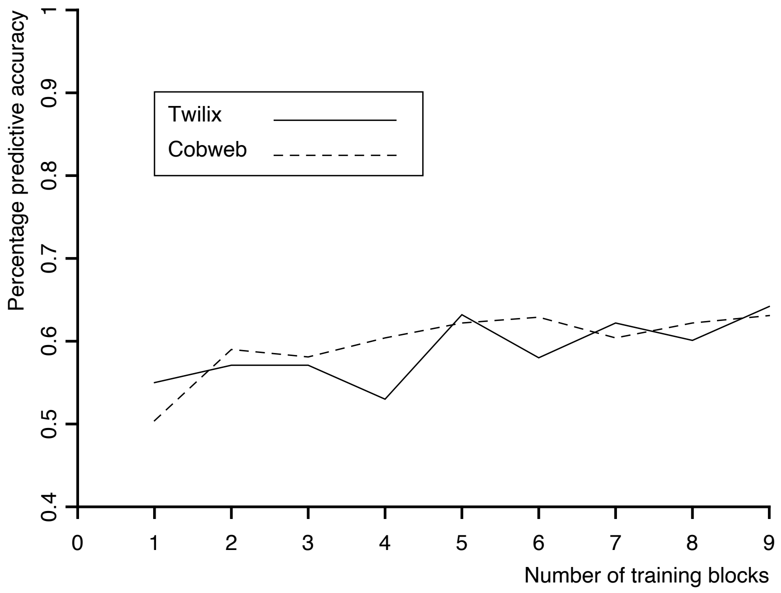
TWILIX combines predictions from different sets by using the most ***confident*** one for each attribute.

## TWILIX: Experimental Results

Martin (1992) compared TWILIX to a Cobweb variant on data for 108 Pittsburgh bridges with 12 discrete attributes.

When predicting one attribute, the new system was much better.





However, when asked to predict all the five design features, their performance was comparable.

## Learning Structured / Relational Concepts

Cobweb assumed an attribute-value notation for data and knowledge, which works well for *tabular* content.

However, many real-world concepts are ***structural*** and ***relational*** in character:

- *E.g., objects have components in a specific configuration*
- *E.g., places comprise elements in a particular layout*

Cobweb’s attribute-value formalism cannot encode such categories without additional representational power.

Thompson and Langley (1991) developed another system – Labyrinth – in response.

## Labyrinth: Representation / Organization

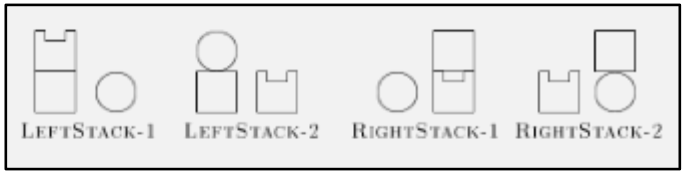
The Labyrinth system extended Cobweb to handle relational content by encoding:

- Instances as composite entities with:
  - *Components described as attribute-value pairs*
  - *Relations among these components*
- Concepts as probabilistic summaries for:
  - *Component categories (probabilities over attribute values)*
  - *Composite categories (probabilities over relations)*

Labyrinth stored separate hierarchies for its composite and component concepts.

# Labyrinth: Representation / Organization

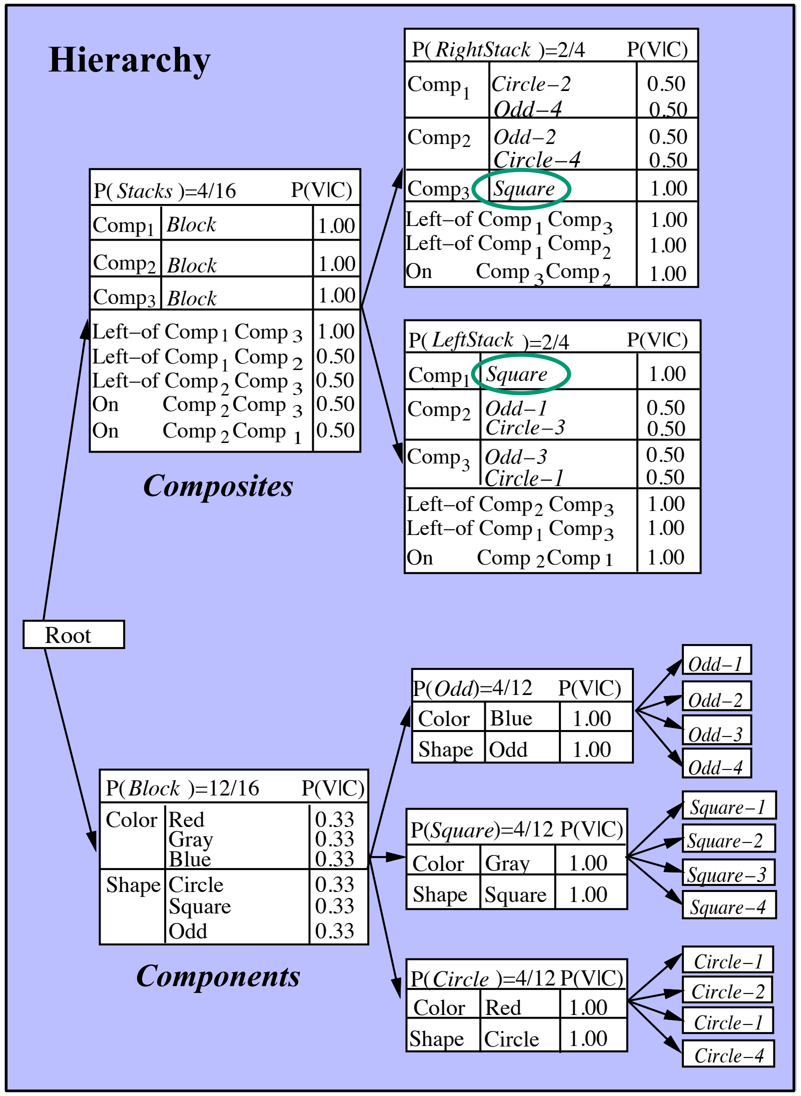
**Instance**



*Labyrinth's composite concepts included not only relations.*

*They also referred to nodes in the component hierarchy.*

*Some were **nonterminal** nodes describing generic categories.*



| P(*Stacks*)=4/16 | | P(V\|C) |
|---|---|---|
| $Comp_1$ | *Block* | 1.00 |
| $Comp_2$ | *Block* | 1.00 |
| $Comp_3$ | *Block* | 1.00 |
| Left–of | $Comp_1$ $Comp_3$ | 1.00 |
| Left–of | $Comp_1$ $Comp_2$ | 0.50 |
| Left–of | $Comp_2$ $Comp_3$ | 0.50 |
| On | $Comp_2$ $Comp_3$ | 0.50 |
| On | $Comp_2$ $Comp_1$ | 0.50 |

| P(*RightStack*)=2/4 | | P(V\|C) |
|---|---|---|
| $Comp_1$ | *Circle–2* | 0.50 |
| | *Odd–4* | 0.50 |
| $Comp_2$ | *Odd–2* | 0.50 |
| | *Circle–4* | 0.50 |
| $Comp_3$ | *Square* | 1.00 |
| Left–of | $Comp_1$ $Comp_3$ | 1.00 |
| Left–of | $Comp_1$ $Comp_2$ | 1.00 |
| On | $Comp_3$ $Comp_2$ | 1.00 |

| P(*LeftStack*)=2/4 | | P(V\|C) |
|---|---|---|
| $Comp_1$ | *Square* | 1.00 |
| $Comp_2$ | *Odd–1* | 0.50 |
| | *Circle–3* | 0.50 |
| $Comp_3$ | *Odd–3* | 0.50 |
| | *Circle–1* | 0.50 |
| Left–of | $Comp_2$ $Comp_3$ | 1.00 |
| Left–of | $Comp_1$ $Comp_3$ | 1.00 |
| On | $Comp_2$ $Comp_1$ | 1.00 |

| P(*Block*)=12/16 | | P(V\|C) |
|---|---|---|
| Color | Red | 0.33 |
| | Gray | 0.33 |
| | Blue | 0.33 |
| Shape | Circle | 0.33 |
| | Square | 0.33 |
| | Odd | 0.33 |

| P(*Odd*)=4/12 | | P(V\|C) |
|---|---|---|
| Color | Blue | 1.00 |
| Shape | Odd | 1.00 |

| P(*Square*)=4/12 | | P(V\|C) |
|---|---|---|
| Color | Gray | 1.00 |
| Shape | Square | 1.00 |

| P(*Circle*)=4/12 | | P(V\|C) |
|---|---|---|
| Color | Red | 1.00 |
| Shape | Circle | 1.00 |

## Labyrinth: Performance / Learning

Like Cobweb, Labyrinth sorts each instance through memory, updating the hierarchy along the way.

To handle structured and relational instances, the system:

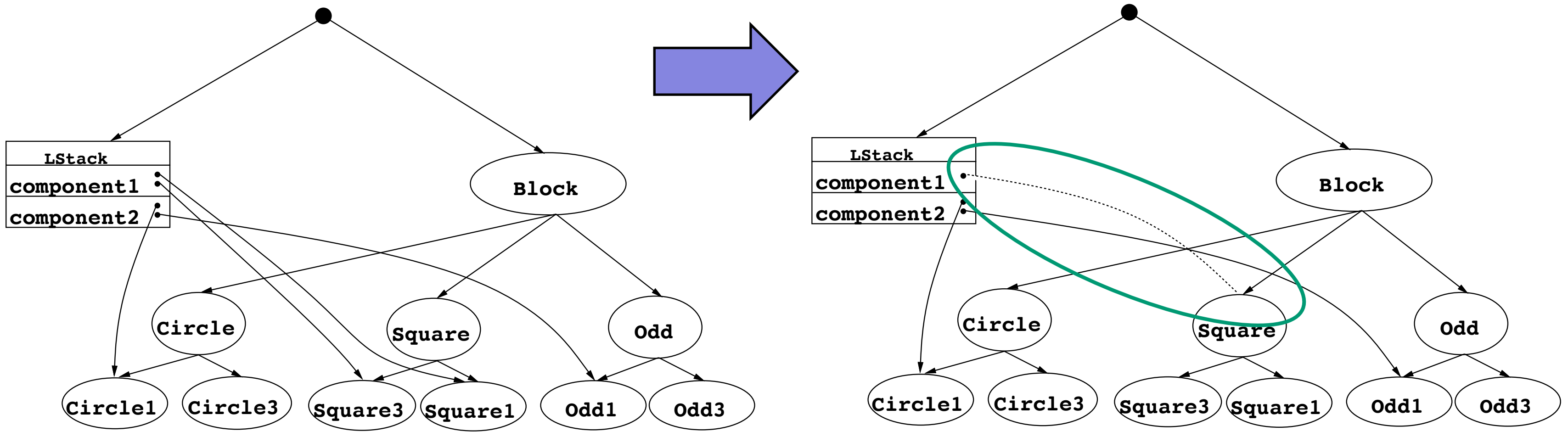
- *Classifies each component based on its description*
- *Uses component categories to redescribe the composite*
- *Classifies the composite based on components and relations*

Because different component mappings are possible, Labyrinth considers them all and selects the best.

Learning is similar to that in Cobweb with one key exception: the need to generalize composite values.

# Labyrinth: Attribute Generalization

Because composite concepts describe components as types, Labyrinth ***generalizes*** over attribute values.



Thus, the system invokes a form of ***representation learning*** to handle structured instances.

Labyrinth uses a variation on category utility to decide when it should take this restructuring step.

## Labyrinth: Negative Results

When developed, Labyrinth seemed like a natural extension to Cobweb for structured / relational domains.

However, the system produced no compelling results because:

- *Attribute generalization required a good component hierarchy*
- *Order effects meant the component hierarchy was* ***unreliable***
- *Restructuring meant the component hierarchy was* ***unstable***

Labyrinth had attempted to build its composite hierarchy on shifting sands, not a solid foundation.

Some later systems used flattened structural descriptions (e.g., MacLellan et al., 2016). to avoid this issue.

## Learning Movement Concepts

Cobweb acquired probabilistic concepts that described static objects or situations.

However, people can induce concepts about dynamic behaviors, including ones for volitional motion:

- *E.g., swinging a club or throwing a ball*
- *E.g., executing a series of dance steps*

For stereotypical activities, we seem to store generalized ***motor schemas*** in long-term memory.

Iba (1991) developed the OXBOW system to represent, use, and acquire such movement concepts.

## OXBOW: Representation / Organization

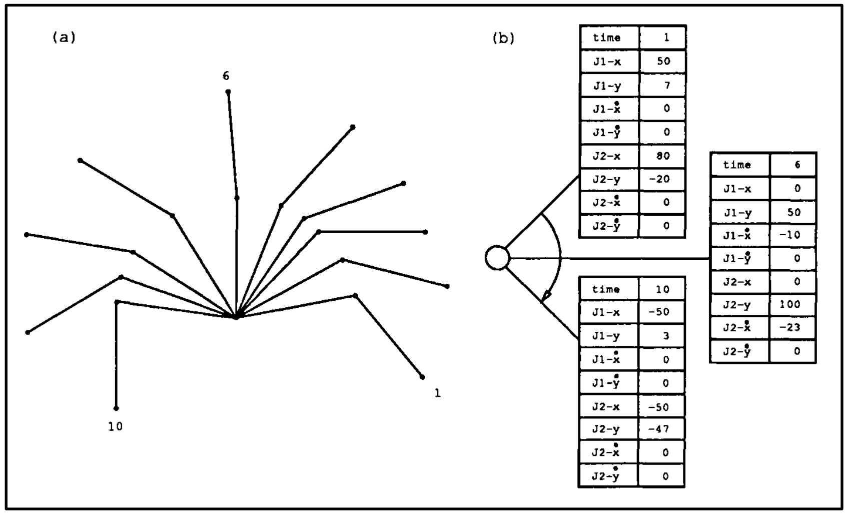
OXBOW organizes concepts in a hierarchy using unsupervised learning, but differs from Cobweb in that:

- Instances comprise a ***sequence of states***, each specifying:
  - *Numeric attribute values for joint positions*
  - *Only a subset of observed states with zero derivatives*
- Concepts in hierarchy also contain state sequences, each with:
  - *The time at which the state occurs (mean / standard deviation)*
  - *Position and velocity for each joint (means / standard deviations)*

The hierarchy is similar to Cobweb's but organizes AND trees that describe motor schemas.

# OXBOW: Representation / Organization

**OXBOW Instance**



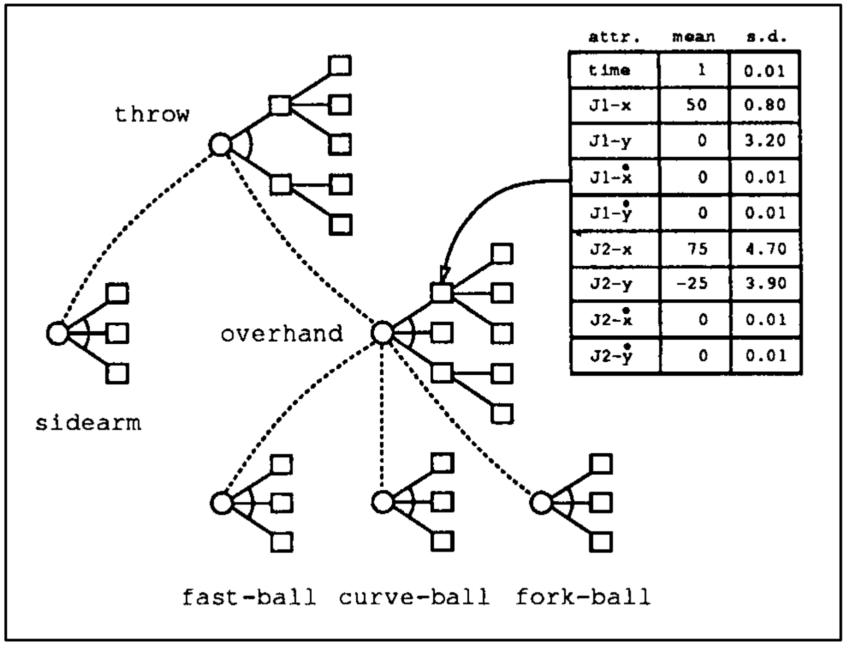
| | |
|---|---|
| time | 1 |
| J1-x | 50 |
| J1-y | 7 |
| J1-$\dot{x}$ | 0 |
| J1-$\dot{y}$ | 0 |
| J2-x | 80 |
| J2-y | -20 |
| J2-$\dot{x}$ | 0 |
| J2-$\dot{y}$ | 0 |

| | |
|---|---|
| time | 6 |
| J1-x | 0 |
| J1-y | 50 |
| J1-$\dot{x}$ | -10 |
| J1-$\dot{y}$ | 0 |
| J2-x | 0 |
| J2-y | 100 |
| J2-$\dot{x}$ | -23 |
| J2-$\dot{y}$ | 0 |

| | |
|---|---|
| time | 10 |
| J1-x | -50 |
| J1-y | 3 |
| J1-$\dot{x}$ | 0 |
| J1-$\dot{y}$ | 0 |
| J2-x | -50 |
| J2-y | -47 |
| J2-$\dot{x}$ | 0 |
| J2-$\dot{y}$ | 0 |

*Each concept specifies a sequence of states, each with an associated time and set of continuous attributes.*

**OXBOW Concept Hierarchy**



| attr. | mean | s.d. |
|---|---|---|
| time | 1 | 0.01 |
| J1-x | 50 | 0.80 |
| J1-y | 0 | 3.20 |
| J1-$\dot{x}$ | 0 | 0.01 |
| J1-$\dot{y}$ | 0 | 0.01 |
| J2-x | 75 | 4.70 |
| J2-y | -25 | 3.90 |
| J2-$\dot{x}$ | 0 | 0.01 |
| J2-$\dot{y}$ | 0 | 0.01 |

*OXBOW employs a sparse encoding of the observed motions that only includes states with zero derivatives.*

## OXBOW: Performance / Learning

OXBOW sorts a state sequence through its concept hierarchy, altering its statistics and structure in the process.

When deciding which of a node's children to select, it:

- Aligns instance and concept states using times and probabilities
  - *Assigning similar instance states to analogous concept states*
  - *But some instance states may still be left unassigned*
- Combines selected states into a flat attribute-value description
  - *Computing each option's score and selecting the best*
  - *Category utility =* $\dfrac{\sum_{k}^{K} P(C_k) \sum_{j}^{J} P(S_{kj}) \sum_{i}^{I} \frac{1}{\sigma_{kji}} \; - \; \sum_{m}^{M} P(S_{pm}) \sum_{i}^{I} \frac{1}{\sigma_{pmi}}}{K}$

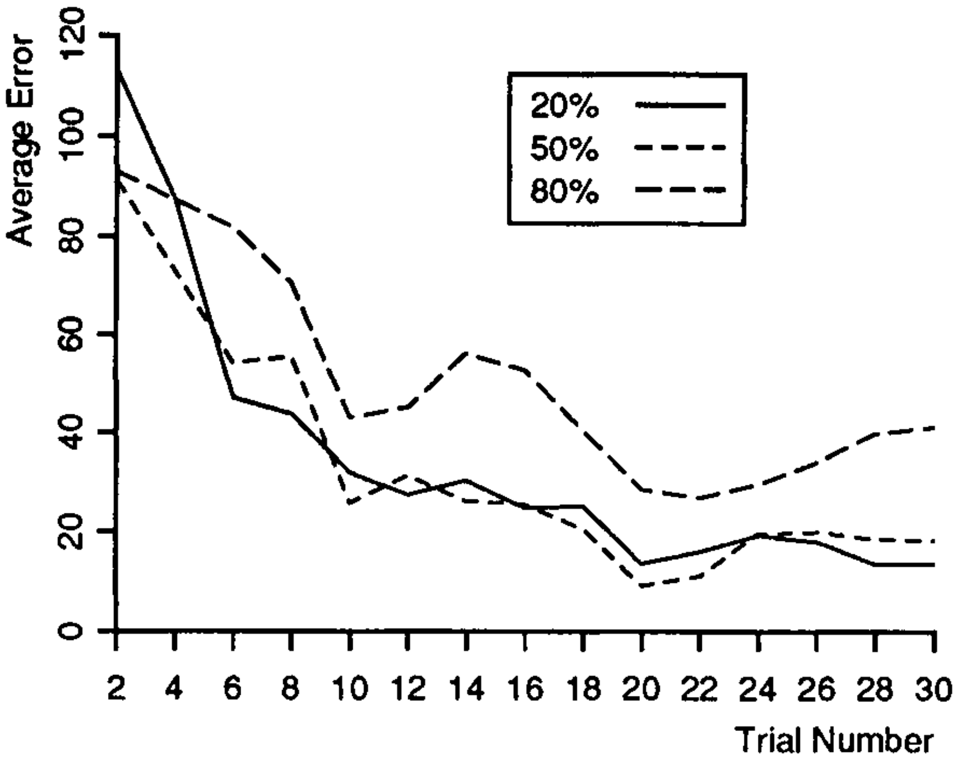
This lets OXBOW incorporate sequences with different numbers of states into the hierarchy.

# OXBOW: Experimental Results

Iba (1991) also reported experimental evaluations of OXBOW's learning behavior.

One study presented the system with instances of four motion types: *salute*, *slap*, *semaphore*, and *wave*.

Different instances of a given category varied in their quantitative values.



Higher variance for attributes' values increased prediction error, but OXBOW still acquired the target concepts.

## Learning Problem-Solving Concepts

Cobweb was designed to learn concepts that support simple classification and prediction.

But humans also learn how to solve multi-step problems that require heuristic search:

- *E.g., proving theorems in logic or geometry*
- *E.g., finding answers to algebra or physics tasks*

Again, Cobweb's representation does not provide the structures to store or incorporate such solutions.

Yoo and Fisher (1991) developed another system – EXOR – to address challenges raised by problem solving.

## EXOR: Representation / Organization

The EXOR system extended Cobweb to learn from solutions to multi-step problems with:

- Training instances that it encoded as:
  - *Relational statements of problems (givens, goal)*
  - *Solutions to these problems (AND trees)*
- Conceptual knowledge that it cast as:
  - *Probabilistic summaries of problem features*
  - *Abstract problem solutions (partial AND trees)*

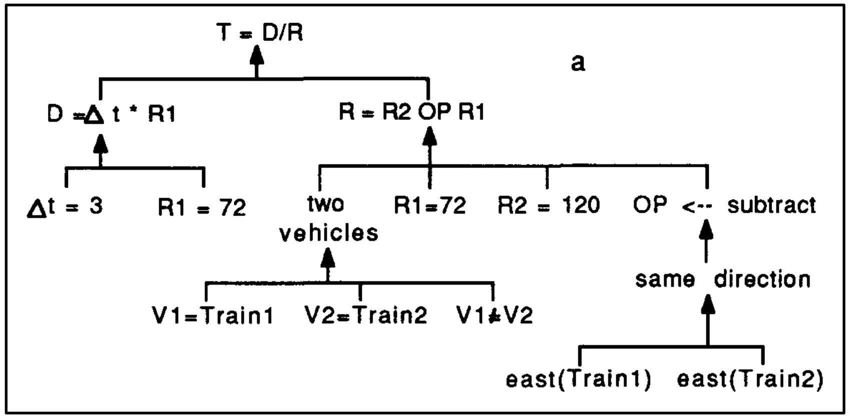
EXOR used the learned concept hierarchy to index these abstract solutions for future use.

# EXOR: Representation / Organization

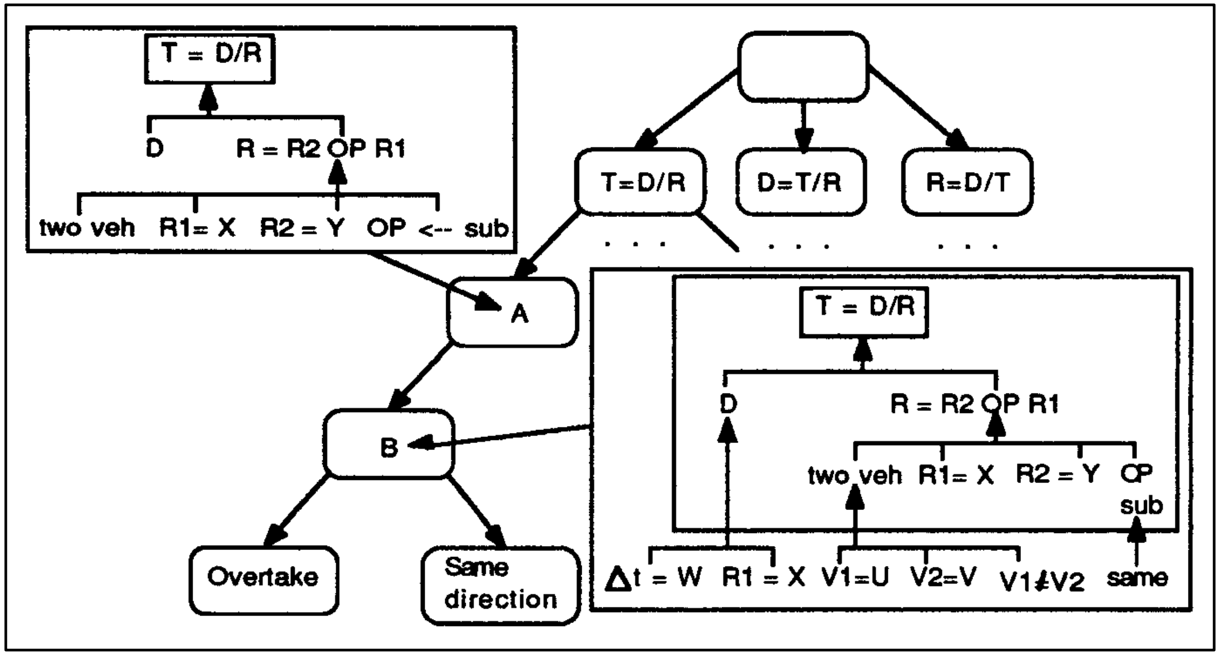
**EXOR Problem**

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

**EXOR Solution**



**EXOR Hierarchy**



## EXOR: Performance / Learning

To solve a given problem, EXOR sorts it through the hierarchy.

At each level, it selects the best-scoring child C and then it:

- *Ensures C's description D is consistent with problem constraints*
- *If not, then it examines the next best child and repeats process*
- *If no children are consistent, then it elaborates using domain rules*
- *If this fails, then it backs up in hierarchy to consider other nodes*

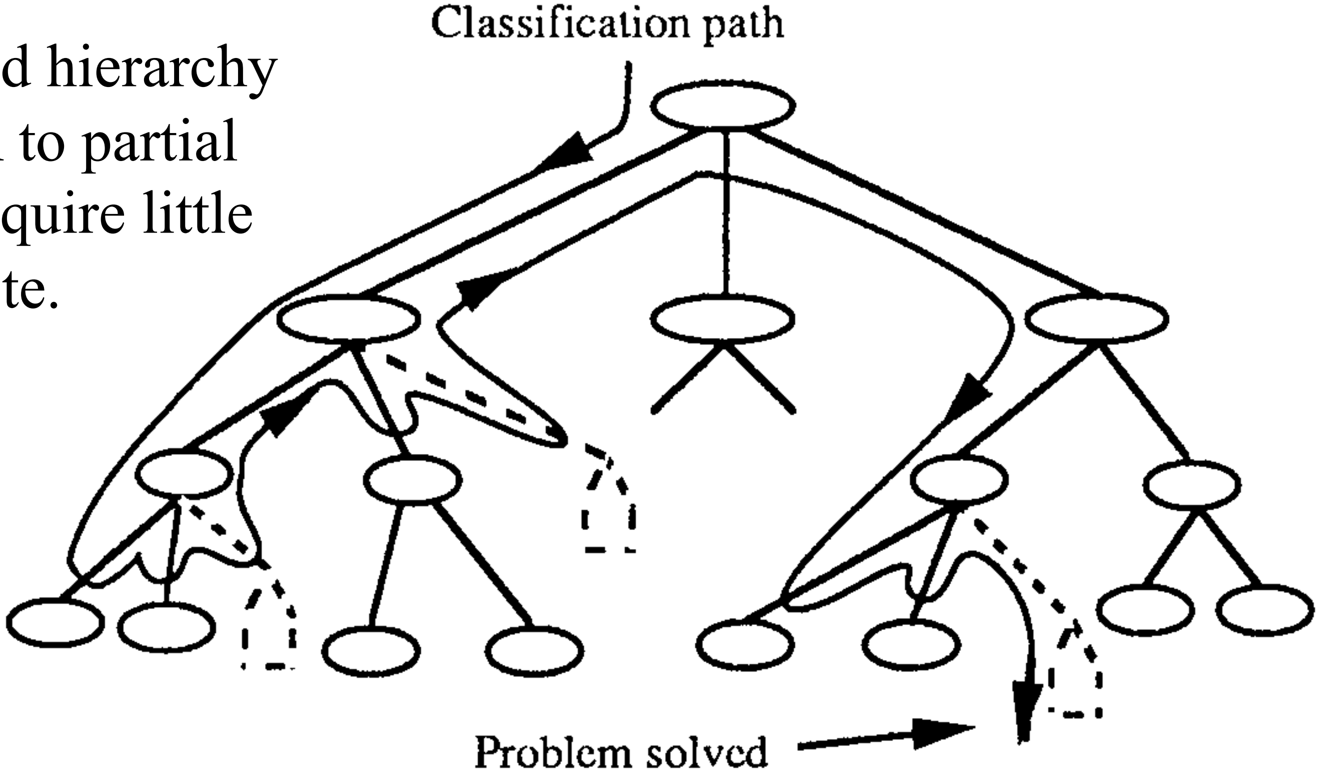
Eventually, this produces an AND tree that solves the problem, ideally with very little search.

Terminal nodes store 'macro-operators' with complete solutions, but higher-level, abstract nodes are also useful.

## Search Reduction in EXOR

We can view this process as replacing search in a *problem space* with search through the *concept memory*.

A well-organized hierarchy will lead EXOR to partial solutions that require little effort to complete.

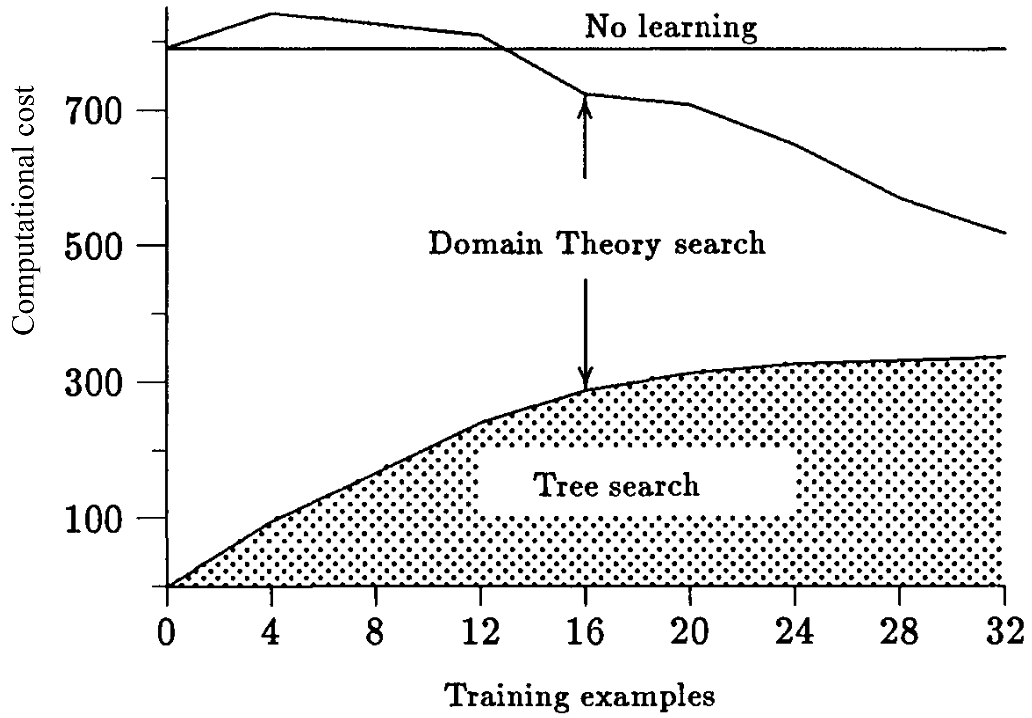


## EXOR: Experimental Results

Yoo and Fisher (1991) studied EXOR's learning behavior on 48 algebra problems.

They used 32 problems for training and reserved the other 16 problems (one of each type) for testing.

The cost to find solutions decreased with experience, with EXOR relying more on classification in later stages.



The system also did substantially better than learning traditional macro-operators with no hierarchy (not shown).

## Other Descendants of Cobweb

Cobweb inspired a variety of other systems that extended its ideas or applied them to new settings:

- *CFIX (Handa, 1990) – Using context in structured domains*
- *No name (Anderson, 1991) – 'Rational analysis' of categorization*
- *ARACHNE (McKusick & Langley, 1991) – Mitigating order effects*
- *COBBIT (Kilander & Jansson, 1993) – Handling concept drift*
- *DAEDALUS (Allen & Langley, 1989) – Search control for planning*
- *BRIDGER (Reich & Fenves, 1991) – Structural design*
- *ARC (Day, 1992) – Search control for constraint satisfaction*
- *ECOBWEB (Reich & Fenves, 1993) – Concept drift, design*
- *TRESTLE (MacLellan et al., 2016) – Structured domains, design*

There were many reasons for the extended gap, but Cobweb's potential was still waiting to be tapped.

# References for Cobweb Extensions

Gennari, J. H., Langley, P., & Fisher, D. H. (1989). Models of incremental concept formation. *Artificial Intelligence, 40,* 11–61.

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

Kilander, F., & Jansson, C. G. (1993). COBBIT – A control procedure for COBWEB in the presence of concept drift. *Proceedings of the European Conference on Machine Learning* (pp. 153–164). Vienna, Austria: Springer-Verlag.

Martin, J. (1992). *Direct and indirect transfer: Explorations in concept formation*. PhD thesis, College of Computing, Georgia Institute of Technology, Atlanta, GA.

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 Mateo, 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 Mateo, CA: Morgan Kaufmann.