Machine Learning, 14, 139–168 (1994)

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

JANUSZ WNEK (WNEK@AIC.GMU.EDU)
RYSZARD S. MICHALSKI (MICHALSKI@AIC.GMU.EDU)
Center for Artificial Intelligence, George Mason University, Fairfax, VA 22030

Abstract. The proposed method for constructive induction searches for concept descriptions in a representation space that is being iteratively improved. In each iteration, the system learns concept description from training examples projected into a newly constructed representation space, using an A^q algorithm-based inductive learning system (AQ15). The learned description is analyzed to determine desirable problem-oriented modifications of the representation space. These modifications include generating new attributes, removing redundant or insignificant ones, and/or agglomerating attribute values into larger units. New attributes are constructed by assigning names to groups of the best-performing *characteristic* rules for each decision class, and then are used to define the representation space for the next iteration. This iterative process repeats until the created hypotheses satisfy a *stopping criterion*. In several experiments on learning discrete functions, the developed AQ17-HCI system consistently outperformed, in terms of the prediction accuracy on new examples, all systems that it was compared to, including the AQ15 rule learning system, GREEDY3 and GROVE decision-list learning systems, and REDWOOD and FRINGE decision-tree learning systems. Although the proposed method was developed for the A^q-based rule learning system, it can potentially be adapted to any other inductive learning system. In this sense, it represents a universal new approach to constructive induction.

Keywords. Concept learning, constructive induction, decision rules, decision trees, decision lists, diagrammatic visualization

1. Introduction

Most research on inductive learning from examples has been concerned with learning concept descriptions from examples in a fixed, a priori defined representation space. This means that attributes relevant to describing the examples have to be provided beforehand, and the learned descriptions are expressed in terms of attributes selected from among them. For this reason, such inductive learning systems are called *selective* (Michalski, 1983; Rendell, 1985). Among such well-known systems are AQVAL/1 (Michalski, 1973), AQ11 (Michalski & Larson, 1978), CART (Breiman et al., 1984), ID3 (Quinlan, 1986a), ASSISTANT (Cestnik et al., 1987), C4.5 (Quinlan, 1989), CN2 (Clark & Niblett, 1989), GREEDY3 & GROVE (Pagallo & Haussler, 1990).

A fundamental limitation of these systems is that they can learn successfully only if the original attributes, or generally *descriptors*, are directly relevant to characterizing the concepts to be learned. These descriptors can be attributes, predicates, functions, relations, transformations, etc. To overcome this limitation, *constructive* inductive learning systems include mechanisms for generating new, more relevant descriptors, as well as for modifying or removing the less relevant descriptors initially provided. Thus, *constructive induction*

performs a *problem-oriented transformation* of the knowledge representation space during the process of inductive learning.

The primary goal of constructive induction may be to improve the overall *prediction accuracy* of the generated concept descriptions, to decrease the *overall complexity* of learned descriptions, or to improve the descriptions according to a combination of both criteria. The prediction accuracy is measured by applying the generated hypothesis to *testing* examples and determining the correctness of the predictions. It is often expressed as a percentage of the testing examples that are correctly classified (a reciprocal measure is an *error rate*, defined as *100% − prediction accuracy*). The overall complexity of a concept description is a function of the complexity of determining the values of attributes (descriptors) used in the description, as well as the complexity of operators involved in it.

The primary goal of the method proposed here is to learn concept descriptions with the highest prediction accuracy on new concept examples. The search for such descriptions is done by iteratively changing representation space and then measuring the prediction accuracy of the descriptions generated in this space. The prediction accuracy is estimated by applying the descriptions to a randomly determined subset of training examples that were not used in the learning process. The changes of the space are based on the analysis of the hypotheses generated in each iteration and the detection of patterns in them. For that reason, this is a "hypothesis-driven" constructive induction method (HCI), as opposed to *data-driven* methods that are based on the analysis of data (concept examples), and *knowledge-driven* methods that explore background knowledge.

The analysis of the hypothesis (in the form of a set of rules) involves determining an *admissible ruleset*, determining various patterns in it, and identifying attributes that can be removed from the original representation space. The admissible ruleset consists of the best-performing rules in a given iteration. This ruleset is used to determine new attributes that can stand for groups of rules ("rule-patterns"), groups of conditions in a rule ("condition-patterns"), or groups of attribute values in a condition ("value-patterns"). New attributes are employed to expand the representation space. Removed attributes contract the representation space. After determining the new representation space, training examples are projected into it, and an inductive learning process is repeated in the new space. This iterative process stops when a generated hypothesis achieves a desired performance level, or some other conditions are satisfied (see "Stopping condition" in section 3.3).

In order to place the proposed method in the family of existing constructive induction methods, section 2 provides a brief summary and classification of such methods. Section 3 gives a description of the proposed method, and section 4 discusses details of constructing new attributes from admissible rulesets. Section 5 explains the method using an example problem and illustrates representation space transformations using a *diagrammatic visualization* method. Section 6 describes the application of the rule learning system AQ17-HCI, implementing the proposed method, to four problems of learning discrete functions. For comparison, an earlier rule learning system AQ15, two decision list learning systems, GREEDY3 & GROVE, and two decision tree learning systems, REDWOOD & FRINGE, are applied to the same problems. Finally, section 7 summarizes the main features of the method, and suggests topics for future research.

2. A classification of constructive induction systems

The idea and the name *constructive induction* were 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. A number of other systems that exhibit constructive induction capabilities have been subsequently developed (e.g., Rendell, 1985; Matheus, 1989; Drastal, Czako, & Raatz, 1989; Wnek & Michalski, 1991).

Systems for constructive induction may employ different strategies for generating new descriptors, or generally, for changing the representation space. Based on the primary strategy employed, existing systems can be divided into four categories: data-driven, hypothesis-driven, knowledge-driven, and multistrategy. Below is a brief characterization of these categories and of selected representative systems in each category.

- ***Data-Driven Constructive Induction Systems (DCI)***

  These systems analyze and explore the input data (examples), particularly the interrelationships among descriptors used in the examples, and on that basis suggest changes in the representation space.

  BACON creates new attributes (variables) that represent simple numerical functions of the original variables. The process of generating new attributes employs heuristics based on the interdependencies between original attributes in the data (Langley, Bradshaw, & Simon, 1983; Langley et al., 1987).

  ABACUS employs methods for splitting data into subgroups, determining equations for each subgroup (in a fashion similar to BACON), and applying methods of symbolic induction for defining the applicability conditions for the equations (Falkenhainer & Michalski, 1990; Greene, 1988; Michael, 1991).

  PLS0 (*P*robabilistic *L*earning *S*ystem) creates new attributes from initial attributes using a form of conceptual clustering performed at three levels of abstraction: object, structure, and group relationships (Rendell, 1985).

  Wyl, IOE (*I*nduction-*O*ver-*E*xplanations) learns structural descriptions of selected concepts in chess and checkers games by first mapping the training examples from a performance-level representation (a chess or checkers board) into a learning-level representation (concepts characterizing game states), generalizing them in this representation, and converting the learned concept back into the performance-level representation for efficient recognition (Flann & Dietterich, 1986; Flann, 1990).

  STAGGER enhances the representation space by generating various Boolean combinations of description elements (attribute-value pairs), and discretizing continuous attributes using a statistical utility function (Schlimmer, 1987).

  AQ17-DCI applies many different logical and mathematical operators to the original attributes to create new "candidate" attributes. The candidate attributes that score high on *an attribute quality function* are added to the original attribute set, and the whole set is employed in the process of inductive generalization (Bloedorn & Michalski, 1991).

FCE (*F*actored *C*andidate *E*limination)—algorithm—starts with a set of initial representation spaces. After detecting inconsistency in hypotheses formulated in these representation spaces, the algorithm creates a Cartesian product of these spaces (Carpineto, 1992).

- ***Hypothesis-Driven Constructive Induction Systems (HCI)***

These systems incrementally transform the representation space by analyzing inductive hypotheses generated in one iteration and then using detected patterns as attributes for the next iteration.

BLIP proposes new "meta-facts" on the basis of rule exceptions that cannot be defined in terms in the given representation (Emde, Habel, & Rollinger, 1983; Morik, 1989; Wrobel, 1989).

CITRE (*C*onstructive *I*nduction on decision *TRE*es) determines a decision tree, and by analyzing it constructs new attributes. Simple facts are combined by constructive operators (Matheus, 1989).

FRINGE improves decision trees by avoiding the duplication of tests in them. New attributes are constructed from "fringes" of the tree, and stand for conjunctions of Boolean attributes (Pagallo & Haussler, 1990).

KLUSTER introduces new relations or concepts if another concept cannot be characterized without it. A definition of the requested 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 representation space.

AM (*A*utomated *M*athematician) changes its representation space by employing predefined heuristics for (1) defining new concepts represented as frames, (2) creating new slots and their values, and (3) adapting concept frames developed in one domain to another domain (Lenat, 1977, 1983).

COPER creates new function arguments by applying rules of dimensional analysis for combining arguments into dimensionless monomials (Kokar, 1985).

AQ15 applies arithmetic transformations (*a-rules*) and/or logical rules (*l-rules*) for constructing new attributes (Michalski et al., 1986).

MIRO applies expert-defined rules ("domain theory") to construct an abstraction space and then to perform induction in this space (Drastal, Czako, & Raatz, 1989).

- ***Multistrategy Constructive Induction Systems (MCI)***

These systems combine different approaches and methods for constructing new representation space. (The strategies combined are specified in the parentheses.)

INDUCE-1 (KCI & DCI) uses rules (selected by a user from a predefined repertoire) and/or built-in procedures for generating new attributes ("meta-attributes"), based on the analysis of structural descriptions of training examples (KCI), and/or of the qualitative dependencies between numerical attributes in the input descriptions (DCI) (Larson & Michalski, 1977; Michalski, 1978, 1983).

STABB (DCI & HCI) uses two procedures to *S*hift *T*o *A B*etter *B*ias. 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 (given in the input or hypothesized), and six transformation operators (HCI). Such inductive transformations are tested against an oracle, which ensures the validity of any transformation (KCI) (Muggleton, 1987).

CIGOL (HCI & KCI) (*LOGIC* backwards) employs "inverted resolution" using Horn clause knowledge representation. New predicates that play the role of subconcepts (or missing premises) are generated from input or hypothesized examples of a high-level predicate by applying the *intra-construction* operator (a form of HCI). A user may name the concept (predicate) or reject the proposed definition (which can be viewed as KCI) (Muggleton & Buntine, 1988).

ALPINE constructs a hierarchy of monotonic, i.e., structure-preserving, abstraction spaces from the operators of a domain (DCI). It uses domain axioms and knowledge about the primary effects of operators to avoid adding unnecessary constraints (KCI) (Knoblock, 1990). The method was integrated with other types of learning in the PRODIGY problem solver (Knoblock, Minton, & Etzioni, 1991).

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) (*C*oncept-*L*earning in an *INT*erative 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 (concept or predicate) (HCI). The user may name the schema or reject it (KCI) (De Raedt & Bruynooghe, 1989).

AQ17 (DCI, HCI & KCI) integrates in a synergistic way constructive induction capabilities of AQ15, INDUCE, AQ17-DCI, and AQ17-HCI (Bloedorn, Michalski, & Wnek, 1993).

3. Hypothesis-driven constructive induction (HCI)

The proposed method is based on repetitively detecting strong "patterns" in the hypotheses generated in one iteration, and then treating them as new attributes in the next iteration. To explain details of the method, we will start by describing the measure of the pattern strength.

3.1. Determining the pattern strength

A pattern can be a group of rules in the learned description, a part of a rule (a conjunction of conditions), or a group of attribute values in a condition of a rule. The strength of a pattern can be determined in many different ways. Below is the measure implemented in the program AQ17-HCI.

In this measure, the strength (σ) of a pattern is a function of the number of positive examples, PCov, and the negative examples, NCov, that are "covered" by the pattern:

$$\sigma(pattern) = f(PCov(pattern), NCov(pattern)) \quad (1)$$

To determine a specific form of the function f, let us observe that the strength of a pattern should be positively related to the number of positive examples covered by it, and negatively related to the number of negative examples covered by it. The way the strength is calculated should also depend on the pattern type—is it a ruleset, a subrule, or a group of attribute values? In addition, the measure of strength may distinguish between types of coverage of concept instances by a given pattern. For example, a concept instance can be covered only by a given pattern (a unique coverage), or can be multiply covered. To reflect this difference, PCov and NCov are expressed not just by single numbers, but by multiple numbers. Here is a simple measure of pattern strength that reflects the above considerations:

$$\sigma(pattern) = \frac{t^+(pattern) + \lambda u^+(pattern)}{t^-(pattern) + 1} \quad (2)$$

where

t^+(pattern), called the *total positive weight*, and t^-(pattern), called the *total negative weight*, are the numbers of positive and negative examples covered by the pattern, respectively;

u^+(pattern), called the *unique weight*, is the number of positive examples uniquely covered by the pattern, i.e., not covered by any other comparable pattern; and

λ is a parameter that controls the relative importance given to these two types of coverage.

When $\lambda = 0$, i.e., when the unique weight is ignored, the above measure of pattern strength (σ) is similar to the *logical sufficiency* (LS) used in the Prospector expert system (Duda, Gasching, & Hart, 1979), and in the STAGGER concept learning system (Schlimmer, 1987).

3.2. Determining an admissible ruleset

The HCI method works iteratively. Each iteration generates a complete and consistent set of rules, i.e., a ruleset that covers all positive examples and none of the negative examples. In order to speed up the process of determining strong patterns, and to avoid searching through rules that are weak and/or of low validity, the method selects rules that have sufficient strength in the generated ruleset. These rules constitute an *admissible ruleset*. The method searches for strong patterns in the admissible ruleset, uses patterns for transforming the representation space, and then moves to the next iteration (the complete method is described in the next section).

When determining the strength of rules in the ruleset representing a concept, expression (2) can be simplified; specifically, the denominator in expression (2) can be ignored. This is so because such a ruleset is consistent with regard to negative examples (no negative examples covered), and therefore t^- (the negative weight) is zero.

Thus, we have

$$\sigma(rule) = t(rule) + \lambda u(rule) \quad (3)$$

where t is the total (positive) weight of a rule in a ruleset (the total number of training examples covered by this rule) and u is the unique weight of a rule in a ruleset (the number of training examples covered only by this rule, and not by any other rule in the ruleset (Michalski et al., 1986). The program's default value for parameter λ is 2, which gives a relatively strong preference to rules with higher unique weights, i.e., rules that have smaller overlap with other rules in a ruleset for a given concept.

To determine an admissible ruleset, rules in the ruleset for a given concept are ordered from the strongest to the weakest. An *admissible* ruleset contains the minimal number of rules from the ruleset whose total relative strength exceeds a predefined threshold:

$$\frac{\sum_{i=1..m} \sigma_i}{\sum_{j=1..n} \sigma_j} \geq TH \tag{4}$$

where σ_i is the strength of rule i defined by equation (3), n is the total number of rules in the current hypothesis, and m ($m \leq n$) is the number of strongest rules (recall that the rules are ordered, and thus $\sigma_i \geq \sigma_{i+1}$). In the program, the TH parameter has the default value 0.67, which means that the admissible ruleset will cover at least two thirds of the training examples of a given concept. Since noisy examples and exceptions are normally covered by low-strength rules, therefore the admissible ruleset can be expected to cover the most "central" portion of the learned concept. This method could be improved by setting the TH parameter on the basis of knowledge of the noise level and of the confidence in the learned hypothesis.

3.3. The AQ-HCI method

The proposed hypothesis-driven constructive induction (AQ-HCI) method combines an inductive rule learning algorithm (A^q) with a procedure for iteratively transforming representation space. In each iteration, the method changes the representation space by adding new attributes, removing insufficiently relevant attributes, and/or agglomerating values of some attributes into larger units. The quality of the hypothesis generated in each iteration is evaluated by applying the hypothesis to a subset of training examples. The set of training examples prepared for a given iteration is split into the primary set (the P set), which is used for generating hypotheses, and the secondary set (the S set), which is used for evaluating the prediction accuracy of the generated hypotheses. Figure 1 presents a diagram illustrating the AQ-HCI method.

In the implemented system, AQ17-HCI, the input consists of training examples of one or more concepts, as well as background knowledge about the attributes used in the examples (which specifies their types and legal value sets). For the sake of simplicity, let us assume that the input consists of positive examples, E^+ and negative examples, E^-, of only one concept. If there are several concepts to learn, examples of each concept are taken as positive examples of that concept, and the set-theoretical union of examples of other concepts is taken as negative examples of that concept.

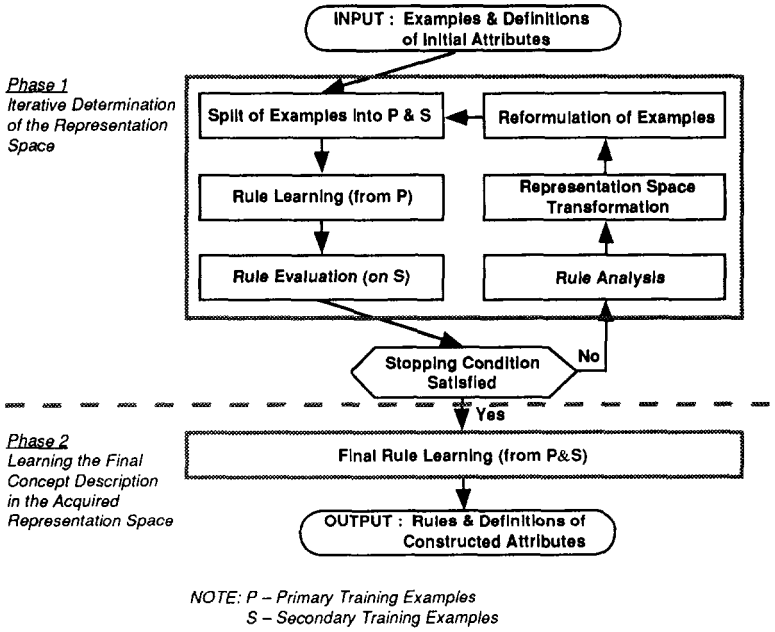

Figure 1. The AQ-HCI method for hypothesis-driven constructive induction.

The method consists of two phases. Phase 1 determines the representation space by a process of iterative refinement. In each iteration, the method prepares training examples, creates rules, evaluates their performance, modifies the representation space, and then projects the training examples into the new space. This phase is executed until the *Stopping Condition* is satisfied. This condition requires that the prediction accuracy of the learned concept descriptions exceeds a predefined threshold or that there is no improvement of the accuracy over the previous iteration. Phase 2 determines final concept descriptions in the acquired representation space from the complete set of training examples. The output consists of concept descriptions and definitions of attributes constructed in Phase 1. Below is a detailed description of both phases, and the basic modules of the method.

Phase 1 consists of six modules. The first, "Split of Examples" module, divides positive and negative training examples into the primary set, P, and the secondary set, S (in the experiments the split was according to the ratios 2/3 and 1/3, respectively). The set of primary positive (negative) examples is denoted P^+ (P^-), and the set of secondary positive (negative) examples is denoted S^+ (S^-). Thus $P = P^+ \cup P^-$, and $S = S^+ \cup S^-$. The primary training set, P, is used for initial rule learning, the secondary set, S, for an evaluation of intermediate rules, and the total set, $P \cup S$, for the final rule learning (in Phase 2).

The "Rule Learning" module induces a set of decision rules for discriminating P^+ from P^-, i.e., a cover COV (P^+/P^-) of positive primary examples against negative primary examples. This is done by employing the AQ15 inductive learning program (Michalski et al., 1986). The program is based on the algorithm A^q for solving general covering problem,

which was described in various sources (e.g., Michalski & McCormick, 1971; Michalski, 1973). For completeness, section 3.4 gives a brief description of the algorithm.

The "Rule Evaluation" module estimates the prediction accuracy of the rules by applying them to the secondary training set, S. The accuracy of the rules in classifying the examples from S is determined by the ATEST procedure implemented in the AQ15 program (Reinke, 1984). If the Stopping Condition criterion is not satisfied, the control passes to the "Rule Analysis" module; otherwise, it passes to Phase 2.

The "Rule Analysis" module determines an admissible ruleset. The "Representation Space Transformation" module analyzes the rules in this ruleset to determine desirable changes in the representation space. It removes redundant or insignificant attributes, modifies existing attributes (by attribute value agglomeration), and generates new attributes (see section 3.5). The "Example Reformulation" module projects all training examples into the new representation space, and the whole inductive process is repeated.

Phase 2 determines the final ruleset by applying the "Rule Learning" module to all training examples projected into the final representation space determined in Phase 1. For each concept, a set of the most specific (ms) rules is induced from all positive examples against all negative examples, i.e., a cover COV_{ms} (E^+/E^-), and the most general (mg) rules of negative examples against positive examples, that is, COV_{mg} (E^-/E^+). The final concept description is built by generalizing the most specific rules for positive examples against the most general rules for negative examples, i.e., determining a cover, COV_{mg} [COV_{ms} (E^+/E^-)/COV_{mg} (E^-/E^+)] (notice that the arguments for the covering algorithm are here not sets of examples, but sets of rules). The description so generated represents an intermediate degree of generalization between the most *specific* positive rules and the most *general* negative rules. For details on generating such concept descriptions, see Wnek (1993).

3.4. Rule learning module

As mentioned earlier, initial and consecutive hypotheses, in the form of rulesets, are generated by the inductive rule learning program AQ15 (Michalski et al., 1986). The program learns rules from examples represented as sequences of attribute-value pairs. Attributes can be multiple-valued and can be of different types, such as symbolic, numerical, or structured (in the latter case, the value set is a hierarchy). The teacher presents the learner a set of examples of each concept to be learned. The program generates a set of general rules (a ruleset) characterizing each class.

A ruleset is equivalent to a disjunctive normal form (DNF) expression with internal disjunction (each rule corresponds to one disjunct). In the standard mode, the program generates rulesets that are consistent and complete concept descriptions, i.e., cover all positive examples and no negative examples. Generated rules 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 AQ15 program is based on the A^q algorithm, which iteratively evokes a *star* generation procedure. 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 the star, 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, if there exists a conjunctive characterization of the concept), the algorithm terminates after the first step of star generation. An efficient procedure for star generation is described in, e.g., Michalski and McCormick (1971).

The AQ15 program has various parameters whose default values can be changed accordingly to learning goals. One parameter, *trim*, controls generality of the learned descriptions without increasing their complexity. Based on this parameter setting, the program can learn either maximal characteristic descriptions or minimal discriminant descriptions (Michalski, 1983). The *maximal characteristic descriptions* are the *most specific* conjunctions characterizing all objects in the given class using descriptors of the given representation. Such descriptions are intended to discriminate the given class from all other possible classes. The *minimal discriminant descriptions* are the *most general* logical products characterizing all objects in the given class using descriptors of the given representation. Such descriptions are intended to discriminate the given class from other classes currently represented in the space.

Another AQ15 parameter, *mode*, controls cover bounds of the learned descriptions. The *intersecting covers* mode produces descriptions that may intersect over areas with no training examples. The *disjoint covers* mode produces descriptions that do not intersect at all.

There are two learning goals in the AQ-HCI method. They are related to the two phases of the method: the iterative determination of the representation space, and learning the final concept description. The first learning goal is to obtain concept descriptions that can serve as an intermediate knowledge for selecting useful patterns for constructing new attributes. Such descriptions should reveal all necessary conditions needed for best classification of available data, and thus help to detect useful value-patterns and condition-patterns. To this end, maximal characteristic descriptions are most desirable.

An additional assumption is made for the purpose of finding proper rule-patterns. Such patterns, if endorsed as new attributes, should not cause ambiguity in the representation space. The ambiguity could have been easily introduced if two or more new attributes created from rulesets of different concept descriptions had overlapping definitions, and at the same time some of existing attributes relevant in describing the overlap were removed. To prevent ambiguity in such explicit cases, the method assumes generating disjoint covers.

The second goal is to obtain final concept descriptions that give the highest performance accuracy. Therefore, Phase 2 generates concept descriptions at an intermediate level of generalization between the most *specific* and the most *general* levels. The descriptions of different classes may overlap over areas occupied by unseen examples (therefore, the "Intersecting Covers" mode is used). Instances from overlapping areas are recognized through a *flexible matching* procedure (Reinke, 1984; Michalski et al, 1986).

In sum, in the AQ-HCI method, the AQ15 program for searching the problem space is combined with processes that change the representation space. The hypotheses generated in Phase 1 are characteristic generalizations of examples (most specific), and used for proposing problem-oriented changes in the representation space. Phase 2 utilizes the resulting space to generate final hypotheses that are at an intermediate level of generalization that is desirable for achieving the highest possible prediction accuracy.

3.5. Representation space transformations

In the proposed method, transformations of the representation space may involve both contraction and expansion operations. Contraction decreases the number of possible instances that can be represented in the space, and expansion increases that number. Contraction can be done by removing attributes, or combining attribute values into larger units. Expansion can be done by adding new attributes or by adding new attribute values to the value sets of the attributes.

3.5.1. Contraction

From the viewpoint of algorithmic efficiency, it is desirable to maximally decrease the representation space while preserving the ability to precisely describe concepts to be learned. The proposed method removes from the space those attributes that can be considered redundant or insignificant. The first of these are defined as those that do not occur in the hypotheses generated by a selective inductive algorithm, and the second of these as those that occur only in rules of low strength. The importance of such a contraction has been confirmed by experiments showing that descriptions generated in properly contracted representation spaces tend to have higher predictive accuracy than descriptions generated in spaces that have not been contracted (Quinlan, 1986b; Subramanian, 1990; Thrun, 1991; Vafaie & De Jong, 1991). The method also agglomerates values of an attribute into larger units. This is done by quantizing continuous attributes, or by creating more abstract values of discrete attributes. These are useful operations, because overly precise attributes can cause overfitting of the hypotheses.

3.5.2. Expansion

When originally given attributes are not directly or sufficiently useful for creating concepts descriptions, it is important to expand the description space by adding to it new attributes. One method for generating new attributes is to invent some new physical processes that allow the measurement of previously unknown or undetectable object properties. This is often very difficult. Another, much simpler method is to search for various, even very complex combinations of the existing attributes that are much more directly useful for describing concepts to be learned. Such combinations are given names and are treated as new attributes. Adding attributes so generated to the representation space is a space-expansion operation. The AQ-HCI algorithm uses different methods for generating such synthesized new attributes (see below). Another space expansion operation used by the algorithm involves detecting subsets of attribute values, called *value-patterns*, that often co-occur in a given class description. The algorithm combines these values into larger units and treats these units as additional values.

New attributes are constructed by detecting strong *condition-patterns* or *rule-patterns*. A strong condition-pattern represents a conjunction of two or more elementary conditions that frequently occur in a ruleset for a given concept, and has high strength. A condition-

Table 1. Examples of various descriptors 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 descriptors		
cv <:: (z = blue v red v white)		(Value-pattern)
ca1 <:: (x = 20) & (y = large)		(Condition-pattern)
ca2 <:: ((x = 75..100) & (y = small))	or (x = 7)	(Rule-pattern)

pattern can be viewed as a useful generalization of rules in which the pattern occurs. A strong rule-pattern is a rule or a subset of rules from a ruleset that has high strength. Such a pattern can be viewed as a useful specialization of the ruleset.

Table 1 shows examples of possible attribute values and attributes constructed in a simple, three-attribute domain. Each new descriptor is based on a different pattern-type found in the hypotheses.

The system searches for patterns in all learned concept descriptions. In each description, only the admissible rules are considered. Each rule depicts certain patterns generalized from examples that reflect an interrelation between attributes, relevance to the learning task, and the representational capabilities of the learning program (representational bias).

The aim of the *Pattern Strength* (σ) function, as defined in equation (2), is to determine which patterns are admissible as new attributes. The function represents a degree to which the presence of an attribute, constructed from the pattern, in the training example indicates the membership of the example to the given concept. Assuming that ($\lambda = 0$), i.e., that the unique weights are not calculated, values of the σ function range from zero to the number of positive examples. When a pattern is not matched by any positive example, the numerator is 0, and then σ(pattern) is equal to 0. Such a pattern is worst for describing the learned concept. When the pattern is not matched by any negative examples, the denominator is 1, and then σ(pattern) is equal to the number of matched positive examples. If the pattern matches more positive than negative examples, then σ(pattern) is greater or equal to 1. Patterns that match more positive than negative examples may be considered useful for constructing new attributes. In the HCI method, pattern selection is more restricted by additional conditions. A pattern is selected if its strength is greater than the strengths of all conditions involved in the description of the pattern.

After patterns are selected, new attributes can be defined. A new attribute definition consists of a name, a defining expression, and a similarity measure for assigning values. The attribute is assigned a unique name after the concept it was created from. The defining expression is the related pattern. The measure can result in assigning real or discrete values from the closed interval 0 to 1. In the simplest case, value 1 is assigned if the pattern is satisfied, and value 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 for the task of texture recognition.

3.6. Summary

There may be many strategies to control attribute generation of various pattern types. In the simplest approach, new attributes are generated based on all types of patterns detected in the initial hypothesis. However, a drawback to this is that introducing new attributes makes the representation space larger and thus more difficult to search. In order to design a strategy for determining patterns for new attribute generation, each type of attribute pattern has to be tested and its properties recognized. This strategy must be based on small and strongly justified changes in the representation space. The selection of attribute pattern can be based on the structure of the hypothesis and the sequence of pattern types already applied.

The process of inducing rules from examples may be repeated several times in different representation spaces in order to achieve the desired prediction accuracy. The complexity of inducing rules from examples in AQ15 is linear with respect to NR. NR is a number of rules needed to describe the concept. The number of negative examples strongly affects the shape of this dependency. 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(T * NR), where T is the number of iterations.

It is assumed from here on that the "Representation Space Transformation" module performs two transformations only: it removes insufficiently relevant attributes from the representation space, and it generates attributes based on rule-patterns in learned hypotheses. A detailed presentation and analysis of other transformations is done by Wnek (1993).

4. A method for generating attributes from rule-patterns

This method generates attributes by agglomerating rules that form a rule-pattern. For each concept learned, a rule-pattern is found, evaluated, and if sufficiently relevant, named and added to the representation space. The rule-pattern is defined as a set of admissible rules abstracted from the learned hypothesis using formula (4). The pattern strength evaluation is done using formula (2). If the strength of the pattern exceeds the strength of all current attributes, then a new attribute is constructed from this pattern. The attribute is assigned a unique name after the concept it was abstracted from.

The new attribute descriptions involve attributes from the current descriptor set and the operators defined within a given representational formalism. Since the AQ17-HCI program uses the VL_1 variable-valued logic formalism (a multi-valued extension of propositional logic), the constructed attributes are VL_1 logical expressions. Therefore, with the new attributes, the system introduces conceptual changes of the representation space. These changes are more complex than a transformation to another space.

Notice that by extending the representation space by introducing new attributes that are logical combinations of other attributes, some areas of the space may represent impossible combinations of attribute values, and some areas may contain a high concentration of concept examples. The higher the concentration of examples of a specific concept in some area, the easier it is to describe and generalize these examples. Such an effect is thus desirable consequence of the change of the representation space (this effect is illustrated later in figure 6).

The constructed attributes conceptually partition the whole representation space into disjoint subconcepts (clusters corresponding to individual values of an attribute). Due to the mechanism of constructing new attributes, training examples of a given concept are agglomerated into two types of subsets: (1) "typical" examples of the concept, and (2) atypical examples of any concept. Typical are those examples that are covered by the attribute constructed from the admissible ruleset of the concept. The remaining examples of the concept cannot be covered by another constructed attribute because all concept descriptions are disjoint. For example, when learning two descriptions of a concept, i.e., Positive and Negative description, two attributes are constructed, P1 and N2. Training examples that have (P1 = 1) are typical for the Positive description, and those with (N2 = 1) are typical for the Negative description. Typical positive examples have (N2 = 0), and typical negative examples have (P1 = 0). Examples with (P1 = 0) and (N2 = 0) are atypical in both Positive and Negative descriptions of a concept. Examples with (P1 = 1) and (N2 = 1) are impossible.

The system is able to use the defined subconcept and its negation in the process of building concept descriptions. This 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 VL_1 expression defining the new attribute.

The ability to introduce new attributes in the form of subconcepts involves two implicit extensions to the representational capabilities of the AQ family systems:

1. Rule set-to-condition operator.
   This 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)

   which stands for (((x = 75,100) & (y = small)) or (x = 7)).
2. Rule set-negation-to-condition operator.
   This 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)

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

5. A simple example illustrating the method

To illustrate the performance of the method, let us describe an experiment on learning a form of the multiplexer function with 3 binary inputs and 8 binary outputs: the *multiplexer-11*

(MX11) problem (Wilson, 1987). The function "switches on" an output (data) line addressed by the input (address) lines. The remaining output lines are irrelevant for the given address.

The address lines are represented by a0, a1, a2 binary attributes, and the data lines are represented by d0–d7 binary attributes. For example, the result of MX11 function on [a0 = 0] [a1 = 1] [a2 = 1] is [d0 = #] [d1 = #] [d2 = #] [d3 = 1] [d4 = #] [d5 = #] [d6 = #] [d7 = #], where "#" represents "0" or "1." An instance described by [a0 = 0] [a1 = 1] [a2 = 1] [d0 = #] [d1 = #][d2 = #] [d3 = 1] [d4 = #] [d5 = #] [d6 = #] [d7 = #] is a positive example of the MX11 concept.

Figure 2 presents the MX11 concept graphically, using the *diagrammatic visualization* method. This method employs a *General Logic Diagram* (GLD), which is a planar representation of a multidimensional space spanned over multivalued discrete attributes[1] (Michalski, 1973; Wnek & Michalski, 1993). Each cell in the diagram represents a combination of the attribute values, i.e., 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 is shaded differently.

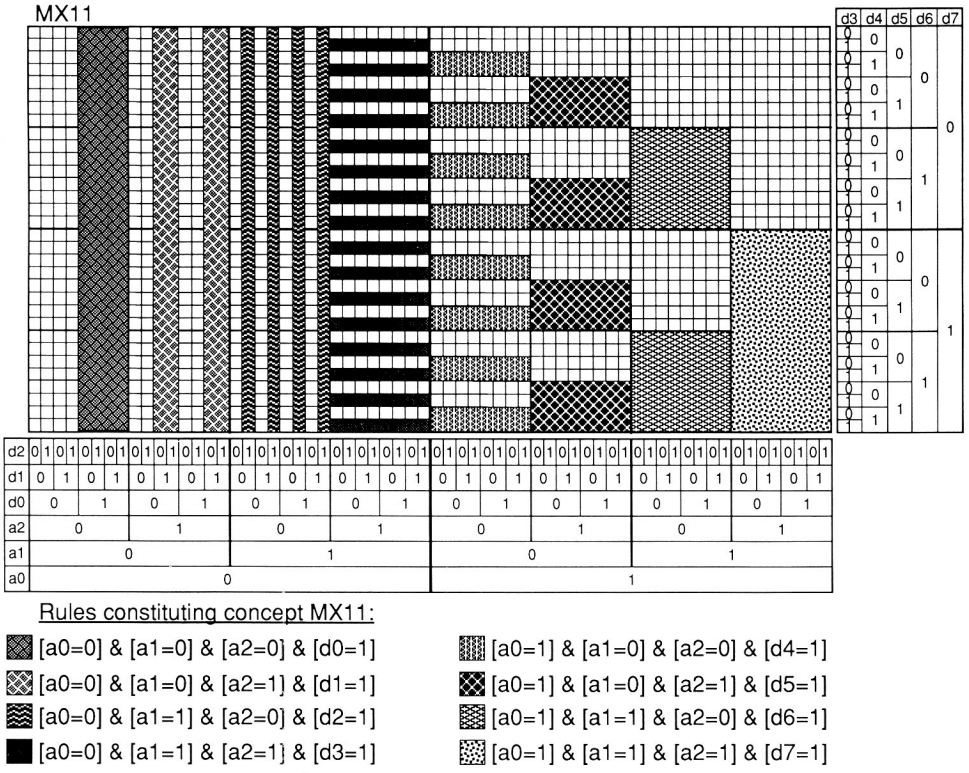

Figure 2. Target concept MX11 in the original representation space.

The diagrammatic visualization method permits one to display both target and learned concepts, individual steps in a learning process, and 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* ($T \backslash L$), while the areas of the learned concept not covered by the target concept represent *errors of commission* ($L \backslash T$). The union of both types of errors represents the *error image*. In the diagrams, errors are marked by slanted lines.

Figure 3 explains the meaning of various cases in concept visualization. Concept images are represented in the diagrams by shaded areas (figures 3A and 3B). If the target and learned concepts are visualized in the same diagram, then the shaded areas represent a learned concept (figure 3C). The error image is represented by slanted areas. It is easy to distinguish between errors of omission and errors of commission in the diagram. 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, and thus the corresponding slanted areas remain white in the background. The shape of the target concept is implicitly indicated by the borders of the correctly learned concept and the errors of omission areas. The correctly learned part of the concept is simply shaded.

The MX11 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 representation space contains 2048 elements. The traning set had 64 (6%) of the positive examples and 64 (6%) of the negative examples. Table 2 shows a sample of the positive and the negative examples.

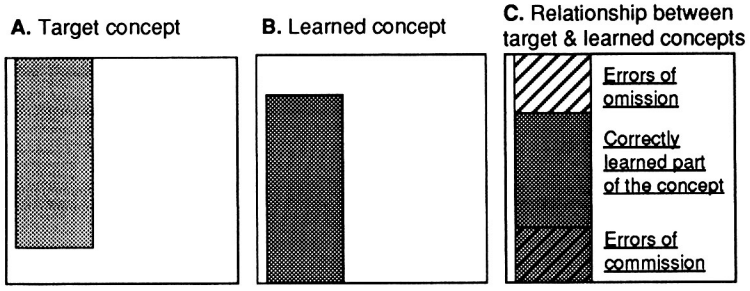

Figure 3. An illustration of a target concept, a learned concept, and their interrelationship.

Table 2. Part of the set of training examples.

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	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	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
...											...										

From these examples the Rule Learning module produced disjoint and maximally characteristic hypotheses of the correct (Pos-Class) and incorrect (Neg-Class) behavior of the multiplexer. The classification rules are shown in table 3. 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.

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

From the hypotheses presented in table 3, the admissible rulesets were selected to constitute the candidate attributes P1 and N2 (table 4). Table 5 shows the definition of the new attributes, and figure 5 shows the coverage of the instance space done by the new attributes.

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

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 by (N2 = 1) in the Neg-Class. Figure 6 summarizes the new learning task in the changed representation space. The new space consists of 7 binary attributes: 5 primary attributes and 2 constructed attributes. A characteristic feature of this representation space is *impossible* instances, i.e., instances that do not have equivalent descriptions in the original space. For example, instances described by the rule ((P1 = 1) & (N2 = 1)) are impossible. This is directly related to the definitions of P1 and N2 (see table 5 and figure 5), and the fact that these attributes were constructed from two disjoint rulesets (section 3.4). This is also in agreement with the intuition that there should not

Table 3. Disjoint and maximally characteristic descriptions of MS11 concept 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)

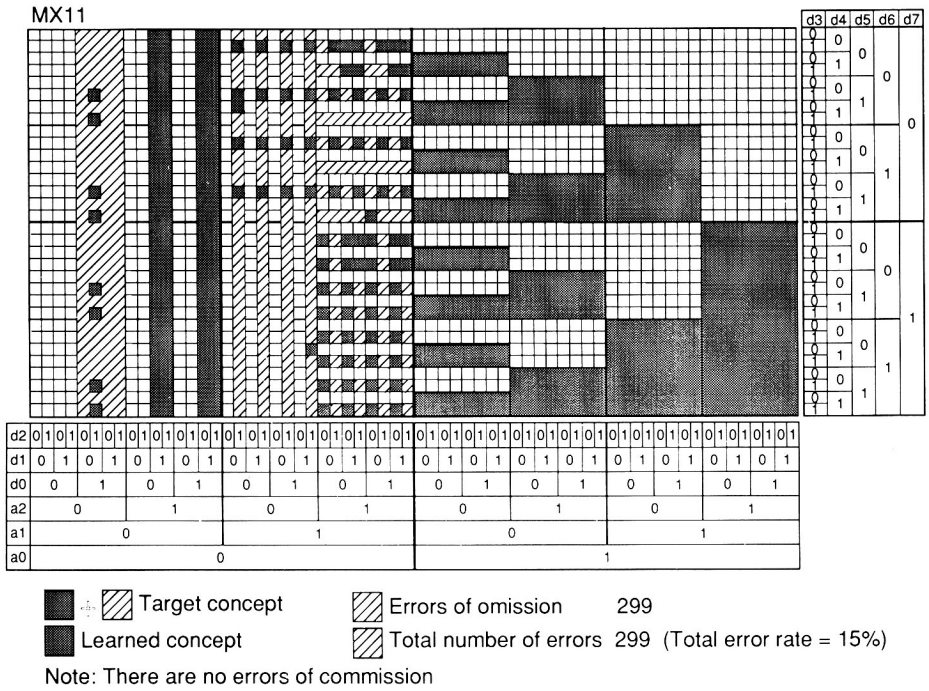

Figure 4. AQ15-learned concept in the original representation space and its relationship to the MX11 target concept (figure 2.)

Table 4. The admissible rulesets selected according to formula (4).

Pos-Class		**Neg-Class**	
1. (a0=1) & (a1=1) & (a2=0) & (d6=1)	(σ:33)	1. (a0=1) & (a1=1) & (a2=1) & (d7=0)	(σ:39)
2. (a0=0) & (a1=0) & (a2=1) & (d1=1)	(σ:33)	2. (a0=0) & (a2=0) & (d2=0)	(σ:36)
3. (a0=1) & (a1=0) & (a2=1) & (d5=1)	(σ:30)	3. (a0=0) & (a1=0) & (a2=1) & (d1=0)	(σ:30)
4. (a0=1) & (a1=1) & (a2=1) & (d7=1)	(σ:30)	4. (a1=0) & (a2=0) & (d1=1) & (d4=0)	(σ:27)
5. (a0=1) & (a1=0) & (a2=0) & (d4=1)	(σ:27)		
$[(\Sigma \sigma_i) / (\Sigma \sigma_j)] = 126 / 191 = 0.66 < TH$	(i=1..4)	$[(\Sigma \sigma_i) / (\Sigma \sigma_j)] = 105 / 192 = 0.55 < TH$	(i=1..3)
$[(\Sigma \sigma_i) / (\Sigma \sigma_j)] = 153 / 191 = 0.80 > TH$	(i=1..5)	$[(\Sigma \sigma_i) / (\Sigma \sigma_j)] = 132 / 192 = 0.69 > TH$	(i=1..4)

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**

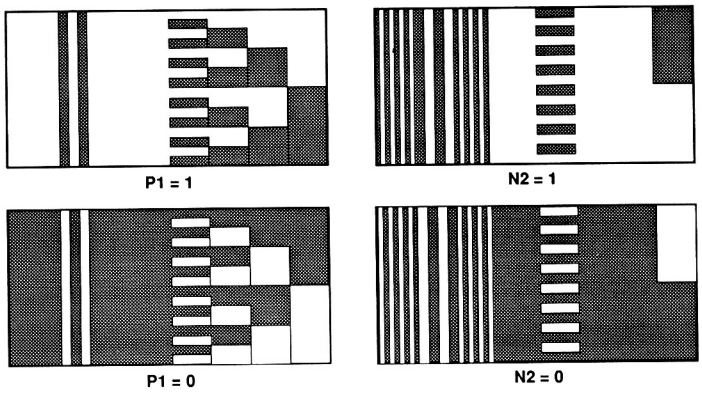

Figure 5. Images of the constructed attributes P1 and N2.

Table 6. A 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	0	1	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	0	1	1	0	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)

be any instances that conform to both "Positive" and "Negative" concept descriptions represented by P1 and N2.

The MX11 target concept image is shown in figure 6A after mapping into the new representation space. For easy identification, areas that correspond to those in figure 2 are marked with the same pattern. Figure 6B 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 6C shows the training examples in the new space. Figure 6D shows the final concept learned.

The learned concept still does not cover exactly the target concept (3 errors of omission and 1 error of commission), but it gives improved prediction accuracy. The learning could be further improved if the generalization were prohibited over impossible areas. Instead of producing a rule (a0 = 0) & (d3 = 1) & (P1 = 0) & (N2 = 0) to cover the four positive examples listed below, the system would be forced to generate a more specific rule:

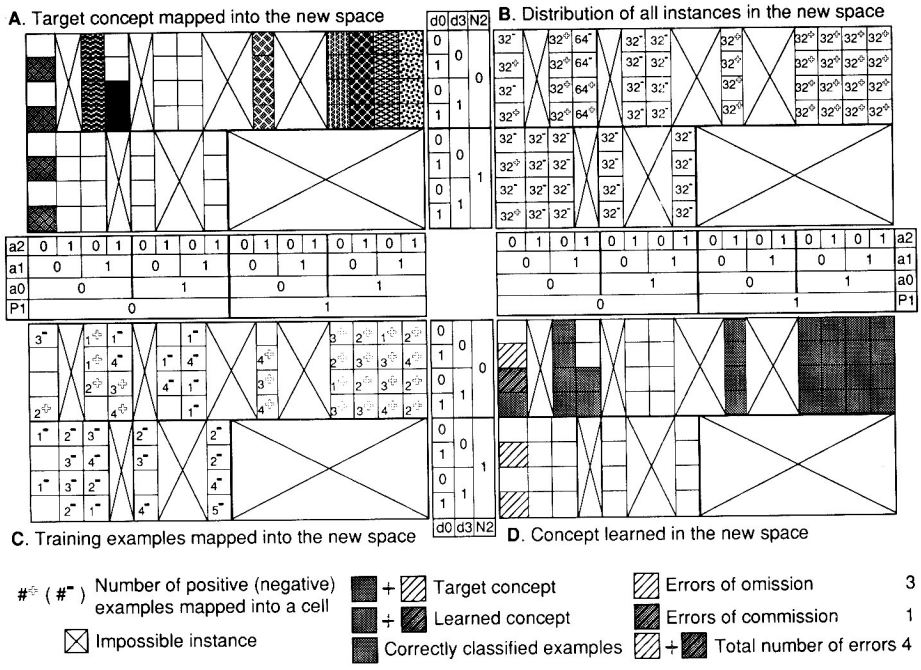

Figure 6. Steps in learning the concept MX11 in the changed representation space.

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. Ongoing research is investigating ways of improving induction in spaces with impossible instances. Another important issue that needs further research involves the utilization of information about the number 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 projected into the original representation space.

6. Experiments

A major measure of the performance of a learning system is the prediction accuracy of the learned concepts on the testing examples. The prediction accuracy is a ratio between

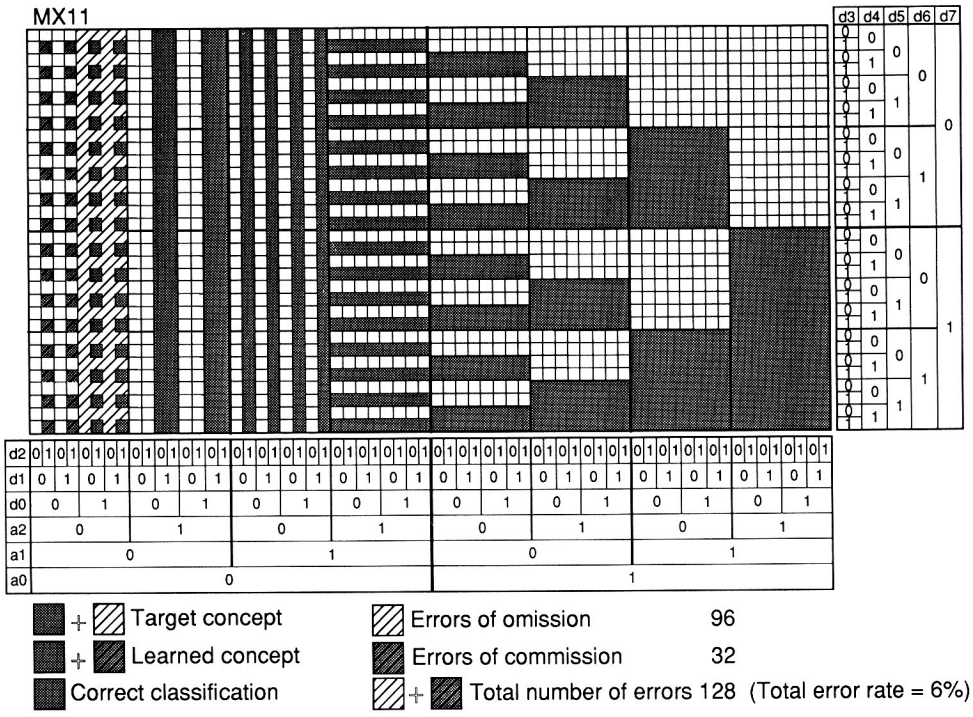

Figure 7. AQ17-HCI-learned concept projected into the original representation space and its relationship to the MX11 target concept (compare with figures 2 and 4).

the number of correctly classified examples from the testing data set and the cardinality of this set. For the sake of comparison with results published previously, we also use the complementary measure of error rate. Error rate is a ratio between the number of incorrectly classified examples from the testing data set and the cardinality of this set. All experiments were run ten times over randomly generated data, and the results were averaged.

The goal of our experiments was to test how well the AQ-HCI method does according to prediction accuracy criterion, and how well it compares to other methods. The following systems were compared: a system implementing the method for generating attributes from rule-patterns, AQ17-HCI; a standard decision rule system, AQ15 (Michalski et al., 1986); two decision lists systems, GREEDY3 and GROVE (Pagallo & Haussler, 1990); and two decision tree systems, one based on a hypothesis-driven constructive induction, FRINGE, and the other based on the standard ID3, REDWOOD (Pagallo & Haussler, 1990).

6.1. Experimental domains

The domains for testing AQ17-HCI and comparison with other methods were four discrete functions: C1 (DNF3), C2 (DNF4), C3 (MX11), and C4 (PAR5). These functions were used by Pagallo & Haussler (1989, 1990) to test several learning methods, specifically

REDWOOD, FRINGE, GREEDY3, and GROVE. In this study, we applied AQ17-HCI and AQ15 and compared results with those reported by Pagallo and Haussler (1989, 1990).

Table 8 provides a characterization of the test domains. The number of training examples was calculated according to the following formula (Pagallo & Haussler, 1990):

$$\frac{K * \log_2(N)}{\epsilon} \tag{5}$$

where N is the number of attributes, K is the number of conditions in the smallest DNF description of the target concept, and ϵ is the maximum error rate of the learned description. The number of conditions in the smallest DNF description of the target concept is the product of the number of rules in the description and the average number of conditions in a rule. In the experiments, ϵ is set to 0.10. The testing set consists of 2000 examples (different from training examples). Following table 8 are the descriptions of the target concepts C1–C4.

6.2. Experimental results

This section compares the performance of the AQ17-HCI and AQ15 systems on concepts C1–C4 on various sets of experimental data. The rules generated by both systems were tested using the ATEST procedure (Reinke, 1984). ATEST views rules as expressions that, when applied to a vector of attribute values, evaluate 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 setttings 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 (τ) of the highest degree of consonance for that rank. When ATEST summarizes the results, it reports the percentage of *first rank decisions* (*flexible match*) ($\tau = 0.02$) as well as the percentage of *only choice decisions* (*100% match*) ($\tau = 0$).

Concepts C1–C4 were learned and tested using data sets characterized in table 8. The summary of the results obtained in ten executions of both systems for each concept learned is presented in table 9. The AQ17-HCI system was able to correctly learn all concepts. The 100% match between the learned concept descriptions and the testing examples shows that all concepts were learned precisely. The AQ15 system did not learn exact descriptions of concepts C1, C2, and C4; however, it was able to recognize them using the flexible matching procedure. The results reported in the "Flexible match" columns are compared with results from other methods in table 11, in the next section.

Table 10 presents results from learning concept C2 using varying numbers of examples in the training set. From the table, it is easy to estimate the number of examples required by a system to achieve a desired prediction accuracy. The table shows that the AQ-HCI method requires a significantly smaller training set to precisely learn the C2 problem. These results are due to better descriptors used in expressing learned concepts both in the learning phase

Table 8. A characterization of experimental domains.

Target concept	No. of attributes	No. of redundant attributes	No. of rules	Average no. of conditions in a rule	No. of training examples	No. of testing examples
C1 (DNF3)	32	12	6	5.5	1650	2000
C2 (DNF4)	64	33	10	4.1	2640	2000
C3 (MX11)	32	21	8	4.0	1600	2000
C4 (PAR5)	32	27	16	5.0	4000	2000

C1 The DNF3 function defined by rules:

x1 x2 x6 x8 x25 x28 ¬x29	=> C1	¬x2 ¬x10 x14 ¬x21 ¬x24	=> C1
x2 x9 x14 ¬x16 ¬x22 x25	=> C1	x11 x17 x19 x21 ¬x25	=> C1
x1 ¬x4 ¬x19 ¬x22 x27 x28	=> C1	¬x1 x4 x13 ¬x25	=> C1

Attributes {x3 x5 x7 x12 x15 x18 x20 x23 x26 x30 x31 x32} are irrelevant, i.e. have random values for each example.

C2 The DNF4 function defined by rules:

x1 x4 x13 x57 ¬x59	=> C2	x18 ¬x22 ¬x24	=> C2
x30 ¬x46 x48 ¬x58	=> C2	¬x9 x12 ¬x38 x55	=> C2
¬x5 x29 ¬x48	=> C2	x23 x33 x40 x52	=> C2
x4 ¬x26 ¬x38 ¬x52	=> C2	x6 x11 x36 ¬x55	=> C2
¬x6 ¬x9 ¬x10 x39 ¬x46	=> C2	x3 x4 x21 ¬x37 ¬x57	=> C2

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} are irrelevant, i.e. have random values for each example.

C3 The C3 function is based on multiplexer-11 function (Wilson, 1987).

¬x1 ¬x2 ¬x3 x4	=> C3	¬x1 ¬x2 x3 x5	=> C3
¬x1 x2 ¬x3 x6	=> C3	¬x1 x2 x3 x7	=> C3
x1 ¬x2 ¬x3 x8	=> C3	x1 ¬x2 x3 x9	=> C3
x1 x2 ¬x3 x10	=> C3	x1 x2 x3 x11	=> C3

Attributes {x12 .. x32} are irrelevant, i.e. have random values for each example.

C4 Parity-5 function with irrelevant attributes .
The function has value *true* on an observation if an even number of relevant attributes {x1..x5} are present, otherwise it has the value *false*.

¬x1 ¬x2 ¬x3 ¬x4 ¬x5	=> C4	¬x1 x2 x3 ¬x4 ¬x5	=> C4
¬x1 ¬x2 ¬x3 x4 x5	=> C4	x1 ¬x2 x3 ¬x4 ¬x5	=> C4
¬x1 ¬x2 x3 ¬x4 x5	=> C4	x1 x2 ¬x3 ¬x4 ¬x5	=> C4
¬x1 x2 ¬x3 ¬x4 x5	=> C4	¬x1 x2 x3 x4 x5	=> C4
x1 ¬x2 ¬x3 ¬x4 x5	=> C4	x1 ¬x2 x3 x4 x5	=> C4
¬x1 ¬x2 x3 x4 ¬x5	=> C4	x1 x2 ¬x3 x4 x5	=> C4
¬x1 x2 ¬x3 x4 ¬x5	=> C4	x1 x2 x3 ¬x4 x5	=> C4
x1 ¬x2 ¬x3 x4 ¬x5	=> C4	x1 x2 x3 x4 ¬x5	=> C4

Attributes {x6 .. x32} are irrelevant, i.e. have random values for each example.

(relations already discovered and stored under new attributes make it possible for a deeper search for dependencies among training data) and the testing phase (the match between an example and a more concise rule results in a higher degree of consonance). The results from this table were combined with results from other methods and are presented in figure 8 in the next section.

Table 9. The experimental results for different problems.

Target concept	Average Error Rate			
	AQ15		AQ17-HCI	
	Flexible match	100% match	Flexible match	100% match
C1	0.3%	1.5%	0.0%	0.0%
C2	0.2%	11.5%	0.0%	0.0%
C3	0.0%	0.0%	0.0%	0.0%
C4	1.6%	18.8%	0.0%	0.0%

Table 10. The experimental results for different numbers of training examples in learning concept C2.

No. of training examples	Average Error Rate in Learning Target Concept C2			
	AQ15		AQ17-HCI	
	Flexible Match	100% match	Flexible Match	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%

Both systems, AQ15 and AQ17-HCI, generate a complete and consistent set of rules from the input examples. Since AQ-HCI involves attributes constructed from AQ15 rules, a question arises: why does AQ17-HCI produce higher accuracy on testing examples? The answer seems to lie in the AQ15 method of generalizing examples. The *extend-against* generalization operator (Michalski, 1983) considers attributes one at a time.[2] 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 to 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 interacting attributes, the systematic transformations in the representation space support the *extend-against* operator in finding more accurate and effective hypotheses.

6.3. Empirical comparison of AQ-HCI with other methods

Figure 8 and table 11 summarize the results obtained in ten executions of all tested algorithms. The results for the REDWOOD, FRINGE, GREEDY3, and GROVE algorithms come from Pagallo and Haussler (1989, 1990).

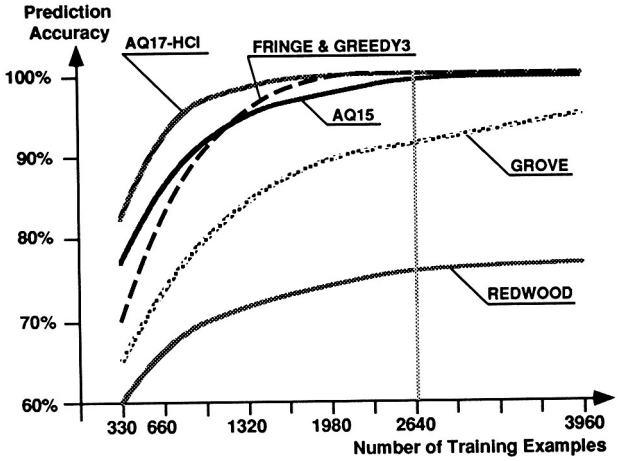

Figure 8. Learning curves for the concept C2 for different systems.

Table 11. Experimental reuslts for testing descriptions of concepts C1–C4 learned by different systems.

Target concept	Average Error Rate					
	Decision TREES (*)		Decision LISTS (*)		Decision RULES (†)	
	REDWOOD	FRINGE	GREEDY3	GROVE	AQ15	AQ17-HCI
C1	7.4%	0.3%	0.6%	1.4%	0.3%	0.0%
C2	24.9%	0.0%	0.0%	7.8%	0.2%	0.0%
C3	13.1%	0.0%	0.5%	3.9%	0.0%	0.0%
C4	36.5%	22.1%	45.8%	41.3%	1.6%	0.0%

*From Pagallo and Haussler (1989, 1990).
†Flexible match.
Concepts C1, C2, C3, and C4 were learned from 1650, 2640, 1600, and 4000 examples, respectively. All concepts were tested on 2000 testing examples.

Figure 8 shows the learning curves for the concept C2. The curves were obtained by measuring and averaging prediction accuracy 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 2640 training examples. However, convergence to 100% is fastest in the case of AQ17-HCI. The exact results for 2640 examples are given in the row describing concept C2 in table 11.

Table 11 shows the results obtained from testing concepts C1–C4 (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.

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

7. Conclusions

The presented AQ-HCI method of constructive induction generates new attributes by analyzing and abstracting inductive concept hypotheses, rather than by directly combining different attributes. In this method the search for new attributes is very efficient, although more limited in the repertoire of attributes that can be constructed by direct, data-driven methods. In our experiments, the proposed method performed very favorably, in terms of prediction accuracy, in comparison to methods employed in such programs as AQ15, REDWOOD, FRINGE, GREEDY3, and GROVE.

In the AQ-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, symbolic numeric, and numeric, as well as structured (where domains are hierarchies). The algorithm detects redundant or insignificant 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 AQ-HCI method has shown to be effective in improving performance accuracy in a wide range of DNF-type problems. Generated attributes are rather complex, and therefore the overall complexity of the descriptions is increased. In future research, we plan to investigate attribute generation based on selected components of the best performing rules rather than 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 (Arciszewski, Dybala, & Wnek, 1992).

The proposed methodology is general, and can be potentially applied not only with the A^q-type rule learning method, but with other inductive learning methods, and with different knowledge representations, such as frames, semantic networks, decision trees, etc. To do so, one needs to identify types of patterns in hypotheses to be searched for, evaluated, and selected as new attributes. It is likely that employing the proposed methodology within any "nonconstructive" inductive learning system will improve its performance. In this sense it represents a universal new approach to constructive induction.

In multiple concept learning, in addition to finding value-patterns, condition-patterns and rule-patterns, it might be desirable to find patterns across class descriptions, *class-patterns*. Such patterns represent conditions that are common for a subset of classes, and distinguish this subset from other classes.

Preliminary studies indicate that intermediate generalizations tend to produce highest prediction accuracy, and therefore we have chosen this type of descriptions in Phase 2 of the method. However, further studies are needed to investigate this problem in a more systematic way.

Acknowledgments

The authors thank Giulia Pagallo for the help in the design of the experiments, Kenneth De Jong and George Tecuci for comments on the earlier version of this article, and Eric Bloedorn and Mike Hieb for the final review.

This research was conducted in the Center for Artificial Intelligence at George Mason University. The Center's research is supported in part by the Advanced Research Projects Agency under Grant No. N00014-91-J-1854, administered by the Office of Naval Research, and the grant No. F49620-92-J-0549, administered by the Air Force Office of Scientific Research, in part by the Office of Naval Research under Grant No. N00014-91-J-1351, and in part by the National Science Foundation under Grants No. IRI-9020266 and CDA-9309725.

Notes

1. The system DIAV implementing the visualization method (Wnek & Michalski, 1993) permits one to directly display description spaces up to 10^6 instances (e.g., about 20 binary attributes). Larger spaces can also be displayed, but their representations have to be projected to subspaces.
2. One way to address this problem can be a lookahead technique to detect interaction between attributes, but this increases computational cost (Rendell & Seshu, 1990).

References

Arciszewski, T., Dybala, T., & Wnek, J. (1992). A method for evaluation of learning systems. *HEURISTICS, The Journal of Knowledge Engineering*, Special Issue on Knowledge Acquisition and Machine Learning, *5*, *4*, 22–31.

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. (1991). Data driven constructive induction in AQ17-PRE: A method and experiments. *Proceedings of the Third International Conference on Tools for AI*. San Jose, CA.

Bloedorn, E., Michalski, R.S., & Wnek, J. (1993). Multistrategy constructive induction: AQ17-MCI. *Proceedings of the Second International Workshop on Multistrategy Learning* (pp. 188–206). Harpers Ferry, WV: Morgan Kaufmann.

Breiman, L., Friedman, J.H., Olshen, R.A., & Stone, C.J. (1984). *Classification and regression trees*. Belmont: Wadsworth.

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.

De Raedt, 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: Morgan Kaufmann.

Duda, R., Gasching, J., & Hart, P. (1979). Model design in the Prospector Consultant System for Mineral Exploration. In D. Michie (Ed.), *Expert systems in the micro electronic age*. Edinburgh: Edinburgh University Press.

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: Morgan Kaufmann.

Falkenhainer, B.C., & Michalski, R.S. (1990). Integrating quantitative and qualitative discovery in the ABACUS system. In Y. Kodratoff & R.S. Michalski (Eds.), *Machine learning: An artificial intelligence approach* (Vol. III). Palo Alto, CA: Morgan Kaufmann.

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.

Flann, N.S. (1990). Improving problem solving performance by example guided reformulation of knowledge. In D.P. Benjamin (Ed.), *Change of representation and inductive bias*. Boston, MA: Kluwer Academic.

Greene, G.H. (1988). *The Abacus 2 system for quantitative discovery: Using dependencies to discover non-linear terms* (Reports of Machine Learning and Inference Laboratory, MLI 88-4). Center for Artificial Intelligence, George Mason University, Fairfax, VA.

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.

Knoblock, C.A. (1990). A theory of abstraction for hierarchical planning. In D.P. Benjamin (Ed.), *Change of representation and inductive bias*. Boston, MA: Kluwer Academic.

Knoblock, C.A., Minton, S., & Etzioni, O. (1991). Integrating abstraction and explanation-based learning in PRODIGY. *Proceedings of AAAI-91* (pp. 541–546). Cambridge, MA: AAAI Press/The MIT Press.

Kokar, M.M. (1985). Discovering functional formulas through changing representation base. *Proceedings of AAAI-86* (pp. 455–459). Philadelphia, PA.

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.

Langley, P.W., Simon, H.A., Bradshaw, G.L., & Zytkow, J.M. (1987). *Scientific discovery: Computational explorations of the creative processes*. Cambridge, MA: The MIT Press.

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). *ACM SIGART Newsletter, 63*, 38–44.

Lenat, D.B. (1977). On automated scientific theory formation: A case study using the AM program. In J.E. Hayes, D. Michie, & L.I. Mikulich (Eds.), *Machine intelligence 9*. New York: Halsted Press.

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. Dissertation, University of Illinois, Computer Science Department, Urbana-Champaign.

Michael, J. (1991). *Validation, verification, and experimentation wth Abacus2* (Reports of the Machine Learning and Inference Laboratory, MLI 91-8). Fairfax, VA: Center for Artificial Intelligence, George Mason University.

Michalski, R.S., & McCormick, B.H. (1971). Interval generalization of switching theory. *Proceedings of the Third Annual Houston Conference on Computer and System Science*, Houston, Texas, April 26–27.

Michalski, R.S. (1973). AQVAL/1—Computer implementation of a variable-valued logic system VL_1 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, IL: University of Illinois, Department of Computer Science.

Michalski, R.S., & Larson, J.B. (1978). *Selection of the most representative training examples and incremental generation of VL1 hypotheses: The underlying methodology and the description of programs ESEL and AQ11* (Technical Report, 867). Urbana-Champaign, IL: Department of Computer Science Department, University of Illinois.

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). Philadelphia, PA.

Mitchell, T.M., Utgoff, P.E., & Benerji, 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*. New York/Berlin: Springer-Verlag.

Muggleton, S. (1987). Duce, an oracle-based approach to constructive induction. *Proceedings of IJCAI-87* (pp. 287–292). Milan, Italy: Morgan Kaufmann.

Muggleton, S., & Buntine, W. (1988). Machine invention of first order predicates by inverting resolution. *Proceedings of the 5th International Conference on Machine Learning* (pp. 339–352). Ann Arbor, MI: Morgan Kaufmann.

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. (1986a). Induction of decision trees. *Machine Learning, 1*, 81–106.

Quinlan, J.R. (1986b). The effect of noise on concept learning. In R.S. Michalski, J.G. Carbonell, & T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 11). Los Altos, CA: Morgan Kaufmann.

Quinlan, J.R. (1989). Documentation and user's guide to C4.5. Unpublished.

Reinke, R.E. (1984). *Knowledge acquisition and refinement tools for the ADVISE meta-expert system*. Master's thesis, University of Illinois, Urbana-Champaign.

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.

Schlimmer, J.C. (1987). Learning and representation change. *Proceedings of AAAI-87* (pp. 511–515). Morgan Kaufmann.

Subramanian, D. (1990). A theory of justified reformulations. In D.P. Benjamin (Ed.), *Change of representation and inductive bias*. Boston, MA: Kluwer Academic.

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, 6*, 1444–1460.

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.

Thrun, S.B., Bala, J.W., Bloedorn, E., Bratko, I., Cestnink, B., Cheng, J., DeJong, K.A., Dzeroski, S., Fahlman, S.E., Hamann, R., Kaufman, K., Keller, S., Kononenko, I., Kreuziger, J., Michalski, R.S., Mitchell, T., Pachowicz, P., Vafaie, H., Van de Velde, W., Wenzel, W., Wnek, J., & Zhang, J. (1991). *The MONK's problems: A performance comparison of different learning algorithms* (Computer Science Reports, CMU-CS-91-197). Pittsburgh, PA: Carnegie Mellon University.

Utgoff, P.E. (1984). *Shift of bias for inductive concept learning*. Ph.D. dissertation, Rutgers University, New Brunswick, NJ.

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.

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., & Michalski, R.S. (1993). *A diagrammatic visualization of learning processes* (Reports of Machine Learning and Inference Laboratory, to appear). Fairfax, VA: George Mason University, Center for Artificial Intelligence.

Wnek, J. (1993). *Hypothesis-driven constructive induction*. Ph.D. dissertation, School of Information Technology and Engineering, George Mason University, Ann Arbor, MI: University of Microfilms, Inc.

Wrobel, S. (1989). Demand-driven concept formation. In K. Morik (Ed.), *Knowledge representation and organization in machine learning*. Berlin Heidelberg: Springer Verlag.

Received February 2, 1992
Accepted June 4, 1992
Final Manuscript January 20, 1993