CONSTRUCTIVE INDUCTION: An Automated Improvement of Knowledge Representation Spaces for Machine Learning

R.S. Michalski and J. Wnek

MLI 93-11

CONSTRUCTIVE INDUCTION:

An Automated Improvement of Knowledge Representation Spaces for Machine Learning

Abstract

Most machine learning methods assume that the original representation space is adequate, that is, the initial attributes or terms are sufficiently relevant to the problem at hand. To cope with learning problems in which this assumption does not hold, the idea of *constructive induction* has been proposed. A constructive induction system conducts two searches, first for an improved representation space, and the second for the "best" hypothesis in this space. Research on constructive induction is reviewed and a method for *hypothesis-driven constructive induction* (AQ-HCI) is presented. The method searches for an adequate representation space by analyzing the hypotheses generated in each step of an iterative double-search learning process. In an experimental study, the method outperformed all learning methods that were tested. Also, it achieved the top performance in solving the so-called Monks' problems that were used as a benchmark in the first international competition of learning programs. The conclusion outlines several open problems in this area.

Key words: symbolic learning, decision rule learning, constructive induction, representational bias

Acknowledgments

The authors thank Giulia Pagallo for the help in the design of the experiments, Kenneth De Jong, George Tecuci, Eric Bloedorn and Mike Hieb for comments.

This research was conducted in the Center for Artificial Intelligence at George Mason University. The Center's research is supported in part by the National Science Foundation under Grants No. CDA-9309725 and IRI-9020266, 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, and in part by the Office of Naval Research under Grant No. N00014-91-J-1351.

1 Introduction

Most research in machine learning has been concerned with learning concept descriptions from examples in an a priori defined representation space. In such methods, the hypothesis learned is expressed using attributes or terms selected from among those initially provided. For this reason, such inductive learning methods are sometimes called *selective* (Michalski, 1983; Rendell, 1985).

To overcome this limitation, the idea of *constructive* induction has been proposed (Michalski, 1978). A constructive induction system performs a double, mutually intertwined search, one for the most suitable representation space, and second for the "best" concept description in this space, for example, the simplest description. The system includes mechanisms for generating new, more relevant descriptors, as well as for modifying or removing less relevant ones from those initially provided. In general, a constructive induction system perform a *problem-oriented transformation* of the original knowledge representation space.

The primary goal of constructive induction may be to maximize the overall *prediction accuracy* of a concept description, to minimize the *overall complexity* of a description, or to optimize the description according to a combination of both criteria. The prediction accuracy is measured by applying the generated concept description 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 number and the complexity of attributes used in the description, and the types of operators employed.

The primary goal of the method described here is to learn a concept description that has the highest prediction accuracy on the unseen concept examples. The search for such a description is done by iteratively changing representation space, and measuring the prediction accuracy of the generated description in this space. The prediction accuracy is estimated by applying the description to concepts examples that were not used in the learning process.

This paper briefly reviews and classifies research on constructive induction, and then presents a new method in which the desirable changes in the representation space are determined by analyzing hypotheses generated in each iteration of the learning process. For that reason, the method represents a *hypothesis-driven* constructive induction (HCI) approach, as opposed to a *data-driven* (DCI) approach, which is based on the analysis of data (concept examples). The presented method employs the well-known AQ inductive learning algorithm, therefore the method is called AQ-HCI.

To relate the presented method to other methods, Section 2 provides a summary and a classification of existing constructive induction methods. Section 3 gives a description of the proposed AQ-HCI method. Sections 4 and 5 examine the performance of the method on two problems. Transformations of the representation space are illustrated by *diagrammatic visualization*. Section 6 describes an experimental comparison of the method with several symbolic and subsymbolic learning systems. 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* was first proposed by Michalski (1978), and implemented in INDUCE-1 system for learning structural descriptions from examples (Larson & Michalski, 1977, Michalski, 1980). 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 descriptors in the process of induction. A number of other systems that exhibit various 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 improving the representation space, in particular, for generating new descriptors[1]. Based on the primary strategy employed, existing constructive induction systems can be divided into four categories: data-driven, hypothesis-driven, knowledge-driven, and multistrategy. Below is a brief characterization of these categories, and selected representative systems in each category.

2.1 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 similar fashion as BACON), and applying methods of symbolic induction for defining the applicability conditions for the equations (Falkenhainer & Michalski, 1990; Greene 1988; Michael, 1991).

PLS0 (Probabilistic Learning System) 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 (Induction-Over-Explanations) 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 (Factored Candidate Elimination) 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).

2.2 Hypothesis-Driven Constructive Induction Systems (HCI)

These systems incrementally transform the representation space by analyzing inductive hypotheses generated in one iteration and 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).

---

1 By descriptors are meant attributes, predicates, functions, relations, transformations, etc. that span the knowledge representation space in which the learning process occurs.

CITRE (Constructive Induction on decision TREes) 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).

AQ17-HCI (an earlier implementation of the system presented here) creates new attributes from admissible sets of rules, i.e. rule patterns (Wnek & Michalski, 1991).

AQ-PRAX introduces new attributes called principal axes. The attributes are defined by AQ-generated rules and a non-linear similarity measure (Bala, Michalski & Wnek, 1992).

2.3 Knowledge-Driven Constructive Induction Systems (KCI)

These systems apply expert-provided domain knowledge to construct and/or verify new representation space.

AM (Automated Mathematician) program changes its representation space by employing predefined heuristics for: (1) defining new concepts represented as frames, (2) creating new slots and/their values, (3) adapting concept frames developed in one domain to another domain (Lenat, 1977, 1983).

SPARC/E (Sequential PAttern ReCognition) program for learning rules in the game Eleusis changes the representation space by *adding derived attributes*. The new attributes make explicit certain commonly known characteristics of playing cards that are likely to be used in an Eleusis rule. Definitions of the attributes are provided by a user (Dietterich & Michalski, 1983, 1985, 1986).

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

2.4 Multistrategy Constructive Induction Systems (MCI)

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

INDUCE-1 (KCI & DCI) applies *inductive generalization rules* selected by a user from a predefined repertoire and/or built-in procedures for generating new attributes. The rules are applied either directly (KCI) or according to the results of the analysis of the properties of the structural descriptions of training examples (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 (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 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 (equivalent to *rule-pattern operator* in AQ17-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 intermediate concepts 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 (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, AQ17-DCI and AQ17-HCI (Bloedorn, Michalski & Wnek, 1993).

3 Representation Space Transformations

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

3.1 Contraction

From the viewpoint of algorithmic efficiency, it is desirable to maximally reduce the representation space, while preserving its ability to adequately describe concepts to be learned. The AQ-HCI method removes from the representation space attributes considered as insufficiently relevant. The latter ones can be either *redundant* or *insignificant*. The redundant attributes are defined as those that do not occur in the concept description (a set of rules) generated by the employed selective induction algorithm (here, AQ). The insignificant attributes are those that occur only in the "low strength" rules (that cover only very small number of examples). The importance of the space 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). Another form of contraction is to agglomerate values of an attribute into larger units. This is done by creating more abstract values of attributes. These are useful operations, because overly precise attributes can cause an *overfitting* of the hypotheses.

3.2 Expansion

When the original representation space is determined to be of insufficient quality, it needs to be extended by adding new attributes. One method for generating new attributes is to invent some new physical processes that allow the measurement of the previously unknown or undetectable object properties. This is often very difficult and is normally done by an expert in a given domain (a physicist, chemist, etc.). Another method is to search a certain functions or relations among the existing attributes for combinations that demonstrate a high relevance to describing concepts to be learned. Such combinations are given names, and serve as new constructed (or derived) attributes. Adding constructed attributes to the representation space is a space expansion operation. The AQ-HCI algorithm generates derived attributes by detecting various "patterns" in the descriptions obtained in consecutive iterations of the learning process. These patterns are described below.

3.3 Patterns in Concept Descriptions

By a *pattern* is meant here a component of a generated concept description that is characteristic of a relatively large number of concept examples. The basic idea behind the AQ-HCI method is to search for different patterns in the hypotheses generated in each iteration of the learning process. Detected patterns are viewed as relevant intermediate knowledge transformations and treated as new derived (constructed) attributes. Each pattern is assigned a *pattern strength*, which reflects the number of examples covered (or explained) by the pattern. Such a pattern strength is viewed as a measure of the importance of the attribute constructed on the basis of this pattern (see Sec. 3.4).

A concept description is in the form of a set of rules (a ruleset). Different pattern types correspond to different components of such a ruleset. There are *value-patterns*, *condition-patterns* and *rule-patterns*. A value-pattern represents a subset of attribute values that satisfies the pattern criterion. A condition-pattern represents a conjunction of two or more elementary conditions. A rule-pattern is a subset of the rules in a given ruleset. A value-pattern or a condition-pattern represents an overgeneralization, if considered by itself as a concept description. A rule pattern represents an overspecialization if considered by itself as a concept description. Thus, patterns represent statements about the input data that can be either an overspecialization or an overgeneralization. These patterns give the learning system powerful mechanisms for knowledge space transformation.

To illustrate the concept of derived attributes and different patterns in a description, let's consider an example. Table 1 shows initial attributes spanning a description space and their value sets. Table 2 shows examples of constructed attributes that correspond to different types of patterns. These patterns are expressed in terms of initial attributes.

Table 1. Examples of attributes.

Attribute	Value set	Explanation
x	1..100	(Attribute x can take values from 1 to 100)
y	small, medium, large	(y is a symbolic attribute)
z	white, red, blue, green, black	(z is a symbolic attribute)

Table 2. Examples of constructed attributes.

Attribute definition	Value set	Explanation
ca1 <:: (z = blue v red v white)	0 or 1	(Value-pattern)
ca2 <:: (x = 20) & (y = large)	0 or 1	(Condition-pattern)
ca3 <:: ((x = 75..100) & (y = small)) or (x < 20)	0 or 1	(Rule-pattern)

The first constructed attribute (ca1) is based on a value-pattern. It states a condition that z should take value blue, red or white. If the condition is satisfied, the attribute takes value 1, otherwise 0. The second constructed attribute, ca2, is based on a condition-pattern. It is a conjunction of two conditions. The third attribute, ca3, is based on a rule-pattern. It is a disjunction of two rules (conjunctive terms). The first is a conjunction of two conditions and the second is a single condition.

Let us now explain our description language. The constructed attributes, as well as *the concept descriptions are expressed in the form of DNF expressions of* variable-valued logic system VL1 (Michalski, 1975). Elementary conditions in such an expression are relations between an attribute and a set of its values:

$$x \# R$$

where x is an attribute, # denotes a relation (=, <, >), R denotes a set of attribute values that is represented as an internal disjunction of values (a v b v ....), or a range (a..b). For example, an elementary condition can be represented in such forms as x =1, x > 3, x = 2..7, x = A v B v D. A VL1 expression is equivalent to a set of conjunctive rules (a ruleset). Such a ruleset can be represented as a two dimensional matrix C(n m), where n is the number of attributes, and m is the number of rules. Each C(i, j) element in this matrix is a set of values that attribute (i) is assigned to in rule (j).

3.4 Determining the Pattern Strength

As mentioned earlier, every pattern in a ruleset is assigned a pattern strength. In the case of learning single concepts, a pattern strength is a function of the number of positive examples, PCov, and the negative examples, NCov, that are covered by (or satisfy) the pattern:

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

To determine a desirable form of the function f, let us observe that the strength of a pattern should be positively correlated with the number of positive examples covered by it, and negatively correlated with the number of negative examples covered by it. In addition, the definition of a pattern strength may distinguish between types of coverage of concept instances by the pattern. For example, a concept instance may be covered only by the given pattern (a unique coverage), or maybe covered also by other patterns (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 (σ) that reflects 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.

$\lambda \geq 0$ is a parameter that controls the relative importance of these two types of coverage.

When $\lambda = 0$, i.e., when the unique weight is ignored, the above measure 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).

The pattern strength function, as defined in equation (2), is used to determine which patterns are admissible as new attributes. The function represents a degree to which an unknown example that satisfies a pattern can be believed to be an instance of the concept.

To explain the σ function (eq. 2), let us assume that $\lambda = 0$, i.e., the unique weights are ignored. In this case, the function ranges from zero to the number of positive examples. When a pattern is not matched by any positive example, σ(pattern) is equal to 0. Such a pattern is least useful for describing the concept. When the pattern is not matched by any negative examples, σ(pattern) equals the number of positive examples. If σ(pattern) is greater than 1, the pattern matches more positive negative than negative examples. Patterns that match many more positive than negative examples may be considered useful for constructing new attributes. In the AQ-HCI method, pattern selection is additionally restricted by a condition that a pattern to be selected should have strength greater than the strengths of all conditions involved in the pattern description.

3.5 Determining an Admissible Ruleset

The AQ-HCI method works iteratively. Each iteration generates a complete and consistent concept description, that is, a ruleset that covers all positive examples and none of the negative examples. To speed up the search for patterns, 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, employs patterns for transforming the representation space, and then moves to the next iteration (the details are described in Section 6).

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

$$\sigma(rule) = t(rule) + \lambda u(rule) \tag{3}$$

where, t is the total weight of a rule in a ruleset (the total number of positive 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 method'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:

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

where σ_i is the strength of rule (i) defined by equation (3), m the total number of rules in the current hypothesis, m' ($m' \leq m$) is the number of the strongest rules (recall that the rules are ordered, thus $\sigma_i \geq \sigma_{i+1}$). The threshold TH is chosen on the basis of an estimate of the possible noise level the data. Noisy examples are typically covered by low-strength rules, therefore the admissible ruleset is selected to cover the most "central" portion of the concept to be learned. The threshold level may also reflect the prior knowledge about the concept. For example, if it is known that the concept is non-DNF, then the threshold can be set very high.

3.6 The AQ-HCI Method

The AQ-HCI method combines an inductive rule learning algorithm (implemented in AQ15) with an HCI procedure for iteratively transforming a representation space. In each iteration, the method changes the representation space by adding new attributes determined on the basis of detected patterns, and removing insufficiently relevant attributes. 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. Fig. 1 presents a diagram illustrating the method.

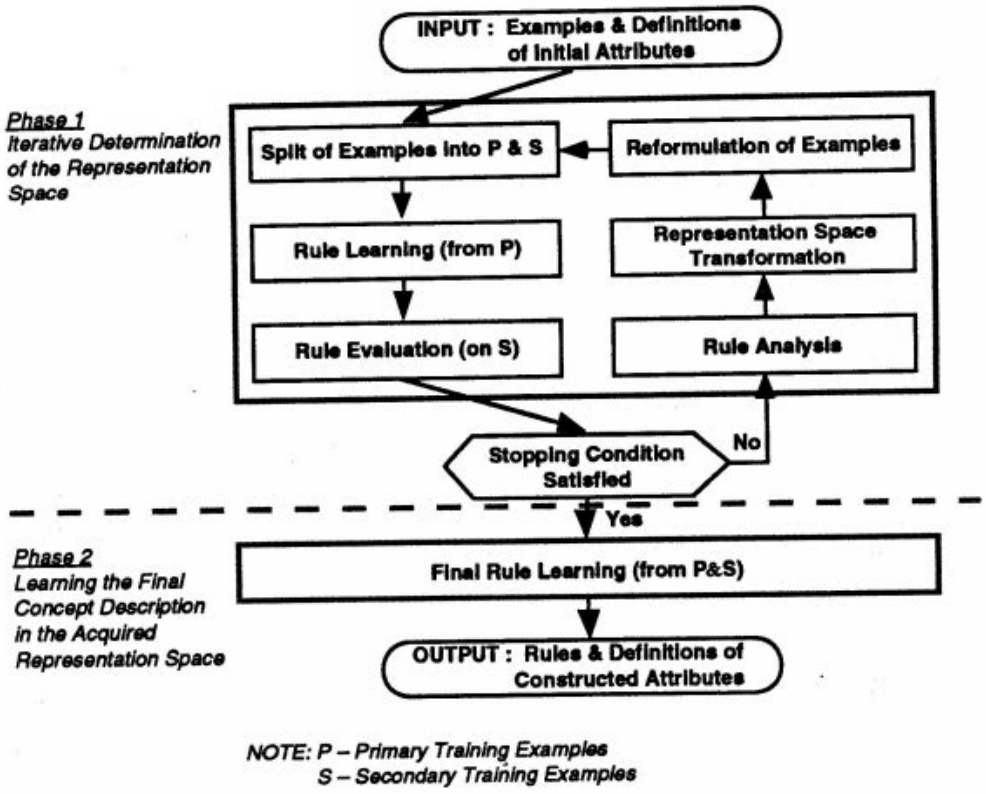

Fig. 1. The method for Hypothesis-driven Constructive Induction.

The input consists of training examples of one or more concepts, and 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.

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 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 total set, $P \cup S$, is used 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 A^q algorithm for solving the general covering problem and for minimizing Boolean functions of a very large number of variables (e.g., Michalski & McCormick, 1971; Michalski, 1973). For completeness, Section 3.7 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 analyzes the rules in the ruleset for each class to determine desirable changes in the representation space. This process includes a determination of the attributes that are to be removed from the space and a generation a constructed attributes that are to be added to the representation space. The last ones represent strong patterns found in the admissible ruleset generated in the current iteration. Section 3.8 gives more details about this process.

The "Representation Space Transformation" module modifies the space according to the findings in the previous module. It removes redundant and insignificant attributes, modifies existing attributes (by attribute value agglomeration), and adds to the space constructed attributes.

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 a set of 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 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. The idea of such an intermediate cover was proposed and implemented by Wnek (1992).

3.7 Rule Learning Module

As mentioned earlier, initial and consecutive hypotheses are generated by the inductive rule learning program AQ15 (Michalski et al., 1986). The program learns rules from examples represented as vectors of attribute-value pairs. Attributes can be multiple-valued and can be of different types, such as symbolic, numerical, cyclic 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 DNF disjunctive normal form 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, 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 (Michalski & 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.

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 intermediate knowledge for selecting useful patterns for constructing new attributes. Such descriptions should reveal the necessary conditions for the best classification of the available data, and thus help 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, may cause ambiguity in the representation space. The ambiguity could be introduced if two or more new attributes, created from rulesets of different concept descriptions, had overlapping definitions, and at the same time some existing attributes, relevant for describing the overlap, were removed. To prevent creating such ambiguity, the method in phase 1 assumes the generation of 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 representation space to generate final hypotheses which are at an intermediate level of generalization that is desirable for achieving the highest possible prediction accuracy (Wnek 1992).

3.8 Rule Analysis Module and Pattern Determination

Given a ruleset representing a concept description generated in a given iteration of the algorithm, the "Rule Analysis" module analyzes the rules to determine desirable changes in the representation space. This process includes a determination of the attributes that should be removed from the space, a modification of some attributes, and a generation a constructed attributes that are to be added to the representation space. The last ones represent strong patterns found in the generated rulesets for each class.

Attributes to be removed are those that are redundant (do not occur in the description) or insufficiently relevant. The latter ones are determined by assigning the *relevance* to each attribute in the concept description. The relevance of an attribute is defined as the sum of the t-weights (the number of positive examples covered) of the rules containing the attribute in the ruleset. There is a parameter that defines which attributes are sufficiently relevant.

The attribute modification and construction are done on the basis of the patterns determined in the rulesets for each class. Here is an algorithm for pattern search:

1. *Determine the admissible ruleset (ARSET).*
   Patterns are searched for only in this ruleset, in order to avoid searching through weak rules.
2. *Determine a logical intersection of the rules in ARSET.*
   This step produces a conjunction of conditions that are common for all rules in the ruleset.
3. *Determine candidate conjunctive patterns*
   The conjunction of conditions determined in the previous step plus all of its subconditions (of the length one or more) are viewed as candidate patterns. A conjunction of two or more conditions constitutes a condition-pattern; single conditions constitute value-patterns.
4. *Determine additional subconjunctive patterns*
   Reduce ARSET by one rule in every possible way and repeat steps 2 and 3 to determine additional patterns. Continue until the ARSET has no rules or the number of candidate patterns reaches an assumed limit.
5. *Select value-patterns and condition-patterns*

   Evaluate the strength of all the generated patterns and select a predetermined number of the strongest ones. The result is a set of value-patterns and condition-patterns.
6. *Determine rule-patterns*

   There are two methods for determining rule-patterns. The early method, "top-down," was employed in our earlier work (Wnek and Michalski, 1991). Here, the whole admissible set was selected as a rule-pattern. The new "bottom up" method works as follows. For each pair of candidate patterns determined in steps 1-4, create a disjunction of them, and evaluate its strength. Repeat this operation for three and more patterns, until the number of patterns reaches a predefined threshold.
7. *Select rule-patterns*

   Evaluate the strength of the determined candidate rule-patterns and select form them a predetermined number of the strongest ones to be rule-patterns.

The so determined patterns are assigned names, and used as attributes for determining a representation space for the next iteration of the algorithm.

The ability to introduce new attributes in the form of subconcepts corresponding to patterns found in subsequently generated rules makes two implicit extensions to the representational capabilities of the AQ family programs:

(1) Ruleset–to–condition operator. This operator substitutes a simple condition for a DNF expression. For example, following the domain description from Table 1, the system can create and use the following condition:

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

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

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

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 the attribute values, and some areas may contain a high concentration of concept examples. The higher the concentration of examples of a specific concept in a some area, the easier it is to describe and generalize these examples. Such an effect is a desirable consequence of the representation space change.

It may be interesting to note, that the idea of the so-called "W-operator" (Muggleton, 1988) is closely related to the detection of condition-patterns in rules for different concept descriptions (called the "inter-construction" W-operator) or rule-patterns (called the "intra-construction" W-operator). The presented concepts of value-patterns, condition-patterns and rule-patterns can be applied not only within the attributional description language used here, but also within a relational description language, e.g., predicate calculus or annotated predicate calculus (Michalski, 1983).

4 Case Study I: Representation Space Contraction

4.1 The Initial Representation Space

The testing domain in this study is the world of robot-like figures in the EMERALD[2] system (Kaufman, Michalski & Schultz, 1989). For simplicity's sake, the robots are described by just six multivalued attributes (Fig. 3A). The attributes are Head Shape, Body Shape, Smiling, Holding, Jacket Color and Tie, and can have 3, 3, 2, 3, 4, and 2 values, respectively. Consequently, the size of event space (the space of all possible robot descriptions) is 3 x 3 x 2 x 3 x 4 x 2 = 432. The space of all possible concepts in this space is 2^{432} -1 (≈10^{143}). Below is a description of a target concept used in the experiment with the total numbers of positive and negative examples:

C_1: *Head is round and jacket is red or head is square and is holding a balloon*

(84 positive, 348 negative)

The concept is presented graphically in Fig. 2B 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 &

---

2 EMERALD is a large-scale system integrating several different learning programs for the purpose of education and research in machine learning. It was developed at the Center for Artificial Intelligence at George Mason University. An earlier version, ILLIAN, was developed at the University of Illinois at Urbana-Champaign.

3 The system DIAV implementing the visualization method permits one to directly display description spaces as many as 10^6 cells (e.g., about twenty binary attributes). Larger spaces can also be displayed but their representations must be projected to subspaces.

Michalski, 1993). Each cell in the diagram represents a combination of the attribute values, e.g., a concept example. For example, the cell A in Fig. 2B represents the following robot description:

Head Shape = round, Body Shape = round, SMiling = yes

HOlding = flag, Jacket Color = green, TIe = no

Positive and negative training examples are marked with + and -, respectively. Concepts are represented as sets of cells. The concept C1 can be described by two rules:

R1: **Head Shape is round** *and* **Jacket Color is red**

R2: **Head Shape is square** *and* **is HOlding balloon**

The rules are represented in the diagram by shaded areas marked R1 and R2.

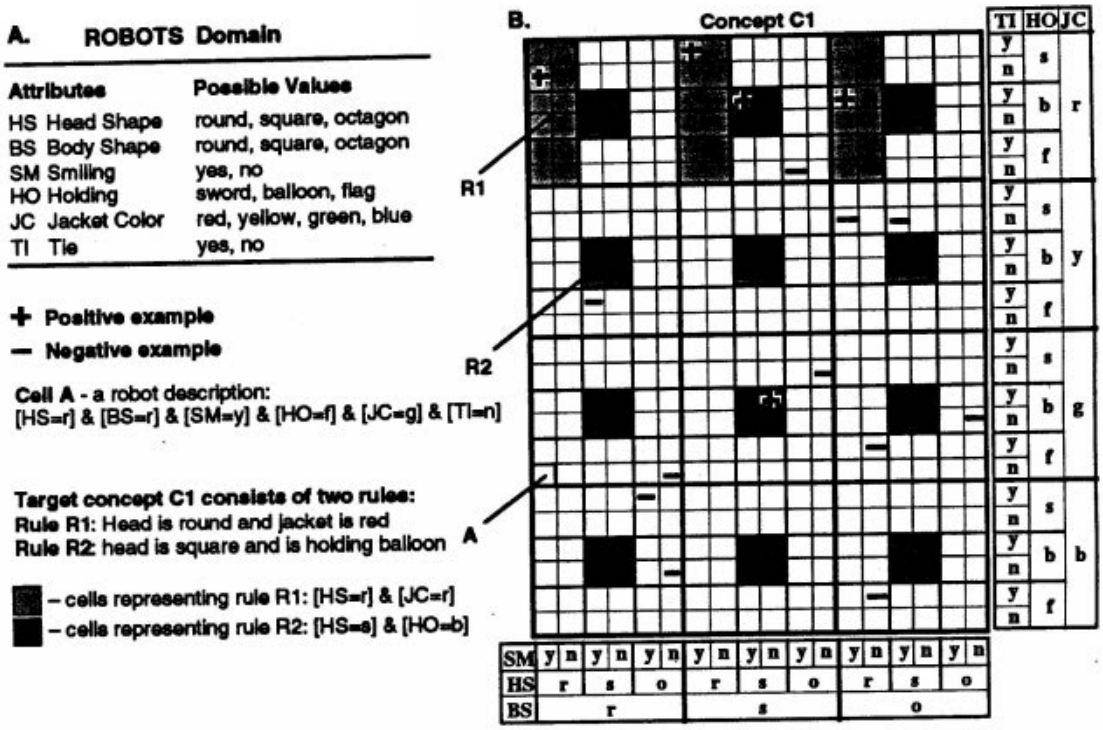

Fig. 2. The target concept and initial training examples in the ROBOTS domain.

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 the *target concept image (T)*. The set of cells representing the learned concept is called the *learned concept image (L)*. The areas of the *target concept* not covered by the *learned concept* represent *errors of omission* ($T \setminus L$), while the areas of the learned concept not covered by the target concept represent *errors of commission* ($L \setminus T$). The union of both types of errors represents the *error image*. In the diagrams, errors are marked by slanted lines.

Fig. 3 explains the meaning of various cases in concept visualization. Concept images are represented in the diagrams by shaded areas, e.g. Figures 3A and 3B. If the target and learned concepts are visualized in the same diagram, then the shaded areas represent learned concept (Fig. 3C). The 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, 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 only shaded.

The descriptions learned by the methods were compared in terms of the *exact error rate*. *Exact error rate* is the ratio between exact error and the size of event space. It is measured as a function of the number of training examples. *Exact error* is defined as the total number of errors of omission and errors of commission, or equivalently the cardinality of the set-difference between the union and the intersection of the target and learned concepts.

$$Exact\_error\_rate = \frac{Exact\_error}{\#\,Event\_space} \tag{7}$$

where $Exact\_error = \#[(T \setminus L) \cup (L \setminus T)] = \#[(T \cup L) \setminus (T \cap L)]$

where $(T \setminus L)$ — error of omission, $(L \setminus T)$— error of commission

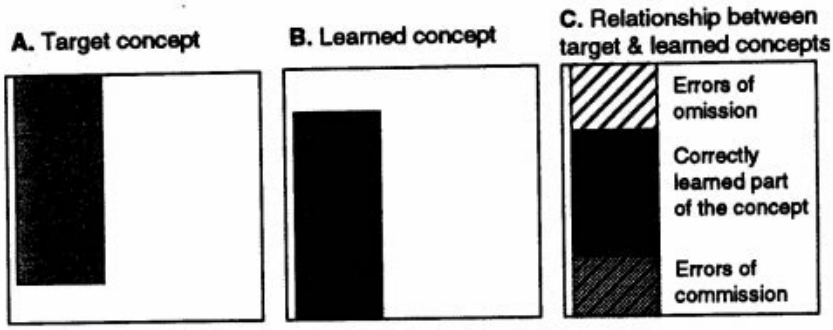

Fig. 3. An illustration of a target concept, learned concept and their interrelationship.

There are many ways to define error rates in order to characterize learning capabilities of a system. Here are three assumptions related to the equation (7). Firstly, for simplicity, we do not make any distinction between errors of omission and errors of commission, which may be important in some real-world domains. Secondly, the equation (7) indicates that if the event space were large and the target concept were relatively small, then the error rate would always be small, and thus, not sensitive to learning errors. Fig. 4 illustrates two cases where the equation (7) would give the same error rate for different learning results. In the first case (Fig. 4A), a system did not learn any part of the target concept and still maintained 2% error rate, or in other words, it was 98% accurate! Since the correct performance of a system is artificially increased by the system's performance on non-examples of a concept, the error estimating method is subjected to the Hempel's paradox (Hempel, 1965; Kodratoff, 1992).

In the second case (Fig. 4B), where the target concept occupies relatively large portion of the event space, the 2% error rate intuitively reflects the true performance of a system. In our experiments the representation space is small and the target concepts cover approximately 30%.

The third assumption related to equation (7) is that in order to get complete insight into the performance of the tested methods we used all examples from the event space to test the performance. In other studies, training examples might have been excluded from the testing phase.

4.2 Steps in Learning Concept C1

The rule learned by AQ17-HCI was exactly equivalent to the target concept (Fig. 5A). It was generated in a transformed, smaller description space. Fig. 6 shows the steps in learning concept C1 by AQ17-HCI. The input to the method is a set of training examples in the original representation space as shown in the diagram A.

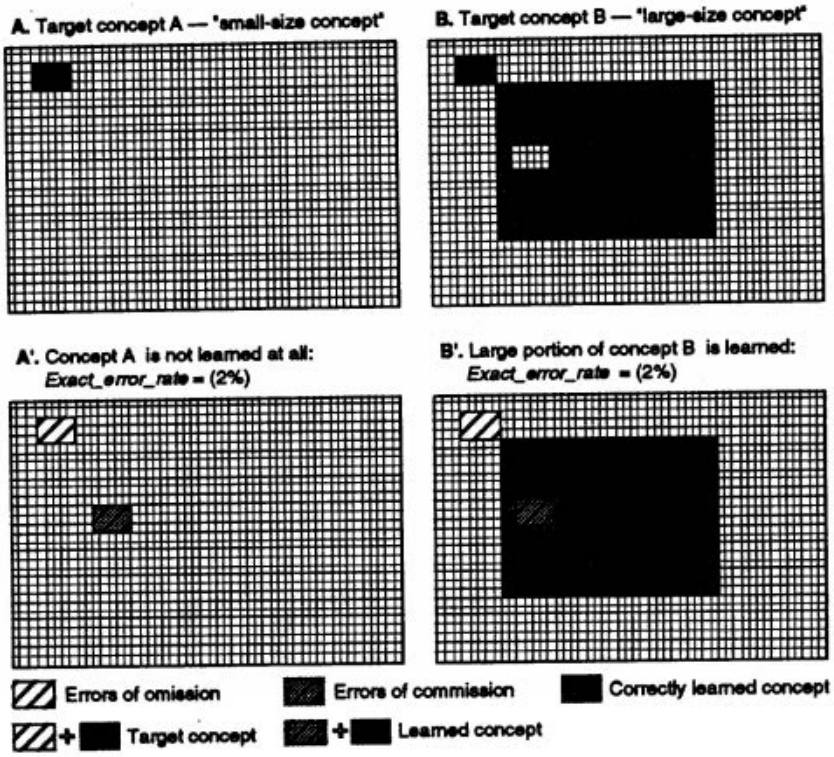

Fig. 4. The dependence of the error rate definition on the relative size of the target concept.

The method divides the training set into primary and secondary examples and employs the AQ15 learning algorithm to induce rules from the primary set of training examples (diagram B). Since the performance test on the secondary training set is not satisfactory, the representation space is reduced to contain relevant attributes only, i.e., those attributes that are present or significant in the induced hypothesis. The method changed the ROBOTS original representation space by removing three irrelevant attributes: Body Shape, SMiling, and TIe (diagram C). The new representation space implied changes in the event space so the number of training examples was decreased by 1. This is due to the fact that two positive examples, E1 and E2, from the original event space have the same description in the new event space.

E1: (round, round, yes, sword, red, no)E2: (round, square, yes, sword, red, yes)

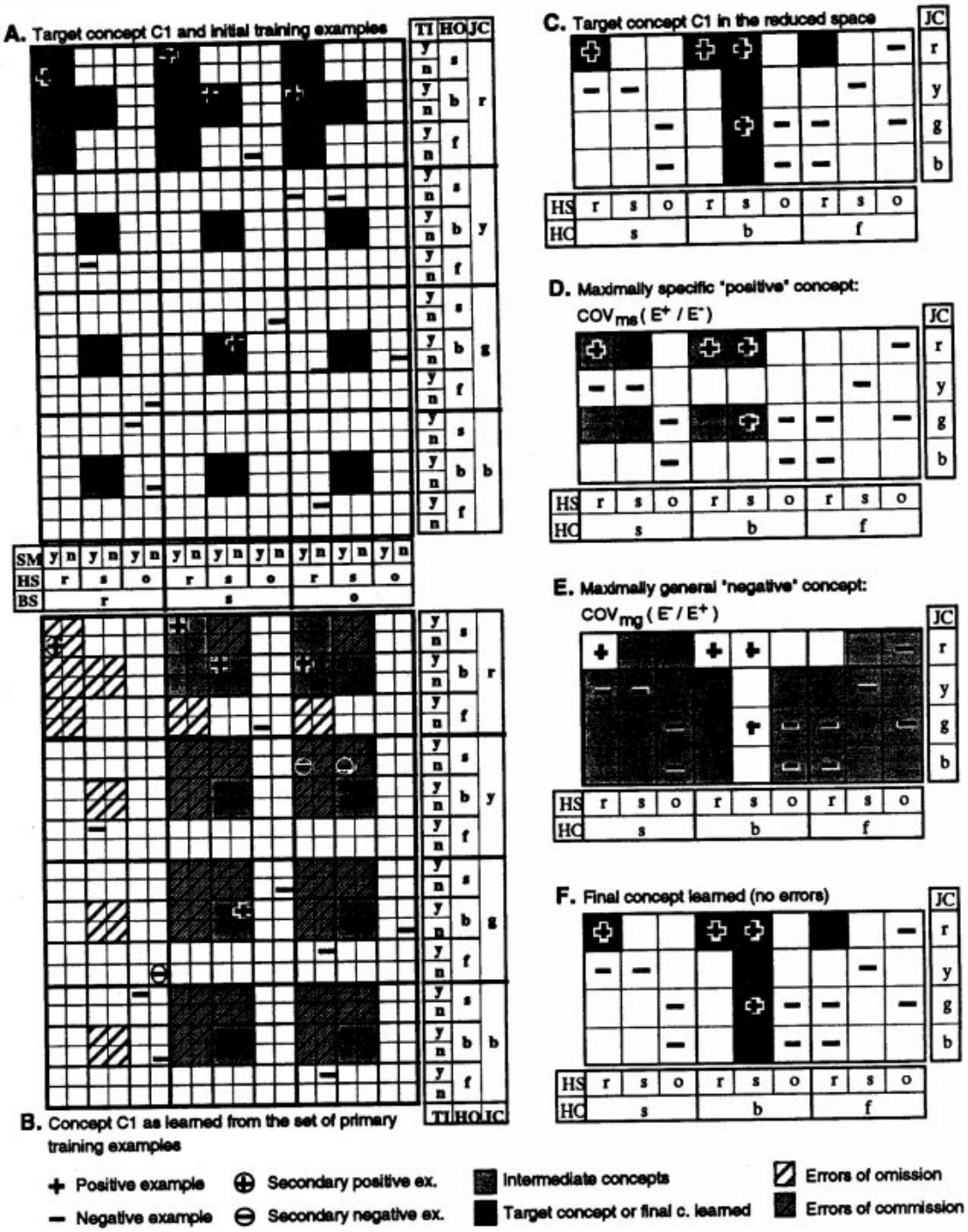

Fig. 5. Steps in learning concept C1 by AQ17-HCI.

Although such an abstracted problem is simpler for learning, the resulting hypothesis is still not accurate (diagram D). At this point, the training data set seems to be insufficient to allow proper learning. The lacking information can however, be induced while taking into consideration both positive and negative hypotheses. Fig. 5, diagrams D and E show two covers, COV_{ms} (E^+/E^-) and COV_{mg} (E^-/E^+), that were generated using all the initial training examples. AQ17-HCI generalized the positive concept description against the negative concept, and by this means improved the learned concept. The concept C1 was learned precisely (Fig. 5, diagram F).

5 Case Study II: Representation Space Contraction and Expansion

To illustrate the use of the ruleset attributes described in Section 3.9, 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.

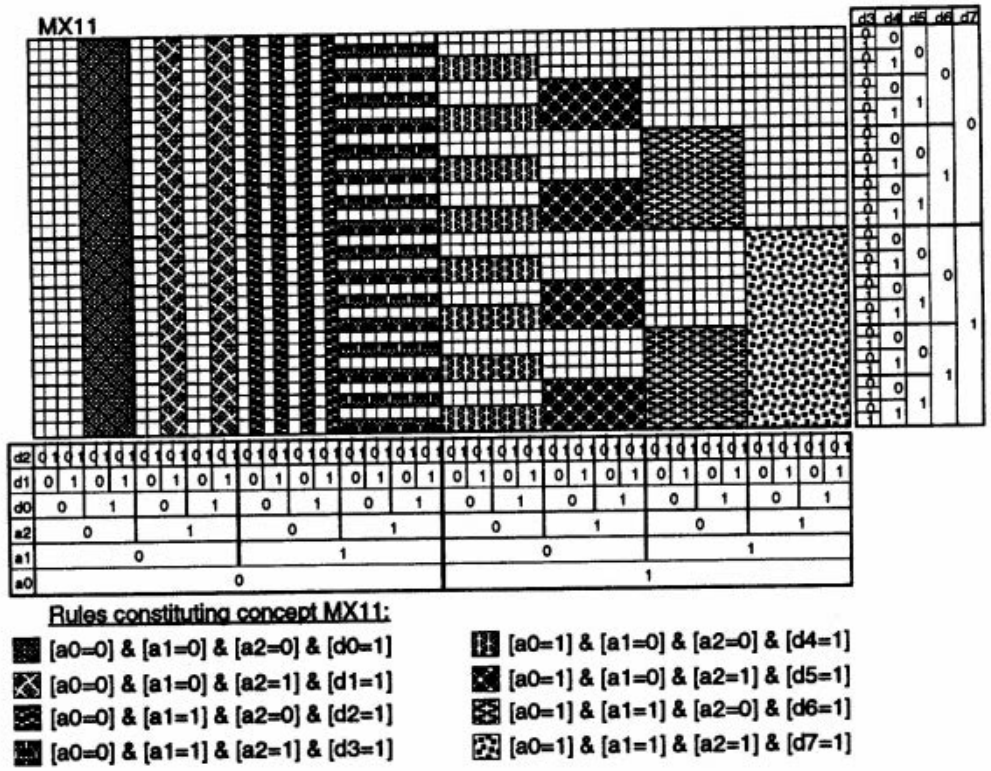

Fig. 6. Target concept MX11 in the original representation space.

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. Fig. 6 presents the MX11 concept graphically, using the diagrammatic visualization method. The MX11 concept is described by eight rules listed at the bottom of the figure. For easy recognition, each rule in the diagram is shaded differently.

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 training set had 64 (6%) of the positive examples and 64 (6%) of the negative examples. Table 3 shows a sample of the positive and the negative examples. 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.

Table 3. Part of the set of training examples.

Positive examples

a0	a1	a2	d0	d1	d2	d3	d4	d5	d6	d7
0	0	1	0	**1**	0	0	0	0	0	0
0	1	0	0	0	**1**	0	0	0	0	1
0	1	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0
1	0	1	0	0	0	0	**1**	**1**	0	1
1	1	0	0	0	0	0	**1**	**1**	**1**	1
...										

Negative examples

a0	a1	a2	d0	d1	d2	d3	d4	d5	d6	d7
0	0	1	0	**0**	1	1	1	1	1	0
0	1	0	0	0	**0**	0	0	0	0	1
0	1	0	1	1	**0**	1	1	1	1	0
1	0	1	0	0	0	0	0	**0**	0	0
1	1	0	0	0	0	0	1	0	**0**	0
...										

Table 4. Disjoint characteristic descriptions of MX11 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)	(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)	(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)

Fig. 7 presents an 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%). The classification rules are shown in Table 4. 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.

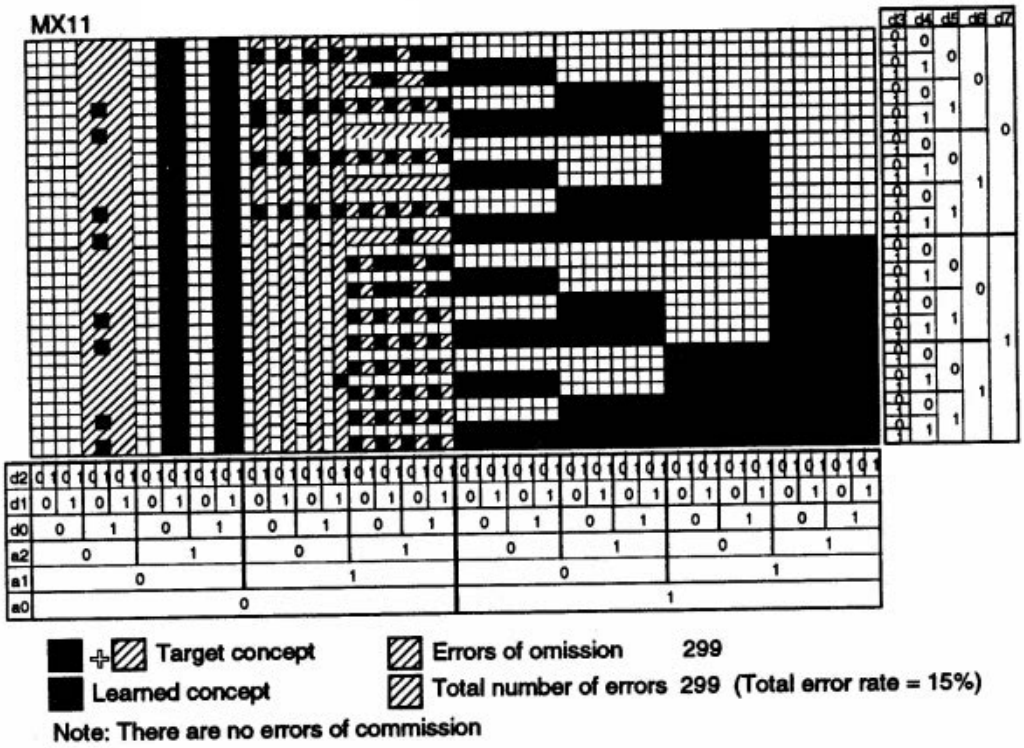

Fig. 7. AQ15-learned concept in the original representation space and its relationship to the MX11 target concept.

From the hypotheses presented in Table 4, the admissible rulesets were selected to constitute the candidate attributes P1 and N2 (Table 5). Table 6 shows the definition of the new attributes and Fig. 8 shows the coverage of the instance space done by the new attributes.

Table 5. The admissible rulesets selected according to formula (4).

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

Table 6. The definition of the constructed attributes P1 and N2.

P1 = 1 if

1. (a0=1) & (a1=1) & (a2=0) & (d6=1) or
2. (a0=0) & (a1=0) & (a2=1) & (d1=1) or
3. (a0=1) & (a1=0) & (a2=1) & (d5=1) or
4. (a0=1) & (a1=1) & (a2=1) & (d7=1) or
5. (a0=1) & (a1=0) & (a2=0) & (d4=1)

P1 = 0 otherwise

N2 = 1 if

1. (a0=1) & (a1=1) & (a2=1) & (d7=0) or
2. (a0=0) & (a2=0) & (d2=0) or
3. (a0=0) & (a1=0) & (a2=1) & (d1=0) or
4. (a1=0) & (a2=0) & (d1=1) & (d4=0)

N2 = 0 otherwise

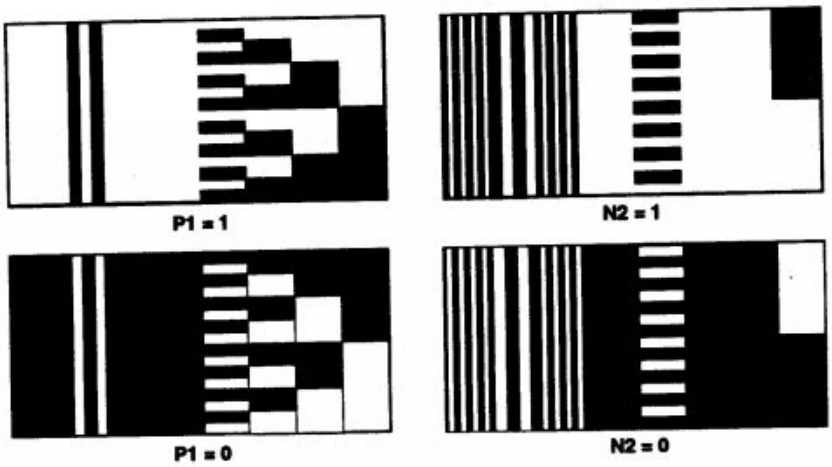

Fig. 9. Images of the constructed attributes P1 and N2.

Once the new attributes are created, they are used to reformulate the training examples (Table 7). 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 value "1" assigned. After examples are reformulated, the whole inductive process is repeated. Table 8 presents the newly inducted rules.

Table 7. 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	1	0	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	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 8. 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, replacing some of the initially given attributes. 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. Fig. 9 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 are *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 and the fact that these attributes were constructed from two disjoint rulesets. This is also in agreement with the intuition that there should not be any instances that conform to both the "Positive" and "Negative" concept descriptions represented by P1 and N2.

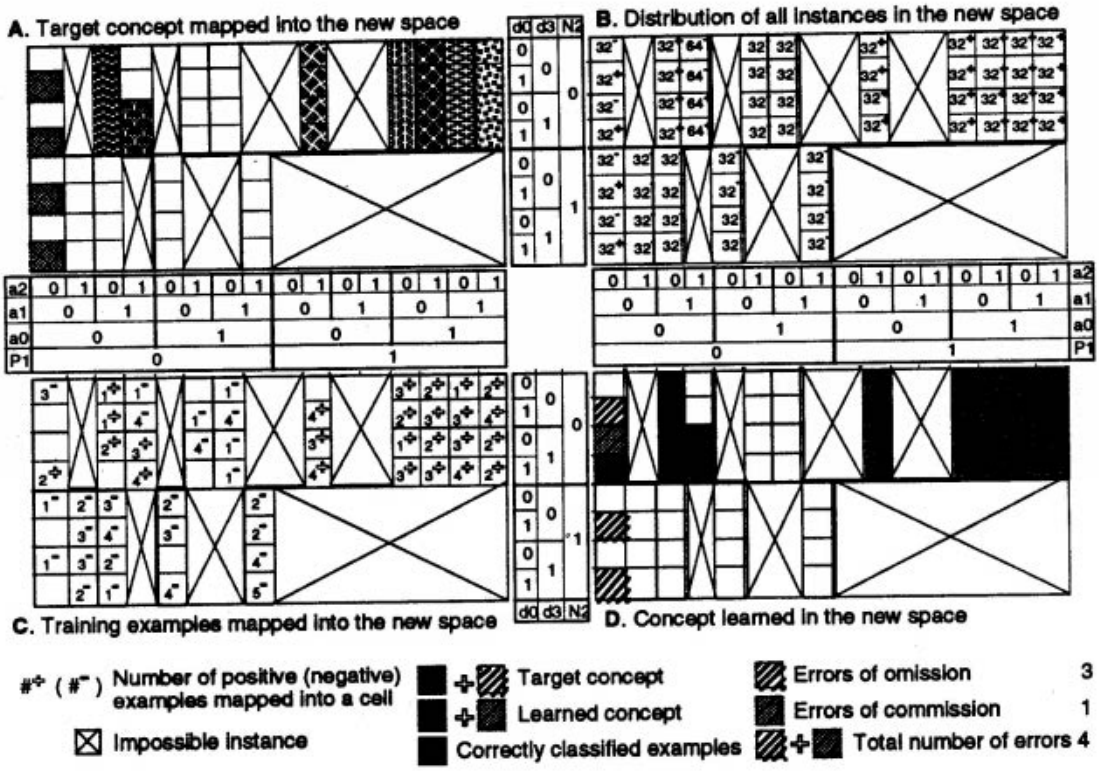

Fig. 9. Steps in learning the concept MX11 in the changed representation space.

The MX11 target concept image is shown in Fig. 9A after mapping into the new representation space. For easy identification, areas that correspond to those in Fig. 6 are marked with the same

pattern. Fig. 9B 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. Fig. 9C shows the training examples in the new space. Fig. 9D 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 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 a 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 does not produce a 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 numbers of original examples mapped into new examples.

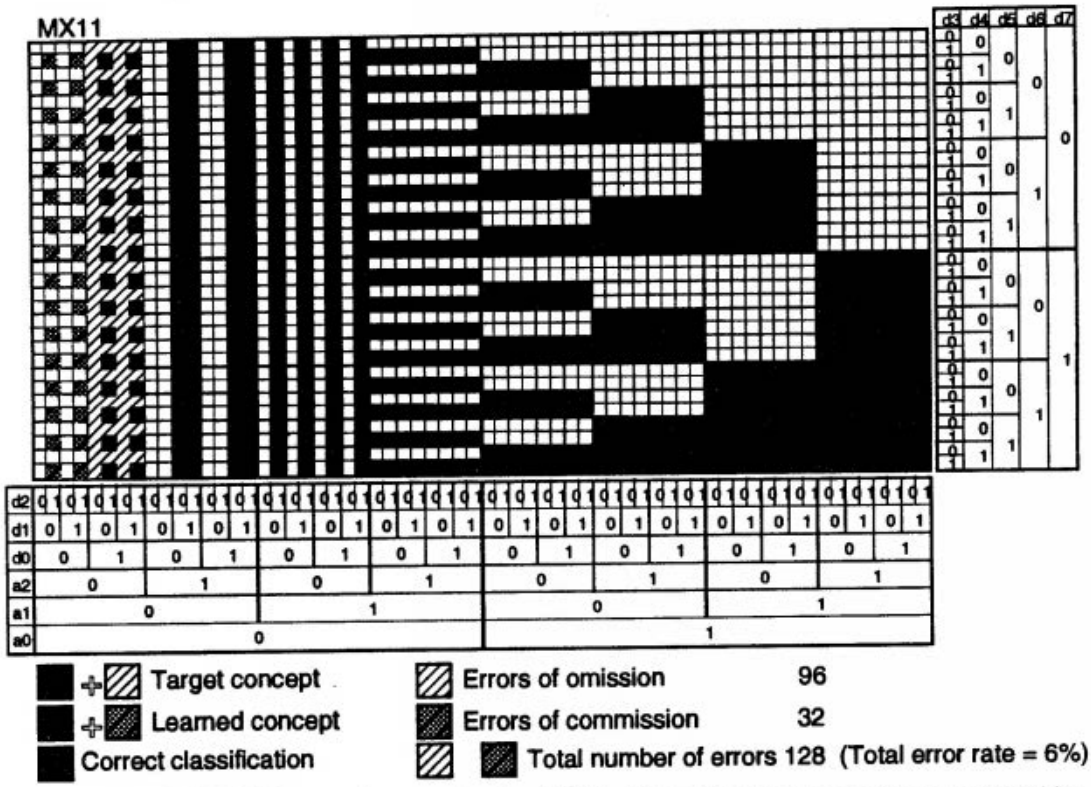

Fig. 10. AQ17-HCI-learned concept projected into the original representation space and its relationship to the MX11 target concept.

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). Fig. 10 shows the final concept image learned by AQ17-HCI in the original representation space. Compare this figure with Figs 6 and 7.

6 Experiments with the AQ-HCI method

6.1 Comparing AQ-HCI with Symbolic Methods

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 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 hypothesis-driven constructive induction, FRINGE, and second, based on the standard ID3, REDWOOD (Pagallo & Haussler, 1990).

6.1.1 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 & Haussler (1989, 1990).

Table 9 provides a characterization of the test domains. The number of training examples was calculated according to the following formula (Pagallo & Haussler, 1990):

$$\frac{K * \log_2(N)}{\varepsilon} \qquad (7)$$

where N is the number of attributes, K is the number of conditions in the smallest DNF description of the target concept, e 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 e is set to 0.10.

The testing set consists of 2000 examples (different from training examples). The training and testing sets are very small in relation to the size of the representation space. For example, in the representation space with 32 binary attributes, there are $2^{32} - 1$ i.e., about 4.3×10^9, possible descriptions of examples. Following Table 9 are the descriptions of the target concepts C1-C4.

Table 9. 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 ex.	No. of testing ex.
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 viewed as irrelevant.

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 viewed as irrelevant.

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 viewed as irrelevant.

6.1.2 Comparing AQ-HCI with the Non-Constructive Induction System AQ15

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 which, 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 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 (t) of the highest degree of consonance for that rank. When ATEST summarizes the results it reports the percentage of *first rank decisions (flexible match)* (t=0.02) as well as the percentage of *only choice decisions (100% match)* (t=0).

Concepts C1-C4 were learned and tested using data sets characterized in Table 9. The summary of the results obtained in ten executions of both systems for each concept learned, is presented in Table 10.

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 10.

Table 10. 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 11 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 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 (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 presented in Fig. 12.

Both systems, AQ15 and AQ17-HCI, generate a complete and consistent set of rules from the input examples. Since 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[4]. This can be an essential obstacle in learning *hard* concepts in the context of a preliminary description of a learned problem (Rendell & Seshu, 1990). Hard concepts are spread out all over the given hypotheses space and require multiple covers.

Table 11. The 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%

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 a hypotheses space support the *extend-against* operator in finding more accurate and effective hypotheses.

6.1.3 Empirical Comparison of AQ-HCI with Other Methods

Fig. 11 and Table 12 summarize the results obtained in ten executions of all tested algorithms. The results for the REDWOOD, FRINGE, GREEDY3, and GROVE algorithms come from (Pagallo & Haussler, 1989, 1990).

Fig. 11 shows the learning curves for the concept C2. The curves were obtained by measuring and averaging prediction accuracy over ten experiments. The measure points were at 330, 660, 1320, 1980, 2640, and 3960 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 12.

Table 12 shows the results obtained from testing concepts C1-C4 (Table 9). 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.

[4] One way to address this problem can be a lookahead technique to detect interaction between attributes, but this increases computational cost (Rendell & Seshu, 1990).

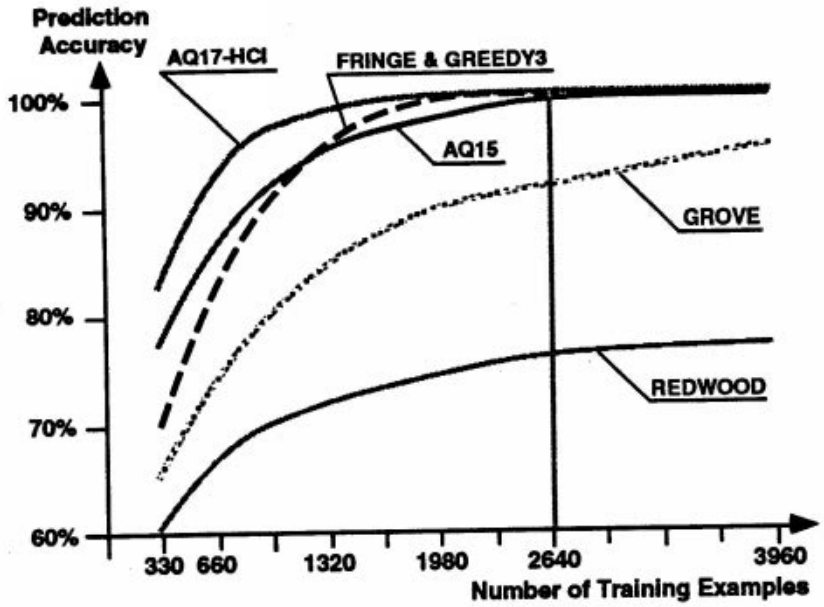

Fig. 11. Learning curves for the concept C2 for different systems.

The results shown in Table 12 suggest that all problems were difficult for the decision tree algorithm implemented in 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 these kinds of problems.

Table 12. Experimental results 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 & Haussler, 1989, 1990), (†) flexible match. Concepts C1, C2, C3, C4 were learned from 1650, 2640, 1600, and 4000 examples, respectively. All concepts were tested on 2000 testing examples.

Finally, Table 13 gives a qualitative comparison of AQ17-HCI with FRINGE. The two systems are compared using the following criteria: representational capabilities (1-2), achievement of constructive induction goals (3), and the overall capabilities (4-7).

Table 13. Qualitative comparison of AQ17-HCI with FRINGE.

	AQ17-HCI	FRINGE
1. Representational formalism		
INPUT: descriptions of examples	VL1	Vector of binary attribute values
Background knowledge	VL1	Not available
OUTPUT: concept description	VL1	Decision tree
Constructed attributes	VL1	Conjunctive expression
2. Operators used to construct new attributes	Value-patterns, e.g. (x1=1, 5..10, 15) Condition-patterns,e.g. (x1=1) & (x2=5) & (x5=4) Rule-patterns, e.g. [(x1=1) & (x2=5)] or [(x3=0) & (x5=7)] or [(x4=8)]	Binary conjunction, e.g. (x1=1) & (x2=0)
3. Constructive induction goal: *- improvement of prediction accuracy* *- reduction of description complexity*	YES The number of rules is reduced but constructed attributes can be complex	YES The tree is smaller but nodes may contain complex conjunctive expressions
4. Facilitating the learning process with information about impossible areas	YES	NO
5. Multiconcept learning	YES	NO
6. Concepts learned that the underlying selective induction learning algorithm could not learn	MONK2 problem PAR5 (parity 5) Improvement in prediction accuracy in learning DNF-type concepts	Improvement in prediction accuracy in learning DNF-type concepts
7. Tested concepts that could not be learned	None	PAR5

6.2 Testing AQ17-MCI on MONK's Problems

This section reports results from a performance comparison of AQ17-HCI with a large number of machine learning algorithms on the so-called MONK's problems. The problems were so called because they were introduced at the Priory Corsendonk in Belgium (a former monastery turned to a conference center), where an International Summer School on Machine Learning was held in 1991 (Thrun et al., 1991). During the School, there were many discussions on the merits and demerits of different machine learning approaches. To help to resolve the disputes, Tom Mitchell (CMU) and Sebastian Thrun (GDM) suggested three problems and asked various research teams in the US and Europe to apply their learning programs to them. A large number of machine learning programs were tested on these problems.

The MONK's problems represent three different types of learning problems. The first problem, M1, is to learn a concept description that can be formulated as a simple DNF expression (a Disjunctive Normal Form). This is a "DNF-type" problem. Because symbolic methods work well on such problems, the M1 problem can be viewed as favoring symbolic learning methods. Problem M2 is to learn an "m-of-n" description (if m out of n features occur in an object, then the object is an instance of the concept). This is a "non-DNF-type" problem, because describing such a concept in the form of a DNF expression would yield a very long expression. This problem favors neural-net learning, because it is easy to express the target description in an artificial neural net. Problem M3 is a DNF-type problem, like M1, but the learning data set contains noise (5%). An expectation was that a neural net might perform better on such a problem, as neural nets are claimed to be more noise-resistant than symbolic methods.

Below is a more detailed characterization of each MONK's problem. It should be noted that in the original formulation of the problems (Thrun et al., 1991), attributes represented characteristics of imaginary robot figures. We changed them here to abstract attributes, x_i, because the nature of the attributes is irrelevant to the methods described.

Problem M1. The target concept (unknown to the learning program) can be simply characterized by a VL1 (variable-valued logic one) expression (Michalski, 1975):

$$[x_2 = 1] \vee [x_4 = x_5] \Rightarrow \mathbf{M1}$$

which can be interpreted: "If for an unknown entity, the attribute x_2 takes value 1, or the attributes x_4 and x_5 take the same value (no matter which one), then—regardless of the values of other attributes—classify the entity as an instance of the concept M1."

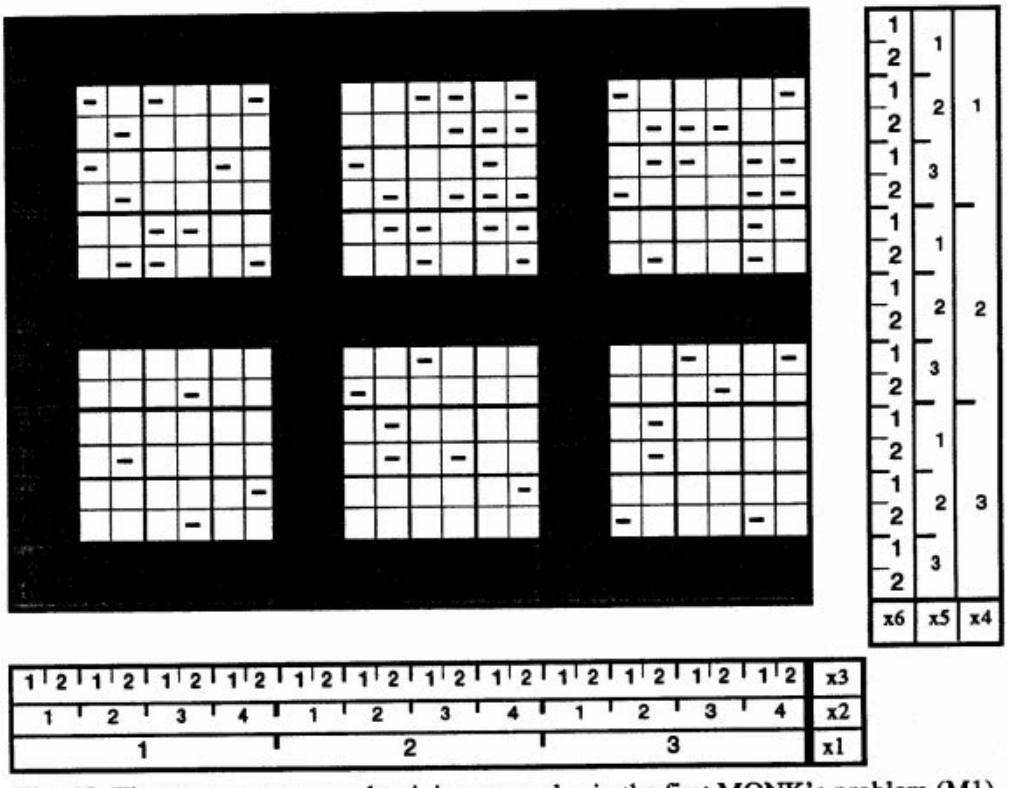

Fig. 12. The target concept and training examples in the first MONK's problem (M1).

The set of training events contained 124 training examples (62 positive and 62 negative), which constituted 30% of the total event space (432 events). Each example was described in terms of six multivalued attributes. The set of testing examples included all possible examples (216 positive and 216 negative). Thus, the accuracies of the learned descriptions (Table 14) were absolute, not estimates.

This problem represents a simple DNF concept, which is visualized in Fig. 12. The diagram in Fig. 12 consists of small cells, each representing one possible example (a vector of attribute values). To read-out the values of attributes $x_1, x_2, \ldots, x_6$, for any example, use the scales at the bottom and the right of the diagram (numbers assigned to different intervals represent possible values of the associated attributes). The dark area in the figure represents the target concept (the concept to be learned), and the white area represents the concept's negation (the set of all possible counterexamples). Cells marked by "+" and "-" correspond to positive and negative concept examples, respectively.

Problem M2. The target concept can be characterized by a sentence:

"If exactly two of six given attributes take their first value for the given entity, then classify the entity as an instance of the class M2."

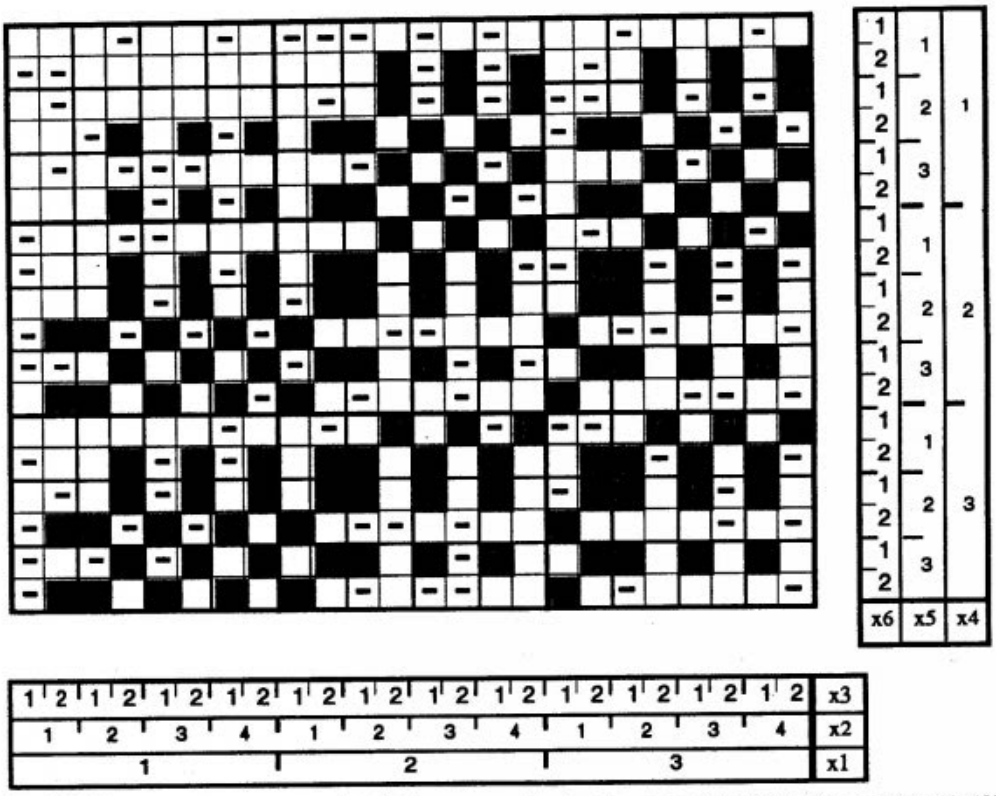

Fig. 13. The target concept and training examples in the second MONK's problem (M2).

It is not important which two of the six attributes take their first value, but it is important that two and only two attributes take such value. The "first value" means that the value sets of the attributes are totally ordered sets, and the first element in the set represents the first value. This concept does

not have a simple DNF representation, and therefore poses a serious problem for typical symbolic learning methods. In the AQ17-DCI program (listed in Table 14), which performs a data-driven constructive induction, this description can be expressed as a special case of a general "counting" condition: **[#Attrs(S, Rel, Value) = R]** which states that the number of attributes in the set S that are in relation Rel (=, ≠, <, >) to value v should be one of the values stated in the set R (in this case, S is the set of all initial attributes, Rel is "=", v is 1, and R consists of just one element, namely 2). This general condition is part of the program's concept description language, similarly as a sigmoid or threshold logic transformation is a part of the language of artificial neural nets.

The training set consisted of 169 examples (105 positive and 64 negative), which represented 40% of the total event space. The testing examples were all possible examples (190 positive and 242 negative). Fig. 13 presents a diagrammatic visualization of the target concept and the training examples for the problem M2.

Problem M3. In this problem, the goal concept can be expressed in VL1 as:

$$[x_2 \neq 4] \;\&\; [x_5 = 1 \vee 2] \;\vee\; [x_1 = 1] \;\&\; [x_2 = 3] \;\Rightarrow\; M3$$

which can be interpreted: to classify an entity to M3, one of two conjunctive conditions must be satisfied: x2 should take value different than 4 and x5 should take value 1 or 2, or x1 should take value 1 and x2 should take value 3.

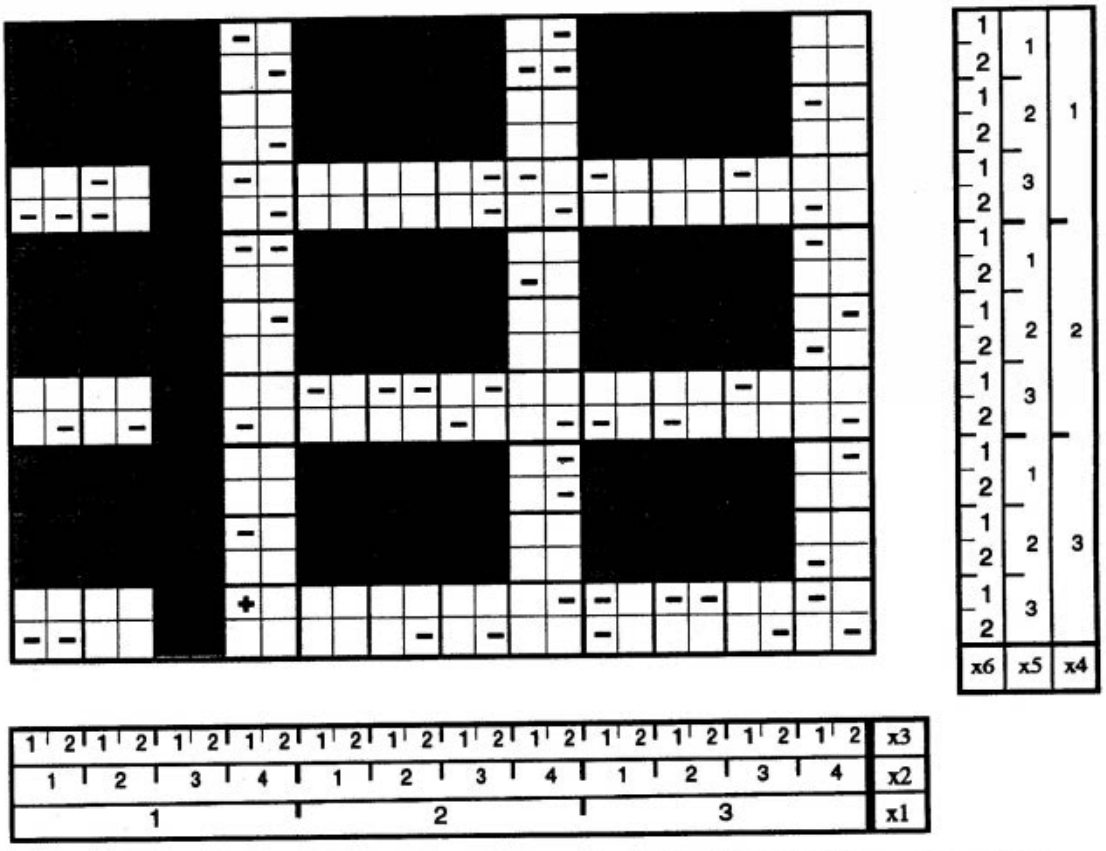

Fig. 14. The target concept and training examples in the third MONK's problem (M3).

The purpose for introducing this problem was to test learning programs' ability to learn from noisy data, that is, from examples that contain errors. There were 122 training examples (62 positive and 60 negative), which represented 30% of the total event space. The testing examples were the set of all possible examples (204 positive and 228 negative). Noise was randomly inserted into the training example set so that 5% of the training examples were misclassified.

Fig. 13 gives a diagrammatic visualization of the target concept and training examples for this problem. The five minuses in the darken area and one plus in the white area represent errors introduced into the training examples.

The tested learning systems fall into four categories:

- Neural Networks: Backpropagation (McClelland & Rumelhart, 1988)
  Cascade Correlation (Fahlman & Lebiere, 1990)

- Decision Trees ID3 (Quinlan, 1986)
  Assistant Professional (Cestnik, Kononenko & Bratko, 1987)
  ID5R (Utgoff, 1990)
  IDL (Van de Velde, 1989)
  ID5R-hat (Utgoff, 1990)
  TDIDT (Quinlan, 1986)
  PRISM (Cendrowska, 1988)

- Inductive Logic Programming (ILP):
  mFOIL (Dzeroski, 1991)

- Decision Rules AQ14-NT (Pachowicz & Bala, 1991)
  AQR, CN2 (Clark & Niblett, 1989)
  AQ15 (Michalski et al., 1986)
  AQ15-GA (Vafaie & De Jong, 1991)
  AQ17-DCI (Bloedorn & Michalski, 1991)
  AQ17-FCLS (Zhang & Michalski, 1993)
  AQ17-HCI (Wnek & Michalski, 1991)

Table 14 shows the results from applying of these systems to the MONKS' problems. The results show that the AQ17-HCI has learned all three concepts with 100% accuracy. While its good performance on problems M1 and M2 is relatively easy to explain, it is rather surprising that it performed so well on the M2 problem, which is non-DNF problem. The reason for the surprise is that the program performs only logic-type transformations on the representations space and does not have the concept of a "counting" condition in its linguistic repertoire (like AQ17-DCI). This counting condition is here a property directly relevant to this problem and it is not easy to express in logical terms. Thus, the program does not seem to be equipped for solving such problems. It may be therefore interesting to see how it expressed the M2 concept. The concept was expressed by a structure of the following rules:

M2 <:: [c4 = 0]& [c7 = 1] & [c8 = 1] & [c9 = 0] or
[c2 = 0] & [c7 = 1] & [c8 = 0] & [c9 = 1] or
[c6 = 0] & [c7 = 0] & [c8 = 1] & [c9 = 1] or
[c1 = 0] & [c3 = 0] & [c5 = 1] & [c6 = 0] or
[c1 = 1] & [c3 = 0] & [c5 = 0] & [c7 = 0] & [c9 = 0] or
[c1 = 0] & [c2 = 0] & [c3 = 1] & [c5 = 0] & [c6 = 1] & [c9 = 0]

where, c1 - c9 are constructed attributes defined as follows:

(c1 = 1) <:: [x4 = 1]
(c1 = 0) <:: [x4 = 2 v 3]
(c2 = 1) <:: [x5 = 1]
(c2 = 0) <:: [x5 = 2 v 3]
(c3 = 1) <:: [x3 = 1]
(c3 = 0) <:: [x3 = 0]
(c4 = 1) <:: [x1 = 1]
(c4 = 0) <:: [x1 = 2 v 3]
(c5 = 1) <:: [x2 = 1]
(c5 = 0) <:: [x2 ≠ 1]
(c6 = 1) <:: [x6 = 1]
(c6 = 0) <:: [x6 = 0]
(c7 = 1) <:: [c3 = 1] & [c6 = 0] or [c3 = 0] & [c6 = 1]
(c8 = 1) <:: [c2 = 1] & [c5 = 0] or [c2 = 0] & [c5 = 1]
(c9 = 1) <:: [c1 = 1] & [c4 = 0] or [c1 = 0] & [c4 = 1]

The first representation space change involved construction of the c1- c6 attributes from value-patterns. These are binary attributes and can be considered intermediate concepts. The introduction of these concepts contracted the representation space from 432 to 64 instances. The second change in the representation was done due to the construction of the c7 - c9 intermediate concepts on the basis on rule-patterns. These concepts have a very regular form. All consist of two rules and involve only two attributes, e.g. c7 is defined by c3 and c6. The attribute values in the second rule are always complementary to those in the first rule. Because of this, such a form of the rule-pattern is called *xor-rule-pattern*. The attributes constructed from xor-rule-patterns merged examples of the non-DNF concept into regions that were easy to describe using DNF-type expressions (Wnek, 1993).

The detection of the xor-patterns in rules generated in the second iteration of the learning process was the key transformation that enabled the program to learn the M2 concept correctly. These results demonstrate that the AQ-HCI method is very versatile and can produce good results even for the problems that appear alien to symbolic methods.

7 Summary

This paper addressed issues of concept learning when the original representation space is of low quality, that is problems of constructive induction. It reviewed various methods and approaches to building constructive induction learning programs capable of a self-improvement of the original space. A hypothesis-driven method for constructive induction, AQ-HCI was described in detail and compared with several other methods. It was interesting to see that a relatively simple mechanism for constructive induction employed in this method was able to significantly improve the learning capabilities of the original AQ-type method.

The AQ-HCI method used the A^q-type rule learning procedure implemented in AQ15 program. The basic idea of the HCI procedure employed in the system is to search for different types of patterns in the generated hypotheses and use them as new attributes. Three types of patterns were identified: value-patterns, condition-patterns, and rule-patterns. If no patterns are found, then the method does not change the representation space and concept descriptions are learned in the original representation space.

The system AQ17-HCI implementing the method was shown to be very effective in improving the performance accuracy in a wide range of both DNF-type, and as well as non-DNF-type concepts. The fact that it works well on some non-DNF problems shows the importance of constructive induction in an automated design of knowledge representation spaces for machine learning. One drawback to the method is that the overall complexity of the descriptions is increased if the generated attributes are complex.

Table 14. Summary of results for MONK's problems.

PARADIGM	PROGRAM	AI Lab	PREDICTION ACCURACY		
			DNF-type (no noise)	Non-DNF (m-of-n)	DNF-type (noise)
Neural Nets	**Backpropagation**	**CMU**	100%	100%	93%
	Cascade Correlation	**CMU**	100%	100%	97%
Decision Trees	**Assistant Professional**	**Slovenia**	100%	81%	100%
	ID3	**Germany**	99%	68%	94%
	ID3 (no windowing)	**Germany**	83%	69%	96%
	ID5R	**Belgium**	82%	69%	95%
	ID5R-hat	**Belgium**	90%	66%	—
	IDL	**Belgium**	97%	66%	—
	TDIDT	**Belgium**	76%	67%	—
	PRISM	**Switzerland**	86%	73%	90%
Relation Learning	**mFOIL**	**Slovenia**	100%	69%	100%
Decision Rules	**AQ14-NT**	**GMU**	100%	77%	100%
	AQR	**Germany**	96%	80%	87%
	CN2	**Germany**	100%	69%	89%
	AQ15	**GMU**	100%	77%	84%
	AQ15-GA	**GMU**	100%	87%	100%
	AQ17-DCI	**GMU**	100%	100%	97%
	AQ17-FCLS	**GMU**	100%	93%	97%
	AQ17-HCI	**GMU**	**100%**	**100%**	**100%**

Experiments were performed at the following laboratories: (CMU) School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA; (**Slovenia**) AI Laboratory, Josef Stefan Institute, Ljubljana,; (**Germany**) Institute for Real-Time Computer Control Systems and Robotics, and University of Karlsruhe, Karlsruhe; (**Belgium**) Artificial Intelligence Laboratory, Vrije Universiteit Brussel, Brussels; (**Switzerland**) AI-Lab, Institute for Informatics, University of Zurich; (GMU) Center for Artificial Intelligence, George Mason University, Fairfax, VA, USA

The presented methodology can be potentially applied not only with the AQ-type rule learning method but with other inductive learning methods that use different knowledge representations, such as VL2, annotated predicate calculus (APC; Michalski, 1983), Horn-clauses, decision trees, etc. To do so, one needs to identify types of patterns in hypotheses that are appropriate for the knowledge representation used, and develop a method for their evaluation and employment as new attributes or intermediate concepts. It is likely that employing the proposed HCI methodology within any "non-constructive" inductive learning system will improve its performance.

In multiple concept learning, it might be desirable to find *class-patterns* that characterize rulesets of different classes. Such patterns would represent conditions that are common for a subset of classes, and distinguish this subset from other classes.

The representation space transformations done by the presented hypothesis-driven constructive induction method are easy to determine and easy to make. They are, however, limited by the representation language used. Thus, they are more limited than those that in which a new repertoire of attributes that can be constructed by direct, data-driven methods. Such data-driven methods require much more search, but can potentially perform any type of transformation that can be described by a mathematical or logical expression. This suggests a new line of research aimed at a synergistic integration of the two approaches.

References

Bala, J.W., Michalski, R.S. and Wnek, J., "The Principal Axes Method for Noise Tolerant Constructive Induction.," *Proceedings of the 9th International Conference on Machine Learning*, pp. 20-29, Morgan Kaufmann, San Mateo, CA, 1992.

Bentrup, J.A., Mehler, G.J. and Riedesel, J.D., "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, 1987.

Bergadano, F., Matwin, S., Michalski R. S. and Zhang, J., "Learning Two-tiered Descriptions of Flexible Concepts: The POSEIDON System," *Machine Learning*, Vol. 8, No. 1, pp. 5-43, Kluwer Academic, Boston, MA, 1992.

Bloedorn, E. and Michalski, R.S., "Data Driven Constructive Induction in AQ17-PRE: A Method and Experiments," *Proceedings of the Third International Conference on Tools for AI.* San Jose, CA, 1991.

Bloedorn, E., Michalski, R.S. and Wnek, J., "Multistrategy Constructive Induction," *Proceedings of the 2nd International Workshop on Multistrategy Learning*, Harpers Ferry, WV, May 26-29, 1993.

Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C.J., *Classification and Regression Trees.* Wadsworth, Belmont, 1984.

Carpineto, C., "Trading off Consistency and Efficiency in Version-Space Induction," *Proceedings of the 9th International Conference on Machine Learning*, pp. 43-48, Morgan Kaufmann, San Mateo, CA, 1992.

Cendrowska, J., "PRISM: An Algorithm for Inducing Modular Rules," in *Knowledge Acquisition for Knowledge-based Systems*, B.R. Gaines and J.H. Boose (Eds.) Academic Press, New York, 1988.

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

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

De Raedt, L. and Bruynooghe, M, "Constructive Induction By Analogy: A Method to Learn How to Learn?" *Proceedings of EWSL-89*, pp. 189-200, Montpellier, France: Pitman, 1989.

Dietterich, T.G. and Michalski, R.S., "Discovering Patterns in Sequences of Events," *Proceedings of the International Machine Learning Workshop*, University of Illinois Allerton House, Urbana, pp. 41-57, June 22-24, 1983.

Dietterich, T.G. and Michalski, R.S. "Discovering Patterns in Sequence of Events," *Artificial Intelligence*, Vol. 25, No 2, pp. 187-232, 1985.

Dietterich, T.G. and Michalski, R.S., "Learning to Predict Sequences," in *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell and T.M. Mitchell (Eds.), Morgan Kaufmann, Los Altos, CA, 1986.

Dietterich, T.G., Hild, H. and Bakiri, G., "A Comparative Study of ID3 and Backpropagation for English Text-to-Speech Mapping," *Proceedings of the 7th International Conference on Machine Learning*, pp. 24-31, Morgan Kaufmann, San Mateo, CA, 1990.

Drastal, G., Czako, G. and Raatz, S., "Induction in an Abstraction Space: A Form of Constructive Induction," *Proceedings of the IJCAI-89*, pp. 708-712, Morgan Kaufmann, San Mateo, CA, 1989.

Duda, R. and Hart, P., *Pattern Classification and Scene Analysis*, New York: Wiley, 1973.

Duda, R., Gasching, J. and Hart, P., "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, 1979.

Dzeroski, S., *Handling Noise in Inductive Logic Programming*. MS Thesis, Dept. of EE and CS, University of Ljubljana, Slovenia, 1991.

Emde, W., Habel, C.U. and Rollinger, C.-R., "The Discovery of the Equator or Concept Driven Learning," *Proceedings of IJCAI-83*, pp. 455-458, Morgan Kaufmann, San Mateo, CA, 1983.

Fahlman, S.E. and Lebiere, C., "The Cascade-Correlation Learning Architecture," in *Advances in Neural Information Processing Systems*, Vol. 2, Morgan Kaufmann, San Mateo, CA, 1990.

Falkenhainer, B.C. and Michalski, R.S., "Integrating Quantitative and Qualitative Discovery in the ABACUS System," in Y. Kodratoff and R.S. Michalski (Eds.), *Machine Learning: An Artificial Intelligence Approach* (Vol. III). Morgan Kaufmann, Palo Alto, CA, 1990.

Flann, N.S. and Dietterich, T.G., "Selecting Appropriate Representations for Learning from Examples," *Proceedings of AAAI-86*, pp. 460-466, Morgan Kaufmann, San Mateo, CA, 1986.

Flann, N.S., "Improving Problem Solving Performance by Example Guided Reformulation of Knowledge," in D.P. Benjamin (Ed.), *Change of Representation and Inductive Bias*. Kluwer Academic, Boston, MA, 1990.

Greene, G.H., "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, 1988.

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

Knoblock, C.A., "A Theory of Abstraction for Hierarchical Planning," in D.P. Benjamin (Ed.), *Change of Representation and Inductive Bias*. Kluwer Academic, Boston, MA, 1990.

Knoblock, C.A., Minton, S. and Etzioni, O., "Integrating Abstraction and Explanation-Based Learning in PRODIGY," *Proceedings of AAAI-91*, pp. 541-546, AAAI Press/The MIT Press, 1991.

Kokar, M.M., "Discovering Functional Formulas Through Changing Representation Base," *Proceedings of AAAI-86*, pp. 455-459, Morgan Kaufmann, San, Mateo, CA, 1985.

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

Langley, P.W., Simon, H.A., Bradshaw, G.L. and Zytkow, J.M., *Scientific Discovery: Computational Explorations of the Creative Processes*. The MIT Press, Cambridge, MA, 1987.

Larson, J.B. and Michalski, R.S., "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*, pp. 38-44, 1977.

Lenat, D.B., "On Automated Scientific Theory Formation: A Case Study Using the AM Program," in *Machine Intelligence*, Vol. 9, J.E. Hayes, D. Michie and L.I. Mikulich (Eds.), Halsted Press, New York, 1977.

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

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

Michael, J., "Validation, Verification, and Experimentation with Abacus2," *Reports of the Machine Learning and Inference Laboratory*, MLI 91-8, Center for Artificial Intelligence, George Mason University, Fairfax, VA, 1989.

Michalski, R.S. and McCormick, B.H., "Interval Generalization of Switching Theory," *Proceedings of the Third Annual Houston Conference on Computer and System Science*, Houston, Texas, April 26-27, 1971.

Michalski, R.S., "A Variable-Valued Logic System as Applied to Picture Description and Recognition," in *Graphic Languages*, F. Nake and A. Rosenfeld (Eds.), North Holland, Amsterdam, 1972.

Michalski, R.S., "AQVAL/1 - Computer Implementation of a Variable-Valued Logic System VL1 and Examples of its Application to Pattern Recognition," *Proceedings of the First International Joint Conference on Pattern Recognition*, pp. 3-17, 1973.

Michalski, R.S., "Variable-Valued Logic and Its Applications to Pattern Recognition and Machine Learning," in *Computer Science and Multiple-Valued Logic Theory and Applications*, D.C. Rine (Ed.), pp. 506-534, North-Holland, Amsterdam, 1975.

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

Michalski, R.S. and Larson, J.B., "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* No. 867, Computer Science Dept., University of Illinois, 1978a.

Michalski, R.S., "A Planar Geometrical Model for Representing Multidimensional Discrete Spaces and Multiple-valued Logic Functions," *Technical Report*, Computer Science Dept., University of Illinois, Urbana, 1978b.

Michalski, R.S. "Pattern Recognition as Rule-Guided Inductive Inference," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-2, No. 4, pp. 349-361, July 1980.

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

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

Michalski, R.S. and Tecuci, G. (Eds.), *Machine Learning: A Multistrategy Approach*, Vol. IV, Morgan Kaufmann, San Mateo, CA, 1993.

Michalski, R.S., "Inferential Theory of Learning: Developing Foundations for Multistrategy Learning," *Machine Learning: A Multistrategy Approach*, Vol. IV, R.S. Michalski and G. Tecuci (Eds.), Morgan Kaufmann, San Mateo, CA, 1993.

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

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

Muggleton, S., "Duce, and Oracle-Based Approach to Constructive Induction," *Proceedings of IJCAI-87*, pp. 287-292, Morgan Kaufmann, San Mateo, CA, 1987.

Muggleton, S. and Buntine, W., "Machine Invention of First Order Predicates by Inverting Resolution," *Proceedings of the 5th International Conference on Machine Learning*, pp. 339-352, Morgan Kaufmann, San Mateo, CA, 1988.

Pachowicz, P.W. and Bala, J., "Improving Recognition Effectiveness of Noisy Texture Concepts," *Proceedings of the 8th International Workshop on Machine Learning*, pp. 625-629, Morgan Kaufmann, San Mateo, CA, 1991.

Pagallo, G. and Haussler, D., "Two Algorithms that Learn DNF by Discovering Relevant Features," *Proceedings of the 6th International Machine Learning Workshop*, pp. 119-123, Morgan Kaufmann, San Mateo, CA, 1989.

Pagallo, G. and Haussler, D., "Boolean Feature Discovery in Empirical Learning," *Machine Learning*, *5*, pp. 71-99, Kluwer Academic, Boston, MA, 1990.

Quinlan, J.R., "Induction of Decision Trees," *Machine Learning, 1*, pp. 81-106, Kluwer Academic, Boston, MA, 1986a.

Quinlan, J.R., "The Effect of Noise on Concept Learning," in R.S. Michalski, J.G. Carbonell and T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*. Morgan Kaufmann, Los Altos, CA, 1986b.

Quinlan, J.R., "Documentation and User's Guide to C4.5," unpublished, 1989.

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

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

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

Schlimmer, J.C., "Learning and Representation Change," *Proceedings of AAAI-87*, pp. 511-515, Morgan Kaufmann, 1987.

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

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

Tecuci, G. and Hieb, M., "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, 1992.

Thrun, S.B., Bala, J., 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. and Zhang, J., "The MONK's Problems: A Performance Comparison of Different Learning Algorithms," *Technical Report*, Carnegie Mellon University, December 1991.

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

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

Utgoff, P.E., "Incremental Learning of Decision Trees," *Machine Learning*, Vol. 4, pp.161-186, Kluwer Academic, Boston, MA, 1990.

Vafaie, H. and De Jong, K., "Improving the Performance of a Rule Induction System Using Genetic Algorithms," *Proceedings of the First International Workshop on Multistrategy Learning*,

Harpers Ferry WV, pp. 305-315, George Mason University, Center for Artificial Intelligence, 1991.

Valiant, L.G., "A Theory of the Learnable," *Communications of the ACM*, Vol. 27, pp. 1134-1142, 1984.

Van De Velde, W., "IDL, or Taming the Multiplexer," *Proceedings of the 4th European Working Session on Learning*, Pitman, London, 1989.

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

Wnek, J., Sarma, J., Wahab, A.A. and Michalski, R.S., "Comparing Learning Paradigms via Diagrammatic Visualization: A Case Study in Single Concept Learning Using Symbolic, Neural Net and Genetic Algorithm Methods," *Proceedings of the 5th International Symposium on Methodologies for Intelligent Systems*, Z.W. Ras, M. Zemankova and M.L. Emrich (Eds.), pp. 428-437, Elsevier, New York, 1990.

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

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

Wnek, J. and Michalski, R.S., "Comparing Symbolic and Subsymbolic Learning: Three Studies," in *Machine Learning: A Multistrategy Approach*, Vol. IV, R.S. Michalski and G. Tecuci (Eds.), Morgan Kaufmann, San Mateo, CA, 1993a.

Wnek, J. and Michalski, R.S., "Hypothesis-driven Constructive Induction in AQ17-HCI: A Method and Experiments," *Machine Learning*, Special Issue on Representation, Kluwer Academic, Boston, MA, 1993b.

Wnek, J. and Michalski, R.S., "A Diagrammatic Visualization of Learning Processes," *Reports of Machine Learning and Inference Laboratory*, Fairfax, VA: George Mason University, Center for Artificial Intelligence, to appear 1993.

Wnek, J. *Hypothesis-driven Constructive Induction*. Ph.D. Dissertation, School of Information Technology and Engineering, George Mason Univ., Fairfax, VA, University Microfilms Int., Ann Arbor, MI, 1993.

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

Zhang, J. and Michalski, R.S., "Combining Symbolic and Subsymbolic Representations in Learning Flexible Concepts: The FCLS System," *Machine Learning*, to appear.