Published in *Computational Intelligence*, Volume 5, Number 2, 1989, pages 67-81.

Explicitly biased generalization

DIANA GORDON
Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory, Code 5510
4555 Overlook Avenue, S.W.
Washington, D.C. 20375, U.S.A.

AND

DONALD PERLIS [1]
Department of Computer Science
University of Maryland
College Park, MD 20742, U.S.A.

During incremental concept learning from examples, tentative hypotheses are formed and then modified to form new hypotheses. When there is a choice among hypotheses, *bias* is used to express a preference. Bias may be expressed by the choice of hypothesis language, it may be implemented as an evaluation function for selecting among hypotheses already generated, or it may consist of screening potential hypotheses prior to hypothesis generation. This paper describes a use of the third method. Bias is represented explicitly both as assumptions that reduce the space of potential hypotheses and as procedures for testing these assumptions. There are advantages gained by using explicit assumptions. One advantage is that the assumptions are meta-level hypotheses that are used to generate future, as well as to select between current, inductive hypotheses. By testing these meta-level hypotheses, a system gains the power to anticipate the form of future hypotheses. Furthermore, rigorous testing of these meta-level hypotheses before using them to generate inductive hypotheses avoids consistency checks of the inductive hypotheses. A second advantage of using explicit assumptions is that bias can be tested using a variety of learning methods.

Key words: machine learning, induction, bias

---

[1] This author is affiliated with the Institute for Advanced Computer Studies at the University of Maryland. This research is supported, in part, by a grant from the Army Research Office.

1. Introduction

Bias plays an important role in the major subfield of machine learning called *empirical induction*. Empirical induction of a concept from examples is performed by formulating hypotheses that approximate an unknown target concept based on experience with instances classified (e.g., by a teacher) as positive or negative examples of the target concept. If learning is *incremental* (on-line rather than batch), hypotheses are formed and then modified to remain consistent with the growing set of known instances. The major modification techniques are generalization and specialization. A hypothesis is consistent with the instances if it logically implies all known positive instances and no known negative instances. The number of hypotheses that are consistent with the instances may be large. Since the purpose of each hypothesis is to predict over future instances, a judicious choice of certain hypotheses over others can improve these predictions, thereby enhancing system performance. Any basis for hypothesis preference is considered bias.[2]

To illustrate bias, this paper will refer to a scenario consisting of a robot in a totally automated factory that has to pick new, raw parts out of a bin. The robot needs to feed each of these parts to a milling machine by ''fixturing'' them (placing the parts into a fixture that clamps them). It has no prior knowledge of which specific part characteristics are correlated with the ability of the part to be successfully fixtured prior to being milled. However, the robot has sufficient visual and tactile sensors to detect whether or not the fixturing operation has succeeded. Therefore, the robot needs to learn the concept of parts that can be successfully held in place by the fixture of this particular machine, i.e., ''fixturable'' parts. Learning this concept will improve the robot's performance because the robot will no longer waste time trying potentially unsuccessful parts, yet will try as many parts as it considers capable of success. Within the bin are parts of size small and large; of material steel, brass, bronze, copper, and aluminum; and of shape sphere, cone, cylinder, cube, and brick. (We assume the robot can recognize these features.) Suppose a small, copper cube has been fed into the machine by the robot and machined successfully. This, then, becomes a positive example. Suppose, further, that a small, brass sphere will not remain secure within the fixture. This becomes a negative example. Many hypotheses, or predictions over future instances, can be made that are consistent with these two examples. One possible hypothesis is that cubes will be successful and spheres will fail. This hypothesis may have been formed because of a belief that the shape of an object is the only feature that is relevant for determining an object's ability to be fixtured. Another possible hypothesis is that copper parts will succeed and brass parts are expected to fail. One of these two hypotheses will be a better predictor than the other. A preference for one of the two hypotheses is considered bias. During incremental induction, hypotheses may be generalized to include new instances. For each new instance, there are usually many, generalized hypotheses that can be formed to include it. Therefore, bias continues to be important as hypotheses are modified during incremental learning.

Three major techniques exist for biasing induction. The first technique consists of using a restricted hypothesis language. *Restricting the language* effectively limits the hypothesis choice because not all hypotheses are expressible. By adding terms to

[2] Called ''inductive bias'' in (Mitchell 1980) or ''preference criteria'' in (Michalski 1983a).

the language, a system with restricted hypothesis language can *shift* (alter) its bias (Utgoff 1986). The second bias technique, called *testing*, is used in systems such as Michalski's AQ and Fisher's COBWEB to evaluate hypotheses that have already been generated (Michalski 1983; Fisher 1987). AQ, for example, bases hypothesis preference on measures such as simplicity or degree of fit to the data. The third bias technique, called *screening*, has previously received less attention than the other two methods. Screening biases induction by reducing the number of hypotheses prior to hypothesis generation. This paper discusses the implementation of a system, called PREDICTOR (Gordon 1988), for which the primary focus is the screening method of bias.[3] Bias in PREDICTOR is represented explicitly both as assumptions that are used to screen the hypotheses and as procedures for testing these assumptions. Screening is performed by generalization heuristics. These heuristics are condition-action pairs. Each heuristic condition consists of a procedure to test a biasing assumption. (An example of a biasing assumption is the irrelevance of the material of parts to their ability to be fixtured. A procedure to test the assumption might consist of trying two objects having the same size and shape but different materials.) The heuristic action consists of applying a generalization operator (such as the elimination of any reference to the material of objects) to the current, as well as future, hypotheses.[4] Although heuristics for selecting generalization operators have occasionally been used in other systems, such as that of Winston (1975), there has not yet been a careful study of this subject. This paper presents new generalization heuristics based on original definitions, as well as comparative evaluations of different heuristics.

Explicit bias in the form used by PREDICTOR offers a couple of advantages for an inductive system in addition to the computational benefit gained from screening prior to generating competing hypotheses. The first advantage is that the bias is a meta-level hypothesis used to generate inductive hypotheses. Since the bias is itself a hypothesis, it may be tested and confirmed or refuted. A system that tests its hypothesis generator has the power to anticipate the form of future hypotheses, thereby reducing the search space for future, as well as current, inductive hypotheses. Furthermore, the testing of the bias may be done using different degrees of rigor at different times in the learning process. A ''cautious'' mode of learning will thoroughly test the assumptions prior to hypothesis generation. If the cautious mode is employed, the inductive hypotheses are guaranteed to be consistent with previous instances (though consistency with future instances is not assured). Therefore, cautious learning avoids the costly process of instance reexamination or any other consistency checks just after generalizing. Furthermore, generalization errors are reduced when cautious heuristics are used. Generalization heuristics that operate in the cautious mode are called *consistency-preserving*. Alternatively, generalization may proceed in a less cautious manner. With less cautious generalization, learning proceeds more rapidly but the number of errors is increased. When generalization errors occur, however, the system can recover gracefully using, once again, the explicit bias. Error resolution methods rigorously retest the biasing assumptions. If any of the assumptions are violated, they can be retracted (which shifts the bias) and the error remedied by *directed backtracking* to a previous hypothesis. Backtracking is used in

---

[3] PREDICTOR is implemented in Quintus Prolog and runs on a SUN workstation.

[4] Eliminating any reference to a feature essentially removes that feature from the hypothesis language. Therefore, we could view this method of screening as performing bias shifts that incrementally add restrictions to the hypothesis language.

lieu of specialization to make hypotheses more specific.[5]

A second advantage of explicit biasing assumptions is that multiple learning methods may be employed for testing the bias. This paper describes how active learning uses the procedures for testing the biasing assumptions in order to propose fruitful examples to try next. The backtracking mechanism, mentioned above, is one method for bias testing. This paper also describes how a domain theory is used to obtain biasing information to augment empirical induction.

The remainder of this paper is divided into seven sections and a summary. The next section presents the unbiased SELECT-INHIBIT method of learning positive and negative hypotheses. Then, in Section 3, we discuss the biasing assumptions and the generalization heuristics, along with original definitions of their corresponding conditions. Heuristics have been developed for three assumptions, namely, the *irrelevance*, *independence*, and *cohesion* of object features for the target concept. These three assumptions have been chosen because we consider them to be significant for any system that empirically learns concepts from examples using features. Section 4 discusses error resolution, active learning, and analytical learning. A detailed example of the system's inputs and outputs are presented in Section 5. We conclude with empirical results followed by discussions of related work and plans for future research. To avoid confusion, the remainder of this paper will refer to the bias as ''assumptions'' and the inductive hypotheses as ''hypotheses''.

2. The SELECT-INHIBIT method of concept learning

Before discussing bias, we first present the initial, unbiased induction algorithm, called ''SELECT-INHIBIT'', along with a hypothesis pair representation. The SELECT hypothesis is maintained for learning the concept. It matches (covers) all known positive examples. The INHIBIT hypothesis, which matches all known negative examples, is kept for learning the negation of the concept. The SELECT and INHIBIT hypotheses may be viewed as sets. An instance description is implied by one of these hypotheses whenever the instance is an element of that set. The SELECT and INHIBIT sets are kept disjoint. One reason for maintaining a pair of disjoint hypotheses is that the costly process of checking hypothesis consistency with instances can, under certain conditions, be avoided. (See Section 3.) Another reason is that two, nonintersecting hypotheses explicitly subdivide the world into a region believed to be positive, a region believed to be negative, and a region for which there is no reason for hypothesizing either positive or negative. Furthermore, input and overgeneralization errors are more easily discovered if the hypotheses are kept disjoint. Maintenance of a negative hypothesis in addition to a positive one can be particularly useful for active learning. When inducing disjunctive concepts using active learning, it is often helpful to focus on negative examples (Bruner *et al.* 1986). This is one of the ways in which the INHIBIT hypothesis is used.

The language for the SELECT and INHIBIT hypotheses includes two classes of lambda abstractions. The first class consists of abstractions whose body is in the form of disjunctions of conjunctions. These abstractions are in a restricted disjunctive normal form (restricted DNF) that has disjuncts consisting of the conjunction of values of features considered relevant. For example, if a small cube and a large sphere are both positive examples, the following hypothesis could be created (we omit the notation

[5] Backtracking may result in the removal of hypothesis language restrictions made earlier.

‘‘lambda(x)’’):

SELECT: { x | ((**val**(size,x,small) & **val**(shape,x,cube)) v
(**val**(size,x,large) & **val**(shape,x,sphere))) }

where ‘‘**val**’’ is the relation that assigns feature values to objects. In other words, relation **val**(r, x, z) means that z is a value of feature r for object x. Although the hypotheses begin with the disjunction of instance descriptions, generalization operators such as dropping features from the hypotheses make these DNF hypotheses more general.

The second class consists of internal disjunctive abstractions (defined in Michalski (1983)). These abstractions are conjunctions of object features with internal disjunction of values (called *internal disjunctive form*, or IDF), e.g.,

SELECT: { x | ((**val**(size,x,small) v **val**(size,x,large)) &
(**val**(shape,x,cube) v **val**(shape,x,sphere))) }.

Note that the second representation does not preserve attribute (feature) dependency information with respect to concept membership. For instance, small spheres are also included, whereas they could be negative examples.

A few more language assumptions must be mentioned. Feature values in these abstractions are assumed to be either nominal or structured (in an ISA-hierarchy). In order to simplify the generalization heuristics, no negation is currently used in either type of abstraction. The absence of negation could cause the production of clumsier hypothesis descriptions. For example, if the target concept were ‘‘not sphere’’ and the domain of possible shapes was large, expressing the negation as the disjunction of alternative shapes could be cumbersome. In fact, if not all possible shapes were known, then the correct concept could not be learned. Nevertheless, for the concepts described in this paper (Section 6), the INHIBIT hypothesis is adequate for expressing negation. The final assumption is that the codomains (set of all possible values) of the features are assumed finite.

PREDICTOR executes a repeating cycle of predictions and updates as new instances appear. One instance is accepted at a time. Every time a new instance is accepted (which may have been requested by the system), a prediction is made based on whether it matches one of the hypotheses. An instance matches a hypothesis if its description, which consists of the conjunction of its feature values, is implied by the hypothesis. There are four cases that may occur. The new instance may match SELECT only, INHIBIT only, neither, or both. When an instance matches neither hypothesis, no prediction is made. For an unmatched, positive example, an associated SELECT hypothesis is created or modified to include the example. Likewise, an INHIBIT hypothesis is created or modified for an unmatched, negative example. The system predicts positive if the SELECT hypothesis matches a new instance and negative if the INHIBIT hypothesis matches it. This prediction could be correct, requiring no action, or incorrect, requiring backtracking through the space of previous hypotheses, described in Section 4. This leaves the case where both hypotheses match the instance. Since PREDICTOR keeps the two hypotheses disjoint, this case never occurs.

PREDICTOR is similar to its predecessors. The system’s cycle of predictions and feedback is quite similar to the mode in which the incremental, concept learning program STAGGER operates (Schlimmer 1986). Maintenance of disjoint hypotheses covering the positive and negative examples can also be done by AQ (Michalski 1983). The unique contribution of PREDICTOR is its explicit bias in the form of hypothesis generating assumptions. In the next section, we discuss PREDICTOR’s

generalization heuristics that bias induction by screening potential hypotheses.

3. Generalization heuristics for biasing induction

In this section, we present the general definition of a generalization heuristic, followed by specific definitions of generalization heuristics associated with irrelevance, independence, and cohesion. First, though, we define axioms used in the system.

3.1. Axioms

To increase clarity this section uses a variable, t, that has values representing incremental time steps. If t=0 is the first time step in incremental induction, when no instances have been accepted, then each inductive time step, t = 1, t = 2,..., is associated with the acceptance of one new training instance, *i(t)*. A training instance consists of an instance description and the information as to whether it is a positive or negative example of the target concept. We let $I_+(t)$ be the set of instances known to be positive at time t, and $I_-(t)$ be the set of instances known to be negative at time t. Then the following axioms hold, where t is universally quantified.

[A1] $I_+(0) = I_-(0) = \emptyset$.
[A2] $(I_+(t+1) = I_+(t) \cup \{i(t)\})$ if $i(t)$ *is positive, else* $(I_+(t+1) = I_+(t))$.
[A3] $(I_-(t+1) = I_-(t) \cup \{i(t)\})$ if $i(t)$ *is negative, else* $(I_-(t+1) = I_-(t))$.
[A4] $SEL(0) = INH(0) = \emptyset$. (SEL stands for SELECT, INH for INHIBIT)
[A5] $(I_+(t) \subseteq SEL(t))$ and $(I_-(t) \subseteq INH(t))$.

These axioms are assumed for the definitions in the remainder of this section.

3.2. General definitions

In this subsection we define a *training scenario*. We also formalize the notion of a generalization heuristic.

A training scenario is a quintuple < I+, I-, SEL, INH, i > (where each quintuple element is a function over t) that satisfies [A1]-[A5]. In the following, we are dealing with a training scenario.

A generalization heuristic, G, consists of a condition, $COND_G$, and an operator, OP_G. A generalization heuristic states that whenever its condition holds, the corresponding generalization operator may be applied. If OP_G is the generalization operator associated with heuristic G, then OP_G, applied at time t, does the following:

$$OP_G(SEL(t), INH(t), i(t)) = (SEL(t+1), INH(t+1)).$$

In addition to determining the form of the hypotheses at time t + 1, the generalization operator, OP_G, also influences the form of the hypotheses at time > (t + 1). Note that given a sequence of training instances and operator applications, the entire training scenario is determined.

Each of the heuristic conditions is a procedural method for testing whether a biasing assumption holds. In addition to using these conditions in the generalization heuristics, PREDICTOR also uses them for active learning and error resolution, described in Section 4. Each generalization operator used in PREDICTOR

corresponds to one of the rules described in Michalski (1983) or Michalski and Stepp (1983). The default generalization operator of the system is *add-feature-values*. Whenever a new instance is accepted, this operator adds the instance description, generalized using the current, biasing assumptions, to the appropriate hypothesis (as a new disjunct) if the hypotheses are in DNF. If the hypotheses are in IDF, feature values are generalized if necessary and added to the internal disjuncts. Further generalization is optional and is performed by generalization heuristics. These heuristics can alter both the bias and the current hypotheses. In order to simplify the directed backtracking (explained in Section 4), PREDICTOR fires at most one of these generalization heuristics for each new instance. If multiple generalization heuristic conditions are satisfied and no metaheuristic (or bias) exists for selecting between them, then the choice between their operators is arbitrary.

Below, we discuss specific examples of generalization heuristics.

3.3. The irrelevance generalization heuristic

This subsection presents a definition of *absolute irrelevance*, followed by the condition and operator associated with the *irrelevance generalization heuristic*. Formal definitions of the irrelevance and independence heuristics may be found in the appendix of this paper.

We have developed definitions of *absolute irrelevance*, *absolute independence*, and *absolute cohesion* that clarify the meaning of irrelevance, independence, and cohesion within a nonincremental learning context. These absolute conditions are idealized conditions for which the universe of objects, the universe of features, the universe of feature values, as well as the classes of all objects, are known prior to the outset of learning. The absolute conditions are the biasing assumptions that PREDICTOR tries to confirm or refute in an incremental setting. The generalization heuristics, which are designed for incremental learning, are based on the definitions of the absolute conditions. The heuristics try to determine whether the absolute conditions hold based on the instances seen so far. If not all instances have been seen, their conclusions may be wrong.

The definition of absolute irrelevance states that feature r is irrelevant to learning concept C if and only if any two objects that have identical values for every feature except, perhaps, feature r, both behave identically with respect to target concept membership regardless of the value of r.

It is now possible to define a condition, $COND_{IRR(r,C,t)}$, that is associated with the irrelevance heuristic and is based on the above definition. The generalization heuristic is defined for the subset of the universe of objects which consists of those objects seen so far. The subscript *IRR(r,C,t)* means that feature r is considered irrelevant to the target concept C at time t. Let $\{r_1, \ldots, r_n\}$ be the set of features considered relevant at time t. We define an *instance description* to be

$$(\mathbf{val}(r_1,x,v_1) \,\&\ldots\&\, \mathbf{val}(r_n,x,v_n)),$$

where v_i is a value in the codomain of r_i for $1 \le i \le n$. Then $COND_{IRR(r,C,t)}$ states that if you take any instance description of an element of the SELECT (resp. INHIBIT) hypothesis, and you change the value of r to *any* other element of its codomain, you still describe an element (if such an object exists) of the SELECT (resp. INHIBIT)

hypothesis.[6]

The operator $OP_{IRR(r,C,t)}$, which is associated with the irrelevance generalization heuristic, is drop-feature(r). The effect of operator application is that feature r is omitted from both the SELECT and INHIBIT hypotheses (current and future).

The irrelevance generalization heuristic, $G_{IRR(r,C,t)}$, states that whenever the irrelevance condition $COND_{IRR(r,C,t)}$ holds, the irrelevance generalization operator $OP_{IRR(r,C,t)}$ may be applied to the irrelevant feature r.

The irrelevance condition is not as strict as absolute irrelevance. This is why $G_{IRR(r,C,t)}$ generalizes. To be certain that absolute irrelevance holds, objects of every combination of values for all features (if such objects exist) would have to be included in either SEL(t), INH(t), or *i(t)*. Since the irrelevance operator generalizes future hypotheses, prediction errors may occur.

As an example of $G_{IRR(r,C,t)}$, assume the codomain of feature ''material'' is {copper, brass}, that of ''size'' is {small, medium, large} and that of ''shape'' is {cylinder, brick, sphere}. Furthermore, suppose the current SELECT hypothesis (at time t) is the following:

SELECT(t):
{ x | ((**val**(size,x,small) & **val**(material,x,copper) & **val**(shape$x$,cylinder))
v (**val**(size,x,small) & **val**(material,x,brass) & **val**(shape,x,cylinder))
v (**val**(size,x,large) & **val**(material,x,copper) & **val**(shape,x,brick)))}

and the next instance has the description, ''(**val**(size, *i(t)*, large) & **val**(material, *i(t)*, brass) & **val**(shape, *i(t)*, brick))''. If ''material'' is ignored, there are two different classes of objects described by the SELECT hypothesis combined with the new instance. The first class includes small cylinders. The second includes large bricks. Furthermore, every small cylinder comes in all materials and every large brick comes in all materials. Therefore, $COND_{IRR(r,C,t)}$ is satisfied, so the system may apply $OP_{IRR(r,C,t)}$ to infer

SELECT(t+1): { x | ((**val**(size,x,small) & **val**(shape,x,cylinder)) v
(**val**(size,x,large) & **val**(shape,x,brick)))}.

''Material'' will henceforth be omitted from SELECT and INHIBIT. Note that since this is a generalization over as yet unseen instances, it could be wrong. For instance, it may turn out that all copper spheres are positive whereas all brass spheres are negative, in which case ''material'' is relevant. This error will be detected by subsequent inconsistency and resolved by backtracking.

In addition to omitting the irrelevant feature from future pairs of hypotheses, the fact that the feature is irrelevant to learning the target concept is added to the database of facts. This statement of irrelevance of a feature can be considered a meta-level hypothesis used to generate inductive hypotheses. If a subsequent prediction error occurs, this metal-level hypothesis about irrelevance can be retested given the new (and larger) set of examples known at that time. This technique of error resolution is described in the Section 4. First, though, we present two more generalization heuristics.

3.4. The cohesion generalization heuristic

A structured feature is one for which there exists a tree of feature values that increase in generality as the tree is ascended from the leaves to the root. Each tree

[6] When internal disjunction is used, this becomes the truth-preserving rule of Michalski (1983).

node may be viewed as a set, or class. Thus, the tree describes a partial ordering of terms, as in Figure 1. The purpose of the cohesion heuristic is to decide when to climb the tree. We next define *absolute cohesion* and the *cohesion generalization heuristic*.

The definition of absolute cohesion states that the subclasses of feature value v are cohesive with respect to target concept membership if and only if any two objects that have identical values for all features except r, and that both have value v (or some subclass) for feature r, behave identically with respect to target concept membership.

We next define cohesion in an incremental setting. The cohesion of class v with respect to the target concept means that subclasses $\{a_1, \ldots, a_n\}$ of class v in the generalization tree of descriptive terms appear to behave as a unit with respect to target concept membership. In other words, cohesion holds if for each description of an object included in the SELECT (resp. INHIBIT) hypothesis, if the value of feature r is changed from a_i to a_j ($1 \leq i, j \leq n$) then any object satisfying this new description is also a member of the SELECT (resp. INHIBIT) hypothesis. For instance, if {cube,brick} is the set of subclasses of ''prism'', and ''small cube'' describes certain elements of the SELECT hypothesis, then ''small brick'' must also describe elements of SELECT in order to satisfy the condition for cohesion of prismatic-shaped objects.

The cohesion heuristic operator is *climb-generalization-tree(r,v)*. This operator replaces the names of all subclasses of feature value v in the hypotheses with v. In the example above, the generalization tree would be climbed to ''small prism''. It is important to note that when a generalization tree exists, irrelevance is a special case of cohesion and drop-feature(r) is a special case of the generalization operator climb-generalization-tree(r,v).

The next subsection presents the third and final generalization heuristic used by PREDICTOR.

3.5. The independence generalization heuristic

This section presents the definition of *absolute independence* and the condition and operator associated with the *independence generalization heuristic*.[7]

In order to simplify the definition of absolute independence, we define the independence of two features. The definition of absolute independence states that features r_1 and r_2 are independent with respect to concept C if and only if whenever you take the descriptions of two objects for which the target concept holds and you switch the values of r_1 (or of r_2) in the two descriptions, you still describe objects for which the target concept holds. The current version of PREDICTOR addresses only the independence of *all* features. If the absolute independence of *all* features holds, then one feature is sufficient for learning the target concept.

The condition associated with the generalization heuristic for independence, $COND_{IND(R,C,t)}$, is based on the definition of absolute independence. If we let $R(t)$ be the set of features considered relevant, then the subscript $IND(R,C,t)$ means that all elements of $R(t)$ are considered independent with respect to C at time t. Let $V_{SEL}(t)$ be the set of all feature values (of all features) currently found on the SELECT hypothesis; and let $V_{INH}(t)$ be defined similarly. Then if *every* instance description formed from feature values in $V_{SEL}(t)$ describes an element of SELECT, and *every* instance description with feature values combined from $V_{INH}(t)$ describes an element

[7] The term ''dependent'' is used here to refer to attribute value dependence with respect to target concept membership. The term does not denote statistical correlations between the attribute values themselves.

of INHIBIT (assuming these objects exist), $COND_{IND(R,C,t)}$ holds. The check for independence may be performed by taking the Cartesian product of values in $V_{SEL}(t)$ or $V_{INH}(t)$. In other words, independence holds for the following hypothesis:

SELECT(t): { x | ((**val**(material,x,brass) & **val**(shape,x,cylinder)) v
(**val**(material,x,copper) & **val**(shape,x,cylinder)) v
(**val**(material,x,brass) & **val**(shape,x,brick)) v
(**val**(material,x,copper) & **val**(shape,x,brick)))}.

Independence *cannot* hold if a combination of feature values distinguishes target concept membership as in the following:

SELECT(t): { x | ((**val**(material,x,copper) & **val**(shape,x,cylinder)) v
(**val**(material,x,brass) & **val**(shape,x,brick)))}

INHIBIT(t): { x | ((**val**(material,x,brass) & **val**(shape,x,cylinder)) v
(**val**(material,x,copper) & **val**(shape,x,brick)))}.

Unlike the two previously described generalization heuristics, the independence heuristic alters the logical form of the hypotheses.[8] Its associated operator, *convert-form*, transforms the current and future hypotheses from DNF to IDF. Although this is a truth-preserving transformation for the current hypotheses if the condition for independence is met, it is not in general truth preserving. The reason for this is that the operator also affects future hypotheses because the bias exists that absolute independence holds. If future hypotheses are in IDF, then a greater degree of generalization will occur.

In practice, PREDICTOR applies this heuristic to only one hypothesis at a time

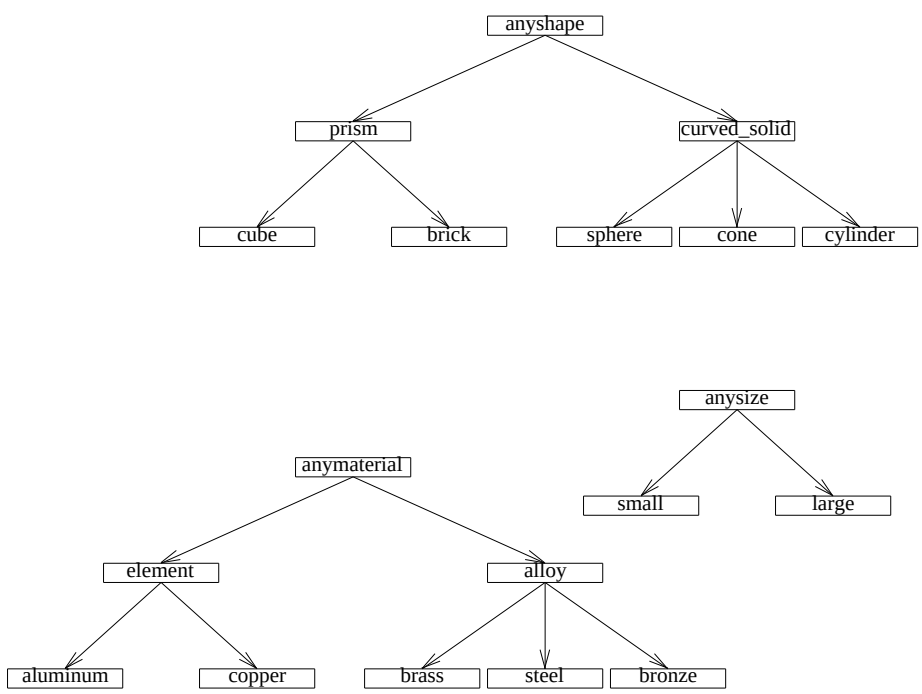

FIG. 1. The generalization tree of descriptive terms.

[8] It is wise to apply the heuristic only after a couple of examples have been seen.

in order to reduce the cost of testing the condition. However, we have still found the independence heuristic to be more expensive and less effective in practice than the other two heuristics. The program of Subramanian and Feigenbaum takes advantage of the independence (called ''factorability'') of a subset of the features rather than requiring all features to be independent as PREDICTOR does (Subramanian and Feigenbaum 1986). Future versions of PREDICTOR should possess a heuristic detecting partial independence. Preliminary tests indicate that PREDICTOR performs well (though the results are not as general) using only cohesion and irrelevance, even when dependencies exist, because cohesion and irrelevance are intra-feature rather than inter-feature generalizations. (See Section 6.) The next subsection describes further work with the cohesion and irrelevance heuristics.

3.6. Cautious and uncautious heuristic modes

Different problem-solving situations may require different degrees of cautiousness for generalizing hypotheses. For example, in our scenario of a robot that fixtures parts in an automated factory, suppose there is another robot that inspects parts. The inspection robot has two bins: one for good (i.e., fixturable) parts, which get transported to the robot that places the parts into the milling machine fixture, and another bin for bad (i.e., not fixturable) parts, which are thrown out. The inspection robot gets raw parts and puts those parts that it believes to be good or bad into their proper bins. We assume it gets feedback very quickly if a part is placed in the wrong bin. If the inspection robot has no reasonable basis for selecting one of the bins (no prediction), it must run a very time-consuming series of tests that yields further information helpful for deciding. One can now imagine two extremes of penalties for overgeneralization (that can result in prediction errors). In the first case, the parts are expensive and breakable. There is a high cost associated with incorrectly predicting that either a part is good *or* that it is bad. If the robot incorrectly predicts the part is good then it could be broken. On the other hand, if the robot incorrectly predicts the part is bad then a perfectly good part will be discarded. Cautious generalization is appropriate, therefore, and when the rationale is weak for predicting good or bad, the robot should run the extra tests (assuming they do not risk breakage). On the other hand, for the second case, suppose the parts are inexpensive and not fragile, yet production time is critical. In this situation, there is a small penalty for overgeneralization and the inspection robot could minimize production time by generalizing more and allowing more prediction errors. Increased generalization of the two hypotheses about good and bad parts would avoid the extra tests and cause learning to proceed more rapidly.

PREDICTOR is capable of selecting the cautiousness of its learning by choosing the strictness of the preconditions for firing its generalization operators. The irrelevance heuristic, as well as the cohesion heuristic, may be used in one of three modes. In the first mode, the full heuristic condition is fired, which is described above, and tests each instance description of the hypothesis (which is a hypothesis disjunct if hypotheses are in DNF). This full heuristic condition checks to see if *all* values of the feature being checked for irrelevance or cohesion can be substituted for the original value in the instance description without describing an object of the wrong class. This mode is the most cautious. It results in the fewest subsequent prediction errors. The second mode, like the first mode, checks each instance description of the hypothesis. However, in the second mode only a subset of the values of the feature in question is tested rather than all values. The third mode, which applies only to DNF hypotheses, ignores the remaining instance description (values of features

other than the one being tested). For example, the third mode of the irrelevance heuristic states that if *any* two disjuncts (which may differ in their values of features other than *f*) of the same hypothesis have different values for feature *f*, then feature *f* is considered irrelevant and dropped from all disjuncts of both hypotheses. This is the least cautious of the modes (called the ''two-disjunct'' mode). Nevertheless, this is quite similar to heuristics that have been used in other inductive systems (e.g., Winston (1975); Mitchell (1978)). The preference between these modes, as well as the number of children to be tested if the second mode is chosen, is decided in PREDICTOR by metaheuristics. These metaheuristics make the selection based on a user-input penalty for prediction errors. The metaheuristics also consider the number of values to be tested when deciding if the first mode is appropriate.

Heuristics used in the first and most cautious mode offer a special benefit. They are consistency-preserving. That is, they are guaranteed not to violate the disjointedness (consistency) of SELECT and INHIBIT (though they do not guarantee future predictions will be correct).[9] Since axiom [A5] holds, if consistency between these two hypotheses is maintained, then it is guaranteed that the SELECT hypothesis will match none of the known negative instances and the INHIBIT hypothesis will match no known positive instances. Consistency-preserving generalization heuristics guarantee that consistency will be maintained whenever they are used for generalizing a hypothesis to include a new instance that matches neither hypothesis. Therefore, there is no need to reconsult previous instances or even to check the disjointness of SELECT and INHIBIT immediately after generalizing with consistency-preserving heuristics. When heuristics that are not consistency-preserving are used, such as the second and third modes of the cohesion heuristic, a language-specific hypothesis consistency check is made by PREDICTOR.

Avoidance of a consistency check is a desirable property because of the computational savings. Like PREDICTOR, the Candidate Elimination Algorithm of Mitchell (1978) also avoids rechecking instances and maintains consistency. However, the Candidate Elimination Algorithm does not combine this advantage with the use of explicit heuristics for biasing the generalization.

This section has concentrated on bias determined by the generalization heuristics. The next section presents other learning strategies of the system that test the same biasing assumptions as the generalization heuristic conditions.

4. Bias and active learning, analytical learning, error resolution

We now discuss PREDICTOR's three methods, besides generalization heuristics, for using the assumptions that bias the learning. The first two, active learning and analytical learning, examine the bias prior to generalizing. The third, error resolution, may re-examine the biasing assumptions after generalization has occurred.

4.1. Active Learning

Although the system has a choice of which mode to use for the generalization heuristics, the strictest (most cautious) mode of checking the conditions is very useful for formulating system requests for future examples in order to increase the speed of

[9] See the appendices for formal proofs that the full irrelevance and independence heuristics are consistency-preserving. We have not included the proof associated with the cohesion heuristic because is is quite similar to the proof for irrelevance.

induction. Those examples which have not yet been seen, but which would satisfy one of the full generalization heuristic conditions if they were of the desired class, are proposed as examples to try next. For example, if a small cube and a large brick are positive examples, PREDICTOR suggests trying a small brick and a large cube. The motivation for this suggestion is that if the proposed examples are both positive, then convert-form can be immediately applied. We have run tests and demonstrated that using the conditions of the heuristics to guide example selection can significantly improve learning. The effectiveness of this approach, however, becomes more limited as the number of possible values for each feature becomes large. PREDICTOR only enables active experimentation when the cost (based on the number of feature values) is below a prespecified level.

There exist a number of variations of active learning that are being investigated by researchers in machine learning. Some systems devise and perform experiments in order to detect reasoning errors (e.g., Carbonell and Gil (1987)). Others use active learning for selecting the next instance(s) in an incremental setting (Sammut and Banerji 1986; Porter and Kibler 1986). The role of active learning in PREDICTOR is the latter. Just as bias may occur prior to or after forming alternative hypotheses, active learners that select a next instance may or may not form the potential hypotheses before selecting desired instances. For example, the program of Sammut and Banerji (1986) forms potential generalizations and then suggests instances to try next whereas Porter and Kibler's (1986) program, like PREDICTOR, generates examples prior to generalizing.

A learner that can take an active role in proposing examples has the potential to significantly reduce the time required to learn a concept. Rather than wading through a large number of instances with low information content, active learners can process high information examples during the early stages of learning.

4.2. Analytical Learning

In addition to active learning, PREDICTOR also employs a variant of a technique called *explanation-based generalization* (EBG) (Mitchell *et al.* 1986), which is a special case of *explanation-based learning* (EBL) (DeJong and Mooney 1986). Whereas empirical learning methods (which are inductive) detect similarities and differences between numerous examples in order to refine hypotheses, explanation-based learning (which is deductive) uses a domain theory to learn from very few instances. EBG is able to generalize from a single instance by using the domain theory to prove that the instance is a member of the target concept. The proof is then generalized to find the weakest precondition under which the instance satisfies the concept. To perform EBG, a domain theory needs to be augmented with the additional information specifying which instance features are considered operational (easily measured or detected). The hypothesis resulting from explanation-based learning will include a subset of the operational features.

There are three different ways in which systems have combined empirical and analytical methods. Some programs, for example Pazzani's (1988), use empirical learning to augment and repair problems with EBL, such as an incorrect or incomplete theory. These systems focus on EBL. For other programs, such as that of Carpineto (1988), the focus is on the empirical learning. EBL provides a justified focus on the relevant features of examples. The third way in which empirical and analytical methods are combined is by giving them equal priority. In Sims (1987), for example, an agenda calls both empirical and analytical tasks as needed.

The focus in our system is on empirical learning and, in particular, on the bias. Only partial information is used from EBL. Similarly to Carpineto (1988), we are interested in gleaning feature relevance information using the domain theory. However, we also use the domain theory to decide feature cohesion and independence information. Unlike previous EBL programs, the parts of the domain theory relevant to both the target concept and the negation of the target concept are explored. This is not only consistent with the use of both SELECT and INHIBIT hypotheses, but it is also considered important if the domain theory is incomplete. Using a theory about the ability of parts to be fixtured, one may, for example, be able to prove that the shape of an object causes its inability to be fixtured but be unable to prove that its shape causes it to succeed in being fixtured. The feature ''shape'', in this case, should not be dropped from the hypotheses for learning about what can be fixtured, since it is relevant.

The Prolog implementation of EBG that is used in PREDICTOR is a variation of Hirsh's program described in Kedar-Cabelli and McCarty (1987). Modifications are as mentioned above. PREDICTOR proves that an instance can be distinguished with respect to target concept membership. A list of relevant, operational features are extracted by the EBG process. Also extracted is the appropriate height within the generalization tree of descriptive terms. For example, suppose the system is learning the concept ''fixturable''. The initial, operational features provided are ''size'', ''color'', ''material'', and ''shape'' of objects and all features are operational. Suppose the first instance, block1, is a large, blue, steel sphere. If the system can derive (deductively) from the domain theory that an object's shape is related to its ability to be fixtured, and that curved-solids are not fixturable (and does not relate color or material to whether or not an object is fixturable), then the results of PREDICTOR's version of EBG might be ''[**val**(size,block1,large), **val**(shape,block1, curved-solid)]'', where ''curved-solid'' is the parent of ''sphere'' in the generalization tree of terms. Generalization (variabilizing the name of the instance) may either be done by EBG or by the empirical learning. Since this step has already been incorporated within our empirical learning, we omit the generalization step of EBG. If the feedback states that block1 is negative, the INHIBIT hypothesis would now be initialized to

INHIBIT: { *x* | (**val**(size,*x*,large) & **val**(shape,*x*,curved-solid)) }.

The induction techniques described in this paper can then be used to modify the hypotheses based on empirical evidence. Unless the results of EBG are subsequently blamed for an error, future SELECT and INHIBIT hypotheses continue to use the EBG information about the irrelevance of all features except ''size'' and ''shape'' as well as the generalization to the term ''curved-solid''. Independence information will be obtained from the domain theory as a side effect of using our EBG version because if all features are independent, then a single feature suffices for describing the target concept. Therefore, the resulting EBG list (if the domain theory is trustworthy) will contain a single feature. Although most implementations use EBG on one instance only, PREDICTOR may use it on multiple instances. The reason for this is that multiple instances may be required in order to get cohesion information for all feature values. PREDICTOR uses the domain theory to get cohesion information for every instance for which this information has not already been learned. For example, suppose in the above example that the second instance is a small, blue, copper brick that is positive. Then a second invocation of EBG might result in

SELECT: { *x* | (**val**(size,*x*,small) & **val**(shape,*x*,prism)) }.

Realistically, most domain theories are not perfect. Imperfect theories can cause

prediction errors. Currently, PREDICTOR leaves the decision to the user whether to blame the domain theory and reject the EBG results if an incorrect prediction occurs. Future versions of the system should include more sophisticated solutions.

4.3. Error resolution

When PREDICTOR makes a prediction error, one of two problems is the cause of this error. Either the instances are inconsistent, i.e., instances with identical attributes are classified differently (one positive, one negative), or one or both of the hypotheses is overly general. When an error occurs, PREDICTOR first checks if the error is due to instance inconsistency. If not, it is assumed that the error is caused by overgeneralization of the hypothesis which matches the newest instance, for which an erroneous prediction was made. We discuss PREDICTOR's methods for handling inconsistent instances first, then its methods for handling overgeneralization.

Inconsistent classification of instances can be caused by noise or language insufficiency. PREDICTOR asks the user whether noise is the cause of the inconsistency. If not, a request from the user for new object attributes is PREDICTOR's remedy for language insufficiency. Note that this is an example of shifting the bias by augmenting the hypothesis language. If the user states that noise has caused the

```
PROCEDURE learn;
 IF more unprocessed instances THEN
   accept a new instance i;
   IF i matches SELECT or INHIBIT [will not match both] THEN
     IF correct_prediction THEN learn
     ELSE [wrong_prediction]
       IF inconsistent instances THEN
         IF due to noise THEN remove noisy instances
         ELSE [not noise] add a new feature ENDIF
         learn
       ELSE [overgeneralization of SELECT or INHIBIT]
         backtrack until consistency achieved;
         [fail this call to ''learn'' and select a
         different generalization heuristic
         on a previous recursive call]
         learn ENDIF
     ENDIF
   ELSE [No match]
     IF i is positive
       THEN fire a generalization heuristic to generalize SELECT
     ELSE [i negative]
       fire a generalization heuristic to generalize INHIBIT ENDIF
     learn
   ENDIF
 ENDIF
END
```

FIG. 2. SELECT-INHIBIT algorithm for concept learning.

inconsistency, then PREDICTOR discards both the newly-introduced instance and the previous one which it contradicts (for which backtracking may be required). This solution is inadequate for more than a minimal amount of noise. For quite noisy data, we plan to eventually add statistical versions of the generalization heuristics.

If PREDICTOR decides that overgeneralization has caused the current prediction error, then an error resolution method called ''directed backtracking'' is employed. Directed backtracking consists of two steps: locating the error and backtracking through the space of hypotheses. During the error locating phase, each of the currently-held biasing assumptions is retested. The full conditions for cohesion (with irrelevance as a special case) and independence are retested, given the new set of instances. This testing procedure is focused, however. Only those generalization heuristics that fired, and are applicable to the current, contradictory instance, are retested. The error detection strategy for cohesion perturbs each of the feature values of the new instance in its instance description. (See Porter and Kibler (1986) for a discussion of perturbation.) If the value of feature *f* for the new instance is altered and an instance of the opposite class is described, then cohesion was incorrectly assumed to hold for *f*. This test may require calls to an oracle (e.g., the user) to request the class of system-formulated examples if the instances seen so far are insufficient to implicate an assumption.[10] The error detecting strategy for independence also focuses on the feature values of the new instance as well as the feature values of previous instances. It rechecks independence given the new feature values.

Shapiro has also developed an algorithm, called *contradiction backtracing*, for isolating a faulty hypothesis (Shapiro 1981). (PREDICTOR's biasing assumptions can be considered meta-level hypotheses.) Shapiro's algorithm uses resolution, followed by a trace of the resolution steps with substitutions performed, in or order to locate an erroneous hypothesis. An oracle is queried to learn the truth of ground sentences. Although both techniques query an oracle and locate faulty hypotheses, they also differ. Contradiction backtracing uses resolution with several hypotheses to isolate a single, bad hypothesis. PREDICTOR's method of error resolution, on the other hand, reevaluates each relevant hypothesis separately with the new data to determine if the hypothesis still holds. Furthermore, contradiction backtracing assumes the user will make the appropriate edit to eliminate the error. PREDICTOR automatically remedies its errors by backtracking to the time at which the first of its erroneous assumptions was invoked.

The seconds stage for PREDICTOR's error resolution is to backtrack through the space of hypotheses. If an assumption is proved wrong, the assumption is retracted, the corresponding generalization heuristic is suppressed, and the system backtracks to the time when this heuristic fired and reprocesses the instances from that point. (Recall that at each time step, a new instance is accepted and a generalization heuristic may fire.) When instances are reprocessed, a different sequence of generalization heuristics can fire than the original sequence. The sequence variation begins when an alternative heuristic to the suppressed heuristic is fired.

PREDICTOR uses the same error resolution methods as described above to reestablish hypothesis disjointedness. However, unlike the case of a wrong prediction, restoring hypothesis consistency may require testing more than one instance.

---

[10] We are currently modifying the directed backtracking for situations where an oracle is unavailable.

Figure 2 summarizes the manner in which PREDICTOR's error resolution methods fit into its learning algorithm. The next section presents a partial trace of the learning algorithm in action.

5. Partial program trace

To better explain the operation of PREDICTOR, we present a partial trace of the program in execution. User input, in the form of routine calls and responses to system queries, are in italics. Comments are in braces. It is assumed that input is from a file of randomly-generated examples and that the generalization tree of Figure 1 is used.[11]

initialize(fixturable). {''fixturable'' is the target concept}
What are all object features? [*size,material,shape*]
What is the penalty level for prediction errors? *5* {The range of possibilities is 1 to 10.}
The following is the order in which heuristics will be tried: [*cohesion-heuristic(material),cohesion-heuristic(size),cohesion-heuristic(shape),independence-heuristic*]
{Generalization heuristics are initially ordered by the system according to error penalty and cost of application. The order in this example, though, was chosen in order to best illustrate the heuristics.}
Do you wish to change this order? *n*
learn(file, no-ebg, fixturable).
{This calls the learning program. Parameter ''file'' specifies the input mode,''no-ebg'' selects the option of not using EBG}
Do you want to pause? *n*
{System inputs from file the block ''sbc'', which is a small, bronze cube. Note that the block name and feature information could alternatively be entered directly rather than from a file if that were the specified input mode.}
No prediction for sbc.
Is sbc a negative or positive example of fixturable? *positive*
Prediction-feedback comparison yields: guess.
Added a new instance description to the positive hypothesis:
select(X, fixturable) ←
val(size,X,small),
val(material,X,bronze),
val(shape,X,cube)
{The target concept is made a parameter of SELECT in the implementation so that the system can eventually be extended to learn multiple concepts.}
Condition for cohesion-heuristic(material) unsatisfied.
Condition for cohesion-heuristic(size) unsatisfied.
Condition for cohesion-heuristic(shape) unsatisfied.
Condition for independence-heuristic(select) unsatisfied. {This heuristic is implemented to check and convert each hypothesis separately.}
A good set of examples to try is the following: {Two examples are suggested by PREDICTOR.}
[[**val**(size,X,small),**val**(material,X,brass),**val**(shape,X,cube)],
[**val**(size,X,small),**val**(material,X,steel),**val**(shape,X,cube)]].
{If some of the recommended examples do not exist, but the rest are positive, the user can input the nonexistent examples as though they were positive.}
Do you wish the examples to be input automatically? *y*
{The system now finds the first examples in the file input which match these descriptions and inputs them immediately. They are positive.}

---

[11] Minor changes have been made for the purpose of readability. Furthermore, for the sake of brevity, the directed backtracking mechanism is not shown here.

.
.
.
Successfully applied cohesion-heuristic(material).
New hypothesis is:
select(X, fixturable) ←
val(size,X,small),
val(material,X,alloy),
val(shape,X,cube)
Do you want to pause? *n*
A good set of examples to try is the following:
[[**val**(size,X,small),**val**(material,X,element),**val**(shape,X,cube)]].
Do you wish the examples to be input automatically? *y*
{The system now inputs a small, copper cube since it is the first instance in the randomly-generated data set that fits the above description. It is positive. A small, aluminum cube is then suggested and, since it is input and is positive, the generalization to element is made.}
.
.
.
Successfully applied cohesion-heuristic(material). {Since the parent of ''alloy'' and ''element'' is ''anymaterial'', the feature ''material'' is dropped}
New hypothesis is:
select(X,fixturable) ←
val(size,X,small),
val(shape,X,cube)
Do you want to pause? *n*
A good set of examples to try is the following:
[[**val**(size,X,large),**val**(shape,X,cube)]].
Do you wish the examples to be input automatically? *y*
{A large, brass cube is now input since it is the first instance to satisfy the description. It is positive.}
.
.
.
Successfully applied cohesion-heuristic(size).
New hypothesis is:
select(X,fixturable) ←
val(shape,X,cube)
{The system now recommends inputting a brick in order to generalize ''shape''. The user rejects this suggestion. Instead, the user lets the system input the next instance in the randomly-generated sequence. This instance happens to be ''lsc'', a large, steel cube. Block ''lsc'' is input now.}
Predicts lsc is positive.
Is lsc a negative or positive example of fixturable? *positive*
Prediction-feedback comparison yields: correct-prediction
.
.
.

The above scenario demonstrates the cycle in which an instance is input, a prediction is made, feedback is accepted from the user, then the two are compared. The comparison result could be ''guess'', ''correct-prediction'' or ''wrong-prediction''. Hypotheses are updated accordingly. Although in the above example, we only update the SELECT hypothesis, the INHIBIT hypothesis is updated in an identical manner. Furthermore, when the generalization tree is climbed for SELECT, it is simultaneously

climbed for INHIBIT.

6. Results

We have run empirical tests of PREDICTOR's performance in order to gain deeper insight into the effectiveness of its approach to bias. The testing has been done to satisfy three different goals. The first goal is to compare the effectiveness of the individual heuristics and modes. The second goal is to determine the effectiveness of the directed backtracking. Our third goal is to compare explicit and implicit bias. For all tests, we have assumed no noise, no new features are introduced, and the explanation-based learning capability is not used.

Ten randomly-generated data sets of 50 instances each are used for all experiments. The feature hierarchies are shown in Figure 1. The concepts that are learned are descriptions of raw, manufacturing parts that are fixturable in a particular milling machine. Three concepts are learned. For concept A, prisms are fixturable and curved-solids are not. Concept A is the easiest to learn because independence of features with respect to target concept membership holds, as well as cohesion for all features. Prisms and cylinders are fixturable objects for concept B. Parts of any other shape will not hold in the fixture. For concept C, small bricks and large cubes can be fixtured, but objects with any other combination of features fail because the fixture has a limited range of adjustability. Concept C is the most difficult to learn because it contains a dependency with respect to target concept membership and cohesion fails to hold for values of two of the features. ''Material'' is irrelevant to determining fixturability for all three concepts.

The graphs of PREDICTOR's performance are in Figures 3 through 7. Percentages are based on instances seen so far. When no prediction is made, this is not considered correct but is included in the total number of opportunities to predict when calculating percentages. In order to present a more detailed view of PREDICTOR's errors and their resolution, instead of showing a graph of percent wrong predictions, we specify the average number of predictions. If no errors occur, the system makes 50 predictions corresponding to the 50 instances. Any additional predictions are associated with backtracking.

Results are averaged over the number of runs shown in each graph. We shall now explain why the graphs differ in the number of runs even though there are 10 unique data sets. The number of runs exceeds 10 on some graphs of PREDICTOR's performance because rather than using metaheuristics for ordering the application of generalization heuristics, we average over two different orders. The reason some curves are associated with fewer than the expected number of runs (e.g., Figure 6) is that we put a computational time limit on the learning (which is 500 predictions). Any run exceeding this limit is aborted and not included in the curves. A small number of runs indicates difficulty learning the concept, i.e., an excessive amount of backtracking has occurred.

We first describe our comparison of the heuristics and their modes. Figures 3, 4, and 5 show the performance of the first and most cautious mode of the cohesion (with irrelevance as special case) and independence heuristics. Figure 6 shows the performance of the third and least cautious mode of the cohesion heuristic, which we call the ''two-disjunct'' heuristic. For this portion of the testing (Figures 3 through 6), chronological backtracking is used rather than directed backtracking. In Figure 2, it is shown that backtracking is implemented by unwinding the recursive calls (where each

call is associated with a new instance). Directed backtracking proceeds backwards to a specific time and instance. Instances are reprocessed from that point and a different sequence of generalization heuristics is tried. The chronological backtracking, on the other hand, proceeds one level at a time and is similar to the depth-first search backtracking described in Nilsson (1980). In addition to using chronological backtracking, no hypothesis consistency checks are made for the runs shown in Figures 3 through 6. (None are needed for the cautious modes.) The reason for using chronological backtracking and omitting consistency checks is that we would like to first compare the heuristics by themselves, using primitive error resolution methods, then we wish to study the effect of adding more sophisticated error resolution techniques. Active learning is used with every run of PREDICTOR. Prior testing has shown that cautious heuristics rarely fire without active learning.

The best overall performance of Figures 3 through 6 seems to be associated with the full cohesion heuristic. In fact, the cohesion heuristic alone outperforms its combination with independence. One problem with the independence heuristic occurs when it is combined with cohesion. If early instances are deceptive, and the independence heuristic fires, the cohesion heuristic is easily satisfied and incorrect generalizations proceed rapidly. If the independence heuristic is used by itself, it does not generalize sufficiently. We are currently investigating improvements to the independence heuristic as discussed in Section 3.

The reason the cautious mode of the cohesion heuristic far outperforms the uncautious mode (Figures 3, 6) on concepts B and C is that the test for cohesion in the latter is insufficiently rigorous. In Figure 7, the rigorous directed backtracking mechanism as well as hypothesis consistency checks are combined with the least cautious (two-disjunct) heuristic mode. We do not show the graph for the full cohesion heuristic with directed backtracking because it is quite similar to Figure 3. The major difference for the cohesion heuristic when the directed backtracking replaces chronological backtracking is that the average number of predictions is reduced from 86 to 55. With directed backtracking added, the difference between the errors of the full cohesion and two-disjunct heuristics is reduced. The average number of errors for full cohesion on concept C is one and for two-disjunct on concept C is three (not shown in graphs). Nonetheless, although the difference in number of errors between two-disjunct and full cohesion is not large, for some applications this difference may be critical. According to our testing so far, the two-disjunct heuristic combined with directed backtracking seems to produce the highest and most consistent success rate of any combination. The small magnitude of the difference between the three curves in Figure 7 is attributable to PREDICTOR's ability to successfully handle different data characteristics using flexible, explicit bias.

Our final goal is to contrast PREDICTOR's explicit bias with the implicit biasing method of another system. The system chosen to depict implicit bias is CEA1, which implements the well-known Candidate Elimination Algorithm (CEA) using the algorithms of Mitchell (1978). CEA1 implements the two-disjunct heuristic and the Version Spaces representation for concepts as described in Mitchell (1978). The generalization tree of Figure 1 is used. The Version Spaces representation takes advantage of the partial ordering of hypotheses and compactly denotes all hypotheses consistent with the instances (and expressible in the language) by the set of maximally general hypotheses (*G boundary*) and the set of maximally specific hypotheses (*S boundary*). CEA1 conducts a pair of breadth-first searches through the space of hypotheses by refining the S and G boundaries based on new instances. When a

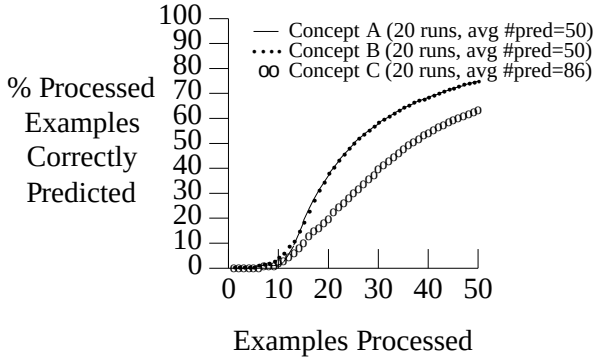

FIG. 3. Cohesion heuristic.

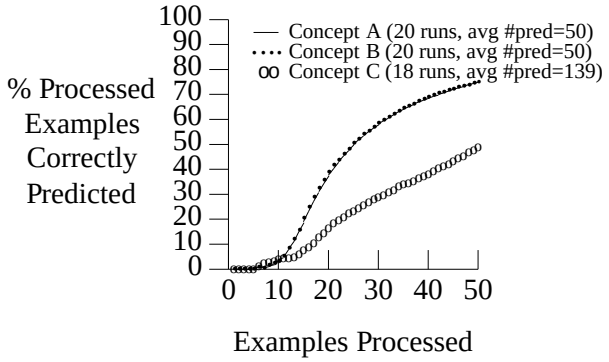

FIG. 5. Cohesion and independence.

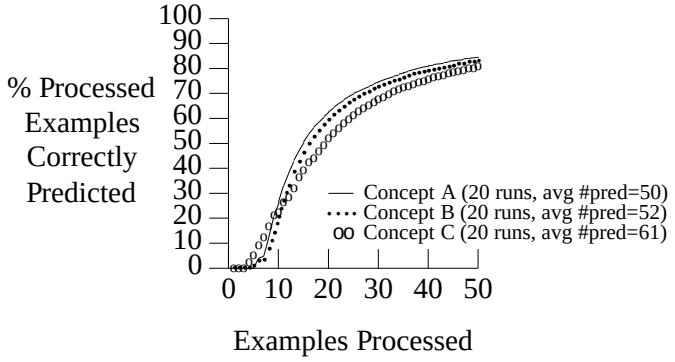

FIG. 7. Two-disjunct, directed backtracking.

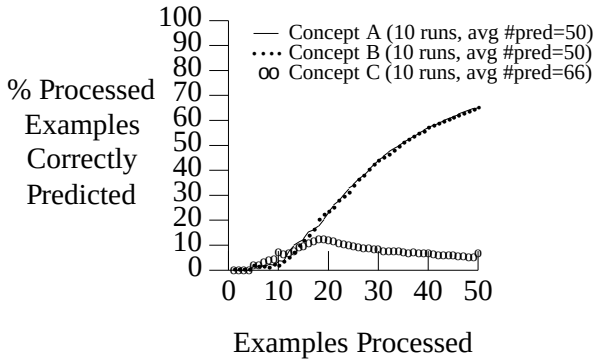

FIG. 4. Independence heuristic.

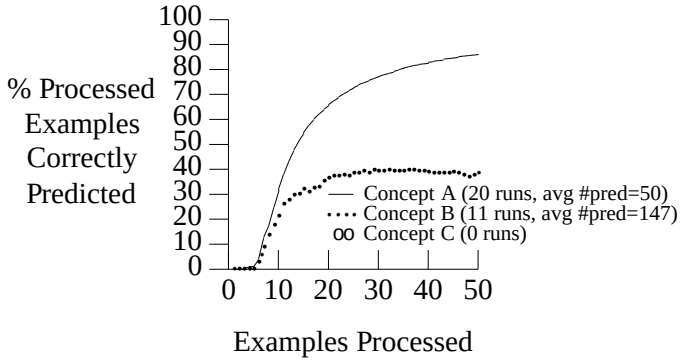

FIG. 6. Two-disjunct heuristic.

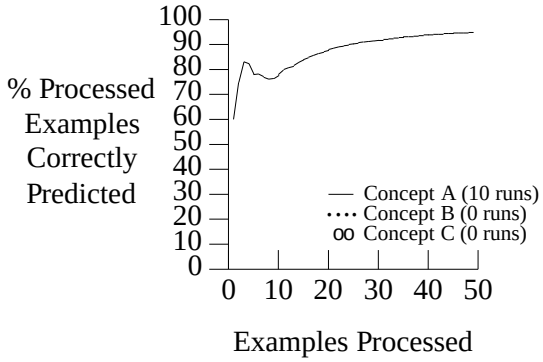

FIG. 8. CEA1.

positive instance is seen, the S boundary is generalized to the most specific generalization consistent with the new instance. Likewise, if a negative instance is encountered then the G boundary is specialized by the least amount possible. Instead of choosing between heuristics, the single, most specific generalization is always used. With CEA1, the two-disjunct heuristic is implemented implicitly for the S boundary because the most specific generalization CEA1 can make when a positive instance and one of the S boundary elements have different values for a feature is to climb the generalization tree to the immediate parent of the differing values. When the S and G

boundaries coincide, the concept has been completely learned. CEA1 is a deductive algorithm. No backtracking is used and so instances need not be saved. An error occurs if the S and G boundaries overlap.

The results of using CEA1 to learn concepts A, B, and C are shown in Figure 8. During initial learning, CEA1's performance differs from that of PREDICTOR because CEA1 predicts on every instance and always begins with a positive instance.[12] Although CEA1 performs very well learning concept A, it is unable to learn either B or C. The curves do not appear because the S and G boundaries overlapped on every trial. The problem is that cohesion does not hold but the generalization operator employed by CEA1 assumes cohesion does hold. Since this assumption is fixed and implicit, CEA1 cannot extricate itself from this problem. There are two possible changes to CEA1 that would resolve the problem.

One solution to CEA1's problem is to alter the hypothesis language. STABB, a program by Utgoff that is an extension of CEA, could augment the hypothesis language by adding new terms (Utgoff 1986). Although this is one important method for systems to shift their bias, if learning new terminology is the *only* method for bias alteration, then the system will soon become cluttered with many useless terms. For example, concept B maintains that spheres and cones are negative but cylinders are positive. (See Figure 1.) If it is discovered while learning concept B that cohesion does not hold for curved-solids, it may be more desirable to recall (as PREDICTOR does through backtracking) that spheres and cones are negative but cylinders are not than to invent a new term for spheres and cones.

A second way to solve CEA1's problem would be to alter its method of generalization. Either its most specific generalization operator could be changed or it might use a generalization heuristic. Suppose that instead of climbing the generalization hierarchy, the operator used by CEA1 takes the disjunction of instance descriptions with feature values not generalized. Then CEA1 would never generalize. Suppose instead that it generalized by forming the internal disjunction of feature values. Then a concept like concept C, which contains a feature dependency, could never be learned. A reasonable alternative would be to embed a rigorous version of the cohesion heuristic in the system in order to thoroughly check for cohesion prior to generalizing. Although this would solve the problem, flexibility would still be missing. When the number of feature values is large, this can be a costly heuristic to apply. We feel it is preferable to allow the system the option of selecting between a cautious heuristic and a less costly heuristic that is coupled with a rigorous and effective backtracking strategy. This can only be done with a flexible system using an explicit representation of the bias.

7. Related work on bias

As mentioned in the introduction, there are three major methods for biasing induction from examples: restricting the language, testing, and screening. PREDICTOR concentrates on the third technique. The systems of Winston (1975), Schlimmer and Granger (1986), and Porter and Kibler (1986) also use the third method. None of these other systems, though, use the definitions of biasing assumptions to test whether they hold. None of the systems varies the level of cautiousness of bias testing.

[12] The small peak is due to the excess of negative instances and the likelihood that the partial match algorithm predicts negative on the second instance.

Finally, the first two systems are passive learners, and analytical learning is not used by any of them. PREDICTOR is unique because it gathers information by both active learning and by traversing a domain theory for the purpose of gaining information to bias empirical induction.

Although PREDICTOR's biasing method offers advantages, more work needs to be done as described in the next section.

8. Future work

Four major directions are planned for future research. The first two are extensions to the current implementation. First, unlike programs such as STAGGER (Schlimmer and Granger 1986), PREDICTOR has only a primitive ability to handle noise. Rather than addressing noise as is currently done, we would like to develop statistical versions of the generalization heuristics and keep both the statistical and nonstatistical versions in the system. A metaheuristic could be implemented that would detect noise and decide which version of the heuristics is appropriate based on the current data. Furthermore, the issue of noise and a weak domain theory should be addressed.

The second proposed extension is to increase PREDICTOR's ability to alter its biases. The system currently has a couple of metaheuristics, such as the one which selects between the different modes of the heuristics. More metaheuristics are being added. It would eventually be desirable to have these metaheuristics open to modification by the system. Performance feedback could be used to generate positive and negative examples for learning at this level, as is done in the system VBMS, described in Rendell *et al.* (1987). Although VBMS selects between full induction algorithms rather than generalization operators, this idea could also be applied to our heuristics for selecting generalization operators.

Another direction for future research is to continue to prove theoretical results like the consistency property described in this paper. Much of the literature on the theory of induction addresses issues such as completeness, convergence rate, and computational efficiency of the induction algorithms in addition to their consistency.[13] (See Gold (1967), Shapiro (1981), and Valiant (1984).) Furthermore, Haussler has done important work in the area of quantifying inductive bias (Haussler 1988). We would like to analyze PREDICTOR's learning algorithms along similar dimensions to those addressed in the literature.

The fourth research direction will consist of further comparisons of the three biasing methods. It is important that we determine the type of data and concepts for which each of these methods is most appropriate.

9. Summary

Empirical induction may be biased by restricting the hypothesis language, by testing, or by screening. PREDICTOR concentrates on the third technique. PREDICTOR unifies generalization, active learning, and error resolution, using the common theme of explicit assumptions and corresponding tests of these assumptions. Furthermore, explicit bias enables PREDICTOR to test its assumptions while varying the degree of rigor of its tests. Although we have only begun to explore the implications

[13] If all possible combinations of all feature values appear in the instances then the concept is guaranteed to be learned. However, this would be true for any consistent learner.

of explicit bias, initial results are encouraging.

Acknowledgements

The authors would like to thank Chitoor Srinivasan, whose constructive comments improved the formal definitions of the bias assumptions. We would also like to thank Jaime Carbonell for suggesting a comparison of the heuristics individually, and John Grefenstette and William Spears for their helpful comments on earlier drafts of this paper. William Spears implemented CEA1, which provided a useful program for comparison.

BRUNER, J., GOODNOW, J. and AUSTIN, G. 1986. A study of thinking. Transaction Books, New Brunswick, NJ.

CARBONELL, J. and GIL, Y. 1987. Learning by experimentation. Proceedings of the Fourth International Workshop on Machine Learning, Irvine, CA. pp. 256-266.

CARPINETTO, C. 1988. An approach based on integrated learning to generating stories from stories. Proceedings of the Fifth International Conference on Machine Learning, Ann Arbor, MI, pp. 298-304.

DEJONG, G. and MOONEY, R. 1986. Explanation-based learning: an alternative view. Machine Learning, **1** : 145-176.

FISHER, D. 1987. Knowledge acquisition via incremental conceptual clustering. Machine Learning, **2** : 139-172.

GOLD, E. 1967. Language identification in the limit. Information and Control, **10** : 447-474.

GORDON, D. 1988. A cautious method for explicitly biasing generalization. Proceedings of the Third Annual Rocky Mountain Conference on Artificial Intelligence, Denver, CO.

HAUSSLER, D. 1988. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. Artificial Intelligence, **36** : 177-221.

KEDAR-CABELLI, S. and MCCARTY, T. 1987. Explanation-based generalization as resolution theorem proving. Proceedings of the Fourth International Workshop on Machine Learning, Irvine, CA, pp. 383-389.

MICHALSKI, R. 1983. A theory and methodology of inductive learning. *In* Machine Learning, Volume 1. *Edited by* R. Michalski, J. Carbonell, and T. Mitchell. Tioga Publishing Co., Palo Alto, CA, pp. 83-134.

MICHALSKI, R. and STEPP, R. 1983. Learning from observation: Conceptual clustering. *In* Machine Learning, Volume 1. *Edited by* R. Michalski, J. Carbonell, and T. Mitchell. Tioga Publishing Co., Palo Alto, CA, pp. 331-363.

MITCHELL, T. 1978. Version spaces: an approach to concept learning. Ph.D. thesis, Stanford University, Stanford, CA.

MITCHELL, T. 1980. The need for biases in learning generalizations. Rutgers University Technical Report Number TR CBM-TR-117, Rutgers University, New Brunswick, N.J.

MITCHELL, T., KELLER, R. and KEDAR-CABELLI, S. 1986. Explanation-based generalization: a unifying view. Machine Learning, **1** : 47-80.

NILSSON, N. 1980. Principles of artificial intelligence. Tioga Publishing Co., Palo Alto, CA.

PAZZANI, M. 1988. Integrated learning with incorrect and incomplete theories. Proceedings of the Fifth International Conference on Machine Learning, Ann Arbor, MI, pp. 291-297.

PORTER, B. and KIBLER, D. 1986. Experimental goal regression: a method for learning problem-solving heuristics. Machine Learning, **1** : 249-286.

RENDELL, L., SESHU, R. and TCHENG, T. 1987. More robust concept learning using dynamically-variable bias. Proceedings of the Fourth International Workshop on Machine Learning, Irvine, CA. pp. 66-78.

SAMMUT, C. and BANERJI, R. 1986. Learning concepts by asking questions. *In* Machine learning, Volume 2. *Edited by* R. Michalski, J. Carbonell, and T. Mitchell. Morgan Kaufmann Publishing Co., Los Altos, CA, pp. 167-192.

SCHLIMMER, J. and GRANGER, R. 1986. Incremental learning from noisy data. Machine Learning, **1** : 317-354.

SHAPIRO, E. 1981. Inductive inference of theories from facts. Yale Research Report 192.

SIMS, M. 1987. Empirical and analytical discovery in IL. Proceedings of the Fourth International Workshop on Machine Learning, Irvine, CA. pp. 274-280.

SUBRAMANIAN, D. and FEIGENBAUM, J. 1986. Factorization in experiment generation. Proceedings of the Fifth National Conference on Artificial Intelligence, Philadelphia, PA, pp. 518-522.

UTGOFF, P. 1986. Shift of bias for inductive concept learning. *In* Machine learning, Volume 2. *Edited by* R. Michalski, J. Carbonell, and T. Mitchell. Morgan Kaufmann Publishers, Los Altos, CA. pp. 107-148.

VALIANT, L. 1984. A theory of the learnable. Communications of the ACM, **27** : 1134-1142.

WINSTON, P. 1975. Learning structural descriptions from examples. *In* The psychology of computer vision. *Edited by* P. Winston. McGraw-Hill Book Company, New York, NY.

Appendix A. Consistency requirement and definition of consistency-preserving

For appendices A, B and C, we assume that for all t, there is no instance kept in $I_+(t)$ whose description is identical to that of an element of $I_-(t)$.

SELECT-INHIBIT Consistency Requirement, or SIC(t):

$$(SEL(t) \cap INH(t)) = \emptyset.$$

Definition of consistency-preserving:

A heuristic, G, is said to be consistency-preserving if for each t such that SIC(t) and $COND_G$ hold, SIC(t+1) also holds, where SEL(t+1) and INH(t+1) are the results of applying OP_G to SEL(t), INH(t), and $i(t)$.

Appendix B. Irrelevance heuristic

This appendix contains a formal definition of the irrelevance heuristic and a proof that this heuristic is consistency-preserving. In order to simplify (shorten) the expressions, we replace **val**(*r,x,v*) with *r(x,v)* in appendices B and C.

B1. Irrelevance heuristic condition

Let $L(t)$=(SEL(t) $\cup \{i(t)\}$) or $L(t)$=(INH(t) $\cap \overline{\{i(t)\}}$) if i is positive. Let $L(t)$=(INH(t) $\cup \{i(t)\}$) or $L(t)$=(SEL(t) $\cap \overline{\{i(t)\}}$) if i is negative. Furthermore, we define $\{r_1,...,r_n\}$ to be the set of features considered relevant as of time (t - 1). Let $1 \leq i \leq n$, where i is the subscript used in the following.
Then, $COND_{IRR(r_i,C,t)} \leftrightarrow$

$$((\forall v_1,...,v_n)(((\exists x \in L(t))(r_1(x,v_1) \,\&...\&\, r_n(x,v_n))) \rightarrow$$
$$((\forall w_i)(\forall y)((r_1(y,v_1) \,\&...\&\, r_i(y,w_i) \,\&...\&\, r_n(y,v_n)) \rightarrow (y \in L(t))))).$$

The full condition is tested if every w_i in the codomain of feature r_i is checked.

B2. Irrelevance generalization operator

Assume $SEL(t) = \{ x \mid \alpha_1(x) \}$ and $INH(t) = \{ x \mid \alpha_2(x) \}$. Let $\beta^r(i)|_i^x$ be the description of a new instance *i(t)* in which relation name r is omitted and in which variable x is substituted for i everywhere in the description. Furthermore, let $\alpha_k{}^r(x)$ be identical to $\alpha_k(x)$ except that relation name r is omitted (and k is 1 or 2). Then the generalization operator, $OP_{IRR(r,C,t)}$, will result in two of the following (based on the class of *i(t)*):

$$SEL(t+1) = \{ x \mid (\alpha_1{}^r(x) \vee \beta^r(i)|_i^x) \} \text{ if } i \textit{ positive}$$

$$INH(t+1) = \{ x \mid (\alpha_2{}^r(x) \,\&\, not(\beta^r(i))|_i^x) \} \text{ if } i \textit{ positive}$$

$$SEL(t+1) = \{ x \mid (\alpha_1{}^r(x) \,\&\, not(\beta^r(i))|_i^x) \} \text{ if } i \textit{ negative}$$

$$INH(t+1) = \{ x \mid (\alpha_2{}^r(x) \vee \beta^r(i)|_i^x) \} \text{ if } i \textit{ negative}$$

Note that for all future instances *i(t+2),...,i(t+n)*, introduced at time *(t+2)*,...,time *(t+n)*, their description will not include feature r either.

B3. Consistency theorem for irrelevance heuristic

Theorem 1: $G_{IRR(r,C,t)}$ is consistency-preserving (when the full condition is tested).
Proof of Theorem 1

We shall prove that for every t, $(SEL(t) \cap INH(t)) = \varnothing$ (SIC(t) holds) after the irrelevance heuristic has been applied if it holds prior to heuristic application. We proceed by induction on t.

Base Step: Since $SEL(0) = INH(0) = \varnothing$ (by axiom [A4]), SIC(0) holds.

Inductive Step: By inductive hypothesis, assume SIC(t) holds. Assume $COND_{IRR(r,C,t)}$ holds. It is necessary to prove that SIC(t+1) holds. In other words, $(SEL(t+1) \cap INH(t+1)) = \varnothing$.

Let $SEL(t) = \{ x \mid \alpha_1(x) \}$ and $INH(t) = \{ x \mid \alpha_2(x) \}$.
Assume $i(t)$ is a positive instance (similar argument if negative) and that $\alpha_i^r(x)$ is identical to $\alpha_i(x)$ except that relation r is omitted for $i = 1, 2$. It is necessary to prove that $\{ x \mid \alpha_1^r(x) \vee \beta^r(i)|_i^x) \} \cap \{ x \mid \alpha_2^r(x) \ \& \ not(\beta^r(i))|_i^x \} = \varnothing$.
There are 4 possible combinations for SEL(t) and INH(t):

Case 1: α_1 and α_2 both include r.
Case 2: α_1 includes r but α_2 does not.
Case 3: α_1 does not include r but α_2 does.
Case 4: α_1 and α_2 both lack r.

By inductive hypothesis, whichever case holds, $(SEL(t) \cap INH(t)) = \varnothing$. Case 4 is the simplest. If $\{ x \mid \alpha_1(x) \} \cap \{ x \mid \alpha_2(x) \} = \varnothing$, where α_1 and α_2 do not include r, then it must be the case that $(SEL(t+1) \cap INH(t+1)) = \varnothing$ since r need not be removed and since any x satisfying β^r is included in SEL and is not included in INH. Now consider cases 1, 2, and 3. According to the definition of the irrelevance condition, if the value of r in the description of an object for which the target concept holds is changed, then the target concept must still hold for any object with this new description. If this is the case, and $(SEL(t) \cap INH(t)) = \varnothing$, then r must not be necessary for distinguishing whether or not the target concept holds. Instead, there must be a set of features currently considered relevant, $\{s_1, s_2, ..., s_n\}$, where $s_i \neq r$ for all $1 \leq i \leq n$, such that the s_i are sufficient for distinguishing target concept membership. Therefore, it is possible to generalize SEL(t) and INH(t) to the SEL(t) and INH(t) of case 4 without losing any information related to goal concept membership. From here on the proof mirrors that of case 4.

Since consistency holds at time t+1, then by the principle of mathematical induction $G_{IRR(r,C,t)}$ is consistency-preserving. □

Appendix C. Independence heuristic

This appendix contains a formal definition of the independence heuristic applied to only one of the hypotheses. This is the form of the heuristic that is implemented in PREDICTOR. We also prove the consistency theorem for $G_{IND(R,L,t)}$, where *L(t)* is $(SEL(t) \cup \{i(t)\})$ if the new instance, *i(t)*, is positive, or *L(t)* is $(INH(t) \cup \{i(t)\})$ if *i(t)* is negative. Although we only prove that the heuristic applied to *one* of the hypotheses preserves consistency, it follows that if consistency is preserved for each hypothesis individually then it is preserved for both. Therefore, it follows from this section that $G_{IND(R,C,t)}$, the independence heuristic defined in the body of this paper which modifies both hypotheses, is consistency-preserving.

C1. Independence condition

The independence condition is the following, where $L(t) = (SEL(t) \cup \{i(t)\})$ if *i(t)* is positive, or $L(t) = (INH(t) \cup \{i(t)\})$ if *i(t)* is negative. Let $R(t) = \{r_1, \ldots, r_n\}$ be the set of all features currently considered relevant. $V_i(t)$ is the set of values of feature $r_i \in R(t)$ which appear in *L(t)*. Subscript *i* has values $1 \le i \le n$.

$$
\begin{aligned}
&COND_{IND(R,L,t)} \leftrightarrow \\
&\quad ((\forall r_i \in R(t))(\forall v_1,\ldots,v_n)(((\exists x \in L(t))(r_1(x,v_1)\ \&\ldots\&\ r_n(x,v_n))) \rightarrow \\
&\quad\quad ((\forall w_i \in V_i(t))(\forall y)((r_1(y,v_1)\ \&\ldots\&\ r_i(y,w_i)\ \&\ldots\&\ r_n(y,v_n)) \rightarrow \\
&\quad\quad\quad (y \in L(t))))))).
\end{aligned}
$$

C2. Independence generalization operator

Let $H(t) = \mathrm{SEL}(t)$ if *i(t)* is positive or $\mathrm{INH}(t)$ if *i(t)* is negative. Let the instance description of *i(t)* be $\beta(i)|_i^x = (r_1(x,v_{i1})\ \&\ldots\&\ r_n(x,v_{in}))$ where *n* is the number of relevant features.

If $H(t) = \{x \mid ((r_1(x,v_{11})\ \&\ldots\&\ r_n(x,v_{1n}))\ \mathrm{v}\ldots\mathrm{v}\ (r_1(x,v_{k1})\ \&\ldots\&\ r_n(x,v_{kn})))\}$
then

$$
\begin{aligned}
H(t+1) = \{x \mid &(((r_1(x,v_{11})\ \mathrm{v}\ldots\mathrm{v}\ r_1(x,v_{k1}))\ \mathrm{v}\ r_1(x,v_{i1}))\ \&\ldots \\
&\&\ ((r_n(x,v_{1n})\ \mathrm{v}\ldots\mathrm{v}\ r_n(x,v_{kn}))\ \mathrm{v}\ r_n(x,v_{in})))\}.
\end{aligned}
$$

If *H(t)* = SEL(*t*), then $\overline{\mathrm{INH}(t+1)} = (\mathrm{INH}(t) \cap \{\beta(i)|_i^x\})$. If *H(t)* = INH(*t*), then $\mathrm{SEL}(t+1) = (\mathrm{SEL}(t) \cap \{\beta(i)|_i^x\})$. Also, *H(t+2)*, *H(t+3)*,... are in internal disjunctive form as well (unless a generalization error is later detected).

C3. Consistency theorem for independence heuristic

We first state and prove a lemma on which a proof of the theorem depends. This lemma states that the DNF and IDF forms are equivalent if the condition holds. Therefore, if the condition holds then the form conversion is truth-preserving for the current hypotheses (though not for future ones).

Lemma 1

Let $H_1(t) = \{x \mid ((r_1(x,v_{11})\ \&\ldots\&\ r_n(x,v_{1n}))\ \mathrm{v}\ldots\mathrm{v}\ (r_1(x,v_{k1})\ \&\ldots\&\ r_n(x,v_{kn})))\}$.
If $COND_{IND(R,H_1,t)}$ holds, then for all y, $y \in H_1(t))$ if and only if $y \in H_2(t))$, where
$H_2(t) = \{x \mid ((r_1(x,v_{11})\ \mathrm{v}\ldots\mathrm{v}\ r_1(x,v_{k1}))\ \&\ldots\&\ (r_n(x,v_{1n})\ \mathrm{v}\ldots\mathrm{v}\ r_n(x,v_{kn})))\}$.

Proof of Lemma 1

($\rightarrow$) This case is true in general, even if $COND_{IND(R,H_1,t)}$ does not hold. Let $y \in H_1(t)$ and let one of the disjuncts of $H_1(t)$ describe y. Then it must be the case that $y \in H_2(t)$ since every feature value of every disjunct of $H_1(t)$ appears in the internal disjunction for that feature in $H_2(t)$. Therefore, $y \in H_1(t)$ implies $y \in H_2(t)$.

($\leftarrow$) Let $y \in H_2(t)$ and let $\gamma(y)$ be a description of y. Let $VALS_1(r_j) = \{v_{1j}, \ldots, v_{kj}\}$ be all values of feature r_j found in the defining condition for $H_1(t)$, where $1 \leq j \leq n$. Since $H_1(t)$ and $H_2(t)$ have the same features and values, the description of y is composed of some combination of feature values of $VALS_1(r_j)$ for all features r_j. If $COND_{IND(R,H_1,t)}$ holds, $\gamma(y)$ implies $y \in H_1(t)$. This is because $COND_{IND(R,H_1,t)}$ implies all permutations of feature values for $r_1, \ldots, r_n$ still describe elements of $H_1(t)$. To show that this is true, we will take an arbitrary combination of feature values from $VALS_1(r_j)$ for all j between 1 and n and show that an object z satisfying this description is in $H_1(t)$. Let $\delta(z)$ be a description of z with this arbitrary combination. If we now change each of these values of features in $\delta(z)$ to various other values from $VALS_1(r_j)$ for each j, we will eventually describe some element of $H_1(t)$ since for every j, $VALS_1(r_j)$ was formed from existing feature values found in the defining condition for $H_1(t)$. Now, according to $COND_{IND(R,H_1,t)}$, we can change each of these values for r_j to new values within $VALS_1(r_j)$ and this newly-produced description will still describe some element of $H_1(t)$. In particular, we can change the values back to those of $\delta(z)$. Therefore, $z \in H_1(t)$. $H_1(t)$ thus has elements described by every permutation of feature values from $VALS_1(r_j)$ for all j. Returning to the proof for y, since every feature value present in $\gamma(y)$ is also in $VALS_1(r_j)$, $1 \leq j \leq n$, $y \in H_2(t)$ implies $y \in H_1(t)$. □

Theorem 2: $G_{IND(R,L,t)}$ is consistency-preserving.
Proof of Theorem 2

We shall prove that $(\forall t)(SEL(t) \cap INH(t)) = \emptyset$. We proceed by induction on t.

Base Step: $SEL(0) = INH(0) = \emptyset$ (by axiom [A4]). Therefore, SIC(0) holds.

Inductive Step: Assuming $(SEL(t) \cap INH(t)) = \emptyset$ and $COND_{IND(R,L,t)}$ holds, we need to show $(SEL(t+1) \cap INH(t+1)) = \emptyset$. Let the new instance, *i(t)*, be positive. The argument is similar for *i(t)* negative. *SEL(t)* is of the form $H_1(t)$ described in Lemma 1. So we shall call it $SEL_1(t)$. But if $SEL_2(t)$ is of the form $H_2(t)$ described in Lemma 1, which is the internal disjunctive form of $SEL_1(t)$, then $SEL_1(t) = SEL_2(t)$ by Lemma 1. Therefore, $(SEL_2(t) \cap INH(t)) = \emptyset$. Since $INH(t+1) = (\mathrm{INH}(t) \cap \{\beta(i) \mid \overline{{}^{x}_{i}}\})$, $(SEL_2(t) \cap INH(t+1)) = \emptyset$. Now, the only difference between *SEL(t+1)* and $SEL_2(t)$ is that the former includes a description of the new instance, *i(t)*. We assume instances that are kept are consistent. Then the inclusion of the description of *i(t)* within $SEL_2(t)$ will not violate consistency. Therefore, $(SEL(t+1) \cap INH(t+1)) = \emptyset$.

Since consistency holds at time (t+1), therefore by the principle of mathematical induction $G_{IND(R,L,t)}$ is consistency-preserving. □