Machine Learning, 6, 7-35 (1991)

Rigel: An Inductive Learning System

ROBERTO GEMELLO (UUNET!I2UNIX!CSELT!GEMELLO)
FRANCO MANA (UUNET!I2UNIX!CSELT!MANA)
CSELT, Artificial Intelligence Department, Via G. Reiss Romoli 274, 10148 Torino, Italy

LORENZA SAITTA (SAITTA@ITOINFO.BITNET)
Universita' di Torino, Dipartimento di Informatica, Corso Svizzera 185, 10149 Torino, Italy

Editor: Pat Langley

Abstract. This paper is aimed at showing the benefits obtained by explicitly introducing a priori control knowledge into the inductive process. The starting point is Michalski's Induce system, which has been modified and augmented. Although the basic philosophy has been changed as little as possible, Induce has been radically modified from the algorithmic point of view, resulting in the new learning system Rigel. The main ideas taken from Induce are the sequential learning of descriptions of each concept against all the others, the Covering algorithm, the Star definition, and the VL_2 representation language. The modifications consist of a new way of computing the Star, the use of a separate body of heuristic knowledge to strongly direct the search, the implementation of a larger subset of the VL_2 language, a reasoned way of selecting the seed, and the use of rules to evaluate the worthiness of the inductive assertions. The effectiveness of Rigel has been tested both on artificial and on real-world case studies.

Keywords. Inductive learning, specialization, generalization, integrated learning

1. Introduction

The capability of inducing plausible rules from examples is a pervasive characteristic of human thought, as well as being a fundamental tool for understanding the world. Thus, it is no surprise that considerable effort has been devoted both to analyzing the philosophical foundations of induction and to constructing systems based on it.

Early AI work on learning focused on inducing general descriptions of concepts from instances (Hayes-Roth & McDermott, 1978; Michalski, 1980; Mitchell, 1982; Vere, 1975; Winston, 1975) and much work has been done on this problem since then, as documented by Carbonell, Michalski and Mitchell (1983). Recently, some researchers (e.g., De Jong & Mooney, 1986; Mitchell, Keller & Kedar-Cabelli, 1986) have proposed deductive methods for acquiring conceptual knowledge; these are complementary rather than alternative approaches to induction, as shown by recent attempts at integration (Bergadano & Giordana, 1988; Danyluk, 1987; Lebowitz, 1986; Pazzani, 1988). Renewed attention has also been given to the computational aspects of induction and to criteria for assessing the credibility of induced concept descriptions (e.g., Haussler, 1987, 1988; Kearns, Li, Pitt & Valiant, 1987; Valiant, 1985), extending ideas from pattern recognition problems (Pearl, 1979; Vapnik & Chervonenkis, 1971).

Future knowledge-based systems will, on one hand, rely on inductive learning from examples (especially in those applications in which the domain knowledge is incomplete, inconsistent, and difficult to formalize) and, on the other, benefit from the use of general and

domain-specific knowledge, to improve the quality of inductive hypotheses and to focus search. For instance, Bergadano and Giordana (1988) have used a body of domain knowledge for these purposes. The main goal of our research is to explore the effects of introducing explicit control knowledge into the inductive process. This should increase the efficiency of the algorithms and the perspicuity of the acquired knowledge; moreover, domain-dependent knowledge, when available, could be exploited to reach better results while retaining a domain-independent learning methodology. This last aspect would greatly enhance the attractiveness of automated learning systems in real-world domains.

Most existing inductive systems only handle concepts that can be described in the propositional calculus, with each instance represented as a vector of attribute-value pairs. Researchers have proposed a number of approaches to this problem, such as constructing decision trees (Quinlan, 1986; Utgoff, 1988) and production rules (Michalski, Mozetic, Hong, & Lavrac, 1986). On the other hand, several domains of application require a more powerful description language, based on first order predicate logic. Unfortunately, handling structured concepts is a much more difficult task (Michalski & Stepp, 1983) and only a few systems have addressed it (Winston, 1975; Vere, 1975; Hayes-Roth & McDermott, 1978; Anderson & Kline, 1979; Michalski & Stepp, 1986; Kodratoff & Ganascia, 1986; Bergadano, Giordana, & Saitta, 1988; Iba, Wogulis, & Langley, 1988).

This paper describes the inductive system **Rigel**[1] (Reasoned Inductive GEneraLization), which is able to learn structured concepts using a first order language while containing the computational complexity. Its starting point is the Induce system (Michalski, 1980, 1983; Michalski & Stepp, 1986). Although radically modifying the algorithmic part of Induce, its basic philosophy has been kept as close as possible to the original, in order to easily test the effects of the introduced innovations.

Section 2 contains an overview of Rigel and an outline of similarities to and differences from Induce. Section 3 illustrates the behavior of Rigel, whereas Section 4 describes the system's approach to generalization and specialization. Section 5 introduces the heuristic knowledge and Section 6 reports the experimental results. Finally, Section 7 presents a brief comparison with related work and Section 8 discusses the benefits and the limitations of the approach and considers directions for future research.

We assume the reader is acquainted with Induce and its terminology. However, for the sake of self-consistency, we give a brief description of AQ and Induce in the Appendix.

2. An overview of Induce

This section briefly reviews the ideas from Induce that are relevant for a comparison with Rigel. The main points taken from Induce are the sequential learning of each concept against all the others, the Covering algorithm, the Star definition, and the VL_2 representation language. All of these aspects are summarized in the following subsections.

2.1. Representation language and performance system

Induce implements a subset of the VL_2 language, allowing conjunction, disjunction, internal disjunction, and quantification over single selector formulas. Each event is represented as

a conjunction of *selectors*, i.e., relational statements that typically contain a predicate descriptor, variables or constants as arguments, and a list of values (see the appendix for more details).

The acquired knowledge base consists of a set of unordered rules, one for each class, whose left-hand side is a disjunctive VL_2 formula and whose right-hand side is the name of a class. A new event E is considered a member of a class C if E's description matches the left-hand side of the rule associated with C. Thus, four possibilities can occur when Rigel applies its acquired knowledge to a new event E. If the event matches only the rule corresponding to its correct class, it is considered correctly classified. If E matches only rules corresponding to other classes than the correct one, it is considered misclassified. If E matches the rule corresponding to its correct class but also matches other rules, it is considered ambiguously classified. Finally, if E does not match any rule, it is considered unclassified.

2.2. The top-level Induce algorithm

Induce is a general-purpose inductive program that transforms symbolic descriptions of events (examples) into general descriptions of the concepts of which the events are instances. For each concept, the algorithm starts by randomly choosing a *seed* from among the instances of the concept. Using this it builds a *star*, which is a set of consistent and non-redundant conjunctive formulas that cover the seed. A formula *covers* an event if and only if the description of the event satisfies the formula.

The procedure for generating the Star starts by considering a partial star that contains a set of formulas, each consisting of just one selector that occurs in the seed. This partial star is trimmed according to some user-defined preference criteria. The remaining formulas are specialized by "and-ing" them with every selector derived from the seed that is directly connected to the formula being considered (i.e., sharing one or more variables with it). A new partial star is thus obtained. If no new formula can be formed in this way, then disconnected selectors are taken into account, allowing formulas which predicate on different variables to be built up. Consistent formulas are put in the *solution* set. This procedure is repeated until a set of solutions of the desired size is obtained.

The conjunctive formulas in the solution set are consistent, but generalizing them may increase their degree of completeness (coverage of positive examples). This is done by using the AQ algorithm, which tries to extend the reference of each formula's selectors without losing consistency.

The previous steps (star computation and generalization with AQ) are iterated, by choosing another seed, until all the positive examples are covered. At each cycle the best conjunctive formula of the star is retained and the positive examples that have already been covered are deleted. The final description of the concept is a disjunction of the conjunctive formulas found in each cycle.

2.3. Star computation

The central idea of AQ and Induce is star computation, which is aimed at finding a consistent formula, covering a set of positive examples, starting from the seed. Induce operates in three steps:

a) *generalization* from the seed to a single selector, by applying the *dropping condition rule* on the seed description;
b) *specialization*, by adding new conjuncts to the current inductive hypotheses;
c) *generalization* using the AQ algorithm.

In the first step, the selected seed is used to obtain from its description the set of all the single selector assertions occurring in it. After that, selective specialization is applied in order to iteratively reaggregate the selectors obtained, with the aim of reaching consistent and possibly complete formulas.

However, the formulas obtained in this way might be too specialized; in fact, no selectors with generalized values (disjunctive values, generalized values of structured domains, intervals of linear domains) are introduced in this phase. Hence, a further generalization step is carried out by the AQ algorithm (Michalski et al., 1986). But AQ can only deal with the VL_1 language (an attribute-value formalism without variables), so the given formulas must be properly converted. A conjunctive VL_2 formula, in the original structured-description space, is transformed into a VL_1 formula, in an attribute-description space, by means of a graph isomorphism process (Michalski, 1980; Michalski & Stepp, 1986). AQ is then applied to the transformed formulas, generating VL_1 generalized formulas, and these generalizations are then reconverted to VL_2 formulas.

This fixed sequencing of generalization and specialization may lead to problems. For instance, some conjunctive descriptions that cover most of the examples may never be found. Such descriptions are important because they supply compact and robust descriptions, avoiding many small disjuncts. Examples of this problem are mentioned in Section 6.

During search, the generated inductive assertions are evaluated by means of the lexicographic evaluation function (LEF), which lets the system's behavior be tested with respect to alternative criteria.

2.4. Seed selection

Induce selects the seed at random, and this choice is justified by the fact that it should be possible to obtain the "correct" description from whatever example is covered. However, this fact can be guaranteed only for the complete Star and not for the reduced Star. In this way, not only can different seeds lead to different amounts of computation, but they can also lead to the early pruning of good hypotheses, which would emerge as such only in later stages of the search.

3. An overview of Rigel

This section presents the general architecture of Rigel, a knowledge-based learning system that consists of an algorithmic component and a knowledge base. The algorithmic part includes the covering algorithm, the seed selection, the inductive search management, and the generalization and specialization policy, whereas the knowledge base includes domain-independent control knowledge, domain-specific background knowledge, and generalization rules.

The control knowledge is a declarative, easily modifiable knowledge base that is aimed at focusing the inductive search of the system; it is domain independent but task dependent: the present control knowledge is oriented to the concept discrimination task. The background knowledge consists of a description of the application domain, including a set of predicates (selectors) used to describe the examples. For each selector, the domain (the set of values it can assume) and the type (nominal, linear, structured) are defined. The generalization rules are represented as a set of operators, whose syntax and semantics will be described in Section 4. The algorithmic part uses this background knowledge and has two goals: to limit the amount of search and to focus attention toward better concept descriptions.

Before explaining Rigel's new ideas in detail, we briefly summarize them here:

- A new way of computing the star searches the space of inductive assertions, starting from the seed, by using interleaved specialization and generalization steps.
- A body of task-dependent heuristic knowledge strongly directs the search.
- A larger subset of the VL_2 language is implemented, including full universal and numerical quantification on complex formulas.
- The seed is selected in a reasoned way, aimed at reducing the probability of starting the search from a scarcely relevant example.
- Modifiable rules are used to evaluate the worthiness of the inductive assertions, thus enabling the user to test alternative criteria easily.

Below we cover each of the issues in greater depth.

3.1. Knowledge representation and performance system

Rigel uses a larger subset of the VL_2 language than Induce. More precisely, a well formed formula (wff) can be defined, in Rigel, as follows:

- A single selector is a wff.
- If V, V_1, V_2 are wffs, then $U = V_1 \wedge V_2$, $W = V_1 \vee V_2$ and (V) are wffs.
- If V is a wff and $\mathcal{J}$ is a set of integers, then:

  $U = \exists. (x_1, x_2, \ldots, x_n)$ (V), $W = \exists. \mathcal{J} (x_1, x_2, \ldots, x_n)$ (V) and
  $Z = \forall. (x_1, x_2, \ldots, x_n)$ (V)

  are wffs, where "$\exists$." and "$\forall$." denote "there exist and are different" and "for all different", respectively.

For example, the formula $W = \exists. (3..5) (x)$ [Shape(x) = Triangle], which states that there exist between 3 and 5 triangles, is a well formed formula of the VL_2 language.

Rigel uses the same performance algorithm as Induce, as described in Section 2.1. That is, it uses its unordered decision rules to match new instances and thus predict their class.

3.2. The top-level Rigel algorithm

As in Induce, given a finite set of concepts, Rigel learns each concept sequentially by computing the cover of that concept's positive examples against the examples of all the other concepts, which are considered to be counterexamples. The covering of a single concept (class) is performed using Michalski's (1980, 1983, 1986) covering algorithm. The innovative points are the seed selection and the star computation.

More precisely, let $\mathcal{K} = \{K_j \mid 1 \leq j \leq J\}$ be a given set of concepts. The Rigel algorithm considers them one at a time and gives as output J sets $COVER_j$ ($1 \leq j \leq J$), each containing a disjunctive description of the associated concept K_j. For each $K_j \in \mathcal{K}$, the sets $POS(K_j)$ and $NEG(K_j)$ contain the positive and negative examples of K_j, respectively. The basic procedure, iterated during the covering cycle, consists of the following steps:

- Initialize $COVER_j$ to ϕ.
- Select a seed from the examples of K_j that are not yet covered.
- Generate the Star of the seed against the counterexamples of K_j.
- Add the best formula found in the Star to $COVER_j$.
- Delete the newly covered examples of K_j from $POS(K_j)$.

The cycle continues until all the examples of K_j have been covered, giving a disjunctive description for this class.

3.3. Seed selection

Rather than selecting an example to compute the star (seed) at random, as in Induce, Rigel selects the best representative of its class. Given a concept $K_j \in \mathcal{K}$, let us consider the set $POS(K_j)$ and the set $HPS(K_j)$, which contain all the formulas constituted by a single selector that were derived considering all the positive examples.

The idea behind the seed selection procedure is to select the seed from among those examples that generate the greatest number of best hypotheses in $HPS(K_j)$. Each hypothesis $h \in HPS(K_j)$ is weighted with a user-defined heuristic function $\nu_{HP}(h)$, which prefers complete and consistent formulas. The set $HPS(K_j)$ is ordered according to decreasing values of $\nu_{HP}(h)$ and trimmed with a threshold τ_0, obtaining a set:

$$BEST(K_j) = \{h \in HPS(K_j) \mid \nu_{HP}(h) \geq \tau_0\}.$$

The number m(e) of hypotheses $h \in BEST(K_j)$ is associated with each example $e \in POS(K_j)$. Let M be the maximum m(e) value. If there is only one example e* corresponding to M, then e* is selected as the seed. Otherwise, let $CS = \{e' \in POS(K_j) \mid m(e') = M\}$ be the set of candidate seeds from among the events in $POS(K_j)$. In this case one would like to select the seed e* in such a way that it could generate the best hypotheses in $BEST(K_j)$. With this aim in mind, the set $HPS(K_j)$ is partitioned into n subsets, which are ordered according to decreasing values of $\nu_{HP}(h)$:

$$HPS(K_j) = HPS_1(K_j) \cup \ldots \cup HPS_n(K_j)$$

where

$$HPS_p(K_j) = \{h \in HPS(K_j) \mid \nu_{HP}(h) = a_p\}$$

and where

$$a_p > a_{p+1} \geq \tau_0, \text{ with } 1 \leq p \leq n - 1.$$

Considering a subset $HPS_p(K_j)$, one can reduce the seed candidates set CS by keeping only those examples that generate the greatest number of hypotheses in $HPS_p(K_j)$. The selection algorithm reduces CS, starting from $HPS_1(K_j)$ and considering the other sets in turn until CS contains only one example, which is selected as seed. If the set of seed candidates still contains more than one example at the end of the reduction process, a random choice is made from among those that have the shortest description.

3.4. Star computation

The most notable differences between Rigel and Induce concern the star computation algorithm. As suggested by Mitchell (1982), one can view inductive learning as a search in the space of inductive assertions. Rigel computes a bounded star by searching the space of the inductive assertions that can be derived from the chosen seed through generalization and specialization. Michalski (1980) defines the star(e | E) of an event e against a set of events E as the "Set of all the alternative non-redundant descriptions covering the event e and not covering any event in E."

The main difference between Rigel, on one side, and Induce and AQ, on the other, relate to the search control strategy and in the learning operators. In particular, Induce uses the *dropping* and *reverse dropping condition rule* and AQ uses the *extension against rule*, whereas Rigel currently uses a larger set of operators, which are discussed in the next section. Moreover, Rigel integrates specialization and generalization at the VL_2 level, performing both activities during a unique inductive search. The specialization and generalization operators are allowed to mix completely freely, in principle, but the heuristic control on the operator activation focuses the operator sequences towards the most promising sequences, dynamically adapting the alternation of specialization/generalization steps. Rigel's use of explicit search, instead of embedding some operators into an algorithm, gives it greater flexibility than Induce. The available operators may be activated and deactivated, and new operators that implement more complex generalization rules can easily be added.

More formally, Rigel's inductive search can be defined as the 5-tuple SEARCH = (S, I, F, Ω, C), where S is a state description, I is the initial state, F is a set of final states, Ω is the operator set, and C is the control strategy. More precisely, the *state description* is a VL_2 formula, with additional information about the positive and negative examples that the formula covers, and about its syntactic form (e.g., number of selectors and presence of quantifiers). Control information lets the system avoid state duplication during expansion.

The *initial state* contains only one element, i.e., the description of the chosen seed. The set of *final states* contains consistent formulas, which are as complete as possible. The user can control the maximum cardinality of this set, as well as the degree of consistency required. The set of the *operators* currently contains six operators, which are described in Section 4.

The *control strategy* lets the system generate new hypotheses by applying operators to the current ones; the coverage of these new hypotheses is tested on all the examples. Starting from a root state (initially the seed), it generates a search tree by creating new states through the application of the operators to the frontier nodes of the tree. Operator application is guided by heuristic conditions. Given a state to expand, the strategy considers all the potentially applicable operators and evaluates their applicability conditions. All operators whose evaluation is considered promising are applied to the state. A beam search limits the explosion of the tree, using an evaluation function $\nu_V(\varphi)$ to select the most promising new states.

4. Generalization and specialization operators

In this section we describe the operators that Rigel uses during its inductive search. An *operator* ω consists of a pair (C, A), where C is a condition of applicability and A is the action. The condition of applicability of an operator ω is a function from the set of states to the unit interval [0, 1]; it supplies an evaluation of the suitability of applying ω to a given state. If this evaluation exceeds a threshold, the operator can be applied. The heuristic conditions are defined in a declarative way, using the heuristic description language introduced in Section 5.

The action implements a generalization or specialization method in a procedural way; given a state and other information specific to the operator, it generates a set of new states. The generation of new states requires the evaluation of the extension of the formulas associated with the states, which can be costly when working with many examples. To overcome this, Rigel uses a look-ahead mechanism, aimed at supplying a low-cost "guess" of the evaluation of the extension. If the extension guessed by the look-ahead is bad, the real extension is not even tested and the operator does not create the state. Bergadano et al. (1988) have employed a similar technique in their system.

4.1. Specialization operators

The current version of Rigel incorporates three specialization operators. The operator ω_A corresponds to the *reverse dropping condition rule*, which Michalski (1983) defines as

$$(\varphi ::> K;\ S ::> K) \mid> \varphi \wedge S ::> K$$

where S is a selector referring to a variable already present in φ (i.e., S is connected to φ). This operator addes a detail to the description of an object. Since an object that occurs in an example corresponds to a variable in a concept description (VL_2 formula), adding

a detail to an object description involves adding a selector referring to that variable. Rigel chooses the selector S from among the set of selectors originally obtained from the seed.

The second operator, ω_J, corresponds to the *reverse dropping condition rule*, but in a different way:

$$(\varphi ::> K;\ \psi ::> K) \mid> \varphi \wedge \psi ::> K$$

where φ and ψ are formulas with no common variables. This operator generates the conjunction of two formulas whose sets of variables are disjointed. Whereas ω_A focuses on the description of an object and tries to refine it, ω_J enlarges the set of considered objects described by a formula, conjoining it with another good formula which refers to possibly different objects. The good formulas, which are candidates to be conjoined, are kept in a set that contains the best formulas generated so far.

The third operator, ω_Q, generates quantified formulas. Given an unquantified formula φ, ω_Q tries to generate from φ some new universally or numerically quantified formulas. Four quantifications are possible:

(a) $\exists.\ (=\ n)(x)\ (\varphi) ::> K$
(b) $\exists.\ (\geq\ n)(x)\ (\varphi) ::> K$
(c) $\exists.\ (\leq\ n)(x)\ (\varphi) ::> K$
(d) $\exists.\ (IN\ (n..m))(x)\ (\varphi) ::> K$

In other words, if there exist (a) exactly n, (b) more or equal than n, (c) less or equal than n, (d) between n and m different objects in a sample satisfying formula φ, then the sample belongs to class K. In addition, a final quantification is:

$$\forall(x)\ (\varphi) ::> K$$

which can be paraphrased: if all objects in a sample satisfy formula φ, then the sample belongs to class K.

Given a generic formula φ, there are usually several quantified formulas that belong to the Star and cover the seed. The operator ω_Q takes into account only the most characterizing ones. In order to do this, it carries out a data-driven activity that strongly focuses the search for possible quantifications. In particular, the positive examples of *K* are allowed to propose the best quantifications.

Given a formula φ that covers an example *e*, there are usually several alternative bindings between the variables in φ and the objects in *e*. A histogram is built up that shows the distribution of the number of bindings in the examples of *K* with respect to φ; Figure 1 presents an example of such a histogram. The x axis represents the number *k* of alternative bindings and the y axis indicates the number of examples that are covered by φ in *k* ways. First the operator analyzes the histogram, detecting a continuous interval [L, R] of nonzero values of k, comprising the seed; then the best form of existential quantification (=n, $\geq$n or $\leq$n) is chosen, depending on the value of L and R. Moreover, if the number of ways in which φ covers the seed equals the total number of possible ways the *m* variables in φ can be bound to the *n* objects in the seed, then the operator proposes the universal quantification.

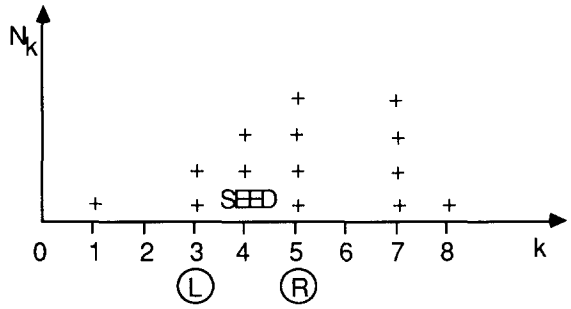


Figure 1. Histogram of the number (N_k) of examples of K_j covered by φ in k different ways.

We should note that every quantified formula is disconnected from all the others. In other words, a quantified formula is closed in that its variables only have meaning inside the formula. Therefore, they can be seen as formulas with no variables, making them available to the ω_J operator during the inductive search.

4.2. Generalization operators

Rigel also incorporates three generalization operators. The first operator, ω_{CV}, is applied only once, at the beginning of the inductive search, to transform the seed description. This operator corresponds to Michalski's (1983) *turning constants to variables rule*, which is defined as follows:

$$\varphi(a_1\ a_2\ \ldots\ a_n) ::> K \mid< \exists.\ (x_1\ x_2\ \ldots\ x_n)\ \varphi(x_1\ x_2\ \ldots\ x_n) ::> K,$$

where $\varphi(a_1\ a_2\ \ldots\ a_n)$ is a VL_2 description containing the constants $a_1, a_2, \ldots, a_n$, which are turned into the corresponding (different) variables $x_1, x_2, \ldots, x_n$.

The second generalization operator, ω_D, is also applied only once at the beginning of the search in order to transform the seed into a set of single selector formulas, which are the starting point for the inductive search. This operator corresponds to a *dropping condition rule*, defined as follows:

$$\varphi \wedge S ::> K \mid< S ::> K,$$

where S is a single selector and φ is the remainder of the formula, which corresponds to Michalski's "context."

The operator ω_G is Rigel's main generalization process. It corresponds to the *consistent extending reference rule*, defined as follows:

$$\left.\begin{array}{l} \varphi \wedge [L(x) = R_1] ::> K \\ \varphi \wedge [L(x) = R_2] ::> K \\ \neg\ \{\varphi \wedge [L(x) = R_3] ::> \neg(K)\} \\ \text{where } (R_1 \neq R_2 \text{ and } R_1, R_2 \subseteq R_3). \end{array}\right| \quad < \varphi \wedge [L(x) = R_3]$$

This operator is a modification of Michalski's (1983) *extending reference rule*. Given an inductive assertion, the operator ω_G generalizes it by extending a selectors' set of values. The generalization takes place under the consistency constraint: each new value is added only if it enhances the number of covered positive examples without covering negative ones. This constraint prevents the system from overgeneralizing the inductive assertion, and so avoids oscillation in the partially ordered set of inductive assertions and focuses the search on discriminant formulas.

The problem of generalizing a formula can be recursively reduced to that of generalizing a single selector. But, given a formula, there are usually very many ways to extend the reference of one of its selectors and it is impossible to hypothesize all of them and subsequently select those which do not violate the consistency condition.

To overcome this difficulty, the operator ω_G performs an analysis of the examples of a concept K to guide its action. Given the formula $\psi = \varphi \wedge [L(x) = R_1]$, the positive and negative coverage of the expression $\varphi \wedge [L(x) = *]$ is computed: the symbol * stands for the domain of attribute L. In this way, all the candidate values for extending the reference of L are represented by means of two histograms, N^+ and N^-, as shown in Figure 2.

Each histogram represents the distribution of positive (N^+) and negative (N^-) examples covered by $\varphi \wedge [L(x) = p]$, $p \in DOM(L)$. From these histograms, one can extract the candidate values that satisfy the consistency condition; a generic value $p \in DOM(L)$ is selected if and only if the N^- histogram has zero value in correspondence at the value ($N^-(p) = 0$). The operator determines the set of values to be added by looking at the histograms for all the values that cover new positive examples and no negative ones. Actually, the consistency constraint can be weakened to accept generalized formulas that also cover some negative examples. This is particularly useful when the system deals with real-world data.

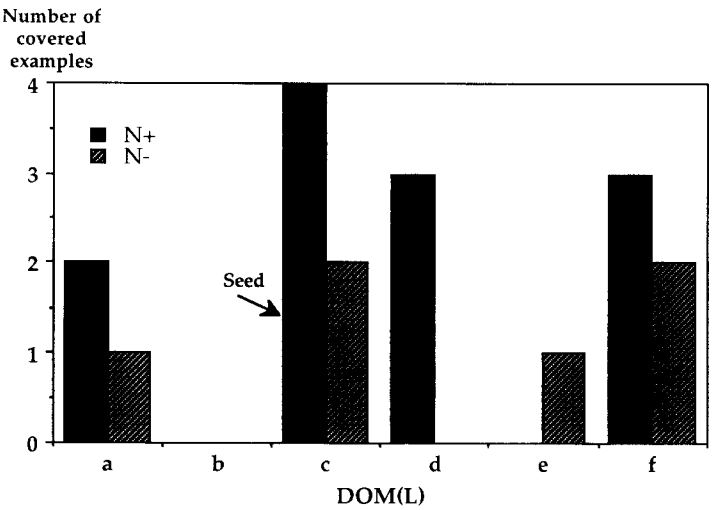


Figure 2. Histogram used to guide the application of the consistent extending reference rule. The x-axis shows the values belonging to the domain DOM(L) of the attribute L and the y-axis reports the numbers N^+ and N^- of positive and negative examples covered by $\varphi \wedge [L(x) = p]$ for each value $p \in DOM(L)$.

4.3. Macro operators

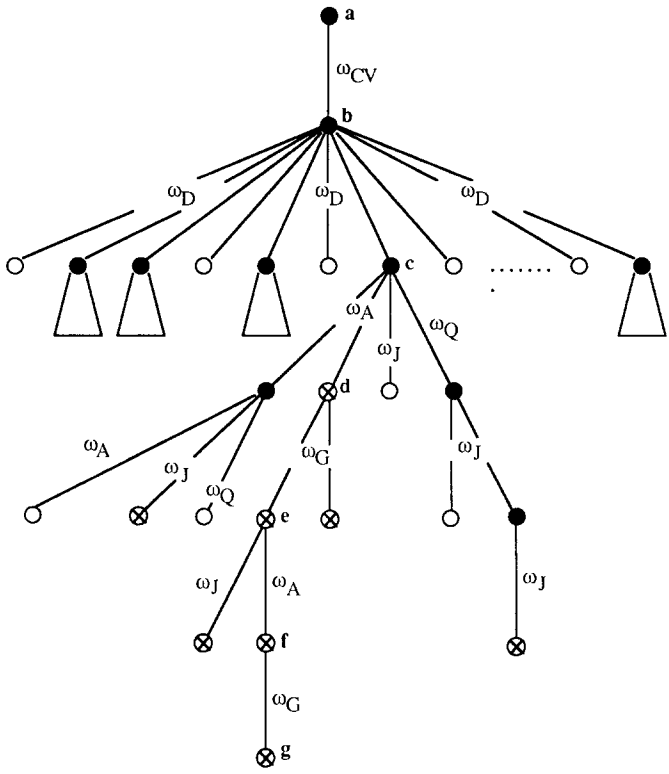
The generalization and specialization operators described above are Rigel's basic tools for exploring the space of inductive assertions. Nevertheless, initial experiments with the system suggest that, under certain conditions, a sequence of basic operators is more effective than a single step. A typical case occurs when the target concept consists of a quantified formula with generalized internal selectors. Obtaining such a formula would require making a generalization step (relaxing the consistency constraints) and then quantifying the generalized formula. However, the intermediate generalized formula may be poor in terms of coverage (e.g., it may cover many negative examples), causing Rigel to discard it before the second step can be applied, even if this step would generate a very good formula.

One solution to this problem involves the use of macro-operators, which implement a sequence of standard operators in a single step. Currently, Rigel contains only one implemented macro-operator, Ω_{GQ}. This is defined as a sequence of the ω_G and ω_Q operators, allowing the generation of complex quantified formulas (i.e., that contain several selectors with generalized values). When Ω_{GQ} is applied, ω_G generates maximally complete formulas without worrying about negative examples; then ω_Q quantifies them, possibly reducing the number of negative examples. Only at this point are the formulas evaluated, by taking into account such factors as completeness and consistency.

4.4. An example of inductive search

To clarify Rigel's use of operators, Figure 3 presents the search tree that the system generates for a simple induction task. The first operator to be applied is ω_{CV}, which transforms the seed description (**a**) into a more general one (**b**) by substituting existentially quantified variables for the constants. The operator ω_D next splits the seed description into a set of one-selector formulas. In this example the solution path starts from the selector [color (X) = red] (node **c**). At this point the applicable operators are: ω_J, which joins the formula in **c** with a formula that predicates on different variables and in this example creates a node that is discarded by the beam (inactive node = white circle); ω_Q, which creates a quantified formula kept in the beam (active node = black circle); and ω_A, which specializes the formula in **c** by adding a new selector, thus creating two nodes, one of which (**d**) is already consistent (final node ≡ consistent formula = crossed circle).

Let us suppose that the formula in **d** is consistent but not yet complete: in this case it would be put in the solution set of the Star (as it is consistent), but also kept in the search, because it could be further developed (specialized and generalized) to achieve completeness while preserving consistency. In the next step, **d** is generalized with ω_G, increasing its positive extension without increasing the negative extension (**e**). In **e** the "weight" descriptor is generalized to the value 2..4 (interval from 2 to 4) without covering negative examples. Next, **e** is specialized with the operators ω_J and ω_A (node **f**), which create a more specialized context in which the "weight" descriptor is generalized from the value 2..4 to the value 2..5 without any loss of consistency. The resulting formula **g**, which is complete and consistent, is selected as the best element of the Star.



● = ACTIVE NODES
○ = INACTIVE NODES
⊗ = FINAL NODES

a = [color(p) = red] [shape(p) = square] [weight(p) = 4] ...
b = ∃(x) [color(x) = red] [shape(x) = square] [weight(x) = 4] ...
c = ∃(x) [weight(x) = 4]
d = ∃(x) [weight(x) = 4] [shape(x) = square]
e = ∃(x) [weight(x) = 2..4] [shape(x) = polygon]
f = ∃(x) [weight(x) = 2..4] [shape(x) = polygon] [color(x) = red]
g = ∃(x) [weight(x) = 2..5] [shape(x) = polygon] [color(x) = red]

Figure 3. Scheme of the inductive beam search process. The figure shows in detail the search space developed starting from one single selector (c). The search space developed from the other selectors is depicted by a triangular shape.

5. Heuristic knowledge

Rigel's inductive search is controlled and focused by a body of heuristic knowledge, represented in terms of a *heuristic description language*. Below we present this language and then describe the heuristics that use it.

5.1. Heuristics description language

The heuristic description language is part of an approximate reasoning framework used in defining Rigel's evaluation criteria and functions. The syntax of an expression is given by the following grammar:

$$\langle \text{expr} \rangle ::= (\text{AND } \langle \text{arg} \rangle^+) \mid (\text{OR } \langle \text{arg} \rangle^+) \mid (\text{NOT } \langle \text{arg} \rangle)$$
$$\langle \text{arg} \rangle ::= \langle \text{linguistic-value} \rangle \mid r \mid \langle \text{expr} \rangle$$
$$\langle \text{linguistic-value} \rangle ::= \lambda$$

The symbol λ denotes a term belonging to a given set Λ and $r \in [0, 1]$ is a real number. Terms correspond to elementary conditions, which can be combined by means of the logical connectives AND, OR, and NOT to build up more complex conditions. These conditions assume a truth value in the interval [0, 1]. For this reason, each λ is the label of a fuzzy set, defined over a given base variable $x \in \mathfrak{D}_x$, whose membership function

$$\mu_\lambda : \mathfrak{D}_x \rightarrow [0, 1]$$

is given as background knowledge. If an expression coincides with some $\lambda \in \Lambda$, then the evaluation of the expression is given by the corresponding μ_λ value. If an expression contains connectives, one must define a semantics for their evaluation. The semantics is the same as that reported by Bergadano et al. (1988), which uses for the connectives AND and OR a pair of corresponding t-norm and to-conorm consistent with De Morgan's laws with respect to the semantics of the negation: $\mu(\text{NOT } \xi) = 1 - \mu(\xi)$.

As an example of a term, let us consider λ = *highpos*. Given a concept K, a set of examples POS(K), and a formula φ, let EXT(φ, POS) be the set of examples of K covered by φ. We will say that *highpos* is true for φ if the ratio $|\text{EXT}(\varphi, \text{POS})|/|\text{POS(K)}|$ is large. More precisely, the evaluation of the truth of *highpos* is given by the μ_λ value obtained from a function like that in Figure 4.

Table 1 shows the set of terms currently defined in order to control the syntactic and coverage aspects of a formula φ in Rigel.

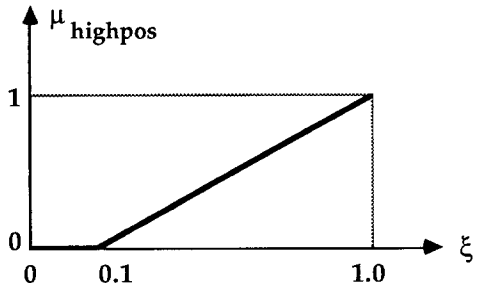


Figure 4. Example of semantic definition of the linguistic term *highpos*. The variable ξ is the ratio between the number of examples (of a concept K) covered by a formula φ and the total number of examples of K.

Table I. Terms and their meaning.

Term	Meaning
highpos	φ covers many positive examples
lowpos	φ covers few positive examples
lowneg	φ covers few negative examples
lowlength	φ contains few selectors
toogen	φ contains some selectors with many values in disjunction
quant	φ is a numerically or universally quantified formula
noquant	φ does not contain any quantified subformulas
final	φ is consistent
complete	φ is complete

5.2. Heuristic functions

Let us now describe some of the heuristic functions that Rigel uses to control its search. The seed selection algorithm calls the function $\nu_{\mathrm{HP}}(\mathrm{h})$ to evaluate the worthiness of an inductive assertion h as the starting point for the inductive search. It works on single selector formulas and examines their statistical relevance. The value of $\nu_{\mathrm{HP}}(\mathrm{h})$ is the truth value of the condition

(*highpos* AND (*lowneg* OR *0.9*)).

Rigel prefers inductive assertions that are quite complete (*highpos*) and consistent (*lowneg*). The constant 0.9 in the OR clause has the effect of lowering the weight of the clause itself. In this way the system can emulate the behavior of various lexicographic evaluation functions (Michalski, 1980).

The second function, $\nu_{\mathrm{V}}(\varphi)$, is used in the beam search. It evaluates the promise of a formula φ for generating a good final description. The value of $\nu_{\mathrm{V}}(\varphi)$ is the truth value of the condition

(*highpos* AND *lowlength* AND (*lowneg* OR (*lowlength* AND (NOT *quant*)))).

This heuristic prefers assertions that cover many positive examples, cover few negative examples, and contain few selectors. At the beginning of the search, i.e., when *lowlength* in the second term is high ($\sim$1), the consistency factor has a very low influence.

A third function, $\nu_{\mathrm{JSET}}(\varphi)$, evaluates the suitability of a formula φ to be joined to another. Rigel uses this to select the formulas to be added to a set (JSET) containing the best formulas generated so far. As ω_{J} is a specialization operator, good candidate formulas for the join are those which are simple and maximally complete. The value of $\nu_{\mathrm{JSET}}(\varphi)$ is the truth value of the condition

(*highpos* AND *lowlength*).

The function $v_{QUANT}(\varphi)$ is used to evaluate the applicability of ω_Q to a formula φ. This heuristic prefers formulas that are simple (a complex quantified formula is not easily readable), complete, and unquantified. The value of $v_{QUANT}(\varphi)$ is the truth value of the condition

(*noquant* AND *lowlength* AND *highpos*).

A final function, $v_{GEN}(\varphi)$, evaluates the opportunity of applying the operator ω_G. The purpose of this operator is to increase the number of positive examples covered. As the values are extended by taking consistency into account, the generalization is likely to fail if the candidate formula covers many negative examples. The *lowneg* term discourages the application of ω_G in such a case. If the ranges of values of the selectors in φ are already very wide (*toogen*), the application of ω_G is also discouraged. The value of $v_{GEN}(\varphi)$ is the truth value of the condition

(*lowpos* AND *lowneg* AND (NOT *toogen*)).

The flexibility of the declarative representation of the evaluation criteria facilitates the testing of different policies, simply by modifying the evaluation functions. Rigel has been tried with several alternatives functions, with the aim of better directing the inductive search, as discussed in Gemello and Mana (1988).

6. Experimental results and discussion

In this section, we describe the results obtained using Rigel on three application domains. The first involves geometric figures, whereas the other two concern real-world applications in the field of image recognition. All the results given here were obtained using a Common Lisp version of Rigel, running on an Explorer Lisp Machine.

6.1. The domain of geometric figures

The first test case is a geometric domain designed to test Rigel's ability to discover complex descriptions. This task requires the interleaved application of the add, join, generalization, and quantification operators. The geometric figures reported in Figure 5 belong to the three classes—C_1, C_2, and C_3—each of which has five examples. Table 2 lists the selectors used in the description language for this problem.

Given these data, Rigel finds a complete and consistent description for each class. The output rules are the best ones according to the currently chosen criteria; the system also finds several others, which the user can examine upon request.[2] Using e_4 as seed, Rigel proposes the following description for class C_1:

$$(\exists.\ (x\ y)[\text{SHAPE}(x) = \text{square}][\text{SIZE}(x) = \text{small}][\text{SHAPE}(y) = \text{circle}][\text{TEXTURE}(y) = \text{shaded}])$$

In other words, all members of the class C_1 contains both a small square and a shaded circle. This description reveals the system's use of the operator ω_J. The program first creates the descriptions of the square and the circle separately and then joins them later.

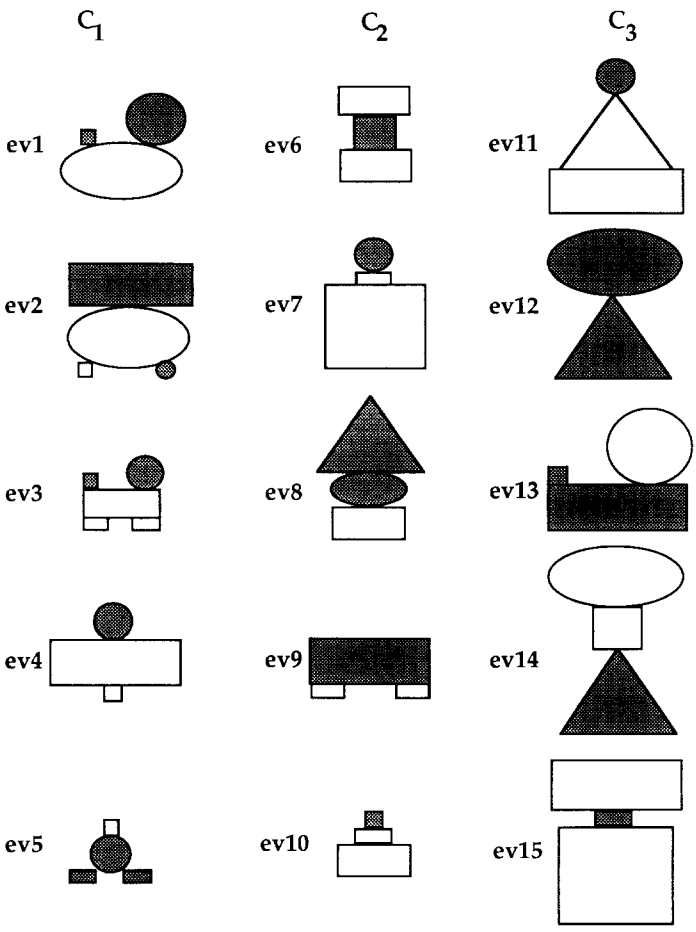


Figure 5. Examples from the domain of geometric figures.

Using the instance e_7 as its seed, Rigel finds the following description for class C_2:

$$\exists.(y)(IN(1..2))[SIZE(y) = small \vee medium][TEXTURE(y) = clear][SHAPE(y) = rectangle])$$

To summarize, members of this class contain one or two clear small-or-medium sized rectangles. Discovering this description requires the use of a quantification operator that works on formulas of arbitrary complexity. Induce cannot acquire this rule, because its implemented language is unable to express quantification over complex formulas. In contrast, Rigel is able to find both descriptions because it employs the numerical quantification operator used *during* the inductive search.

Moreover, the presence of generalized selectors points out the utility of the macro-operator Ω_{GQ}. In this case the generalization cannot take place under the consistency condition, because example e_3 of class C_1 includes some clear small-or-medium sized rectangles. The

Table 2. Descriptors used for the geometric figures in Figure 5.

Descriptor	Type	Meaning	Domain
SIZE(x)	Linear	Size of the object x	{Small, Medium, Large}
TEXTURE(x)	Nominal	Texture of the object x	{Clear, Shaded}
SHAPE(x)	Structured	Shape of the object x	{Triangle, Square, Rectangle, Circle, Ellipse, Polygon (|> Triangle, Square, Rectangle), Curved (|> Circle, Ellipse)}
ONTOP(x y)	Nominal	Object x is on y	{True, False}

macro-operator Ω_{GQ} is able to generate the final description by relaxing the consistency condition in the generalization step, thus generating an intermediate generalization and specializing it with the following quantification operator.

Finally, starting from instance e_{14}, Rigel discovers a discriminant description for the class C_3:

$$(\exists.(x\ y)[\text{ONTOP}(x\ y)][\text{SHAPE}(y) = \text{triangle} \vee \text{rectangle}][\exists.(z)(=2)[\text{SIZE}(z) = \text{large}]))$$

In this case, the class is described by the presence of a triangle or a rectangle under another object, as well as by the presence of two large objects. This shows that Rigel can find descriptions by searching contexts for generalizing consistent but incomplete descriptions. In fact, the system discovers that the presence of a triangle under another object is sufficient to discriminate some examples but then continues the search, joining the numerical property and generalizing the obtained formula. In this way, it finds a final description that covers all the examples of the class. This also shows the utility of a interleaving generalization and specialization during the actual search process.

Finally, the role of the seed selection algorithm deserves analysis. To this aim, an experiment was run in which each example was selected as seed in turn, while the beam size was set (by trial and error) to the minimum value that ensured Rigel would find the desired final descriptions. The examples that led the inductive search to generate the minimum search space were the best seeds. Table 3 reports the number of hypotheses considered in each case. The italicized rows show the examples that Rigel selected as its seed, revealing that its strategy for seed selection leads to considerably less search than random selection. Moreover, the time consumed by the seed selection process represents only 0.3% of the total time needed to discriminate the three classes.

6.2. An image recognition domain

The other two testbeds are taken from the domain of image recognition. These cases represent interesting tests on Rigel's ability to deal with real data. All the results reported in this section have been obtained using the performance system described in Section 2.1. The first problem involves discriminating among the five flat metallic pieces shown in Figure 6. The examples are generated from a set of 2D images of the objects, acquired by a TV camera placed above the table supporting the pieces. The pixel map is analyzed by a low-level module (Mangilli & Viano, 1987) that describes the contours in terms of angles, straight

Table 3. Evaluation of the seed selection algorithm on the data from Figure 5.

Seed	Beam	Number of Hypotheses
ev1	2	130
ev2	2	93
ev3	2	103
ev4	***2***	***83***
ev5	3	94
ev6	5	104
ev7	***1***	***55***
ev8	5	106
ev9	1	56
ev10	5	123
ev11	2	50
ev12	2	48
ev13	29	415
ev14	***2***	***56***
ev15	30	399

lines, and curves, as shown in Figure 7. Associated with each primitive is a set of features that are invariant with respect to rotation and translation.

Due to random variations in the input images, resulting from rotation, illumination, and noise in the low-level signal processing, the results of the segmentation process differ from case to case. Figure 8 shows the complete set of eight training instances of the class H_1 that were generated by the feature extractor and given to Rigel.

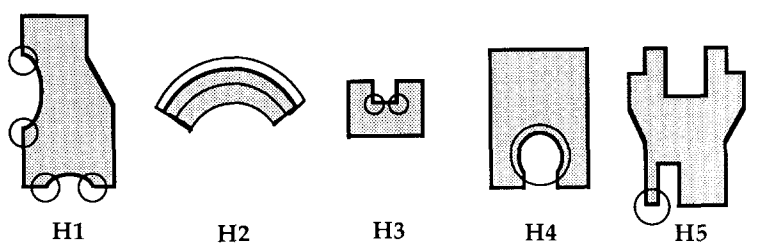


Figure 6. The metallic pieces to be discriminated in the image recognition problem. The discriminant features learned by Rigel are encircled.

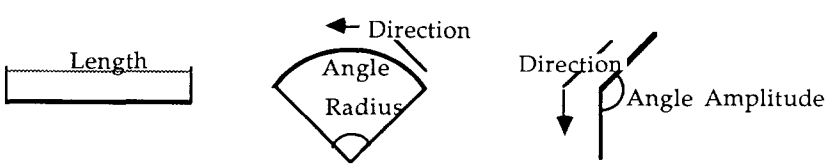


Figure 7. Primitive features extracted by low-level image processing.

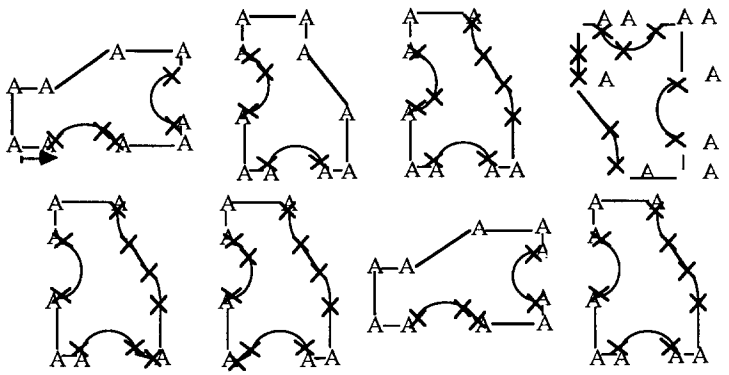


Figure 8. The segmentation generated for the H_1 class images. The capital letter A represents the angle, the marker X the curve delimitation and the segment represents the straight line. Due to random variations in the input images and noise in the low-level image processing, the segmentation differs from image to image, both in the number of segments and in the labelling.

Table 4 summarizes the descriptors that Rigel uses in this task. These let one construct a symbolic description of the input examples, starting from a point chosen at random and following the image contour in a counterclockwise direction. For example, a partial description of the first example in Figure 8 can be stated as:

[TYPE(p_1) = straight_line][LENGTH(p_1) = 37.6][NEXT(p_1 p_2)][TYPE(p_2 = angle] [AMPLITUDE-A(p_2) = obtuse][ANGLE-A(p_2) = 2.263][DIRECTION(p_2) = left]...

Rigel was given a total of 55 input examples, consisting of 8 for the concept H_1, 12 for H_2, 10 for H_3, 13 for H_4, and 12 for H_5. From these data, the system induced a set of conjunctive rules expressing sufficient conditions to discriminate the images of the pieces. Table 5 shows the simplest (shortest) rules proposed by Rigel to discriminate the five classes, along with English paraphrases for each rule.

Table 4. Descriptors used for the images in Figure 6.

Descriptor	Type	Meaning	Domain
TYPE(p)	Nominal	Primitive type	{angle, straight_line, curve}
LENGTH(p)	Linear	Straight line length	(0. .150) pixel
DIRECTION(p)	Nominal	Curve or angle direction	{left, right}
AMPLITUDE-C(p)	Nominal	Amplitude of the curve	{acute, right, obtuse, straight_angle, round_angle}
AMPLITUDE-A(p)	Nominal	Amplitude of the angle	{acute, right, obtuse, straight_angle, round_angle}
ANGLE-C(p)	Linear	Curve numerical amplitude	(0. .6.28) radians
ANGLE-A(p)	Linear	Angle numerical amplitude	(0. .3.14) radians
RADIUS(p)	Linear	Radius of the curve	(0. .620) pixel
NEXT(p_1 p_2)	Nominal	Primitive p_2 follows primitive p_1	{true false}

Table 5. Decision rules discovered by Rigel.

Rule	Paraphrase
(∃.(x)[ANGLE-A(x) = 2.04 .. 2.36]) ::> H1	There exists an angle with amplitude between 2.04 and 2.36 rad
(∃.(x)[RADIUS(x) = 86.3 .. 132.0] [ANGLE-C(x) = 0.63 .. 1.26] [DIRECTION(x) = left]) ::> H2	There exists a left-curve with a radius between 86.3 and 132.0 pixels and whose amplitude is between 0.63 and 1.26 rad
(∃.(x)(1..2)[ANGLE-A(x) = right] [DIRECTION(x) = right]) ::> H3	There exist one or two right angles whose direction is right
(∃.(x)[AMPLITUDE-C(x) = round_angle]) ::> H4	There exists a curve with an amplitude round angle
(∃.(x)[AMPLITUDE-A(x) = acute]) ::> H5	There exists an angle with amplitude acute

These rules have also been experimentally found to be those minimizing the classification time. Moreover, Rigel found many other discrimination rules, including the more complex rule:

$$(\exists.(x\ y)[\text{TYPE}(x) = \text{angle}][\text{AMPLITUDE-A}(x) = \text{obtuse}][\text{NEXT}(x\ y)]$$
$$[\text{TYPE}(y) = \text{curve}][\text{AMPLITUDE-C}(y) = \text{acute}]$$
$$(\exists(\geq 3)(x)[\text{AMPLITUDE-C}(x) = \text{right} \vee \text{acute}])) ::> \text{H1}$$

This can be paraphrased as "There exists an angle whose amplitude is obtuse followed by a curve whose amplitude is acute, and there exist more than three right or acute curves."

The above limited experiment was run to test Rigel's ability to find rules starting from corrupted data, rather than to test its predictive power. Nevertheless, the data were separated into training set (70%) and test set (30%). This splitting was performed randomly 20 times; in each case the learned rules were used to classify the test set and the accuracy was measured. In the majority of the cases (60%), classification was achieved with 100% accuracy and without any ambiguity. In the other cases (40%), classification was somewhat less correct (from 75% to 90%), with some ambiguous classification occurring. Overall, the average classification accuracy was 97% over the 20 runs, though this sample is not large enough for statistical evaluation.

The second task was also chosen from the image recognition domain, this one concerned with discriminating among 18 capital letters (A, B, C, D, E, F, G, H, L, M, N, P, R, S, T, U, V, Z). This task is more difficult than the previous one because some of the letters are very similar, requiring more complex rules. Rigel was given 16 images for each letter, but only 70% of them were used as training examples, with the remaining 30% being used as the test set. In other words, 198 examples were given to Rigel to perform the learning step, and a classification system used the learned rules to classify the 90 test patterns.

The resulting performance in the test set, evaluated as an average over 10 runs, can be summarized as follows: 89% of the cases were correctly classified (i.e., only rules corresponding to the correct class were matched by these instances); 2% of the cases received an ambiguous classification (i.e., rules of the correct class were matched but some others

were as well); 7% of the cases were not classified (i.e., no rule was matched); and 2% of the cases received an erroneous classification (i.e., only rules corresponding to wrong classes were matched). The reported performance level seems quite respectable on a real-world domain the system was not initially designed to handle.

6.3. Sensitivity analysis

We have argued that one of Rigel's advantages is its declarative representation of knowledge for search control. Different domains may require more or less effort to separate classes, involve different induction operators, and so forth. One can often interview a domain expert to obtain estimates of these characteristics, which can then be encoded to constrain the search process. If not, one carries out an initial phase of experimental tuning to determine reasonable settings for a given domain.

This engineering approach is suitable for the semi-automated construction of knowledge-based systems, but it sidesteps the interesting scientific question of Rigel's sensitivity to these parameter settings. To answer this question, we carried out two experiments that varied the system's parameters. The first study compared two runs of Rigel on the image recognition task that involved metallic objects. One run used the same set of parameters as originally used for the domain of geometric figures, whereas for the other run we changed 7 of the 12 total tunable parameters. In the latter case Rigel acquired rules that differed only slightly from those found with the original tuned parameters values. Moreover, these rules performed quite well, with only one example of class H_4 being ambiguously classified.

The second experiment was more systematic, varying the setting for two parameters and measuring accuracy of the acquired rules on the metallic pieces domain. In particular, the two parameters were the amplitude of the beam search, which was varied from 5 to 30 (with increments of 5), and the applicability threshold of one of the learning operators, which was increased from 0.5 to 1 (with an increment of 0.1) until the operator was deactivated. This gave a total of 36 combinations, and we performed two runs for each of them, repeating the entire process for each of Rigel's operators.

When run on the same training and test sets as described above, Rigel generally acquired the same rules independent of the parameter setting. The system did find different rules when the join operator w_J was deactivated, but their performance did not sensibly change. Most likely, these results are due to the simplicity of the discrimination rules for this domain, but they are still encouraging.

7. Related work

In addition to Rigel and Induce, only a few other inductive learning systems have the ability to deal with structured examples. Early work by Plotkin (1970), Vere (1975, 1980), and Winston (1975) focused on methods for acquiring a concept by characterizing a set of examples, using counter-examples to constrain the search when possible. These approaches represented concepts as conjunctive first-order logical formulas containing positive literals, although Vere (1980) also allowed disjunctions and negations ("counterfactuals") for representing exceptions. These methods were limited along a number of dimensions. None allowed

for many-to-one variable bindings, none could handle noisy training data, and some employed exponential enumerative algorithms to search the space of concept descriptions. Taken together, these limitations made the systems unsuitable for use in real domains.

Hayes-Roth and McDermott (1978) responded to some of these issues by describing an alternative method that searched for the most specific generalization of a set of examples of a given concept. One advantage of their approach was that it explicitly stated the heuristic criteria used to choose among alternative generalizations. Thus, their Sprouter system fits well into Mitchell's (1982) view of generalization as search through a space of concept descriptions. Like the other system mentioned above, Sprouter carried out a specific-to-general search strategy, incorporating operators for dropping conditions and the turning constants to variables.

Kodratoff and Ganascia (1986), in their "structural matching" approach, augment the inductive learning process with deductive inference. Their approach uses background knowledge to construct concept descriptions that contain features and relations that are not explicitly present in the training examples. However, their method suffers from a high computational complexity, which is made even worse when noise is present in the data.

Iba, Wogulis, and Langley's (1988) Hillary uses an incremental hill-climbing strategy to achieve lower complexity. From their perspective, concept learning involves a search for descriptions that satisfy opposing criteria, which can be explicitly stated as a modifiable evaluation function. Specifically, Hillary learns disjunctive concepts from examples and counter-examples, attempting to trade off simplicity of the concept description (low number of disjuncts) with coverage (low error rate). Experimentally, the system showed some ability to handle noise that occurred from misclassified training examples.

Like all the systems described above, Rigel's basic induction method involves a search from specific concepts to more general ones. However, the use of numerical quantification during this search, together with the extending reference rule, lets Rigel acquire descriptions that the other systems cannot find. In addition, the interleaving of general-to-specific induction operators with specific-to-general ones allows a more flexible search for useful concepts. Finally, Rigel incorporates a body of declarative knowledge that states criteria for guiding each step of the search, which helps the system reduce the resources it requires. This aspect bears similarities with Hillary, though the current system has a more extensive search control scheme.

Finally, Rigel owes several ideas to Bergadano, Gemello, Giordana, and Saitta's (1988, 1989) ML-SMART, from which the inductive operators and declarative heuristic criteria have been taken. Nevertheless, the two systems differ in important ways. For instance, ML-SMART uses a general-to-specific strategy for searching the space of concept descriptions. The system can also learn several concepts at the same time, and rather than representing its acquired knowledge as a set of flat if-then rules, it structures this knowledge into a rule network.

8. Conclusions

This paper has described Rigel, an inductive system for the automated acquisition of knowledge for concept discrimination tasks. Starting from a set of classified examples of a given set of concepts (classes), the system learns a set of rules that let it discriminate each class

from the others. We have tested Rigel on several classic induction tasks, including Michalski's (1980) trains domain, geometric figures and chemical formulas (Michalski & Stepp, 1986). In all cases the system has learned the descriptions reported in the literature, and it has often discovered additional solutions. We have also applied Rigel to the domain of image recognition with encouraging results. At present we are applying the system to a larger task, namely the diagnosis of faults in a telecommunication network.

However, the current version of the system has a number of limitations, including conceptual problems, which could be corrected by extensions, and fundamental problems, which require a different approach. Among the first type of problem is the fact that Rigel is not *incremental*, in the sense that it cannot take into account initial knowledge provided by the teacher or by a previous run on another set of examples. This ability would be very useful in constructing knowledge-based systems, letting an expert supply initial hypotheses of discrimination rules and letting the learning system refine them.

Further, the current system does not carry out any kind of *deductive learning*. An extended Rigel might use a domain theory to guide the inductive process. The system is also unable to interact with the user for advice and guidance, although the user can provide precedence constraints on the descriptors used during the inductive search. Finally, the semantics of the descriptors is Boolean and not a continuous-valued one. Such a facility would allow the system to handle borderline cases, especially in noisy domains.

A more fundamental problem involves Rigel's inability to generate intermediate decision rules, which could discriminate subsets of classes. That is to say, the learned knowledge is represented as a set of one-level rules. Constructive learning activity is essentially limited to quantification: the system is not able to introduce new descriptors and use them in a discovery process.

As several of these problems have been overcome with a different approach taken in a parallel project (Bergadano et al., 1988), future research on Rigel will focus only on certain issues, such as increasing the system's ability to handle noise. With this in mind, we have designed a characterization module and integrated it with the discrimination module (Gemello & Mana, 1989). We are studying the effects of this integration, both from the viewpoint of evaluating knowledge quality and of obtaining computational benefits. The source of these benefits is the reduction in the number of examples that must be handled during the discrimination phase; in fact, these examples can be limited to a set of precisely defined "near misses" (Winston, 1975). We also plan to explore approaches to introducing incremental learning capabilities into the system, by letting it start from a set of tentative rules, possibly given by a domain expert.

In summary, Rigel seems to be a useful tool for learning rule in structured domains, as it couples the richness of a first order representation language with contained computational complexity.

Appendix

Here we summarize the methodology proposed by Michalski (1983) for inductive learning. The framework is based on the representation language (VL), in which the predicate notation is replaced by the relational statement notation (*selector*):

$$S = [T_1 \# T_2],$$

where T_1 is the *referee*, T_2 is the *reference*, and # is a *relation*. An example satisfies a relational statement S iff the value, computed by evaluating the referee T_1 on the example, is in relation # with the reference T_2. Two versions of VL have been introduced: VL_1, which is an *attribute-value language* where T_1 is an attribute name, and VL_2, which is a *first order language*, where T_1 is a predicate in first order logic.

In Michalski's methodology, the concepts are learned one at a time. Given a set $\mathcal{K} = \{K_1, \ldots, K_n\}$ of concepts to be learned, one defines a set $POS(K_i)$ of positive examples and a set $NEG(K_i)$ of negative examples for each concept K_i. Single concept learning is performed by computing the covering set $COV(POS(K_i)|NEG(K_i))$ of the set $POS(K_i)$ against the set $NEG(K_i)$. The covering set is defined as the set of concept descriptions so that there is at least one description that covers each example $e \in POS(K_i)$ and so that no such description exists for any example $e \in NEG(K_i)$.

The covering algorithm uses the concept of a *star* for an example $e \in POS(K_i)$, called the *seed*, against the set $NEG(K_i)$. A star is the set of all nonredundant descriptions of K_i that cover the seed and do not cover any counterexample. In practical problems the star may contain many descriptions and will be computationally intractable. Therefore, the star concept is replaced by the *reduced star* concept, which contains no more than a fixed number of descriptions. The best descriptions are retained in the reduced star, in accordance with a lexicographic preference criterion provided by the teacher.

At each cycle of the covering algorithm, the covering set is updated with the best description chosen from among those found in the reduced star. If the covering set is partial (if the examples in $POS(K_i)$ are not all covered), then the algorithm continues computing the covering of the uncovered examples, until it reaches a total covering. Thus, the covering set can be viewed as a disjunction of descriptions generated by different stars. Michalski has proposed two different algorithms, AQ and Induce, for computing the star. As we discuss below, they differ in their representation language and in their generalization rules.

A.1. The AQ algorithm

The AQ algorithm uses the VL_1 representation language, in which the simplest entity is the *selector* $[x_i \# R_i]$; x_i is an attribute name, $\# \in \{=, \neq\}$ is a connective relation, and R_i is a set containing values connected by internal disjunction. The attributes needed to describe the application domain are given by the teacher. A domain is associated with each attribute and is described as the set of values that the attribute can assume. The domain is classified in accordance with the relation existing among its values: *nominal* if no relation between the values exists, *linear* if there is a linear relation among the values, and *structured* if an IS-A hierarchy links the values.

A well-formed formula in VL_1 is called a *complex* and is composed of a conjunction of selectors:

$$[x_1 \# R_1] [x_2 \# R_2] \ldots [x_m \# R_m].$$

Given a set $\{x_1\ x_2\ \ldots\ x_n\}$ of variables and an example of the application domain (i.e., an event e), the latter is described by means of an n-selector conjunction:

$$e = [x_1 = v_1]\ [x_2 = v_2]\ \ldots\ [x_n = v_n].$$

Actually, it is sufficient to have an n-ary vector in which the i-th element contains the value associated with the i-th variable.

The AQ algorithm uses the *extension against rule* and the *extending reference rule* as its inductive operators. The first of these generalizes an attribute in the context of a complex:

$$\varphi \wedge [x = R_1] ::> K,\quad \psi \wedge [x = R_2] ::> \neg K \quad |< \varphi \wedge [x \neq R_2] ::> K$$

where $R_1 \cap R_2 = \phi$. In contrast, the *extending reference rule* generalizes an attribute by extending the set of values it can assume:

$$\varphi \wedge [x = R_1] ::> K \quad |< \varphi \wedge [x = R_2] ::> K,$$

where $DOM(x) \supseteq R_2 \supseteq R_1$ and DOM(x) denotes the domain of x.

The AQ algorithm computes the Star G(e|E) of an example e against a set E. The star G will be a set of complexes that satisfy the consistency condition. Given the set of variables $\{x_1\ x_2\ \ldots\ x_n\}$, an example e, and a set of examples $E = \{e_1\ e_2\ \ldots\ e_n\}$, the *elementary star* $G(e|e_i')$ is the set of maximally general complexes that cover e and do not cover e_i. Intuitively, the elementary Star represents each possible way of discriminating e from e_i by means of conjunctive formulas in VL_1. The computation of $G(e|e_i')$ is realized in the following way:

(a) Let $e = (r_1\ \ldots\ r_n)$ and $e' = (r_1'\ \ldots\ r_n')$.
(b) Let $x_{j_1}, \ldots, x_{j_k}$ $(1 \leq j_i \leq n)$ be the variables whose values differ in e and e'.
(c) Generate k complexes in accordance with the *extension against rule*

$$G(e|e') = \{[x_{j_i} \neq r_{j_i}'] \mid i \in [1\ ..\ k]\}$$

The Star G(e|E) is then defined as the conjunction of all possible ways of discriminating e from every example e_i belonging to E. Formally, G(e|E) is defined as

$$G(e|e') = \bigwedge_{e_i \in E} G(e|e_i)$$

In order to obtain nonredundant complexes in the partial star, the conjunction task is performed under the absorption law condition (i.e., $A \wedge A = A$). As an example, let us consider two elementary stars $G(e|e_i) = cpx_1 \vee cpx_2$ and $G(e|e_j) = cpx_1 \vee cpx_3$. The star of e against the set $\{e_i, e_j\}$ can be computed as follows:

$$G(e|\{e_i\ e_j\}) = (cpx_1 \vee cpx_2)(cpx_1 \vee cpx_3) = cpx_1 \vee cpx_2cxp_1 \vee cpx_2cpx_3.$$

The descriptions contained in the star use only values already present in the examples. During the star computation, each variable is considered as belonging to a nominal domain, and domain knowledge is used to simplify the descriptions that are found. For this reason, AQ applies the *extending reference rule* to each description, so as to search for the best internal disjunction.

AQ computes a reduced star, in that it retains only the m best complexes of the partial star in accordance with the preference criterion defined by the teacher.

A.2. The Induce algorithm

This algorithm works on descriptions in the VL_2 representation language, an extension of VL_1 that can manipulate expressions containing variables and universally or numerically quantified formulas. The simplest entity allowed in VL_2 is the selector [L # R], where L is a n-ary function (descriptor), # is a connective belonging to the set $\{= \neq \leq \geq < >\}$ and R is a set of values of L. The values in R are considered to be internal disjuncts. A domain type and a range of values are associated with each descriptor.

Induce considers only a subset of the VL_2 language, namely the set of well-formed formulas (wff) consisting of selectors and of formulas built up by means of conjunction, disjunction, internal disjunction, and existential quantification. Each example (event) e, containing the objects $a_1, \ldots, a_m$, is described as a conjunction of selectors involving the a_i's.

For its inductive process, Induce uses the *dropping condition rule*, the *extending reference rule*, and the *extension against rule*. The first of these operators generalizes a formula by dropping a selector:

$$\varphi \wedge [L(x) = v] ::> K \quad |< \varphi ::> K$$

The other rules are the same as those defined for the AQ algorithm.

The Induce algorithm also computes the Star G(e|E) of an example e against the set E. The algorithm consists of two steps: the first one uses only selectors of the seed to obtain some descriptions that do not cover any counterexample, and in the second step it generalizes these descriptions. In the first step, the *dropping condition rule* is applied to the seed description until single selector assertions are obtained. Afterwards, each single selector hypothesis is repeatedly specialized using the *reverse dropping condition rule.*

The second step implements the maximal generalization of the formulas found in the first step. This is accomplished by applying the *extension reference rule.* The application of this rule is computationally more expensive on VL_2 expressions than on VL_1 ones. For this reason, the Induce algorithm uses the AQ algorithm for the second step. Each VL_2 description is used to generate corresponding VL_1 examples. Given the VL_2 formula φ, a new formula φ^* is created. This has the same conjunction defined by φ, but each involved selector is maximally generalized; for example, [color(X) = red] is replaced by [color(X) = *], where * represents the whole domain of color. Examples and counterexamples for the AQ system are generated by computing each possible binding between φ^* and the VL_2 examples and counterexamples, respectively. Each binding represents an event expressed in VL_1.

The algorithm sketched above computes a reduced star. As in AQ, the reduced star is obtained by keeping the m best generated formulas.

Acknowledgments

The authors are grateful to all the referees for their careful evaluation and many useful suggestions.

Notes

1. Rigel is also a first order magnitude star of the Orion constellation.
2. Rigel acquired concepts C_1, C_2 and C_3 in 11, 14 and 18 seconds, respectively.

References

Anderson, J.R., & Kline, P.J. (1979). A learning system and its psychological implications. *Proceedings of the Sixth International Joint Conference on Artificial Intelligence* (pp. 16–21). Tokyo, Japan: Morgan Kaufmann.

Bergadano, F., Giordana, A., & Saitta L. (1988). Automated concept acquisition in noisy environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 10,* 555–577.

Bergadano, F., Gemello, R., Giordana, A., & Saitta L. (1989). SMART: A problem solver for learning from examples. *Fundamenta Informaticae.*

Bergadano, F., & Giordana, A. (1988). A knowledge intensive approach to concept induction. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 305–317). Ann Arbor, MI: Morgan Kaufmann.

Bergadano, F., & Giordana, A. (in press). Guiding induction with domain theories. In R.S. Michalski & Y. Kodratoff (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 3). San Mateo, CA: Morgan Kaufmann.

Carbonell, J.G., Michalski, R.S. & Mitchell, T.M. (1983). *Machine learning: An artificial intelligence approach* (Vol. 1). San Mateo, CA: Morgan Kaufmann.

Danyluk, A.P. (1987). The use of explanations for similarity-based learning. *Proceedings of the Tenth International Conference on Artificial Intelligence* (pp. 274–276). Milan, Italy: Morgan Kaufmann.

De Jong, G., & Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning, 1,* 145–176.

Gemello, R., & Mana, F. (1988). Controlling inductive search in RIGEL learning system. *Proceedings of the International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems.* Urbino, Italy: Springer Verlag.

Gemello, R., Mana, F., & Viano, G. (1988). Inducing conceptual discrimination rules from examples: An application to image recognition. *Proceedings of the Third International Symposium on Methodologies for Intelligent Systems* (pp. 313–321). Torino, Italy: North-Holland.

Gemello, R., & Mana, F. (1989). An integrated characterization and discrimination scheme to improve learning efficiency in large data sets. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 719–724). Detroit, MI: Morgan Kaufmann.

Hayes-Roth, F., & McDermott, J. (1978). An interference matching technique for inducing abstractions. *Communications of ACM, 21,* 401–410.

Haussler, D. (1987). Learning conjunctive concepts in structural domains. *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 466–470). Seattle, WA: Morgan Kaufmann.

Haussler, D. (1988). Quantifying inductive bias: AI learning algorithms and Valiant's learning framework, *Artificial Intelligence, 36,* 177–221.

Iba, W., Wogulis, J., & Langley, P. (1988). Trading off simplicity and coverage in incremental concept learning. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 73–79). Ann Arbor, MI: Morgan Kaufmann.

Kearns, M., Li, M., Pitt, L., & Valiant, L. (1987). Recent results on Boolean concept learning. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 337–352). Irvine, CA: Morgan Kaufmann.

Kodratoff, Y., & Ganascia, J.G. (1986). Improving the generalization step in learning. In J.G. Carbonell, R.S. Michalski, & T.M. Mitchell (Eds.), *Machine Learning: An artificial intelligence approach* (Vol. 2). San Mateo, CA: Morgan Kaufmann.

Langley P. (1987). A general theory of discrimination learning. In D. Klahr, P. Langley, & R. Neches (Eds.), *Production system models of learning and development*. Cambridge, MA: MIT Press.

Lebowitz, M. (1986). Integrated learning: Controlling explanation. *Cognitive Science, 10*, 219–240.

Mangili, R., & Viano, G. (1987). A dynamic programming approach to knowledge based contour segmentation. *Proceedings of the Fourth International Conference on Image Analysis and Processing* (pp. 136–147). Cefalu', Italy.

Michalski, R.S. (1983). A theory and methodology of inductive learning. In J.G. Carbonell, R.S. Michalski, & T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 1). San Mateo, CA: Morgan Kaufmann.

Michalski, R.S. (1980). Pattern recognition as a rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 2*, 349–361.

Michalski, R.S., & Stepp, R.E. (1986). *Induce 3: A program for learning structural descriptions from examples* (Internal Report). Urbana-Champaign: University of Illinois, Department of Computer Science.

Michalski, R.S., Mozetic, I., Hong, J., & Lavrac, N. (1986). *The AQ15 inductive learning system: An overview and experiments* (Intelligent Systems Group Report). Urbana-Champaign: University of Illinois, Department of Computer Science.

Michalski, R.S., & Larson, J.B. (1983). *Incremental generation of VL_1 hypotheses: The underlying methodology and the description of the program AQ11* (Intelligent Systems Group Report). Urbana-Champaign: University of Illinois, Department of Computer Science.

Michalski, R.S., & Stepp, R.E. (1983). How to structure structured objects. *Proceedings of the International Machine Learning Workshop* (pp. 156–160). Monticello, IL.

Mitchell, T.M. (1982). Generalization as search. *Artificial Intelligence, 18*, 203–226.

Mitchell, T.M., Keller, R.M., & Kedar-Cabelli, S.T. (1986). Explanation-based generalization: A unifying view. *Machine Learning, 1*, 47–80.

Pazzani, M.J. (1988). Integrating explanation-based and empirical learning methods in OCCAM. *Proceedings of the Third European Working Session on Learning* (pp. 147–165). Glasgow, UK.

Pearl, J. (1979). Capacity and error estimate for Boolean classifiers with limited complexity. *IEEE Transaction on Pattern Analysis and Machine Intelligence, 10*, 350–355.

Plotkin, G.D. (1970). A note on inductive generalization. In B. Meltzer & D. Michie (Eds.), *Machine Intelligence* (Vol. 5). Edinburgh, UK: Edinburgh University Press.

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

Utgoff, P. (1988). ID5: An incremental ID3. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 107–120). Ann Arbor, MI: Morgan Kaufmann.

Valiant, L. (1985). Learning disjunctions of conjunctions. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 560–566). Los Angeles, CA: Morgan Kaufmann.

Vapnik, V.N., & Chervonenkis, A. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications, XVI*, 264–280.

Vere, S. (1975). Induction of concepts in the predicate calculus. *Proceedings of the Fourth International Joint Conference on Artificial Intelligence* (pp. 351–356). Tbilisi, USSR: Morgan Kaufmann.

Vere, S. (1980). Multilevel counterfactuals for generalizations of relational concepts and productions. *Artificial Intelligence, 14*, 139–164.

Winston, P. (1975). Learning structural descriptions from examples. In P.H. Winston (Ed.), *The psychology of computer vision*. New York: McGraw Hill.