HYPOTHESIS-DRIVEN CONSTRUCTIVE INDUCTION IN AQ17: A METHOD AND EXPERIMENTS

by

J. Wnek
R. S. Michalski

In Reports of Machine Learning and Inference Laboratory, Center for Artificial Intelligence, George Mason University, MLI 92-02, January 1992.

HYPOTHESIS-DRIVEN CONSTRUCTIVE INDUCTION in AQ17-HCI: A Method and Experiments

J. Wnek and R. S. Michalski

MLI92-2

HYPOTHESIS-DRIVEN CONSTRUCTIVE INDUCTION IN AQ17-HCI:

A Method and Experiments

Janusz Wnek and Ryszard S. Michalski

Artificial Intelligence Center
George Mason University
Fairfax, VA 22030

MLI 92-2

January 1992

Hypothesis-Driven Constructive Induction In AQ17-HCI: A Method and Experiments

Abstract

A method for constructive induction is described that generates new problem-relevant attributes by analyzing and abstracting iteratively created inductive concept hypotheses. The method, called HCI (hypothesis-driven constructive induction), first creates a set of initial rules from the training examples using a standard AQ rule-learning algorithm. The rules generated are then analyzed and evaluated according to a *rule quality* criterion. The analysis determines which of the original attributes are irrelevant, and reduces the original representation space. It also determines the best-performing rules for each decision class, and assembles them into sets that are assigned names, and treated as new attributes. These new attributes are then used to reformulate the training examples from the previous step, and the whole inductive process is repeated. This iterative process stops when the performance accuracy of the last generated rules exceeds a predefined performance threshold. In several experiments on learning various well-defined transformations, the AQ17-HCI system implementing the method consistently and significantly outperformed, in terms of the predictive accuracy, the AQ15 rule learning system, GREEDY3 and GROVE decision list learning systems, and REDWOOD and FRINGE decision tree learning systems.

Key words: Concept learning, constructive induction, decision rules, diagrammatic visualization

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 Center for Artificial Intelligence at George Mason University. The Center's research is supported in part by the Defense Advanced Research Projects Agency under the grant administrated by the Office of Naval Research No. N00014-91-J-1854, in part by the Office of Naval Research under the grant No. N00014-91-J-1351, and in part by the National Science Foundation under the grant No. IRI-9020226.

1. Introduction

Most systems for inductive learning from examples create concept descriptions in terms of the attributes that are selected from those present in the original examples. Well-known examples of such *selective* inductive systems include AQVAL/1 (Michalski, 1973), ID3 (Quinlan, 1983), ASSISTANT (Cestnik et al., 1987), and CN2 (Clark and Niblett, 1989). In contrast to such systems, a *constructive* inductive learning system generates new attributes, or generally *descriptors*, and employs them in the hypothesized description. These descriptors can be attributes, predicates, terms, operators, etc., and are supposed to be more relevant to the learning problem than those initially given. A constructive learning algorithm thus performs a *problem-oriented transformation of the knowledge representation space* (Michalski, 1978; Rendell, 1985; Matheus, 1989; Drastal, Czako & Raatz, 1989; Wnek & Michalski, 1991).

The idea and the name *constructive induction* was first proposed by Michalski (1978), and implemented in the INDUCE-1 system for learning structural descriptions from examples (Larson & Michalski, 1977). INDUCE-1, and subsequent versions, e.g., INDUCE-4 (Bentrup, Mehler & Riedesel, 1987), used various *constructive generalization* rules and procedures to generate new problem-oriented descriptors (Michalski, 1983). These descriptors were then employed together with the original ones in the process of induction.

Subsequently, a number of other systems that exhibit certain constructive induction capabilities have been developed. These systems employ different strategies for constructing new descriptors. Based on the major trend and a source for generating new descriptors, the existing systems can be divided into four categories: data-driven, hypothesis-driven, knowledge-driven, and multistrategy-driven, constructive induction. The border line between the strategies is not precise, the same way, as there is no precise distinction between empirical learning and knowledge-based learning, and where the annotation is a matter of proportion and primary source of power.

- **Data–Driven Constructive Induction Systems (DCI).**
  These systems analyze and explore the input data, specifically, interrelationships among attributes, examples, concepts, etc., in order to determine new descriptors. Examples of such systems are:

BACON represents information at varying levels of descriptions, with higher levels summarizing the levels below them. BACON's constructive abilities rely on detecting interdependencies between numerical attributes (Langley, Bradshaw & Simon, 1983).

PLS0 (Probabilistic Learning System) creates new attributes from initial attributes using a form of conceptual clustering performed on three levels of abstraction (Rendell, 1985).

Wyl learns structural descriptions of checkers and chess concepts by first mapping the training examples from performance representation into a learning representation, generalizing them there, and converting the learned concept back into a performance representation for efficient recognition (Flann & Dietterich, 1986).

STAGGER generates new discrete attributes in order to utilize continuously valued attributes. It also increases the representational capabilities by forming Boolean combinations of attribute values (Schlimmer, 1987).

AQ17-DCI system applies a variety binary and multi-argument operators to original attributes in the search for new problem-oriented attributes. Best performing constructed attributes are added to the original attribute set, and the whole set is employed in the process of inductive generalization (Bloedorn & Michalski, 1991).

FCE (Factored Candidate Elimination) algorithm assumes a set of initial representation spaces. After detecting inconsistency in all version spaces, it shifts to a larger size representations. The new representation is a product of the existing ones (Carpineto, 1992).

- **Hypothesis-Driven Constructive Induction Systems (HCI).**
  Such systems determine new attributes by analyzing recursively generated inductive hypotheses. Examples of systems in this category are:

BLIP invents new concepts on the basis of rule exceptions which cannot be defined in terms of the given representation. Hypotheses and the negation operator are used to construct new concepts (Emde, Habel & Rollinger, 1983; Morik, 1989; Wrobel, 1989).

FRINGE was designed for decision trees to avoid the duplication of tests in a decision tree. New attributes are constructed from "fringes" of the tree and are in conjunctive form (Pagallo & Haussler, 1990).

KLUSTER introduces new relations or concepts if another concept cannot be characterized without it. A definition of the demanded concept or relation is learned using initial examples (Kietz & Morik, 1991).

- **Knowledge-Driven Constructive Induction Systems (KCI).**
  These systems apply expert-provided domain knowledge to construct and/or verify new attributes. Examples of systems in this category are:

AM (Automated Mathematician) program changes its representation by using a set of predefined heuristics, on three levels of abstraction: (1) by defining a new concept, (2) by creating new attribute of a concept, (3) by shifting between the schemes (Lenat, 1977, 1983).

AQ15 applies rules for constructing new attributes that were specified by an expert in the arithmetic and/or logic form (Michalski, Mozetic, Hong & Lavrac, 1985).

MIRO applies an expert specified set of rules (a domain theory) to construct an abstraction space and then to perform induction over this space (Drastal, Czako & Raatz, 1989).

- **Multistrategy Constructive Induction Systems (MCI).**
  These systems combine different approaches and methods for constructing new descriptors. Examples of systems in this category are:

INDUCE-1 (DCI & KCI) performs selective and constructive induction simultaneously. It uses a variety of rules and procedures for generating additional attributes ("meta-attributes") from structural descriptions of the training examples (KCI), along with a qualitative dependency determination between attributes (DCI) (Larson & Michalski 1977; Michalski, 1978, 1983).

STABB (DCI & HCI) uses two procedures to Shift To A Better Bias. The *least disjunction* procedure changes the representation by examining only the training examples and the current description language (DCI). The *constraint back-propagation* procedure builds new representation based on hypotheses (operator sequences) verified by LEX's critic (HCI). STABB was incorporated into the existing LEX program (Mitchell, Utgoff & Banerji, 1983) to provide LEX with constructive abilities (Utgoff, 1984, 1986).

Duce (HCI & KCI) suggests domain features to a user (or oracle) on the basis of a set of example object descriptions (from an input or hypothesized) and six transformation operators. Such inductive transformations are tested against an oracle which ensures the validity of any transformation (Muggleton, 1987).

CIGOL (HCI & KCI) (LOGIC backwards) is based on inverting the mechanism of resolution in first-order Horn clause representation. New predicates, that play role of sub-concepts (or missing premises), are generated from input or hypothesized examples of a high-level predicate by applying the *intra-construction* operator (HCI). A user may name the concept (predicate) or reject the proposed definition (KCI) (Muggleton & Buntine, 1988).

CITRE (HCI & KCI) (Constructive Induction on decision TREes) learns decision tree and uses it to construct new features. The use of domain knowledge is limited to simple facts (ground literal clauses) that define legal relationships between constructive operands (Matheus, 1989).

NeoDisciple (HCI & KCI) introduces new concepts in the form of example explanations provided by an expert (KCI) (DISCIPLE, Tecuci & Kodratoff, 1990) , and creates new features based on the similar definitions in the knowledge base to reduce the inconsistency in the learned rules (HCI) (Tecuci, 1992; Tecuci & Hieb, 1992).

CLINT (HCI & KCI) (Concept-Learning in an INTeractive way) learns concepts using an inductive and/or abductive method. If the learned rules match a predefined schemata then a

user is presented with the partially instantiated schema (HCI). The user may name the concept (predicate) or reject the proposed definition (KCI) (De Raedt & Bruynooghe, 1991).

AQ17 (DCI, HCI & KCI) integrates in a synergistic way constructive features of AQ15, INDUCE, AQ17-DCI, and AQ17-HCI (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 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).

The goal of constructive induction may be to simplify the generated concept descriptions, or to improve their predictive accuracy. The simplicity of a description is measured by computing the number of attributes and the number & type of operators used in the description. The predictive accuracy is measured by applying the hypothesis 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. This is done by supplementing a generation of an inductive hypothesis with various changes in the representation space. The changes are based on the analysis of the hypothesis and discovered patterns. 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, where the need and the character of required changes is detected by analyzing the hypothesis. In AQ17, changes in the representation space can be data-, hypothesis-, and knowledge-driven (Bloedorn, Michalski & Wnek, 1992). Here we describe the HCI part only.

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 can be of different types, such as symbolic, numerical, 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's output is a set of general classification rules for each class. The rules are equivalent to expressions in disjunctive normal form with internal conjunction. These rules cover all 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 is repeated. 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 HCI method, the AQ algorithm for searching the problem space is combined with processes that change the representation space. The AQ generated hypothesis serves as intermediate knowledge for proposing problem-oriented changes in the representation and making the following search more accurate and more time effective.

2. Method Description

The proposed method for hypothesis-driven constructive induction (HCI) combines a standard inductive rule learning algorithm AQ with a method for iteratively generating new attributes based on the analysis of the hypotheses obtained in the previous iteration. At each iteration, the method changes the representation space with respect to the set of attributes available for the given iteration.

Although the method was applied and implemented using the AQ rule learning algorithm, it can potentially be combined with any type of rule learning method. It thus represents a universal new

approach to constructive induction of rule-based descriptions. The primary purpose of the method is to increase the predictive accuracy of the generated hypothesis, rather than to minimize the *total complexity* of the description. By the total complexity of the description we mean a measure that combines the complexity of not only the final hypothesis (which in the method is often very simple), but also the complexity of all the intermediate descriptions. Below is a diagram illustrating the method.

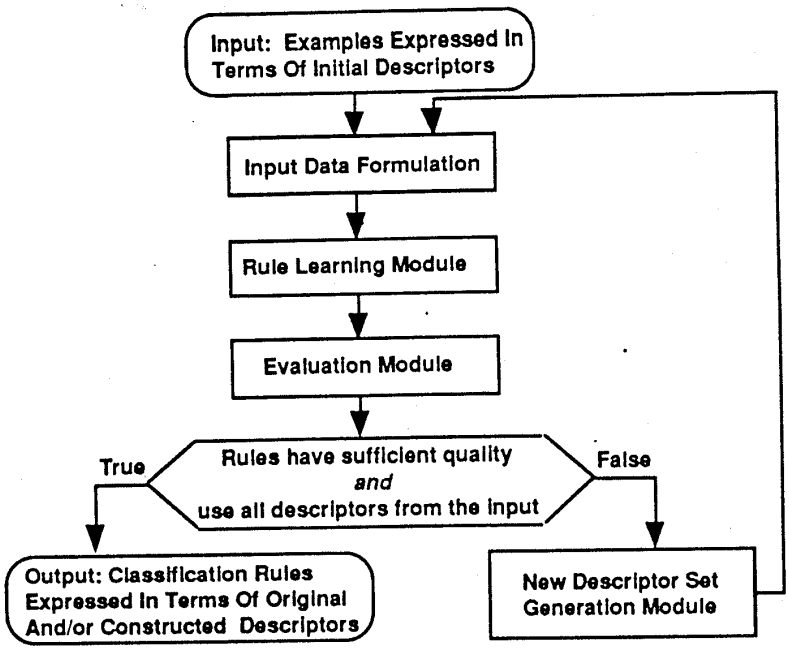

Figure 1. Hypothesis-driven constructive induction learning algorithm.

The HCI algorithm starts with creating a set of classification rules from given examples of concepts using a selective algorithm (Figure 1). (At this stage, the examples are formulated using all initially available descriptors.) This step is performed in the *rule learning module*. The rules are next examined in the *evaluation module*, in order to detect the need for a change in the representation space. If such a need is detected, a new descriptor set is generated, and the input data is reformulated according to those changes. Rule generation starts again. This iterative process stops when correct rules were generated using relevant set of descriptors, or the repertoire of representational changes was exhausted.

One way of evaluating learned description is to check completeness and consistency of the learned rules. In the case of the VL1 representation and the AQ-type learning strategy however, the completeness and consistency with the training data is always guaranteed (in extreme, a concept can be described as a disjunction of all training examples). Therefore, two other criteria are used: (1) an expected degree of generalization, and (2) a performance accuracy. The degree of generalization is an easy-to-define and to test, user-supplied recommendation of an expected number of rules in the hypothesis. If the learning algorithm fails to express a concept to be learned, it is assumed that the representation is not adequate and the representation change is made. The method may use the second criterion, the performance accuracy, if the first one is not specified by a user (Arciszewski, Dybala & Wnek, 1992).

However, even if the rule quality is sufficient, the change in the representation may be requested based on the quality of attributes used in the hypothesis. This criterion stems from the fact that the same selective algorithm may build different hypotheses depending on attributes present in representing data. 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.[1]

In order to select a representative set of attributes, attribute quality is measured. However, attribute utility measures based on the training data do not take into consideration how the attributes are interrelated or whether attributes are relevant to the learning task unless a complex search is performed. Attribute evaluation based on a hypothesis analysis makes the proper evaluation plausible since the hypothesis already reflects a mapping between the training set and the representational capabilities of a learning program (representational bias).

In general, the transformation of a descriptor set may involve generalization or specialization. One may generalize the set, by dropping (removing) attributes or narrowing their domains, or specialize the set, by adding new attributes or extending domains of existing attributes. More specifically, construction of a new attribute set may involve searching for one of the four types of patterns: null-pattern, condition-pattern, rule-pattern, and rule-set-pattern. Null-pattern of an attribute is detected when the attribute is used in few or no conditions of the hypothesis. In this case, the attribute is removed. Condition-pattern is detected, if the analyzed hypothesis contains conditions that repeatedly involve the same values of a single attribute. The third type is defined by conjunctions of conditions (rules). This is called a rule-pattern. The fourth type is found in sets of rules, i.e., disjunctions of rules, and is called a rule-sets-pattern. In multiple concept learning, it might be feasible to find patterns across class descriptions, *class-patterns*.

Table 1 shows examples of possible attributes constructed in a simple, three-attribute domain based on different patterns found in hypotheses.

Table 1. Examples of attributes constructed from different patterns.

A domain	
x = 1..100	(Numeric attribute x has integer values from 1 to 100)
y = small, medium, large	(Symbolic attribute y)
z = white, red, blue, green, black	(Symbolic attribute z)
Examples of constructed attributes	
c1 <:: (z = blue, red, white)	(condition-pattern)
c2 <:: (x = 20) & (y = large)	(rule-pattern)
c3 <:: ((x = 75,100) & (y = small)) or (x = 7)	(rule-set-pattern)

The mapping in an attribute definition can result in assigning real or discrete values from the closed interval 0 to 1. The simplest mapping assigns value 1 if the pattern is satisfied and 0 otherwise. More sophisticated measures may express the distance between an instance and a pattern in real values. For example, Bala, Michalski and Wnek (1992) use rule-pattern attributes with a real valued similarity measure in the task of texture recognition.

There may be many strategies to schedule attribute generation of various pattern types. One strategy could try new attributes based on all types of patterns generated once from the initial hypothesis. However, at the same time the new attributes are introduced to the descriptor set, the representation becomes more redundant. Moreover, the more new attributes the more redundancy in the representation space. This correlation is intuitive and easy to prove. Also, initial testing of this strategy supplemented with the attribute removal mechanism has shown that the final classification rules have strong tendency to deteriorate. The explanation may follow from the fact that in such a case, the primary discrimination power of initial attributes is dispersed among many newly generated attributes.

In order to design reliable strategy, each type of pattern attributes has to be tested and its properties recognized. The strategy has to be based on small and strongly justified changes in the

[1] Pachowicz (1990) improves learned descriptions by removing those examples from the training set that are hypothetically noisy. Vafaie & De Jong (1991) use a genetic algorithm to preselect a set of attributes for AQ15. Learning in the modified domain is faster and gives better concept descriptions in terms of predictive accuracy.

representation space. The selection of the type of attribute pattern can be based on the structure of the hypothesis and the sequence of pattern types already applied.

In Section 3, the implementation of the HCI method, with respect to attributes constructed from patterns found in rule-sets, is presented in the AQ17-HCI rule learning system. In Section 4, an exemplary problem is solved using the system. The changes in the representation space are visualized using diagrams. In Section 5, experiments in four domains are presented. AQ17-HCI is compared with two selective, inductive learning algorithms, as well as three HCI algorithms utilizing the second type of patterns (patterns in rules). Finally, Section 6 summarizes the method.

3. Method Implementation

Learning in the *rule learning module* is performed by the AQ15 program. In the first place, if the change in the representation is requested, the HCI method performs abstraction of the representation space by identifying and removing those attributes that were not found in hypotheses at all, or that were found in rules that cover marginal number of training examples, "small disjuncts," only.

If the learned hypothesis is still not accurate then new attributes are introduced. In order to do this, the learning system takes advantage of induced concept descriptions by abstracting them and forming new, problem-relevant descriptors.

Both the changes in the representation are totally separated from the induction from examples. Especially the process of creating new attributes relies on the completed rule induction from examples and therefore it would not be feasible to perform it in parallel with induction.

The heuristic for constructing attributes includes extracting a part of a classification hypothesis that contains *best rules*. This is done by sorting all rules from the output hypothesis according to a simple utility measure, *tu*, and selecting the minimum number of rules with the highest utility that satisfy the inequality (1).

$$\frac{\sum tu_{best}}{\sum tu_{all}} \geq TH1 \quad \ldots\ldots (1)$$

$$tu = t + (\lambda * u) \quad \ldots\ldots (2)$$

Since rules in the hypothesis may overlap over the same training examples, the *tu* rule utility function depends on the number of positive examples covered by the rule (t), as well as the number of positive examples that are uniquely[2] covered by the rule (u). The λ constant determines the importance of the exclusiveness of the coverage. Both λ and TH1 were determined empirically and preset to 2 and 0.67, respectively. With $\lambda = 2$, a preference is given to rules with higher uniqueness. For TH1 = 0.67, the constructed attribute will at least cover 2/3 of training examples of a given concept. Assuming that noise and exceptions are usually covered by rules with low *t* and *u* weights, then the constructed attribute covers only fundamental portion of the learned concept. However, the best solution for setting TH1 parameter would be a dynamic evaluation based on knowledge about noise level and the confidence in the learned hypothesis.

The new attribute descriptions involve attributes from the current descriptor set and operators defined within a given representational formalism. Since the implementation of AQ17-HCI program is done using the VL1 variable-valued logic formalism (an extension of propositional logic) thus the constructed attributes are VL1 logical expressions. Therefore, with the new attributes, the system introduces conceptual changes within the representation rather then shifting to another representational formalism. The whole instance space is conceptually partitioned into regions that exhibit a *strong* concept membership and an *undecided* concept membership. For example, in the case of binary concepts, the new attributes partition the instance space into three regions, instances with a *strong positive* concept membership, instances with a *strong negative* concept membership, and instances with an *undecided* concept membership. Value "1" of such attribute indicates instances belonging to the fundamental part of the given concept, and value "0"

[2] Uniqueness means that no other rule in the hypothesis covers the same examples.

characterizes *not decided* instances, e.g., exceptions, noise. Real type values could be used to express various degrees of similarity to defined subconcepts.

The system is able to use the defined subconcept and its negation in the process of building concept descriptions. It is done through the reformulation of the training data using all relevant original attributes and newly generated attributes. For each training example the values of the new attributes are calculated by evaluating the VL1 expression characterizing the new attribute.

The introduction of new attributes in the form of subconcepts made two implicit extensions to the AQ17-HCI representational capabilities that were not present in AQ15:

(1) Rule-set–to–condition operator.
The operator substitutes a DNF expression with an attribute value. For example, following the domain description from Table 1, the system is able to create and use the following condition:

(c3 = 1)
that stands for (((x = 75,100) & (y = small)) or (x = 7)).

(2) Rule-set-negation–to–condition operator.
The operator substitutes a negated DNF expression with an attribute value. From the conceptual point of view, this operator plays important role in negating the created subconcepts. For example:

(c3 = 0)
that stands for: (not (((x = 75,100) & (y = small)) or (x = 7)))
and is equivalent to: (((x ≠ 75,100) or (y ≠ small)) & (x ≠ 7))

From the 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 induction is 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.

4. Example

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*.

Figure 2 presents the MX11 concept graphically using a method for *diagrammatic visualization*. This method employs a *General Logic Diagram* (GLD) which is a planar representation of a multi-dimensional space spanned over multivalued discrete attributes[3] (Michalski, 1973; Wnek and Michalski, 1992). Each cell in the diagram represents a combination of the attribute values, e.g., a concept example. Concepts are represented as sets of cells. They are depicted in the diagrams by shaded areas. For example, the MX11 concept is described by eight rules listed at the bottom of Figure 2. For easy recognition, each rule in the diagram was differently shaded.

The diagrammatic visualization method permits one to display the target and learned concepts, individual steps in a learning process, and the errors in learning. The set of cells representing the target concept (the concept to be learned) is called *target concept image (T)*. The set of cells representing the learned concept is called *learned concept image (L)*. The areas of the *target concept* not covered by the *learned concept* represent *errors of omission (T \ L)*, while the areas of

[3] The system DIAV implementing the visualization method (Wnek and Michalski, 1992) permits one to directly display description spaces up to 10^6 cells (e.g., about twenty binary attributes). Larger spaces can also be displayed but their representations have to be projected to subspaces.

the learned concept not covered by the target concept represent *errors of commission (L \ T)*. The union of both types of errors represents the *error image*. In the diagrams, errors are marked by slanted lines.

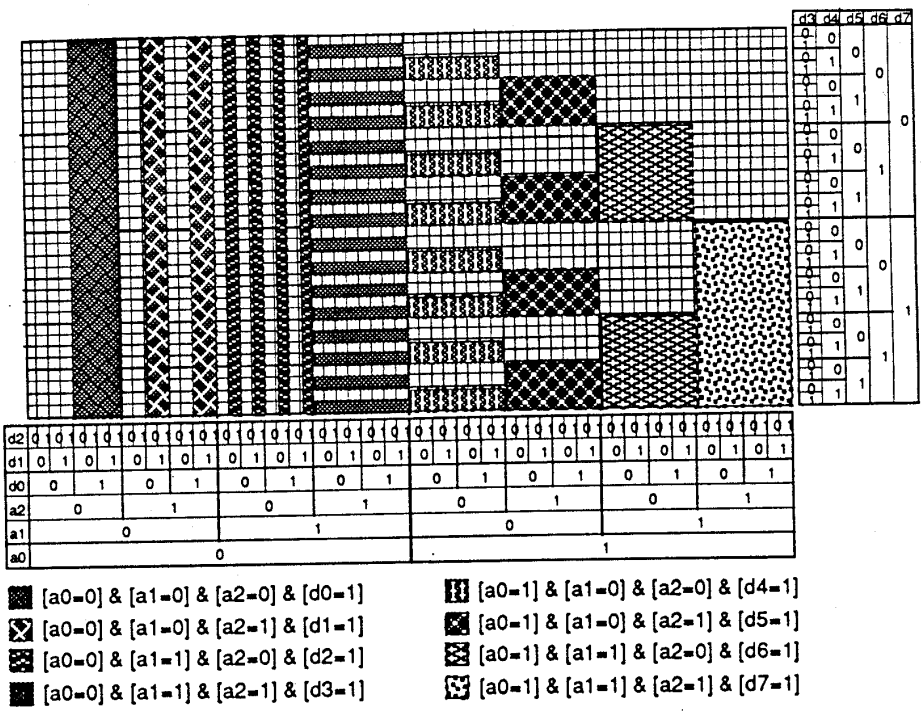

Figure 2. Diagrammatic visualization of the rules constituting MX11 concept.

Figure 3 explains the meaning of various cases in concept visualization. Concept images are represented in the diagrams by shaded areas, e.g. Figure 3-A. If the target and learned concepts are visualized in the same diagram, then the shaded areas represent learned concept (Figure 3-B). Error image is represented by slanted areas. It is easy to distinguish between errors of omission and errors of commission. Since errors of commission are part of a learned concept, the corresponding areas on the diagram are both shaded and slanted. Errors of omission are not part of the learned concept thus the corresponding slanted areas remain white in the background. The location of the target concept is implicitly indicated by correctly learned concept and errors of omission. The parts of the target concept that were correctly learned are shaded only.

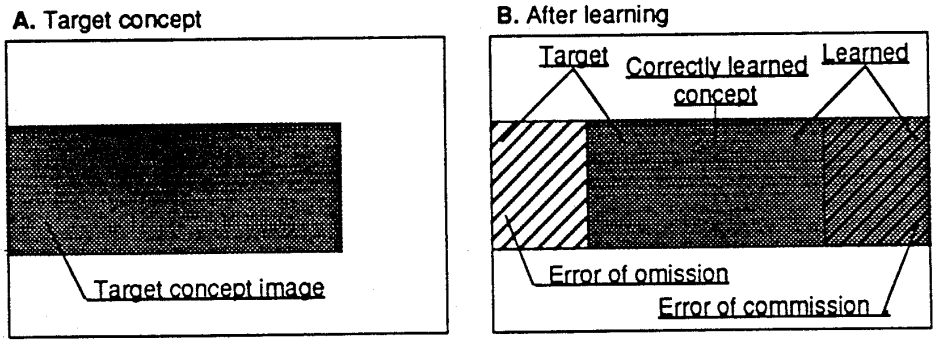

Figure 3. Interpretation of various areas in the diagrammatic visualization.

e MX11 function has the value of the data bit indexed by the address bits. In the experiment,
: input examples were encoded in terms of 11 binary attributes. Thus, the description space
9 ntains 2048 elements. The training set had 64 (6%) of the positive examples and 64 (6%) of the
:gative examples. Table 2 shows a sample of the positive and the negative examples. The
tributes a0, a1, a2 describe address lines, and d0-d7 describe data lines.
From these examples, the rule generation module produced non-overlapping hypotheses of the
:orrect (Pos-Class) and incorrect (Neg-Class) behavior of the multiplexer. The classification rules
supplemented with the *total* and *uniqueness* weights are shown in Table 3.

Table 2. A part of the set of training examples

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

Table 3. The rules induced by AQ15 from the training examples.

Pos-Class if

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

Neg-Class if

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

Figure 4 presents the AQ15 learned concept description in the context of the target concept. The total number of errors measured over the whole representation space is 299, which gives 15% error rate. (The error rate for overlapping covers is 20%).

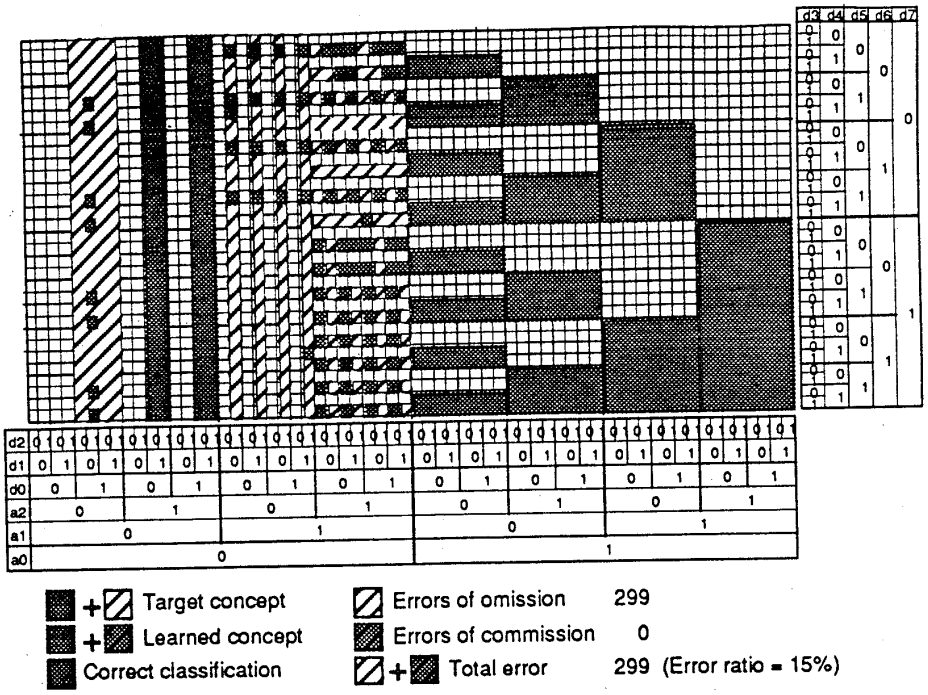

Figure 4. Visualization of the MX11 (Pos-Class) concept learned by AQ15.

Pos-Class and Neg-Class are hypotheses in the k-DNF form. Each rule in the hypotheses is accompanied with *t* and *u* weights that represent total and unique numbers of training examples covered by a rule. From these hypotheses, the best rules were selected to constitute the candidate attributes P1 and N2 (Table 4). The new attributes are named after the concept descriptions they were abstracted from, and consecutive numbers. Table 5 shows the definition of the new attributes and Figure 5 shows the coverage of the instance space done by the new attributes.

Table 4. Best rules according to the formula (1).

Pos-Class		**Neg-Class**	
1. (a0=1) & (a1=1) & (a2=0) & (d6=1)	(tu:33)	1. (a0=1) & (a1=1) & (a2=1) & (d7=0)	(tu:39)
2. (a0=0) & (a1=0) & (a2=1) & (d1=1)	(tu:33)	2. (a0=0) & (a2=0) & (d2=0)	(tu:36)
3. (a0=1) & (a1=0) & (a2=1) & (d5=1)	(tu:30)	3. (a0=0) & (a1=0) & (a2=1) & (d1=0)	(tu:30)
4. (a0=1) & (a1=1) & (a2=1) & (d7=1)	(tu:30)	4. (a1=0) & (a2=0) & (d1=1) & (d4=0)	(tu:27)
5. (a0=1) & (a1=0) & (a2=0) & (d4=1)	(tu:27)		
$\Sigma\, tu_{best-1} = 126$; $\Sigma\, tu_{all} = 191$; $[(\Sigma\, tu_{best-1}) / (\Sigma\, tu_{all})] = 0.66$		$\Sigma\, tu_{best-1} = 105$; $\Sigma\, tu_{all} = 192$; $[(\Sigma\, tu_{best-1}) / (\Sigma\, tu_{all})] = 0.55$	
$\Sigma\, tu_{best}=153$; $\Sigma\, tu_{all} =191$; $[(\Sigma\, tu_{best}) / (\Sigma\, tu_{all}) = 0.80$		$\Sigma\, tu_{best} = 132$; $\Sigma\, tu_{all} = 192$; $[(\Sigma\, tu_{best}) / (\Sigma\, tu_{all}) = 0.69$	

Table 5. The definition of the constructed attributes P1 and N2.

P1 = 1 if	**N2 = 1 if**
1. (a0=1) & (a1=1) & (a2=0) & (d6=1) or	1. (a0=1) & (a1=1) & (a2=1) & (d7=0) or
2. (a0=0) & (a1=0) & (a2=1) & (d1=1) or	2. (a0=0) & (a2=0) & (d2=0) or
3. (a0=1) & (a1=0) & (a2=1) & (d5=1) or	3. (a0=0) & (a1=0) & (a2=1) & (d1=0) or
4. (a0=1) & (a1=1) & (a2=1) & (d7=1) or	4. (a1=0) & (a2=0) & (d1=1) & (d4=0)
5. (a0=1) & (a1=0) & (a2=0) & (d4=1)	
P1 = 0 otherwise	**N2 = 0 otherwise**

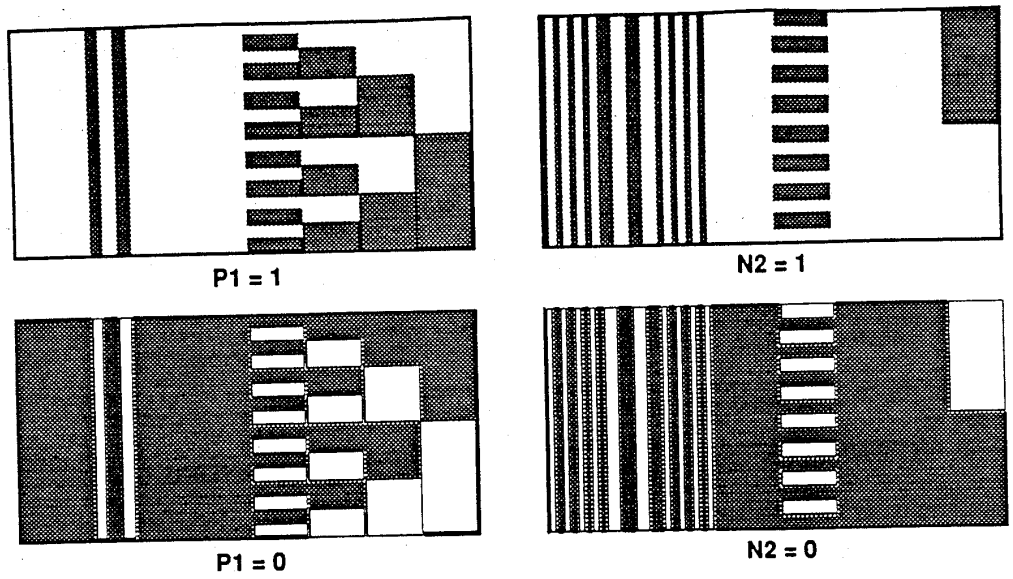

Figure 5. Images of the constructed attributes.

Once the new attributes are created, they are used to reformulate the training examples (Table 6). For each old training example the new P1 and N2 attribute values has been added. Note that, if the new attribute originated in the given class then it mostly has value "1" assigned, and "0" value otherwise. After the reformulation is done the whole inductive process is repeated. Table 7 presents newly inducted rules.

Table 6. The part of the reformulated training set.

Positive examples											**Negative examples**									
a0	**a1**	**a2**	**d0**	**d1**	**d2**	**d3**	**d4**	**d5**	**d6**	**d7**	**P1**	**N2**	**a0**	**a1**	**a2**	**d0**	**d1**	**d2**	**d3**	**d4**	**d5**	**d6**	**d7**	**P1**	**N2**
0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1	1	1	1	0	0	1
0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1
0	1	0	1	1	1	1	1	1	1	0	0	0	0	1	0	1	1	0	1	1	1	1	0	0	1
1	0	1	0	0	0	0	1	1	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	1	1	1	1	0	1	1	0	0	0	0	0	1	0	0	0	0	0
.....																									

Table 7. Decision rules with the constructed attributes.

Pos-Class if	
1. (P1=1) or	(t:51, u:51)
2. (a0=0) & (d3=1) & (P1=0) & (N2=0) or	(t:11, u:11)
3. (a0=0) & (a1=1) & (a2=0) & (d3=0) & (P1=0) & (N2=0)	(t:2, u:2)

Neg-Class if	
1. (N2=1) or	(t:44, u:44)
2. (a0=1) & (P1=0) & (N2=0) or	(t:12, u:12)
3. (a0=0) & (a1=1) & (a2=1) & (d3=0) & (P1=0) & (N2=0) or	(t:5, u:5)
4. (a0=0) & (a1=0) & (a2=0) & (d0=0) & (d3=0) & (P1=0) & (N2=0)	(t:3, u:3)

As expected, the new attributes were used in the output hypotheses in both Pos and Neg classes. We can observe that large portions of training examples were covered by the rules (P1=1) in the Pos-Class, and (N2=1) in the Neg-Class. Figure 6 summarizes the new learning task in the changed representation space.

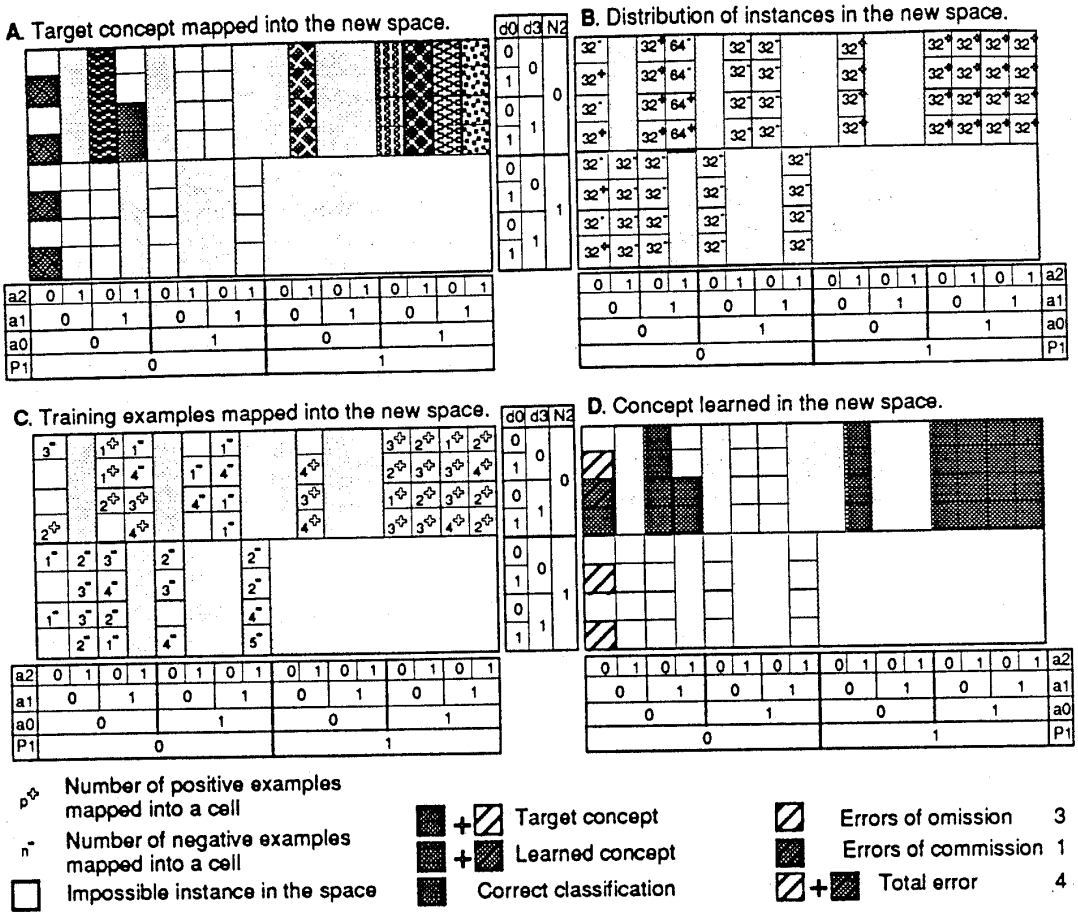

Figure 6. Learning the concept MX11 in the changed representation space.

The MX11 target concept image is shown in Figure 6-A after mapping into the new representation space. For an easy identification, areas that correspond to those in Figure 2 are marked with the same pattern. There exists interdependency between P1, N2 and original attributes, therefore the space spanned over that area excluded *impossible* combinations of instance descriptions. For example, the whole area described by the rule ((P1=1) & (N2=1)) is impossible since the intersection of the two attributes is empty (see P1, N2 definitions in Table 5 or their visualization in Figure 5). Figure 6-B shows all instances of the MX11 concept mapped into the new space. One cell in the new space represents 32 or 64 original examples depending on the rule describing the new cell. Figure 6-C shows the training examples in the new space. Figure 6-D shows the final concept learned. The learned concept still does not cover exactly the target concept (3 errors of omission and 1 errors of commission) but it gives a better performance accuracy. The learning could be further improved if the generalization were prohibited over impossible areas. Instead of producing the rule (a0=0) & (d3=1) & (P1=0) & (N2=0) to cover the four positive examples listed below, the system would be forced to generate more specific rule.

1. (a0=0) & (a1=1) & (a2=1) & (d0=0) & (d3=1) & (P1=0) & (N2=0)
2. (a0=0) & (a1=1) & (a2=1) & (d0=1) & (d3=1) & (P1=0) & (N2=0)
3. (a0=0) & (a1=1) & (a2=0) & (d0=0) & (d3=1) & (P1=0) & (N2=0)
4. (a0=0) & (a1=0) & (a2=0) & (d0=1) & (d3=1) & (P1=0) & (N2=0)

The rule would not cause the only commission error. The only remaining uncovered example (4) could be generalized to form the rule: (a0=0) & (a1=0) & (a2=0) & (d0=1) & (P1=0), and therefore eliminate the three omission errors. The ongoing research investigates ways of improving induction in spaces with impossible instances. Another important issue that needs further research is utilization of the information about the numbers of original examples mapped into new examples.

The final hypothesis produced by AQ17-HCI was tested against the testing set. The result was 94% accuracy (compared with 85% accuracy from rules generated by AQ15 without constructive induction). Figure 7 shows the final concept image learned by AQ17-HCI in the original representation space.

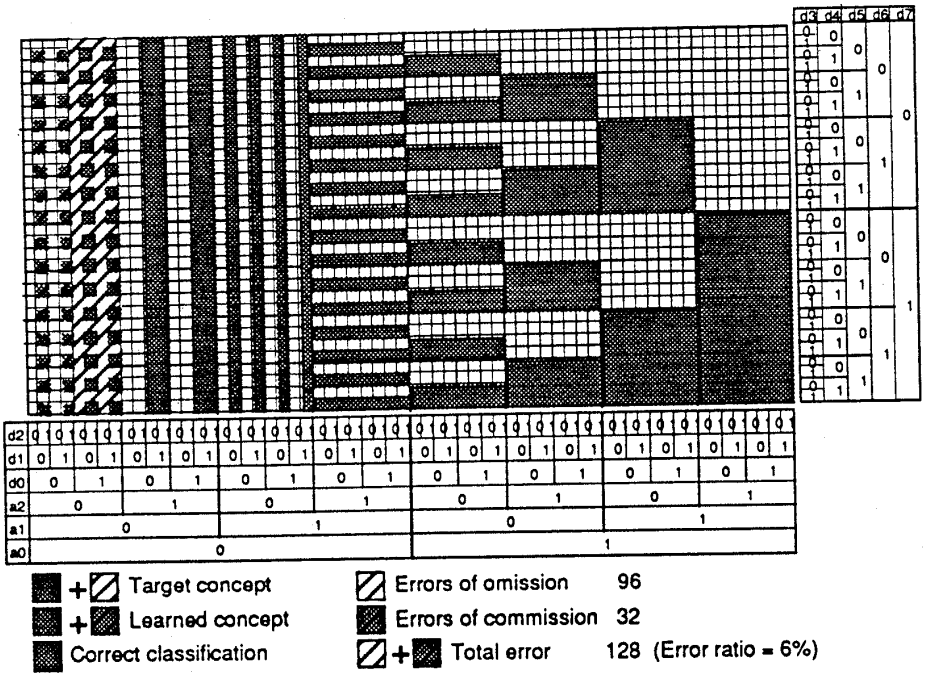

Figure 7. Concept learned by AQ17-HCI after conversion to original representation space.

5. Experiments

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 & Haussler, 1990).

5.1 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). Table 8 provides a summary 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.

Table 8. Summary description of experimental domains

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

DNF3 Boolean function defined by the expression:

x1x2x6x8x25x28¬x29 or x2x9x14¬x16¬x22x25 or
x1¬x4¬x19¬x22x27x28 or ¬x2¬x10x14¬x21¬x24 or
x11x17x19x21¬x25 or ¬x1¬x4x13¬x25

Attributes x3 x5 x7 x12 x15 x18 x20 x23 x26 x30 x31 x32 have random values for each example.

DNF4 Boolean function defined by the expression:

x1x4x13x57¬x59 or x18¬x22¬x24 or x30¬x46x48¬x58 or
¬x9x12¬x38x55 or ¬x5x29¬x48 or x23x33x40x52 or
x4¬x26¬x38¬x52 or x6x11x36¬x55 or ¬x6¬x9¬x10x39¬x46 or
x3x4x21¬x37¬x57

Attributes x2 x7 x8 x14 x15 x16 x17 x19 x20 x25 x27 x28 x31 x32 x34 x35 x41 x42 x43 x44 x45 x47 x49 x50 x51 x53 x54 x56 x60 x61 x62 x63 x64 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 bits[4].

Attributes x12 .. x32 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 x6 .. x32 have random values for each example.

---

[4] 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.

5.2 *Experimental results*

This section compares the performance of the AQ15 and AQ17-HCI programs on a set of experimental data. 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 an instance of a concept.

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 instances of a single class. This is done by establishing equivalence classes among the rules that were tested on those instances. Each equivalence class (called a rank) contains rules whose degrees of consonance were within a specified tolerance (tau) of the highest degree of consonance for that rank. When ATEST summarizes the results it 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 9). Table 10 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)).

Table 9. The experimental results for different problems.

Target concept	Average Error Rate			
	AQ15		AQ17-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 10. The experimental results for different numbers of training examples in learning DNF4.

No. of training examples	Average Error Rate in Learning Target Concept DNF4			
	AQ15		AQ17-HCI	
	1st Rank	100% match	1st Rank	100% match
330	29.6%	48.2%	27.2%	48.2%
660	7.7%	24.8%	2.4%	9.4%
1320	1.8%	16.4%	0.2%	4.3%
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%

5.3 *Empirical comparison of HCI with other methods*

Figure 8 and Table 11 summarize 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).

Figure 8 shows the learning curves for the concept DNF4. The curves were obtained by measuring and averaging error rates over ten experiments for each measure point. The measure points were 330, 660, 1320, 1980, 2640, and 3960 of training examples. Four systems, AQ15, FRINGE, GREEDY3, and AQ17-HCI obtain 100% performance accuracy when supplied with 3960 training examples. However, convergence to 100% is the fastest in the case of AQ17-HCI.

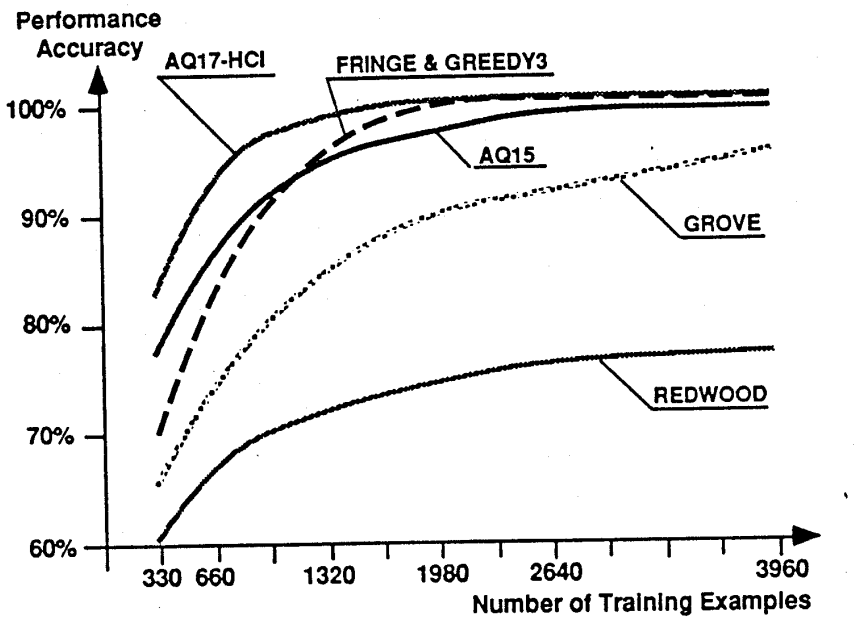

Figure 8. Learning curves for the concept DNF4.

Table 11 shows results obtained for the numbers of training examples listed in Table 8. AQ17-HCI with hypothesis-driven constructive induction capabilities has completely learned all the target concepts. 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) learned all the concepts.

Table 11. The experimental results.

Target concept	Average Error Rate					
	Decision TREES (*)		Decision LISTS (*)		Decision RULES (†)	
	REDWOOD	FRINGE	GREEDY3	GROVE	AQ15	AQ17-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%

(*) from (Pagallo and Haussler, 1989, 1990), (†) 1st rank decisions.

6. Discussion and conclusions

Both AQ15 rules and the AQ17-HCI constructed rules provide a complete and consistent coverage of the input examples. Since HCI involves attributes constructed form 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 attributes[5]. 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 given hypotheses space and require multiple covers.

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 abstracting concept descriptions, 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 11 suggest that 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.

The presented HCI method of constructive induction generates new attributes by analyzing and abstracting inductive concept 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. 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 hypothesis 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. On the other hand, generated attributes are rather complex and therefore, the overall complexity of the descriptions may be increased. In future research, we plan to investigate attribute generation based on selected components of the best performing rules rather than the entire rules. This could potentially lead to both a further 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.

---

[5] 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)

References

Arciszewski, T., Dybala, T. & Wnek J. (1992). Method for Evaluation of Learning Systems. To appear in *Journal of Knowledge Engineering "Heuristics"*.

Bala, J.W., Michalski, R.S. & Wnek, J. (1992). The Principal Axes Method for Noise Tolerant Constructive Induction. *Proceedings of the Ninth International Conference on Machine Learning*, (pp. 20-29). Aberdeen, Scotland: Morgan Kaufmann.

Bentrup, J.A., Mehler, G.J. & Riedesel, J.D. (1987). *INDUCE 4: A Program for Incrementally Learning Structural Descriptions from Examples*. (Reports of the Intelligent Systems Group, ISG 87-2). Urbana-Champaign: University of Illinois, Department of Computer Science.

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

Bloedorn, E., Michalski, R.S. & Wnek, J. (1992). AQ17 - A Multistrategy Constructive Learning System. (Reports of Machine Learning and Inference Laboratory).Fairfax, VA: George Mason University, Center for Artificial Intelligence.

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

Carpineto, C. (1992). Trading off Consistency and Efficiency in Version-Space Induction. *Proceedings of the 9th International Conference on Machine Learning*, (pp. 43-48). Aberdeen, Scotland: Morgan Kaufmann.

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

Clark, P., & Niblett, T. (1989). The CN2 Induction Algorithm. *Machine Learning*, *3*:261-284.

DeRaedt, L. & Bruynooghe, M. (1989). Constructive Induction By Analogy: A Method to Learn How to Learn? *Proceedings of EWSL-89*, (pp. 189-200). Montpellier, France: Pitman.

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

Emde, W., Habel, C.U. & Rollinger, C.-R. (1983). The Discovery of the Equator or Concept Driven Learning. *Proceedings of IJCAI-83*, (pp. 455-458). Karlsruhe, Germany.

Flann, N.S., & Dietterich, T.G. (1986). Selecting Appropriate Representations for Learning from Examples. *Proceedings of AAAI-86* (pp. 460-466). Philadelphia, PA: Morgan Kaufmann.

Kietz, J.U. & Morik, K. (1991). Constructive Induction of Background Knowledge. *Proceedings of IJCAI-91 Workshop on Evaluating and Changing Representation in Machine Learning*, (pp. 3-12). Sydney, Australia.

Langley, P., Bradshaw, G.L. & Simon, H.A. (1983). Rediscovering Chemistry With the BACON System. In R.S. Michalski, J.G. Carbonell & T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*. Los Altos, CA: Morgan Kaufmann.

Larson, J.B. & Michalski, R.S. (1977). Inductive Inference of VL Decision Rules. Invited paper for the Workshop in Pattern-Directed Inference Systems, May 23-27, Hawaii. Published in *ACM SIGART Newsletter, 63*, 38-44.

Lenat, D.B. (1983). Learning from Observation and Discovery. In R.S. Michalski, J.G. Carbonell & T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*. Los Altos, CA: Morgan Kaufmann.

Matheus, C. (1989). *Feature Construction: An Analytic Framework and Application to Decision Trees*. Ph.D. Thesis, Urbana-Champaign: University of Illinois.

Michalski, R.S. (1973). 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*. (pp. 3-17).

Michalski, R.S. (1978). *Pattern Recognition as Knowledge-Guided Computer Induction*. (Technical Report No. 927). Urbana-Champaign: University of Illinois, Department of Computer Science.

Michalski, R.S. (1983). A Theory and Methodology of Inductive Learning. In R.S. Michalski, J.G. Carbonell & T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*. Los Altos, CA: Morgan Kaufmann.

Michalski, R.S., Mozetic, I., Hong, J. & Lavrac, N. (1986). The Multi-Purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains. *Proceedings of AAAI-86*. (pp. 1041-1045).

Mitchell, T.M., Utgoff, P.E. & Banerji, R. (1983). Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics. In R.S. Michalski, J.G. Carbonell & T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*. Los Altos, CA: Morgan Kaufmann.

Morik, K. (1989). Sloppy Modeling. In K. Morik (Ed.), *Knowledge Representation and Organization in Machine Learning*. Springer-Verlag.

Muggleton, S. (1987). Duce, and Oracle-Based Approach to Constructive Induction. *Proceedings of IJCAI-87*, (pp. 287-292). Milan, Italy: Morgan Kaufmann.

Pachowicz, P. (1991). Application of Symbolic Inductive Learning to the Acquisition and Recognition of Noisy Texture Concepts. In *Applications of Learning and Planning Methods*.

Pagallo, G. & Haussler, D. (1989). Two Algorithms that Learn DNF by Discovering Relevant Features. *Proceedings of the 6th International Machine Learning Workshop*, (pp. 119-123), Ithaca, NY: Morgan Kaufmann.

Pagallo, G. & Haussler, D. (1990). Boolean Feature Discovery in Empirical Learning. *Machine Learning, 5*, 71-99.

Quinlan, J.R. (1986). Induction of Decision Trees. *Machine Learning, 1*, 81-106.

Reinke, R.E. (1984). *Knowledge Acquisition and Refinement Tools for the ADVISE Meta-expert System*. Master's Thesis, University of Illinois.

Rendell, L. (1985). Substantial Constructive Induction Using Layered Information Compression: Tractable Feature Formation in Search. *Proceedings of IJCAI-85*, (pp. 650-658).

Rendell, L. & Seshu, R. (1990). Learning Hard Concepts Through Constructive Induction: Framework and Rationale. *Computational Intelligence, 6*, 247-270.

Rivest, R. (1987). Learning Decision Lists. *Machine Learning 2*, 229-246.

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

Tecuci, G. & Kodratoff, Y. (1990). Apprenticeship Learning in Imperfect Theory Domains. In Y. Kodratoff & R.S. Michalski (Eds.), *Machine Learning: An Artificial Intelligence Approach* (Vol. III). Palo Alto, CA: Morgan Kaufmann.

Tecuci, G. (1992). Automating Knowledge Acquisition as Extending, Updating, and Improving a Knowledge Base. *IEEE Transactions on Systems, Man and Cybernetics, 22, 4*.

Tecuci, G. & Hieb, M. (1992). Consistency Driven Knowledge Elicitation within a Learning Oriented Representation of Knowledge. *AAAI-92 Workshop Notes on Knowledge Representation Aspects of Knowledge Acquisition*, (pp. 183-190), San Jose, CA.

Utgoff, P.E. (1984). *Shift of Bias for Inductive Concept Learning*. Ph.D., Rutgers University.

Utgoff, P.E. (1986). Shift of Bias for Inductive Concept Learning. In R.S. Michalski, J.G. Carbonell & T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach* (Vol. II). Los Altos, CA: Morgan Kaufmann.

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

Vafaie, H. & De Jong, K. (1991). Improving the Performance of a Rule Induction System Using Genetic Algorithms. *Proceedings of the First International Workshop on Multistrategy Learning*, (pp. 305-315). Harpers Ferry WV: George Mason University, Center for Artificial Intelligence.

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

Wnek, J. & Michalski, R.S. (1991). Hypothesis-Driven Constructive Induction in AQ17 - A Method and Experiments. *Proceedings of IJCAI-91 Workshop on Evaluating and Changing Representation in Machine Learning*, (pp. 13-22). Sydney, Australia.

Wnek, J. (1992). *Transformation of Version Space with Constructive Induction: The VS* Algorithm*. (Reports of Machine Learning and Inference Laboratory, ML-92-01). Fairfax, VA: George Mason University, Center for Artificial Intelligence.

Wrobel, S. (1989). Demand-Driven Concept Formation. In K. Morik (Ed.), *Knowledge Representation and Organization in Machine Learning*. Berlin Heidelberg: Springer Verlag.