<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0</td>
<td>Introduction</td>
<td>10.5</td>
<td>Knowledge and Learning</td>
</tr>
<tr>
<td>10.1</td>
<td>A Framework for Symbol-Based Learning</td>
<td>10.6</td>
<td>Unsupervised Learning</td>
</tr>
<tr>
<td>10.2</td>
<td>Version Space Search</td>
<td>10.7</td>
<td>Reinforcement Learning</td>
</tr>
<tr>
<td>10.3</td>
<td>The ID3 Decision Tree Induction Algorithm</td>
<td>10.8</td>
<td>Epilogue and References</td>
</tr>
<tr>
<td>10.4</td>
<td>Inductive Bias and Learnability</td>
<td>10.9</td>
<td>Exercises</td>
</tr>
</tbody>
</table>

George F Luger

**ARTIFICIAL INTELLIGENCE 6th edition**

Structures and Strategies for Complex Problem Solving
Fig 10.1 A general model of the learning process
Fig 10.2 Examples and near misses for the concept “arch.”
Fig 10.3 generalization of descriptions to include multiple examples.

a. An example of an arch and its network description

b. An example of another arch and its network description
Fig 10.3  generalization of descriptions to include multiple examples (cont’d)

c. Given background knowledge that bricks and pyramids are both types of polygons

```
   polygon
      isa
   brick
      isa
   pyramid
```

d. Generalization that includes both examples

```
   arch
      part
      part
      part
   brick
   supports
   polygon
```
Fig 10.4 Specialization of a description to exclude a near miss. In 10.4c we add constraints to 10.4a so that it can’t match with 10.4b.
Fig 10.5 A concept space.
Defining *specific to general search*, for hypothesis set $S$ as:

Begin  
Initialize $S$ to the first positive training instance;  
$N$ is the set of all negative instances seen so far;  

For each positive instance $p$  
  Begin  
  For every $s \in S$, if $s$ does not match $p$, replace $s$ with its most specific generalization that matches $p$;  
  Delete from $S$ all hypotheses more general than some other hypothesis in $S$;  
  Delete from $S$ all hypotheses that match a previously observed negative instance in $N$;  
  End;  

For every negative instance $n$  
  Begin  
  Delete all members of $S$ that match $n$;  
  Add $n$ to $N$ to check future hypotheses for overgeneralization;  
  End;  

End
In this algorithm, negative instances lead to the specialization of candidate concepts; the algorithm uses positive instances to eliminate overly specialized concepts.

Begin
Initialize \( G \) to contain the most general concept in the space; \( P \) contains all positive examples seen so far;

For each negative instance \( n \)
Begin
For each \( g \in G \) that matches \( n \), replace \( g \) with its most general specializations that do not match \( n \);
Delete from \( G \) all hypotheses more specific than some other hypothesis in \( G \);
Delete from \( G \) all hypotheses that fail to match some positive example in \( P \);
End;

For each positive instance \( p \)
Begin
Delete from \( G \) all hypotheses that fail to match \( p \);
Add \( p \) to \( P \);
End;
End
Fig 10.6 The role of negative examples in preventing overgeneralization.
Fig 10.7 Specific to general search of the version space learning the concept “ball.”

S: {}

Positive: obj(small, red, ball)

S: {obj(small, red, ball)}

Positive: obj(small, white, ball)

S: {obj(small, X, ball)}

Positive: obj(large, blue, ball)

S: {obj(Y, X, ball)}
The algorithm specializes $G$ and generalizes $S$ until they converge on the target concept. The algorithm is defined:

Begin  
Initialize $G$ to be the most general concept in the space;  
Initialize $S$ to the first positive training instance;

For each new positive instance $p$  
Begin  
Delete all members of $G$ that fail to match $p$;  
For every $s \in S$, if $s$ does not match $p$, replace $s$ with its most specific generalizations that match $p$;  
Delete from $S$ any hypothesis more general than some other hypothesis in $S$;  
Delete from $S$ any hypothesis more general than some hypothesis in $G$;  
End;

For each new negative instance $n$  
Begin  
Delete all members of $S$ that match $n$;  
For each $g \in G$ that matches $n$, replace $g$ with its most general specializations that do not match $n$;  
Delete from $G$ any hypothesis more specific than some other hypothesis in $G$;  
Delete from $G$ any hypothesis more specific than some hypothesis in $S$;  
End;
Fig 10.8 General to specific search of the version space learning the concept “ball.”

G: \{obj(X,Y,Z)\}

G: \{obj(large, Y, Z), obj(X, white, Z), obj(X, blue, Z), obj(X, Y, ball), obj(X, Y, cube)\}

G: \{obj(large, Y, Z), obj(X, white, Z), obj(X, Y, ball)\}

G: \{obj(large, white, Z), obj(X, white, Z), obj(X, Y, ball)\}

G: \{obj(large, white, Z), obj(X, white, Z), obj(X, Y, ball)\}

G: \{obj(X, Y, ball)\}

Negative: obj(small, red, brick)

Positive: obj(large, white, ball)

Negative: obj(large, blue, cube)

Positive: obj(small, blue, ball)
Fig 10.9 The candidate elimination algorithm learning the concept “red ball.”
Fig 10.10 Converging boundaries of the G and S sets in the candidate elimination algorithm.
Fig 10.11 A portion of LEX’s hierarchy of symbols.
Fig 10.12 A version space for OP2, adapted from Mitchell et al. (1983).
Table 10.1 Data from credit history of loan applications

<table>
<thead>
<tr>
<th>NO.</th>
<th>RISK</th>
<th>CREDIT HISTORY</th>
<th>DEBT</th>
<th>COLLATERAL</th>
<th>INCOME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>high</td>
<td>bad</td>
<td>high</td>
<td>none</td>
<td>$0 to $15k</td>
</tr>
<tr>
<td>2.</td>
<td>high</td>
<td>unknown</td>
<td>high</td>
<td>none</td>
<td>$15 to $35k</td>
</tr>
<tr>
<td>3.</td>
<td>moderate</td>
<td>unknown</td>
<td>low</td>
<td>none</td>
<td>$15 to $35k</td>
</tr>
<tr>
<td>4.</td>
<td>high</td>
<td>unknown</td>
<td>low</td>
<td>none</td>
<td>$0 to $15k</td>
</tr>
<tr>
<td>5.</td>
<td>low</td>
<td>unknown</td>
<td>low</td>
<td>none</td>
<td>over $35k</td>
</tr>
<tr>
<td>6.</td>
<td>low</td>
<td>unknown</td>
<td>low</td>
<td>adequate</td>
<td>over $35k</td>
</tr>
<tr>
<td>7.</td>
<td>high</td>
<td>bad</td>
<td>low</td>
<td>none</td>
<td>$0 to $15k</td>
</tr>
<tr>
<td>8.</td>
<td>moderate</td>
<td>bad</td>
<td>low</td>
<td>adequate</td>
<td>over $35k</td>
</tr>
<tr>
<td>9.</td>
<td>low</td>
<td>good</td>
<td>low</td>
<td>none</td>
<td>over $35k</td>
</tr>
<tr>
<td>10.</td>
<td>low</td>
<td>good</td>
<td>high</td>
<td>adequate</td>
<td>over $35k</td>
</tr>
<tr>
<td>11.</td>
<td>high</td>
<td>good</td>
<td>high</td>
<td>none</td>
<td>$0 to $15k</td>
</tr>
<tr>
<td>12.</td>
<td>moderate</td>
<td>good</td>
<td>high</td>
<td>none</td>
<td>$15 to $35k</td>
</tr>
<tr>
<td>13.</td>
<td>low</td>
<td>good</td>
<td>high</td>
<td>none</td>
<td>over $35k</td>
</tr>
<tr>
<td>14.</td>
<td>high</td>
<td>bad</td>
<td>high</td>
<td>none</td>
<td>$15 to $35k</td>
</tr>
</tbody>
</table>
Fig 10.13 A decision tree for credit risk assessment.
Fig 10.14 a simplified decision tree for credit risk assessment.
The induction algorithm begins with a sample of correctly classified members of the target categories. ID3 constructs a decision tree according to the algorithm:

```
function induce_tree (example_set, Properties)
begin
if all entries in example_set are in the same class
  then return a leaf node labeled with that class
else if Properties is empty
  then return leaf node labeled with disjunction of all classes in example_set
else begin
  select a property, P, and make it the root of the current tree;
  delete P from Properties;
  for each value, V, of P,
  begin
    create a branch of the tree labeled with V;
    let partition_v be elements of example_set with values V for property P;
    call induce_tree(partition_v, Properties), attach result to branch V
  end
end
end
```
Fig 10.15 A partially constructed decision tree.

Fig 10.16 Another partially constructed decision tree.
Table 10.2 The evaluation of ID3

<table>
<thead>
<tr>
<th>Size of Training Set</th>
<th>Percentage of Whole Universe</th>
<th>Errors in 10,000 Trials</th>
<th>Predicted Maximum Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>0.01</td>
<td>199</td>
<td>728</td>
</tr>
<tr>
<td>1,000</td>
<td>0.07</td>
<td>33</td>
<td>146</td>
</tr>
<tr>
<td>5,000</td>
<td>0.36</td>
<td>8</td>
<td>29</td>
</tr>
<tr>
<td>25,000</td>
<td>1.79</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>125,000</td>
<td>8.93</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Fig 10.17 Specific and generalized proof that an object, X, is a cup.

Proof that obj1 is a cup

- cup(obj1)
  - liftable(obj1)
    - light(obj1)
    - small(obj1)
  - holds liquid(obj1)
    - part(obj1, handle)
    - part(obj1, bowl)
    - concave(bowl)
    - points_up(bowl)

Generalized proof that X is a cup

- cup(X)
  - liftable(X)
    - light(X)
    - small(X)
  - holds liquid(X)
    - part(X, handle)
    - part(X, W)
    - concave(W)
    - points_up(W)
Fig 10.18 An explanation structure of the cup example.
Fig 10.19 An analogical mapping.
Fig 10.20 The steps of a CLUSTER/2 run.

After selecting seeds (step 1).

After generating general descriptions (steps 2 and 3). Note that the categories overlap.

After specializing concept descriptions (step 4). There are still intersecting elements.

After eliminating duplicate elements (step 5).
Fig 10.21 A COBWEB clustering for four one-celled organisms, adapted from Gennari et al. (1989).
A COBWEB algorithm is defined:

cobweb(Node, Instance)
begin
  if Node is a leaf
    then begin
      create two children of Node, L1 and L2;
      set the probabilities of L1 to those of Node;
      initialize the probabilities for L2 to those of Instance;
      add Instance to Node, updating Node’s probabilities;
    end
  else begin
    add Instance to Node, updating Node’s probabilities;
    for each child, C, of Node, compute the category utility of the clustering
      achieved by placing Instance in C;
    let S1 be the score for the best categorization, C1;
    let S2 be the score for the second best categorization, C2;
    let S3 be the score for placing instance in a new category;
    let S4 be the score for merging C1 and C2 into one category;
    let S5 be the score for splitting C1 (replacing it with its child categories)
  end
  If S1 is the best score
    then cobweb(C1, Instance) % place the instance in C1
  else if S3 is the best score
    then initialize the new category’s probabilities to those of Instance
  else if S4 is the best score
    then begin
      let Cm be the result of merging C1 and C2;
      cobweb(Cm, Instance)
    end
  else if S5 is the best score
    then begin
      split C1;
      cobweb(Node, Instance)
    end;
end
Fig 10.22 Merging and splitting of nodes.
Fig 10.23 A sequence of tic-tac-toe moves. Dashed arrows indicate possible move choices, down solid arrows indicate selected moves, up solid arrows indicate reward, when reward function changes state’s value.
Fig 10.24 Backup diagrams for (a) $V^*$ and (b) $Q^*$, adapted from Sutton and Barto (1998).
Fig 10.25 A step.

Fig 10.26 An example of a 4 x 4 grid world, adapted from Sutton and Barto (1998).