

92-3

**Hypothesis-driven Constructive
Induction in AQ17:
A Method and Experiments**

Janusz Wnek and Ryszard S. Michalski

**P92-2
MLI 92-2**

Hypothesis-driven Constructive Induction in AQ17: A Method and Experiments

Janusz Wnek and Ryszard S. Michalski
GMU Center for Artificial Intelligence
4400 University Dr.
Fairfax, VA 22030, USA
{wnek, michalski}@aic.gmu.edu

Abstract

This paper presents a method for constructive induction in which new problem-relevant attributes are generated by analyzing consecutively created inductive hypotheses. The method starts by creating a set of rules from given examples using the AQ algorithm. These rules are then evaluated according to a rule quality criterion. Subsets of the best-performing rules for each decision class are selected to form new attributes. These new attributes are used to reformulate the training examples used in the previous step, and the whole inductive process repeats. This iterative process ends when the performance accuracy of the rules exceeds a predefined threshold. In several experiments on learning different well-defined transformations, the method consistently outperformed (in terms of predictive accuracy) the AQ15 rule learning method, GREEDY3 and GROVE decision list learning methods, and REDWOOD and FRINGE decision tree learning methods.

1 Introduction

Most programs for inductive learning from examples create descriptions in terms of attributes that are selected from those present in the original examples. Such programs, e.g., AQVAL/1 (Michalski, 1973), ID3 (Quinlan, 1983), ASSISTANT (Cestnik et al., 1987), CN2 (Clark and Niblett, 1989), do not create new attributes or, in general, concepts in the process of learning. In contrast to such *selective* inductive programs, a *constructive* induction program generates and uses new concepts in the hypothesized description. These new concepts (or descriptors) can be attributes, predicates, terms, operators, etc., and are more relevant to the learning problem than those initially given. A constructive learning program thus performs a problem-oriented transformation of the knowledge representation space (Michalski, 1978).

The idea of constructive induction was first proposed by Michalski (1978), and implemented in the INDUCE.1 program for learning structural descriptions from examples. INDUCE.1 used user-defined inference rules and procedures to generate new descriptors. These descriptors were then employed together with the original ones in the process of induction.

Subsequently, a number of other systems have been developed that exhibit certain constructive induction capabilities. In general, the constructive induction methods can be divided into four categories:

- **Data-Driven (DCI)** - by analyzing and exploring the input data. For example:

INDUCE (Michalski, 1978)
 LEX (Mitchell, Utgoff & Banerji, 1983)
 AM, EURISCO (Lenat, 1983)
 BACON (Langley, Bradshaw & Simon, 1983)
 STAGGER (Schlimmer, 1987)
 AQ17-DCI (Bloedorn & Michalski, 1991)

- **Hypothesis-Driven (HCI)** - by analyzing generated hypotheses. For example:

FRINGE (Pagallo & Haussler, 1990) - for decision trees
 AQ17-HCI (Wnek & Michalski, 1991) - for decision rules

- **Knowledge-Driven (KCI)** - by applying expert-provided knowledge For example:

INDUCE (Michalski, 1978)
 AQ15 (Michalski, Mozetic, Hong & Lavrac, 1985)
 MIRO (Drastal, Czako & Raatz, 1989)
 DUCE, CIGOL (Muggleton, 1987)

- **Multistrategy (MCI)** - by combining different methods. For example:

PLS0 (Rendell, 1985) - DCI, KCI
 CITRE (Matheus, 1989) - DCI, KCI
 AQ17 (combines a whole spectrum of methods, such as DCI, HCI, A-rules, L-rules and GDN; Bloedorn, Michalski & Wnek, 1992).

In an attribute-based learning system, the abstraction level of attributes strongly affects the complexity of the hypothesized rules. By employing high-level attributes, the concept representation can be greatly simplified. Such attributes may, however, be encoded as very complex functions of low-level primitives. In such cases, to improve the efficiency, these complex functions have to be compiled (Flann and Dietterich 1986). One important goal of constructive induction is therefore to discover attributes that lead to the maximal simplification of the generated hypotheses. Another goal is to discover attributes that produce the best performing hypotheses, that is, to emphasize the predictive accuracy of generated hypotheses, rather than their simplicity.

The predictive accuracy can be measured by applying the hypotheses to the testing data, and determining the correctness of the predictions.

The primary goal of the method proposed in this paper is to increase the predictive accuracy of the hypotheses. The underlying idea is to generate new attributes by analyzing the hypotheses initially created by a selective induction process, and then by consecutive learning steps. For that reason, the method is called a "hypothesis-driven" constructive induction (HCI). The method is a major component of the multistrategy constructive induction system AQ17 (Bloedorn, Michalski and Wnek, 1992).

Initial and consecutive selective hypotheses are generated by the rule learning program AQ15 (Michalski et al., 1986). AQ15 learns rules from examples represented as sequences of attribute-value pairs. Attributes can be multi-valued and be of different types, such as nominal, linear or structured (in which the value set is a hierarchy). The teacher presents to the learner a set of examples of each of the concepts to be learned. The program outputs a set of general decision rules (that are equivalent to DNF expressions) for each class. These rules cover all the examples of a given class and none of other classes' examples (i.e., they are consistent and complete descriptions). The rules generated optimize a problem-dependent "criterion of preference." In the case of noisy data, the program may generate only partially consistent and/or complete rules.

The program is based on the AQ algorithm, which iteratively employs a *star* generation procedure (Michalski et al., 1986). A *star of an example* is the set of the most general alternative rules that cover that example, but do not cover any negative examples. In the first step, a star is generated for a randomly chosen example (a *seed*), and the "best" rule in it, as defined by the preference criterion, is selected. All examples covered by that rule are removed from further consideration. A new seed is then selected from the yet-uncovered examples, and the process repeats. The algorithm ends when all positive examples are covered. If there exists a single rule that covers all the examples (that is, there exists a conjunctive characterization of the concept), the algorithm terminates after the first step. In the presented method, the above algorithm is combined with a process of iteratively generating new attributes, and using them in subsequent learning steps.

2 A description of the method

As mentioned above, the proposed HCI method ('a hypothesis-driven constructive induction') determines new problem-relevant attributes by analyzing the currently held inductive hypotheses (Figure 1). Its primary goal is to determine new attributes that produce a hypothesis with the highest predictive accuracy. A natural extension of this goal is detection and removal from the input data any irrelevant attributes.

Basic steps of the proposed HCI method are:

1. Induce rules for each decision class from a subset of training examples using a selective inductive program (AQ15)
2. Analyze the rules to identify irrelevant attributes
3. For each decision class generate one candidate attribute that corresponds to a subset of the highest quality rules
4. Modify training examples by adding newly generated attributes and removing irrelevant ones
5. Induce rules from the modified training set
6. Evaluate the predictive accuracy of the rules on remaining training examples. If the performance does not exceed a predefined threshold, go to step 2.
7. Induce rules from the complete training set using all relevant initial and derived attributes.

The steps 1, 5, and 7 are performed by the AQ15 program. In the step 2, the HCI method identifies those attributes that were not found in hypotheses, or were found in rules that cover marginal number of training examples, “small disjuncts,” only. The heuristic for constructing attributes in Step 3 includes extracting a part of a classification hypothesis which contains *best rules*. This is done by sorting all rules from the output hypothesis according to the so-called *tu-weights*, and selecting the maximum number of rules with the highest *tu-weights* that satisfy the inequality:

$$[(\sum tu_{best}) / (\sum tu_{all})] < TH1 \quad (1)$$

where

$$tu = t\text{-weight} + (2 * u\text{-weight}) \quad (2)$$

and *t-weight* denotes the total number of positive examples covered by the rule, and *u-weight* denotes the number of positive examples that are uniquely covered by the rule (i.e., no other rule in the hypothesis covers those examples). The constant “2” in the formula (2) was determined experimentally (it assures that the rules with higher *u-weight* are preferred in a constructed attribute). The TH1 threshold is a program parameter, and is set to 0.65.

The number of *best rules* defined by the TH1 threshold can be extended by those remaining rules that do not deviate much from already selected, with respect to *tu-weights*. If $tu_{best-min}$ is a lowest weight among rules already selected than only rules with *tu-weights* greater than $(TH2 * tu_{best-min})$ will be added to constructed attribute. The TH2 threshold was equal to 0.65. The motivation for the TH1 and TH2 thresholds arises from an assumption about the nature of constructed attributes: they should convey to the next learning step as much useful knowledge of the learned problem as possible. In this setting, a fundamental part of an intermediate concept is stored as new

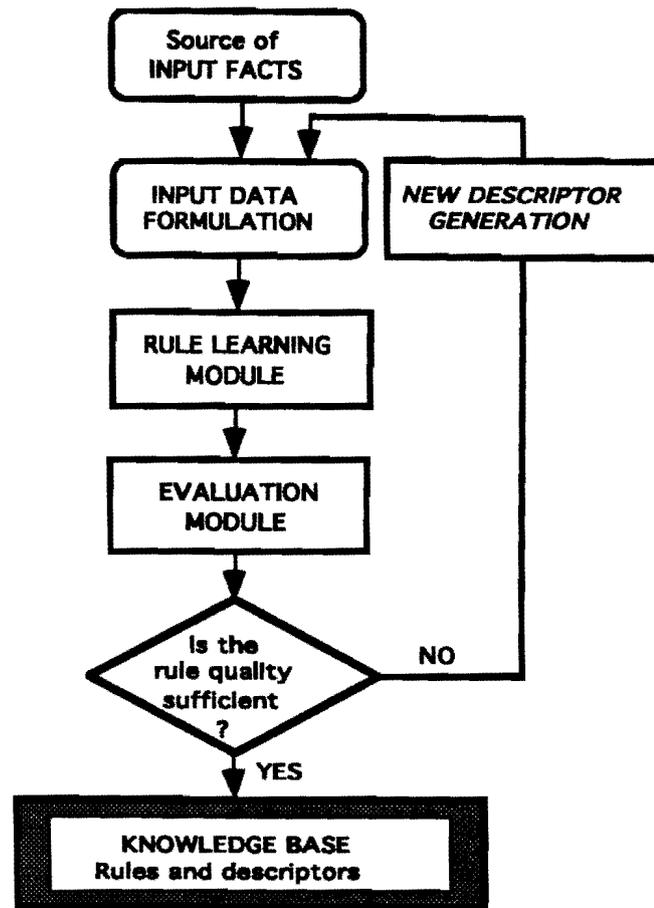


Figure 1. HCI method built into a rule learning system

attribute. The remaining part of a concept, which holds exceptions or noise in the data, will be covered in following iterations. The role of the TH2 threshold is to assure that *all strong rules* from a hypothesis will form a new attribute.

Initially, there are only two values assigned to the candidate attribute that characterize a class membership. More values can be added if the same attribute description occurred in other classes. The new attributes are logical expressions involving previous attributes. After new attributes are determined, the training set is updated with new attribute values. For each training example the values of the new attributes are calculated by evaluating the logical expression characterizing the new attribute.

From this outline of the method one can see that the process of inducing rules from examples may be repeated several times in order to achieve the desired predictive accuracy. This adds complexity

to the learning algorithm depending on how many times steps 2-6 are repeated. The complexity of inducing rules from examples in AQ15 is $O(PN)$, where P is a number of positive examples and N is a number of negative examples (every positive example is generalized against all negative examples). The complexity of forming a new attribute is linear with respect to the number of rules in the hypothesis. As this constant effort is made for every iteration, the overall complexity is $O(PNT)$, where T is the number of iterations.

New attributes introduced in each iteration in the form of learned sub-concepts from the searched hypotheses space make the learning problem easier. The training set was already pre-partitioned by problem oriented and statistically significant attributes. Most of the training examples are covered with the new attributes, and only exceptions require building additional concept descriptions.

3 Exemplary problem

To illustrate the performance of the constructive search, we describe an experiment on learning a multiplexer function with 3 inputs and 8 outputs: the so-called *multiplexer-11 problem* (Wilson, 1987). For each positive integer k , there exists a multiplexer function defined on a set of $k + 2^k$ attributes or bits. The function can be defined by thinking of the first k attributes as *address bits* and the remaining attributes as *data bits*.

The function has the value of the data bit indexed by the address bits. In the experiment, the input examples were encoded in terms of 11 binary attributes. Thus, the description space contains 2048 elements. The training set had 64 (6%) positive examples and 64 (6%) of the negative examples. Table 1 shows a sample of positive and negative examples. The attributes a_0, a_1, a_2 describe address lines, and d_0-d_7 describe data lines. From these examples, the rule generation phase (Step 1) produced rules for the correct (POS-Class) and incorrect (NEG-Class) behavior of the multiplexer. The rules are shown in Table 2.

Positive examples											Negative examples										
a0	a1	a2	d0	d1	d2	d3	d4	d5	d6	d7	a0	a1	a2	d0	d1	d2	d3	d4	d5	d6	d7
0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	0
0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
0	1	0	1	1	1	1	1	1	1	0	0	1	0	1	1	0	1	1	1	1	0
1	0	1	0	0	0	0	1	1	0	1	1	0	1	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0
...

Table 1. A part of the training set of examples

POS-Class if

NEG-Class if

1. (a0=1) & (a1=1) & (a2=0) & (d6=1) or (t:11, u:6)	1. (a0=1) & (a1=1) & (a2=1) & (d7=0) or (t:13, u:5)
2. (a0=0) & (a1=0) & (a2=1) & (d1=1) or (t:11, u:5)	2. (a0=0) & (a2=0) & (d2=0) or (t:12, u:7)
3. (a0=1) & (a1=0) & (a2=1) & (d5=1) or (t:10, u:6)	3. (a2=0) & (d3=0) & (d4=0) & (d7=1) or (t:11, u:2)
4. (a0=1) & (a1=1) & (a2=1) & (d7=1) or (t:10, u:4)	4. (a0=0) & (a1=0) & (a2=1) & (d1=0) or (t:10, u:8)
5. (a0=1) & (a1=0) & (a2=0) & (d4=1) or (t:9, u:5)	5. (a0=1) & (a2=1) & (d5=0) & (d7=0) or (t:10, u:2)
6. (a1=1) & (d4=1) & (d6=1) & (d7=1) or (t:8, u:1)	6. (a1=0) & (a2=0) & (d1=1) & (d4=0) or (t:7, u:4)
7. (a1=1) & (a2=1) & (d3=1) & (d7=1) or (t:8, u:1)	7. (a0=1) & (a1=1) & (a2=0) & (d6=0) or (t:7, u:3)
8. (a0=0) & (a2=1) & (d1=1) & (d5=0) or (t:8, u:1)	8. (d0=0) & (d3=0) & (d5=0) & (d6=0) or (t:6, u:1)
9. (a2=0) & (d1=0) & (d2=1) & (d3=1) or (t:6, u:2)	9. (a0=0) & (a1=1) & (a2=1) & (d3=0) or (t:5, u:5)
10. (a0=0) & (a1=1) & (d3=1) & (d4=0) or (t:6, u:2)	10. (d1=0) & (d2=1) & (d3=0) & (d5=0) or (t:5, u:1)
11. (d0=0) & (d3=0) & (d4=1) & (d5=1) or (t:6, u:1)	11. (a0=1) & (a1=0) & (d5=0) & (d7=1) (t:4, u:4)
12. (a2=1) & (d0=1) & (d2=0) & (d5=1) (t:3, u:1)	

Table 2. Rules induced by AQ15 from examples

1. (a0=1) & (a1=1) & (a2=0) & (d6=1) or (tu:23)
2. (a0=0) & (a1=0) & (a2=1) & (d1=1) or (tu:21)
3. (a0=1) & (a1=0) & (a2=1) & (d5=1) or (tu:22)
4. (a0=1) & (a1=1) & (a2=1) & (d7=1) or (tu:18)
5. (a0=1) & (a1=0) & (a2=0) & (d4=1) (tu:19)

$$\sum tu_{\text{best}} = 103; \sum tu_{\text{all}} = 166; [(\sum tu_{\text{best}}) / (\sum tu_{\text{all}})] = 0.62$$

$$\sum tu_{\text{best}+1} = 113; \sum tu_{\text{all}} = 166; [(\sum tu_{\text{best}+1}) / (\sum tu_{\text{all}})] = 0.68$$

Table 3. Best rules from POS-Class according to the formula (1)

POS-Class and NEG-Class are hypotheses in the k-DNF form. Each rule in the hypotheses is accompanied with t-weights and u-weights that represent total and unique numbers of training examples covered by a rule. For the above POS-Class hypothesis, the rules presented in Table 3 were chosen to constitute the candidate attribute Pos0, (Step 3): (here we present attribute generation for POS hypothesis only). Table 4 shows the definition of the new attribute Pos0.

Pos0=1	if (a0=1) & (a1=1) & (a2=0) & (d6=1) or (a0=0) & (a1=0) & (a2=1) & (d1=1) or (a0=1) & (a1=0) & (a2=1) & (d5=1) or (a0=1) & (a1=1) & (a2=1) & (d7=1) or (a0=1) & (a1=0) & (a2=0) & (d4=1)
Pos0=0	otherwise

Table 4. The definition of the Pos0 attribute

Table 5 shows the modified training set. For each old training example a new Pos0 attribute value has been added (Step 4). Table 6 presents rules induced from the modified training set (Step 5).

Positive examples											Negative examples												
a0	a1	a2	d0	d1	d2	d3	d4	d5	d6	d7	Pos0	a0	a1	a2	d0	d1	d2	d3	d4	d5	d6	d7	Pos0
0	0	1	0	1	0	0	0	0	0	0	1	0	0	1	0	0	1	1	1	1	1	0	0
0	1	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0
0	1	0	1	1	1	1	1	1	1	0	0	0	1	0	1	1	0	1	1	1	1	0	0
1	0	1	0	0	0	0	1	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0

Table 5. The part of the modified training set

POS-Class if	NEG-Class if
1. (Pos0=1) or (t:51, u:45)	1. (a0=1) & (Pos0=0) or (t:34, u:12)
2. (a0=0) & (a1=1) & (a2=1) & (d3=1) or (t:7, u:5)	2. (a2=0) & (d2=0) & (Pos0=0) or (t:20, u:8)
3. (a2=0) & (d1=0) & (d2=1) & (d3=1) or (t:6, u:2)	3. (a1=0) & (d5=0) & (Pos0=0) or (t:18, u:4)
4. (a0=0) & (a1=1) & (d2=1) & (d7=0) or (t:5, u:2)	4. (d3=0) & (d5=1) & (d7=1) & (Pos0=0) or (t:13, u:4)
5. (a0=0) & (d0=1) & (d1=1) & (d2=1) (t:5, u:1)	5. (a2=1) & (d1=0) & (d2=1) & (d4=1) & (Pos0=0) (t:10, u:2)
	6. (a2=1) & (d0=1) & (d1=0) & (d6=0) & (Pos0=0) (t:7, u:2)

Table 6. Decision rules with the constructed attribute

Target concept	No. of attributes	No. of classes	No. of rules	Average rule length	No. of training examples	No. of testing examples
DNF 3	32	2	6	5.5	1650	2000
DNF 4	64	2	10	4.1	2640	2000
MX 11	32	2	8	4.0	1600	2000
PAR 5	32	2	16	5.0	4000	2000

Table 7. Target functions

As expected, the new attribute was used in the output hypothesis for both POS and NEG classes. We can observe that most of the training examples were uniquely covered by a rule (Pos0=1). Also, due to changes in the hypotheses space, a new useful rule (i.e., a part of the target concept description) was induced. (See rule (2) in the POS-Class hypothesis in Table 6). The ongoing research investigates ways of detecting and incorporating useful items into constructed attributes.

The final hypotheses produced by AQ17-HCI were tested against the testing set. The result was 85.6% accuracy (to be compared with 74% accuracy from rules generated by AQ15 without constructive abilities, e.g. rules obtained in Step 1).

4 Experiments with the method

A major measure of the performance of a learning algorithm is the classification accuracy of the learned concepts on the testing examples. The goal of our experiments was to test how well the method does according to this criterion, and how well it compares to other methods: standard decision rule algorithm - AQ15, standard decision tree algorithm - REDWOOD, and algorithms with constructive abilities: FRINGE, GREEDY3, and GROVE (Pagallo and Haussler, 1989 & 1990).

4.1 Experimental domains

The domains for testing AQ17-HCI and comparison with other methods were four Boolean functions: DNF3, DNF4, MX11, and PAR5. The same functions were used to test decision tree algorithms: REDWOOD (based on ID3) and FRINGE, and decision list algorithms: GREEDY3 and GROVE (Pagallo and Haussler, 1989, 1990).

DNF3 $x_1x_2x_6x_8x_{25}x_{28}\neg x_{29}$ or $x_2x_9x_{14}\neg x_{16}\neg x_{22}x_{25}$ or
 $x_1\neg x_4\neg x_{19}\neg x_{22}x_{27}x_{28}$ or $\neg x_2\neg x_{10}x_{14}\neg x_{21}\neg x_{24}$ or
 $x_{11}x_{17}x_{19}x_{21}\neg x_{25}$ or $\neg x_1\neg x_4x_{13}\neg x_{25}$

Attributes $x_3 x_5 x_7 x_{12} x_{15} x_{18} x_{20} x_{23} x_{26} x_{30} x_{31} x_{32}$ have random values for each example.

DNF4 $x_1x_4x_{13}x_{57}\neg x_{59}$ or $x_{18}\neg x_{22}\neg x_{24}$ or $x_{30}\neg x_{46}x_{48}\neg x_{58}$ or
 $\neg x_9x_{12}\neg x_{38}x_{55}$ or $\neg x_5x_{29}\neg x_{48}$ or $x_{23}x_{33}x_{40}x_{52}$ or
 $x_4\neg x_{26}\neg x_{38}\neg x_{52}$ or $x_6x_{11}x_{36}\neg x_{55}$ or $\neg x_6\neg x_9\neg x_{10}x_{39}\neg x_{46}$ or
 $x_3x_4x_{21}\neg x_{37}\neg x_{57}$

Attributes $x_2 x_7 x_8 x_{14} x_{15} x_{16} x_{17} x_{19} x_{20} x_{25} x_{27} x_{28} x_{31} x_{32} x_{34} x_{35} x_{41} x_{42} x_{43} x_{44} x_{45} x_{47} x_{49} x_{50} x_{51} x_{53} x_{54} x_{56} x_{60} x_{61} x_{62} x_{63} x_{64}$ have random values for each example.

MX11 multiplexer-11 function ($k=3$) (Wilson, 1987).

For each positive integer k , there exists a multiplexer function defined on a set of $k + 2^k$ attributes or bits. The function can be defined by thinking of the first k attributes as *address bits* and the last attributes as *data bits*. The function has the value of the data bit indexed by the address bits¹.

Attributes $x_{12} .. x_{32}$ have random values for each example.

PAR5 parity-5 function.

For each positive integer k , there exists an even parity function defined on a set of k attributes. The function has value *true* on an observation if an even number of attributes are present, otherwise it has the value *false*.

Attributes $x_6 .. x_{32}$ have random values for each example.

¹ In experiments with multiplexer function, Pagallo and Haussler (1989, 1990) classified an example as *positive* when the value of the function was 1 and negative for the value 0. However, according to the definition, both values: 0 and 1 are valid values of the function. Thus, each multiplexer function needs an additional bit to indicate whether the value of the function was properly assigned. For the sake of comparability of the results of the HCI method with other methods (Pagallo and Haussler; 1989, 1990; VanDeVelde, 1989) we used the same, simpler multiplexer function. This function learns how to "switch on" or "set to 1" the addressed line.

Target concept	Average % error			
	AQ15		HCI	
	1st Rank	100% match	1st Rank	100% match
DNF3	0.3	1.5	0.0	0.0
DNF4	0.2	11.5	0.0	0.0
MX11	0.0	0.0	0.0	0.0
PAR5	1.6	18.8	0.0	0.0

Table 8. The experimental results for different problems

No. of training examples	Average % error in learning target concept DNF4			
	AQ15		HCI	
	1st Rank	100% match	1st Rank	100% match
330	29.6	48.2	14.9	35.4
660	7.7	24.8	1.8	7.0
1320	1.8	16.4	0.0	0.0
1980	0.8	13.6	0.0	0.0
2640	0.2	11.4	0.0	0.0
3960	0.2	10.5	0.0	0.0

Table 9. The experimental results for different numbers of training examples in learning DNF4

Table 7 provides a short description of the test domains. The number of randomly generated training examples was 1650, 2640, 1600, and 4000 for DNF3, DNF4, MX11, PAR5 respectively, as in (Pagallo and Haussler, 1989). For each problem, 2000 random examples (independent from training examples) were used to test learned hypotheses.

4.2 Experimental results

Here we compare the performance of the AQ15 and AQ17-HCI programs. The rules generated by both programs were tested using the ATEST program (Reinke, 1984). ATEST views rules as expressions which, when applied to a vector of attribute values, evaluates to a real number. This number is called the *degree of consonance* between the rule and the event. The method for arriving at the degree of consonance varies with the settings of the various ATEST parameters. Rule testing is summarized by grouping the results of testing all the events of a single class. This is done by establishing equivalence classes among the rules that were tested on those events. Each equivalence class (called a rank) contains rules whose degrees of consonance were within a specified tolerance (τ) of the highest degree of consonance for that rank. When ATEST summarizes the results it

Target concept	Average % error					
	Decision TREES (*)		Decision LISTS (*)		Decision RULES (†)	
	REDWOOD	FRINGE	GREEDY3	GROVE	AQ15	HCI
DNF3	7.4	0.3	0.6	1.4	0.3	0.0
DNF4	24.9	0.0	0.0	7.8	0.2	0.0
MX11	13.1	0.0	0.5	3.9	0.0	0.0
PAR5	36.5	22.1	45.8	41.3	1.6	0.0

Table 10. The experimental results (*) from (Pagallo and Haussler, 1989, 1990) (†) 1st rank decisions

reports the percentage of *1st rank decisions* ($\tau=0.02$) as well as the percentage of *only choice decisions* (*100% match*) ($\tau=0$). In our experiments we used ATEST with its default parameters.

In learning DNF functions, the HCI method strongly outperformed AQ15 in terms of performance accuracy (Table 8). Table 9 shows that the HCI method requires a significantly smaller training set to precisely learn the DNF4 problem. These results are due to better descriptors used in expressing learned concepts both in the learning phase (relations already discovered and stored under new attributes make it possible for a deeper search for dependencies among training data) and the testing phase (match between an example and a more concise rule results in a higher degree of consonance (Reinke, 1984)).

4.3 Empirical comparison of HCI with other methods

Table 10 summarizes the results obtained in ten executions of the tested algorithms. The results for the REDWOOD, FRINGE, GREEDY3, and GROVE algorithms come from (Pagallo and Haussler, 1989, 1990).

AQ17 with hypothesis-driven constructive induction capabilities has learned all the target concepts from fewer than the assumed number of examples (Table 7).

REDWOOD and GROVE did not learn any concept with 100% accuracy. FRINGE and GREEDY3 learned three concepts but failed to learn the PAR5 concept. It is worth noting that the standard decision rule system AQ15 (without constructive induction abilities) learned all the concepts.

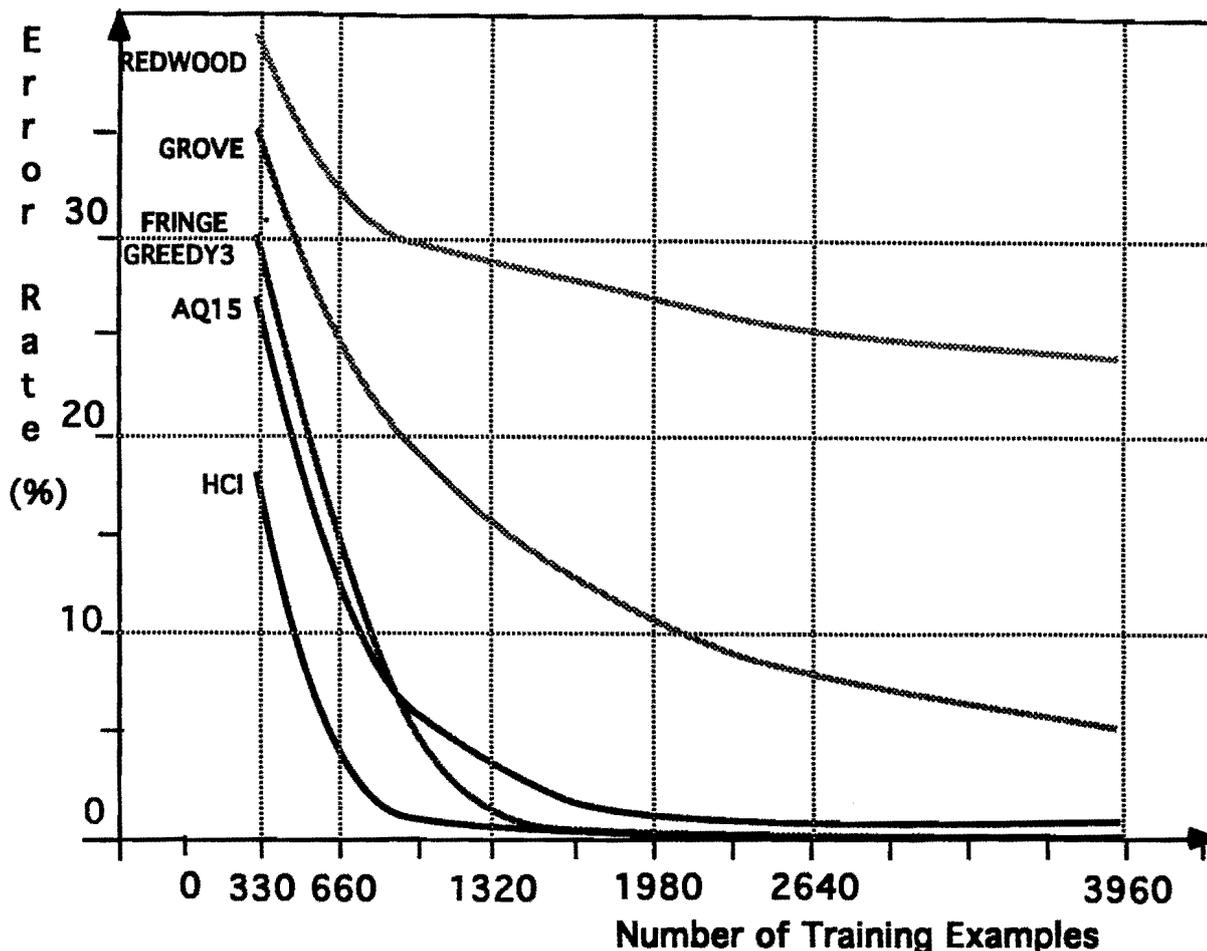


Figure 2. Learning curves for DNF4

5 Discussion

Both AQ15 rules and the HCI constructed rules provide a complete and consistent coverage of the input examples. Since HCI involves attributes constructed from AQ15 rules then a question arises: why AQ17-HCI produces higher accuracy on testing examples?

The answer seems to lie in the AQ15's strategy to generalize examples. The *extend-against* operator considers attributes one at a time and is limited to the initial attributes². This can be an essential obstacle in learning *hard* concepts in the context of preliminary description of a learned problem (Rendell & Seshu, 1990). Hard concepts are spread out all over the hypotheses space and require multiple covers.

² One way to address this problem can be a lookahead technique to detect interaction between attributes, but this increases computational cost (Rendell and Seshu, 1990)

In order to merge those regions and make the induction process simpler, a learning algorithm has to detect possible attribute interactions, and construct new attributes that capture those interactions. A closer look at AQ17-HCI shows that it does exactly this. By selecting subsets of the best performing rules as new attributes, the method takes advantage of already detected attribute interactions and uses them in converting a hard problem to an easier one by just enlarging the initial attribute set. Since new attributes combine initial interacting attributes, the systematic transformation in a hypotheses space support the *extend-against* operator in finding more accurate and effective hypotheses.

The results shown in Table 10 suggest that the all of the problems were *hard* for the standard decision tree algorithm REDWOOD. The reason is that the decision tree structure does not capture interactions between attributes. Only FRINGE which places conjunctions of initial attributes in the nodes of the decision tree, thus acting more like AQ15, was able to partially overcome those difficulties.

The AQ15 algorithm was able to find almost perfect solutions. This suggests that the structure of this algorithm supports solving this kind of problems.

6 Changes in the Representation Space

The HCI method changes the representation space by removing irrelevant attributes and/or adding new attributes. Attribute removal is meant in a different way than attribute selection in *selective* learning. The same selective algorithm may build different hypotheses depending on whether an attribute is present in the training set .

A domain description, expressed as a set of attributes, and a concept, expressed as a set of training examples, are similar in terms of potential noise. In a domain description, irrelevant attributes play the same role as noisy examples in training data. In both cases, removing noise from the training set speeds up and improves induction (Pachowicz, 1990; Vafaie & De Jong, 1991). However, without seeing the whole training sample, it is as hard to select a proper attribute, as it is to decide whether an example is noisy. One could argue that there are certain measures used by selective algorithms that allow the estimation of the utility of a particular attribute. Unfortunately, those measures, do not take into consideration how the attributes are interrelated, or whether attributes are ambiguous.

The solution for finding relevant attributes is analogous to that of finding relevant examples in a training set (Pachowicz, 1990). The HCI method incrementally removes from consideration those attributes that were not found in hypotheses, or were found in “small disjuncts” only. Thus, the

constructive learning algorithm incrementally adjusts representation space with the knowledge extracted from learned hypotheses. Rendell & Seshu (1990) also identify the need for two complementary measures for estimating the discriminatory ability of a set of attributes, μ_s and μ_x , to be applied during and after hypothesis induction.

The second constructive mechanism, attribute generation, also changes the representation space. It is worth noting however, that this is only data reformulation without changes in the dimensionality of the problem space, as opposed to attribute removal operation. In the HCI method, as well as in the other constructive induction methods, the constructed attributes have two or more values. This suggests that the problem space grows with additional attributes. However, there is only one value in effect for each of the instances from the problem space. In this case, the new attributes express relations between already existing attributes. In other words, since the new attributes are functions defined in a problem space, they do not expand the problem space.

Rendell & Seshu (1990) outline three levels of attribute construction: patterns (conjunctions of initial attributes), pattern classes (disjunctions of patterns), and pattern groups (descriptions of classes). Patterns are built from initial (ground) attributes, pattern classes are built from patterns, and pattern groups are built from pattern classes. Pattern construction is not discussed here, since it is an integral part of AQ algorithm (Michalski, 1973). The HCI method selects *best* patterns (rules) and builds pattern classes. Building pattern groups and the application of this method to recognize 24 classes of textures is addressed in (Bala, Michalski & Wnek, 1992).

7 Conclusion

The presented HCI method of constructive induction generates new attributes on the basis of an analysis of the hypotheses, rather than by directly combining different attributes. This way the search for new attributes is very efficient, although it is more limited in the repertoire of the attributes that can be constructed by direct, data-driven methods (Bloedorn and Michalski, 1991). In our experiments, the proposed method performed very favorably, in terms of performance accuracy, in comparison to methods employed in such programs as AQ15, REDWOOD, FRINGE, GREEDY3, and GROVE.

In the HCI method, new attributes correspond to subsets of best performing rules obtained in the previous iteration of the method. This is a real advantage of the method because it can easily handle problems with attributes of any type, such as Boolean, nominal, linear, as well as structured (where domains are hierarchies). The algorithm detects irrelevant attributes among those used in a primary description of a problem as well as those introduced during the attributes' generation

process. Initial and new attributes are examined according to classification abilities and new hypotheses building is based on the most relevant attributes.

The presented HCI method has shown to be effective in improving performance accuracy of an inductive system. It has also led to a simplification of learned descriptions.

On the other hand, generated attributes are rather complex. In future research, we plan to investigate attribute generation based on selected components of the best performing rules rather than the entire rule. This could potentially lead to both a rapid improvement of the accuracy, as well as to a greater simplification of the overall complexity of the hypotheses. We also plan to test the method on different types of learning problems in order to determine its strongest areas of applicability.

Acknowledgements

The authors thank Giulia Pagallo for the help in the design of the experiments, and Kenneth De Jong and George Tecuci for comments on the earlier version of this paper.

This research was done in the GMU Center for Artificial Intelligence. Research of the Center is supported in part by the Defense Advanced Research Projects Agency under the grants administered by the Office of Naval Research No. N00014-87-K-0874 and No. N00014-91-J-1854, in part by the Office of Naval Research under grants No. N00014-88-K-0226, No. N00014-88-K-0397 and No. N00014-91-J-1351, and in part by the National Science Foundation grant No. IRI-9020266.

References

Bala, J., Michalski, R.S. and Wnek, J., "The Principal Axes Method for Noise Tolerant Constructive Induction," *submitted to Machine Learning Conference*, Scotland, 1992.

Bloedorn, E. and Michalski, R.S., "Data-driven Constructive Induction in AQ17-DCI: A Method and Experiments," *Reports of Machine Learning and Inference Laboratory*, Center for Artificial Intelligence, George Mason University, 1992.

Bloedorn, E., Michalski, R.S. and Wnek, J., "AQ17 - A Multistrategy Constructive Learning System," to appear in *Reports of Machine Learning and Inference Laboratory*, Center for Artificial Intelligence, George Mason University, 1992.

Blumer, A., Ehrenfeucht, A., Haussler, D., Warmuth, M.K., "Occam's Razor," *Information Processing Letters*, **24**, pp. 377-380, 1987.

Cestnik, B., Kononenko, I. and Bratko, I., "ASSISTANT 86: A Knowledge Elicitation Tool for Sophisticated Users," *Proceedings of EWSL-87*, Bled, Yugoslavia, pp. 31-45, 1987.

Clark, P., and Niblett, T., "The CN2 Induction Algorithm," *Machine Learning*, **3**, pp. 261-284, 1989.

Drastal, G., Czako, G. and Raatz, S., "Induction in an Abstraction Space: A Form of Constructive Induction," *Proceedings of the IJCAI-89*, pp. 708-712, Detroit, MI, August 1989.

- Flann, N.S., and Dietterich, T.G., "Selecting Appropriate Representations for Learning from Examples," *Proceedings of AAAI-86*, Philadelphia, PA, pp. 460-466, 1986.
- Langley, P, Bradshaw, G.L. and Simon, H.A., "Rediscovering Chemistry With the BACON System," in *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell and T.M. Mitchell (eds.), Morgan Kaufmann, 1983.
- Lenat, D.B., "Learning from Observation and Discovery," in *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), Morgan Kaufmann, 1983.
- Matheus, C., "Feature Construction: An Analytic Framework and Application to Decision Trees," *Ph.D. Thesis*, University of Illinois, 1989.
- Michalski, R.S., "AQVAL/1 - Computer Implementation of a Variable-Valued Logic System VL1 and Examples of its Application to Pattern Recognition," *Proceedings of the First International Joint Conference on Pattern Recognition*, Washington, D.C., pp. 3-17, 1973.
- Michalski, R.S., "Pattern Recognition as Knowledge-Guided Computer Induction," Computer Science Dept., University of Illinois, Urbana, 1978.
- Michalski, R.S., Mozetic, I., Hong, J. and Lavrac, N., "The Multi-Purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains," *Proceedings of AAAI-86*, pp. 1041-1045, 1986.
- Mitchell, T.M., Utgoff, P.E. and Banerji, R., "Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics," in *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (eds.), Morgan Kaufmann, 1983.
- Muggleton, S., "Duce, and Oracle-Based Approach to Constructive Induction," *Proceedings of IJCAI-87*, pp.287-292, Milan, Italy, 1987.
- Pachowicz, P., "Application of Symbolic Inductive Learning to the Acquisition and Recognition of Noisy Texture Concepts," in *Applications of Learning and Planning Methods*, November 1991.
- Pagallo, G. and Haussler, D., "Two Algorithms that Learn DNF by Discovering Relevant Features," *Proceedings of the 6th International MLW*, Ithaca, pp. 119-123, 1989.
- Pagallo, G. and Haussler, D., "Boolean Feature Discovery in Empirical Learning," *Machine Learning*, 5, pp. 71-99, 1990.
- Quinlan, J.R., "Induction of Decision Trees," *Machine Learning* 1, pp. 81-106, 1986.
- Reinke, R.E., "Knowledge Acquisition and Refinement Tools for the ADVISE Meta-expert System," *Master Thesis*, University of Illinois, 1984.
- Rendell, L., "Substantial Constructive Induction Using Layered Information Compression: Tractable Feature Formation in Search," *Proceedings of IJCAI-85*, pp. 650-658, 1985.
- Rendell, L. and Seshu, R., "Learning Hard Concepts Through Constructive Induction: Framework and Rationale," *Computational Intelligence*, 6, pp. 247-270, 1990.
- Rivest, R., "Learning Decision Lists," *Machine Learning* 2, pp. 229-246, 1987.

Schlimmer, J.C., "Incremental Adjustment of Representations in Learning," *Proceedings of the Fourth International Machine Learning Workshop*, pp.79-90, 1987.

Van de Velde, W., "IDL, or Taming the Multiplexer," *Proceedings of the 4th EWSL-89*, France, pp. 211-225, 1989.

Vafaie, H. and De Jong, K., "Improving the Performance of a Rule Induction System Using Genetic Algorithms," in *Proceedings of the First International Workshop on Multistrategy Learning*, pp. 305-315, Harpers Ferry WV, 1991.

Wilson, S.W., "Classifier Systems and the Animat Problem," *Machine Learning*, 2, pp. 199-228, 1987.

Wnek, J. and Michalski, R.S., "Hypothesis-driven Constructive Induction in AQ17: A Method and Experiments," *Proceedings of the IJCAI-91 Workshop on Evaluating and Changing Representations*, K. Morik, F. Bergadano and W. Buntine (Eds.), pp. 13-22, Sydney, Australia, 1991.

Wnek, J., "Transformation of Version Space with Constructive Induction: The VS* Algorithm," *submitted to the Machine Learning Conference*, Aberdeen, Scotland, 1992.