Queries for Bias Testing

Diana F. Gordon

Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory, Code 5510
Washington, D.C. 20375
gordon@aic.nrl.navy.mil

Abstract

Selecting a good bias prior to concept learning can be difficult. Therefore, dynamic bias adjustment is becoming increasingly popular. Current dynamic bias adjustment systems, however, are limited in their ability to identify erroneous assumptions about the relationship between the bias and the target concept. Without proper diagnosis, it is difficult to identify and then remedy faulty assumptions. We have developed an approach that makes these assumptions explicit, actively tests them with queries to an oracle, and adjusts the bias based on the test results.

1 Introduction

Bias is a fundamental aspect of any supervised concept learner. Numerous papers have noted this importance (e.g., Mitchell 1980; Haussler 1988). The type of bias that we discuss here is the choice of a hypothesis language. The hypothesis language defines the space of hypotheses. A *strong* bias defines a small hypothesis space; a *weak* bias defines a large hypothesis space; a *correct* bias defines a space that includes the target concept. A strong correct bias, e.g., one with fewer features, is generally desirable because it reduces the number of hypothesis choices and thereby promotes rapid convergence to the target concept.

The bias can be adjusted (shifted) dynamically during incremental concept learning by strengthening the bias when possible and weakening it to regain correctness. Recently, interest has grown in systems that dynamically shift the bias (e.g., Utgoff 1986; Rendell 1990; Spears & Gordon 1991). These systems, however, are limited in their ability to identify erroneous assumptions about the relationship between the bias and the target concept. Proper diagnosis aids in the recovery from faulty assumptions. We have developed an approach to bias adjustment that addresses this need for proper diagnosis. Our method consists of a bias tester and adjuster that can be added to an incremental concept learner to improve the learner's performance.

Unlike previous approaches to bias testing, our approach uses formal definitions of assumptions about the bias, called *biasing assumptions*, to guide an analysis of *why* the bias is inappropriate (e.g., too weak, or incorrect) for learning the target concept. An example of a biasing assumption is the irrelevance of a feature for learning the target concept. The bias tester performs this analysis (called a *biasing assumption test*) by actively testing the bias with queries to an oracle. Each query is a request to an instance generator for a new instance. For example, the irrelevance of a feature might be tested by querying an oracle for the class (positive/negative) of instances having different values of that feature. The bias adjuster then records the analysis results and adjusts the bias accordingly. If a biasing assumption holds, the adjuster strengthens the bias, e.g., by removing the irrelevant feature from the hypothesis language. Otherwise, the adjuster weakens the bias or allows the bias to stay the same if no adjustments are needed.

Our approach has three primary advantages. First, the bias tests are composed of queries. Queries can accelerate learning significantly (see Gordon 1990; 1992). Second, our approach is designed to be incorporated into an existing concept learner. Third, our approach diagnoses the bias to find and record specific erroneous biasing assumptions. This enables the bias to be *minimally weakened* as well as corrected. Minimal weakening is most advantageous when a stronger bias is desirable. In that case, bias strengthening along with minimal bias weakening can enable very rapid acquisition of the target concept (see Gordon 1990; 1992).

In our framework, the bias is the set of features and their values in the hypothesis language. These values appear in *value trees* (e.g., see Figure 1), which are input by a user or knowledge engineer who is somewhat familiar with the domain. Value trees are

typically called *generalization trees* because parent nodes are more general than their child nodes. Training instances are described in terms of leaf node values. Throughout this paper, we assume the concept learner begins with hypotheses described in terms of the instance language and evolves its hypotheses (perhaps using the value trees) in a specific-to-general direction. *Generalization* increases the generality of values within a particular hypothesis; *abstraction* increases the generality of the hypothesis language. The concept learner can use value trees for generalization. Our approach to bias testing and adjustment uses value trees for abstraction. Bias strengthening implies removal (i.e., abstraction) of a feature or feature value distinction from the hypothesis language. This shrinks the hypothesis space. Bias weakening implies the restoration of features or feature value distinctions. This weakening undoes abstraction and enlarges the hypothesis space. Bias weakening is defined to be minimized when the features and feature value distinctions that are restored to the language are restricted to those that *must* be restored to correct the bias.

The drawback of our approach is that it requires an oracle that can respond to queries during learning. The oracle can be either a human or the environment. In either case, it is not always practical to require an oracle. Humans may be too busy to answer questions. Furthermore, the use of the environment as an oracle is impractical if lives are at stake. For example, it is unreasonable to query whether a new chemical weapon is effective at killing people. On the other hand, it *is* practical to query whether small doses of Vitamin C cure the common cold.

Using Figure 1, we can see how a bias may be strengthened, weakened, and minimally weakened. Suppose the bias is all the trees in Figure 1, and the target concept states that **small bricks** are positive and instances of any other description are negative. The bias might be strengthened by removing all features other than "size" from the hypothesis language. This bias is incorrect because "shape" information is also needed to learn the target concept. One way to weaken and correct the bias is to restore the original language. Alternatively, we can minimally weaken the bias by restoring parts of the "shape" value tree but none of the "material" tree. Within the "shape" tree, we restore the "cube"/"brick" distinction and above, and restore the "curved-solid" node, but do not restore any child of the "curved-solid" node. Removing a distinction strengthens the bias to create an abstraction, whereas restoring it weakens the bias to undo the abstraction.

Section 2 formally defines two important biasing assumptions and then collapses them into one. Section 3 presents and analyzes algorithms to test the collapsed assumption. Section 4 summarizes and explains empirical results. Finally, Sections 5 and 6 present related work and a summary of the paper.

2 Biasing Assumptions

When supervised concept learners shift their bias, they typically make an implicit biasing assumption that the bias shift is correct for learning the target concept. Our approach makes each biasing assumption explicit, and associates each assumption with an abstraction operator. If the assumption holds, the corresponding abstraction operator can fire.

We assume two abstraction operators: *climb-value-tree(f,a)* and *remove-feature(f)*. The climb-value-tree(*f,a*) operator replaces values of feature *f* that are lower in the value tree (e.g., "cube" and "brick" in Figure 1) with a value *a* (e.g., "prism") that is higher in the tree throughout the hypothesis language. The remove-feature(*f*) operator eliminates feature *f* from the hypothesis language. We associate a *cohesion* assumption with climb-value-tree(*f,a*). Cohesion implies that the values below *a* in the value tree of *f* are unnecessary for predicting the target concept membership of instances. We associate an *irrelevance* assumption, which is equivalent to cohesion at the root node of a value tree, with drop-feature(*f*). Irrelevance implies that the feature to be removed is unnecessary for predicting the target concept membership of instances.

The following are the formal definitions of the two biasing assumptions. These definitions are tailored for an incremental concept learning context. We assume one new instance is accepted at a time and all previous instances are saved. Furthermore, we assume that the instances are not noisy and that the instance features are sufficient to distinguish positive from negative instances, though perhaps not ideal for learning the target concept. We abbreviate the set of all known positive instances at time t with POS(t), the set of all known negative instances at time t with NEG(t), the set of all positive instances with POS, and the set of all negative instances with NEG. We abbreviate the new instance at time t with $i(t)$, the target concept with TC, the irrelevance biasing assumption with $IRR(f,TC,t)$ for feature f, and the cohesion biasing assumption with $COH(a,TC,t)$ for value a.

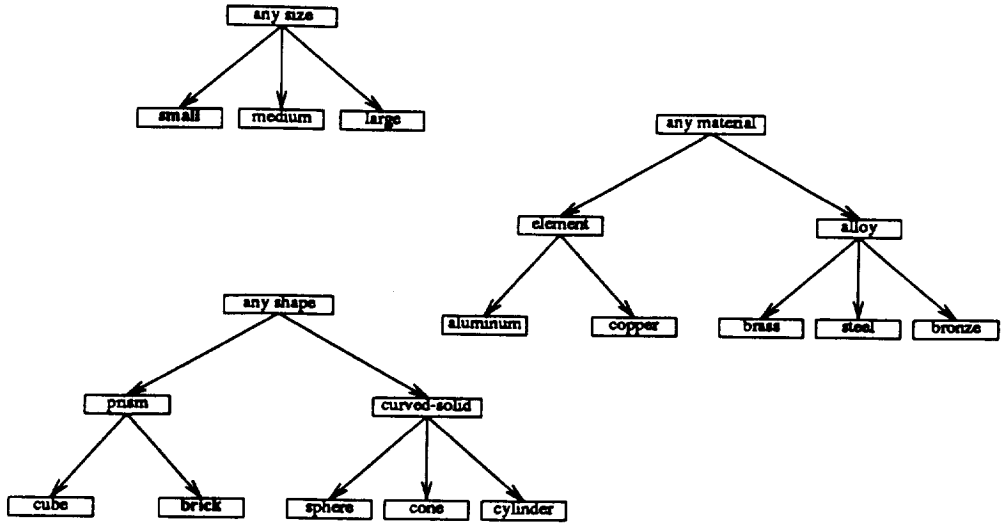

FIG. 1. Value trees.

For the following definitions, if $i(t)$ is positive, we let $L(t)$=(POS(t) $\cup$ $\{i(t)\}$) and L=POS or we let $L(t)$=NEG(t) and L=NEG. Likewise, if $i(t)$ is negative, we let $L(t)$=(NEG(t) $\cup$ $\{i(t)\}$) and L=NEG or we let $L(t)$=POS(t) and L=POS.

Let $\{f_1,...,f_n\}$ be the set of features considered relevant as of time (t - 1). Let $1 \le i \le n$, where i is the subscript used in the following definitions. Finally, we define $f_i(x,v_i)$ to mean that the value of feature f_i for instance x is v_i. Although the instance language consists of value tree leaf nodes, a nonleaf node can also be used to describe an instance, though not uniquely. We allow v_i to be either a leaf or nonleaf node in the following definitions. The formal definition of the irrelevance biasing assumption is:

$$IRR(f_i,TC,t) \leftrightarrow$$
$$((\forall v_1,...,v_n)(((\exists x \in L(t))(f_1(x,v_1) \,\&...\&\, f_n(x,v_n))) \rightarrow$$
$$((\forall w_i)(\forall y)((f_1(y,v_1) \,\&...\&\, f_i(y,w_i) \,\&...\&\, f_n(y,v_n))$$
$$\rightarrow (y \in L))))).$$

In other words, f_i is considered irrelevant to learning TC at time t if changing the value of f_i in any known instance x always yields a (new or old) instance whose classification (positive/negative) is the same as that of x.

Next, we define the cohesion biasing assumption. The cohesion of value a with respect to the target concept, $COH(a,TC,t)$, means that the descendent nodes $\{a_1,\ldots,a_l\}$ below value (node) a in the value tree appear to behave equivalently with respect to target concept membership. Let $A = \{a_1,\ldots,a_l\}$. Let $1 \le i \le n$ and $1 \le j,k \le l$. The formal definition is:

$$COH(a,TC,t) \leftrightarrow$$
$$((\forall v_1,...,v_n)(\forall a_j \in A)(((\exists x \in L(t))$$
$$(f_1(x,v_1) \,\&...\&\, f_i(x,a_j) \,\&...\&\, f_n(x,v_n))) \rightarrow$$
$$((\forall a_k \in A)(\forall y)((f_1(y,v_1) \,\&...\&\, f_i(y,a_k) \,\&...\&\, f_n(y,v_n))$$
$$\rightarrow (y \in L))))).$$

In other words, cohesion holds for value a for learning TC at time t if the replacement of one descendent value of a with another descendent value in any known instance x always yields a (new or old) instance whose classification is the same as that of x. Note that irrelevance is a special case of cohesion that occurs when a is the root node of a value tree. Therefore, these two assumptions can be collapsed into one. Let us call the collapsed assumption $IRR\text{–}COH(a,TC,t)$. The definition of this collapsed assumption is identical to that of $COH(a,TC,t)$.

3 Queries for Testing the Biasing Assumptions

The definition of $IRR\text{–}COH(a,TC,t)$ presented in the last section has been translated into algorithmic biasing assumption tests. This section presents the algorithms for these tests. There are two types of

biasing assumption tests, corresponding to the two times at which tests are executed. Each type is associated with a separate algorithm. One type of test (in Section 3.1) executes before bias shifting, and the other type (in Section 3.2) executes after bias shifting.

Like the definition on which they are based, our biasing assumption tests are tailored for incremental concept learning. If an assumption test is satisfied, then the corresponding biasing assumption is considered valid, and therefore it is "safe" to implement the abstraction corresponding to this assumption. If the test is not satisfied, corrective action may be required.

We assume that our approach to bias testing and shifting is added to an incremental concept learner that maintains two Disjunctive Normal Form (DNF) hypotheses: one that covers all previously seen positive instances and one that covers all previously seen negative instances. The flow of control begins when a query to the instance generator requests a new instance. When the new instance is received by the concept learner, the learner uses its hypotheses to predict the class of this instance. The learner then consults an oracle to find out the true class of the instance. If enough instances have been seen at this time to complete an assumption test, the bias is shifted according to the test results. Next, the learner updates its hypotheses to preserve *completeness* and *consistency*. Completeness implies the positive hypothesis covers all known positive instances and the negative hypothesis covers all known negative instances. Consistency implies the positive hypothesis covers no known negative instances and the negative hypothesis covers no known positive instances. These steps are repeated until the user decides the target concept has been learned. For more details see (Gordon 1992).

We introduce four types of queries to facilitate concept learning with bias shifts: *bias strengthening queries*, *bias weakening queries*, *counterexample queries*, and *random instance queries*. An assumption test is a sequence of bias strengthening queries or a sequence of bias weakening queries. Bias strengthening queries test abstractions before they are made; bias weakening queries retest abstractions after they have been made. The last two types of queries are not part of assumption tests, but they are useful for other reasons. The purpose of counterexample queries is to find out *whether* the bias is incorrect. If an incorrectness is found, the bias weakening queries then determine *why* the bias is incorrect. The purpose of the random instance queries is to generate instances for concept learning when none of the other queries applies. All queries except the bias weakening queries request instances not previously seen. Bias weakening queries try to use previously seen instances before generating new ones because they retest previously held assumptions, and the necessary instances to do this are often already present.

Random instance and counterexample queries are simple, so we describe them first. Random instance queries are requests to the instance generator for randomly generated (previously unseen) instances. Counterexample queries can provide counterexamples because they are requests for randomly generated (unseen) instances that are covered by one of the hypotheses. A negative instance covered by the positive hypothesis is a counterexample, and a positive instance covered by the negative hypothesis is a counterexample.

3.1 Bias Strengthening Queries

Bias strengthening queries test whether the biasing assumption associated with a potential abstraction holds. To do this, these queries test nodes of the value tree below the potential abstraction. If these values do not seem useful for distinguishing target concept membership, the abstraction is made.

The assumption tests that use bias strengthening queries and are executed prior to abstraction may be as rigorous as desired. Tests that use more queries are more rigorous. (Gordon 1990) describes a method for varying the rigor of these tests. Increasing the rigor can reduce the number of prediction errors, but it can also significantly increase the cost of the tests. For example, suppose we wish to test the cohesion of value a of feature f. Let us consider how we may vary the rigor of tests that are based on the formal definition of $IRR\text{-}COH(a,TC,t)$ in Section 2. We can increase the rigor with which we test the cohesion assumption by the following two methods: (1) Increase the number of values a_k from A to substitute for the original value a_j before assuming cohesion holds; (2) Increase the number of instances x whose f value is varied before assuming cohesion holds. (Note that each x corresponds to a unique choice of values for v_1 through v_n.) Either of these methods will increase the number of queries.

Here, we describe an algorithm that does not have very rigorous assumption tests and is therefore not excessively costly. It is not very rigorous because only one sibling of the original value is tested for each

hypothesis disjunct before making an abstraction, and because hypothesis disjuncts, rather than instances, have their values varied. There are typically far fewer disjuncts than instances. Our algorithm for generating bias strengthening queries to test biasing assumptions is the following:

Repeat the following until no more unseen, uncovered instances i' can be generated:
  For each instance feature f do
    Find the value a that is the parent node of the value of f in some (arbitrary) disjunct of one of the two hypotheses. The value a is a potential abstraction to be tested. If this value has been tested previously, then select another value for a from another disjunct.
    For each (positive, negative) hypothesis h do
      For each disjunct *disj* of h do
        (1) Find $f(x,v)$, a conjunct of *disj*. If none exists, or if v is not a child of a, try another disjunct.
        (2) Set SIBLINGS equal to the set of all siblings (which share a parent node a) of v in the value tree of f. These siblings are children of a.
        (3) Select $s \in$ SIBLINGS.
        (4) Replace "$f(x,v)$" in *disj* with "$f(x,s)$" to form d'. Then form a potential instance i' that satisfies d' by translating d' to the language of the instances, which consists of leaf values in the value trees. When translating to leaf values, the choice of a descendent of a higher level value is random. Check to see that i' is not already covered by the hypotheses and has not been seen yet. If i' is uncovered and unseen, request i' from the instance generator and continue. Otherwise, try other choices of leaf value descendents until they have all been tried. If the descendents have all been tried, then go to step (3) to find another sibling.
        (5) Accept i' from the instance generator and consult the oracle for the class of i'.
      Endfor
    Endfor
    (If the assumption test for abstraction a has succeeded at this point, the bias adjuster makes the abstraction.)
  Endfor

This algorithm executes a sequence of biasing assumption tests. Each biasing assumption test corresponds to a test of an abstraction a. This test consists of a sequence of queries that vary the values in the hypothesis disjuncts. To ensure that these queries do not overlap with the other query types, bias strengthening queries only request instances that are not covered by either of the hypotheses and have not yet been seen. To form each bias strengthening query, this algorithm selects one feature f of one disjunct of one hypothesis h and alters the value of this feature. This is a form of *perturbation* (Porter & Kibler 1986). The value of f that is perturbed is a child, i.e., an immediate descendent, of the abstraction a in the value tree. The new value obtained through perturbation is a sibling of the original value, i.e., both values are children of a in the value tree. This new value is substituted for the old value in the hypothesis disjunct, and an instance that matches this description and has random values for unspecified features is requested.

Let us assume the algorithm is testing abstraction a. For each hypothesis h, suppose perturbing the value of f in all disjuncts of h yields only instances whose class is the same as that of h. Then nodes in the value tree below a do not seem to be useful for distinguishing positive from negative instances (though this assumption might later be proven wrong). Thus, the abstraction to a is permissible and can therefore be made by the bias adjuster. In other words, the biasing assumption test for abstraction a is satisfied. On the other hand, if an instance of a different class is generated, the abstraction cannot be made because the biasing assumption test is not satisfied.

Because we assume the hypotheses begin with the language of the instances, which consists of leaf values in the value trees, this algorithm tests abstractions one value tree level at a time, beginning one level up from the leaves. For example, when using the "material" tree of Figure 1, the language shift to "alloy" would be tested before the language shift to the root node "any material".

We can now see how this algorithm corresponds to the formal definition of $IRR\text{-}COH(a,TC,t)$ in Section 2. For each disjunct d of hypothesis h, we let x (from the assumption definition) be any instance covered by d. We also let v_1 through v_n be the feature values present in d. To perturb the value of feature f_i, our algorithm selects an a_k for the sibling value. If the substitution of a_k into d yields a new instance whose classification differs from that of x, the assumption being tested does not hold. On the other hand, if the

substitution of a_k's into *every* disjunct of h yields only instances of the same class as x, our assumption passes the test and is considered to hold. The perturbation values a_k are children of node a in a value tree, where a is the abstraction being tested. If a is the root node of the value tree, our algorithm tests the irrelevance of f. Otherwise, our algorithm tests the cohesion of a.

An upper bound on the number of bias strengthening queries generated by this algorithm is $O(F * D * d)$, where F is the number of instance features, D is the maximum number of disjuncts in the two hypotheses, and d is the maximum depth of a value tree. The branching factor of the value tree is not included in the cost of this algorithm because to test each abstraction only one sibling value is requested for each disjunct.

To illustrate bias strengthening queries, suppose we are testing the feature "size", where the features and trees of Figure 1 are used. Furthermore, suppose feature "material" is considered irrelevant and has been removed from the hypothesis language, and the current hypotheses are:

POS HYP: { x |
((size(x,small) & shape(x,brick))
v
(size(x,large) & shape(x,sphere))) }

NEG HYP: { x |
(size(x,medium) & shape(x,cylinder)) }.

Then a bias strengthening query to test the abstraction to "any size" is formed by using the first disjunct of the positive hypothesis to request a **medium brick** (or **large brick**) that has a randomly chosen value for "material". Then the second disjunct of the positive hypothesis is used to request a **small sphere** (or **medium sphere**), and the only disjunct of the negative hypothesis is used to request a **small cylinder** (or **large cylinder**), each with randomly chosen values for "material". If the first two instances are positive and the third instance is negative, then "size" is considered irrelevant and is removed from the hypothesis language by the bias adjuster. An abstraction is created when "size" disappears from the hypothesis language. On the other hand, if any of the requested instances has a different classification than the hypothesis from it was derived (e.g., the first instance is negative), then the abstraction is not created and "size" remains in the hypothesis language.

3.2 Bias Weakening Queries

Because the algorithm of Section 3.1 does not exhaustively test the biasing assumptions (e.g., only *one* sibling value per disjunct is tested before creating an abstraction), abstractions made after running this algorithm cannot be guaranteed to be correct. Therefore, bias weakening queries are needed to retest the abstractions after a prediction error in case the prediction error is due to an incorrect abstraction rather than a generalization error. These queries perturb the values of the description of the instance for which a wrong prediction has been made to isolate erroneous abstractions that might have caused the error.

Suppose a wrong prediction is made on a new instance i, and H is the hypothesis whose class differs from that of i. Then the following is our algorithm for generating bias weakening queries following a wrong prediction:

Form the set A of all abstractions present in the hypothesis language (and in H) that apply to i. These are the abstractions to be tested. Elements of A are feature-value pairs.
For each $(f,a) \in A$ that is not already known to be faulty do
  Let L be the set of all leaves in the value tree for f that are below a.
  For each $v \in L$ do
    (1) Substitute v for the corresponding feature value in the description of i to form a new description of i'. If i' has not yet been seen, ask the concept learner to predict then get the actual class (from the oracle) of i'. Otherwise, use the known class of i'.
    (2) If the class of i' is the same as that of i then loop again to find another element of L. Otherwise, record (f,a) as being faulty and abort this loop through the values of L to get another element of A.
  Endfor
Endfor

An abstraction applies to an instance if it has a value of feature f that is more general than the value of f in the instance. In other words, the abstraction must be a value tree ancestor of the value of f in the instance.

This algorithm retests the biasing assumption associated with each abstraction a. The values for perturbation are leaf nodes in the value tree below the abstraction a being tested. A feature value of i is varied by substituting a perturbation value into the

description of i and requesting an instance of this new description from the instance generator along with the class of the requested instance. If perturbing any of the values of i causes the generation of a new instance of a different class than i, then the abstraction being tested by perturbation is faulty. This abstraction is faulty because it removes a distinction that is below it in the value tree and that is necessary for predicting target concept membership.

Although bias weakening queries do not retest *all* biasing assumptions, they rigorously retest all biasing assumptions associated with abstractions that apply to i. Therefore, they identify all biasing assumption errors that caused the error in predicting the class of i. By doing so, these queries correct the bias in a way that enables the concept learner to regain consistency and completeness with respect to all previous instances, including i. We consider this testing to be rigorous because *all* descendent (leaf) values are tested until a value is found that disallows the abstraction. If none is found, the concept learner regains consistency and completeness without bias shifts.

Similarly to Section 3.1, we can see how this algorithm corresponds to the formal definition of $IRR\text{-}COH(a,TC,i)$ in Section 2. This algorithm alters the value of f in the new instance i for which a wrong prediction has been made to create queries that request new instances. The algorithm then tests whether these newly-created instances have the same classification as i. The instance i plays the role of x in the definitions, and the perturbation values are the a_k's. If the abstraction a being tested is a root node value of a value tree, this algorithm tests the irrelevance of f. Otherwise, the algorithm tests the cohesion of a.

A is the set of all abstractions that apply to i. If F is the number of instance features, then the maximum size of A is F. This is because, for each instance feature f, only one ancestor (in the value tree) of the value of f in i will be in the hypothesis language at a particular time, and there are at most F features in the hypothesis language. In other words, for each f, there exists at most one $(f,a) \in A$. Furthermore, for each $(f,a) \in A$, this algorithm tests all leaf node descendents of a. There are at most b^d of these descendents, where d is the maximum depth and b is the maximum branching factor of any value tree. Therefore, an upper bound on the number of queries generated by this algorithm is $O(F * b^d)$. This algorithm is executed for each instance i for which the concept learner makes a wrong prediction.

To illustrate bias weakening queries, we continue with the example in Section 3.1. Suppose the bias strengthening queries cause "size" to be considered irrelevant and removed from the hypothesis language. The hypotheses are now:

POS HYP: { x | ((shape(x,brick)) v
(shape(x,sphere))) }

NEG HYP: { x | (shape(x,cylinder)) }.

If a **large copper brick** is not among the known instances, and a counterexample query requests one, and this instance is negative, then the concept learner will incorrectly predict the class of this instance. Bias weakening queries now perturb the description of this instance to determine the source of the prediction error. Perturbation to test the abstraction to "any size" might result in a request for an instance that is a **small copper brick**. If this example is positive, the assumption that "size" is irrelevant for distinguishing target concept membership is incorrect. If this instance is negative, the next query might be a request for a **medium copper brick**.

After bias weakening queries identify incorrect biasing assumptions, the bias adjuster weakens the bias to correct it. In the example just described, "size" would have to be restored to the hypothesis language to distinguish the **small copper brick** that is positive from the **large copper brick** that is negative. After the bias has been corrected, the concept learner relearns the instances to reform the hypotheses with the new language bias. The bias weakening queries enable bias weakening to be minimized because they identify the incorrect biasing assumptions. *All* assumptions not proven incorrect (in the past or the present) can be assumed to hold and can therefore be preserved during relearning.

3.3 Order of the Queries

The order for selecting the queries is as follows. The first query is a random instance query; the next query is a bias strengthening query. Bias strengthening queries continue as the default unless one of the following holds: (1) *complacency* occurs; (2) the concept learner makes a prediction error; or (3) no bias strengthening query can be formed.

Complacency is defined to occur when the concept learner has made four consecutive, correct predic-

tions.[1] A string of correct predictions indicates either that the concept has been learned or that counterexamples to the current hypotheses should be sought. Since it is not possible to know for certain whether the correct concept has been learned, counterexample queries occur in response to complacency. Bias weakening queries are the response to prediction errors. Once the diagnosis performed by these queries is completed, the bias strengthening queries resume until complacency occurs. If none of the other queries can be formed, random instance queries are generated. Counterexample queries can be formed only if they generate unseen instances that are covered by the current hypotheses. Bias strengthening queries can be formed only if they generate unseen instances that are *not* covered by the current hypotheses.

4 Results and Cost/Benefit Analyses

In this section, we summarize previously published empirical results. We then explain these results from three perspectives: system bias appropriateness, a query cost analysis, and a query benefit analysis. Finally, we present an example that illustrates why our approach is effective.

4.1 Empirical Results

We have added an implementation of our approach to bias testing and shifting to an incremental concept learner to form a system called PREDICTOR (Gordon 1990; 1992). In the experiments of (Gordon 1992), PREDICTOR's performance is compared with that of a baseline system called Iba's Algorithm Concept Learner (IACL), which is based on an algorithm from (Iba 1979). PREDICTOR is built on top of IACL by extending IACL's bias shifting capabilities. PREDICTOR is identical to IACL in all ways except two: the former system tests the bias prior to bias shifting whereas the latter does not, and the former system prefers a stronger bias than the latter system. Both systems consult an oracle to answer membership queries. IACL's membership queries are requests for randomly chosen instances. Also, both systems can shift the bias. However, IACL, like ID3 (Quinlan 1986) and a number of other concept learners, interleaves hypothesis selection and term (feature) selection. For IACL, bias shifting is not a deliberate, high-priority task as it is for PREDICTOR.

[1] The number of correct predictions needed was chosen by empirical tests.

The empirical experiments of (Gordon 1992) demonstrate that when its method is appropriate, PREDICTOR produces an order of magnitude improvement in the rate of convergence to the target concept and its negation over IACL. The empirical experiments of (Gordon 1990) demonstrate that, when appropriate, PREDICTOR has a better convergence rate than all the other systems with which it has been compared (IACL, a variant of ID3, and a version of AQ described in Michalski et al. 1986). If inappropriate, however, PREDICTOR produces a performance degradation with respect to IACL and the other systems. PREDICTOR's method is appropriate precisely when one would expect it to be - when bias shifting is the most expedient action to take to learn the target concept, i.e., there is a large disparity between the instance language and the language in which the target concept can be expressed most succinctly.

It is easy to see how this disparity would be likely to occur in many real-world learning situations. Most objects are described in terms of primitive features. It is reasonable to expect that a knowledge engineer, who is familiar with the domain in which concept learning will occur, would be aware of a number of potentially useful abstractions but would not be certain which abstractions are relevant for learning the concept. Therefore, this engineer might have the system begin with the known primitive features, but provide the system with potential abstractions. When provided with a set of potentially useful value trees, PREDICTOR's queries can isolate those abstractions which are correct and thereby expedite the learning process.

From an experimental study described in (Gordon 1992), we have learned that PREDICTOR has a synergistic effect between its bias shifting method and its bias tests. The system's bias shifting method promotes this synergy because few bias strengthening queries are required before an abstraction occurs. (Recall from Section 3.1 that the number is proportional to the number of hypothesis disjuncts.) Furthermore, the system minimally weakens the bias after a set of bias weakening queries (see Gordon 1992). In both cases, PREDICTOR is able to capitalize on the query results to gain and maintain a strong bias.

4.2 System Bias Appropriateness

In this paper, we use the term "bias" synonymously with "hypothesis language bias". This is the bias that PREDICTOR adjusts. In this section, we will discuss a meta-level bias, namely, the *system*

bias. To avoid confusion with the hypothesis language bias, hereafter, we refer to the system bias as the *system policy* or simply the *policy* as in (Provost & Buchanan 1992). At some level, all concept learning systems have some form of fixed system policy. For example, even the most adjustable system cannot implement all possible bias adjustment methods. The choice of bias adjustment methods is a fixed system policy. No one policy can be best for learning every target concept. Therefore, we need to develop an understanding of the appropriateness of system policies for learning different classes of concepts.

PREDICTOR's policy is its implicit assumption that abstraction is appropriate to try (by strengthening the language bias) and to retain (by minimally weakening the language bias) as much as possible. Strengthening and minimally weakening the bias are appropriate when they are the correct actions to take in the context of the current instance language and target concept.

PREDICTOR's bias strengthening and weakening queries, which form the majority of the system's queries, are geared entirely toward gathering information for bias shifting. This system uses its queries to reorder the instances to increase its information gain about the hypothesis language bias early in the learning process. Bias shifting is its priority. If this priority matches the task, PREDICTOR usually outperforms all other systems with which it is compared. This is because once PREDICTOR's queries have achieved their goal of a strong correct bias, the number of instances required to converge on the target concept is often significantly reduced by this bias shift (see Section 4.4).

Although IACL can adjust the hypothesis language, PREDICTOR generally far outperforms IACL when bias shifting is important. This is because IACL's policy places a higher priority on finding less general hypotheses and also on maintaining hypothesis consistency and completeness with previous training instances than it does on bias shifting (see Gordon 1992). This system only shifts the bias when it decides this is the best way to achieve its other priorities. IACL's queries are requests for randomly-generated instances. Therefore, this system does not favorably order the instances for gathering information about the bias. The other systems with which PREDICTOR has been compared have problems that are similar to IACL's.

In the next two sections, we present cost/benefit analyses that further explain the empirical results.

4.3 Query Costs

The upper bounds on PREDICTOR's queries, presented in Sections 3.1 and 3.2, seem somewhat high. Why does this system generate fewer queries (converge earlier) than IACL when bias shifting is appropriate for learning the target concept? Why does it generate more queries than IACL when it is inappropriate? In this section, we present a rough cost comparison between the number of queries generated by PREDICTOR and the number generated by IACL. To simplify our analysis and avoid a confusion between the effects of generalization and those of abstraction, let us suppose that no generalization is needed for learning the target concept.[2]

In Section 3.1, we show that an upper bound on the number of bias strengthening queries is $O(F * D * d)$, where F is the number of instance features, D is the maximum number of hypothesis disjuncts, and d is the maximum depth of any value tree. According to Section 3.2, an upper bound on the number of bias weakening queries generated in response to each wrong prediction on a new instance is $O(F * b^d)$, where b is the maximum branching factor of any value tree.

Suppose W wrong predictions are made prior to convergence.[3] Then an upper bound on the number of bias weakening queries generated to resolve W wrong predictions is $O(W * F * b^d)$. Therefore, an upper bound on the total number of bias strengthening and weakening queries prior to convergence to the target concept and its negation is $O((F * D * d) + (W * F * b^d))$.

The total number of PREDICTOR's queries also includes random instance and counterexample queries. However, in this analysis we ignore the cost of these two types of queries because at most four random instance queries occur before the counterexample queries activate (see Section 3.3), and the counterexample queries typically do not take long to find a counterexample. Once a counterexample is found, the bias weakening and then strengthening queries resume.

---

[2] This simplification does not significantly affect our analysis.

[3] PREDICTOR also executes bias weakening queries at another time besides when it makes a wrong prediction. Nevertheless, this does not significantly alter our cost estimates. We therefore omit a discussion of this topic. See (Gordon 1992) for details.

IACL's bias shifts are triggered by the order of the training instances (see Gordon 1992). A serendipitous order will enable the system to make the correct abstractions early. However, since this order is random, it cannot be guaranteed to be helpful. Furthermore, as mentioned in Section 4.2, IACL's bias shifts occur when IACL decides this is the best way to achieve its other goals. The main problem with this approach is that it does not offer much help to a system that needs to recover from incorrect abstractions. Often, what is really an abstraction problem is treated as a generalization problem by the system. As a result, incorrect abstractions often linger, thereby preventing correct abstractions from being made. In the worst case, this would cause the number of queries required for convergence to the target concept and its negation to equal the total number of instances, which is $O(b^{d*F})$, where b^d is the maximum number of leaf nodes in any value tree.

Comparing the upper bounds for the two systems, we note that as F increases, IACL's performance should degrade much more rapidly than that of PREDICTOR. Furthermore, the upper bound cost for IACL depends on the data structures but not on the target concept. On the other hand, W and D in the formula for PREDICTOR's upper bound cost depend, at least in part, on the target concept.

D depends almost entirely on the target concept. W, on the other hand, depends both on the target concept and the bias appropriateness. If we assume the data structures and target concept are fixed, and we wish to analyze the effectiveness of bias shifting, then we need to focus on the W component of the cost formula. In particular, as W approaches zero, PREDICTOR's cost upper bound approaches a polynomial. W tends toward zero as a greater proportion of the biasing assumptions made by PREDICTOR are correct, e.g., most features are irrelevant and therefore the irrelevance assumption holds frequently. This is precisely the situation in which empirical experiments have shown PREDICTOR outperforms IACL.

Likewise, when most of the biasing assumptions are incorrect, W can become very large. In the worst case, we would have the same situation as we have with IACL, where all instances would have to be seen to learn the target concept. Again, empirical experiments confirm this analysis.

Our analysis of IACL's upper bound cost is much like that of the other systems with which PREDICTOR has been compared because these systems are also not designed for bias testing and shifting. Their system policies favor other tasks.

4.4 Query Benefits

One of the goals of our method for bias testing and shifting is to reduce the number of features in the hypothesis language. Relevant results from computational learning theory can provide a rough estimate of the benefits of strengthening the bias in this way.[4] The *sample complexity* is the number of instances required to converge to the target concept. (Haussler 1988) has shown that sample complexity relates directly to the Vapnik-Chervonenkis (VC) dimension of a hypothesis space, which is a measure of the expressiveness of the hypothesis language. A less expressive language implies a stronger bias and a lower VC-dimension. Haussler measures convergence in the Probably Approximately Correct (PAC) framework of learnability, which assumes the error $\varepsilon > 0$ and the confidence $(1 - \delta) < 1$. In this framework, a concept is expected to be learned approximately with high probability.

According to (Haussler 1988), given a fixed ε and δ the minimum sample complexity is directly proportional to the VC-dimension. Furthermore, if the concept hypothesis is in k-DNF (which is true for many concept learners), then

$$VC\text{-}dim(H) \leq 4ks \log(4ks\sqrt{n}),$$

where H is the hypothesis space, n is the number of features in the instance language, s is a bound on the number of terms (disjuncts), $k \leq n$, and $s \leq \binom{n}{k}$.

We can now apply this theoretical estimate of sample complexity to partially explain the empirical results summarized in Section 4.1. In our framework, empirical performance is measured in terms of *absolute convergence* to the target concept, rather than PAC convergence. Absolute convergence implies the error ε is 0 and the confidence $(1 - \delta)$ is 1. In other words, we require that the target concept be learned precisely. We also require that the negation of the target concept be learned precisely for absolute convergence to hold. (These requirements are meaningful for applications where the cost of making an error is high.)

In our framework, the maximum number of literals per term, k, is equal to the number of (relevant)

[4] Only a "rough" estimate can be provided because the computational learning theory makes assumptions, such as instance selection from a fixed distribution, that are not met in our framework.

features in the hypothesis language. Also, we use a different bound on s for our framework because when there are irrelevant features, n no longer affects the number of disjuncts. If we assume there is a maximum of v values for any of the k relevant features, $s \leq v^k$. When PREDICTOR discovers irrelevant features and strengthens the bias (by reducing k), Haussler's inequality (with the new upper bound on s) predicts that the system will reduce the VC-dimension and thus reduce the sample complexity. With the new bound on s, Haussler's inequality also predicts a reduction in sample complexity when v is reduced by abstractions that are made based on cohesion assumptions. Furthermore, PREDICTOR's ability to minimally weaken the bias enables it to *retain* a hypothesis space with a low VC-dimension.

In summary, when bias strengthening is desirable, the cost of using queries to gather information about the bias can be offset by the benefit of a reduction in the sample complexity gained by having a stronger bias. In Section 4.3, we showed how in this situation the costs are also reduced. So, when appropriate, PREDICTOR's method can yield lower costs and increased benefits and a better performance in comparison with IACL and other systems. When inappropriate, the costs increase and the benefits are reduced and system performance degrades with respect to the performance of IACL and other systems.

4.5 Illustrative Example

Let us examine a very simple illustration of how the queries and bias shifts together can result in a synergy that reduces the convergence rate. We will focus primarily on the value of the bias weakening queries and minimal bias weakening. Assume we have two concept learners, CL-Q and CL. They differ only in that CL-Q uses the bias weakening queries and performs minimal bias weakening, whereas CL does not. CL's method for bias weakening is to make the hypothesis language equal to the instance language. CL's motivation for bias weakening is the same as that of CL-Q, namely, to resolve prediction errors. Other than the bias weakening queries of CL-Q, we assume both systems request random instances from an oracle.

For simplicity, let us further suppose that both systems are given a strong bias beforehand by the system implementor and their only bias shifting task is to weaken the bias if they discover (by a wrong prediction) that the bias is incorrect. Also, for simplicity, we assume neither system generalizes. They only learn concepts using bias adjustments.

The data structures given to both systems are the value trees of Figure 1, except that feature "size" is now restricted to having the values "small" and "large", and the value "curved-solid" has no child values. The target concept, as in Section 1, states that **small bricks** are positive and instances of any other description are negative. Finally, both systems begin with a hypothesis language bias which states that "shape" is the only relevant feature.

Both systems begin by requesting the same two instances: a **small aluminum brick**, which is positive, and a **large steel curved-solid**, which is negative. The current hypothesis now held by both systems is:

POS HYP: { x | (shape(x,brick)) }

NEG HYP: { x | (shape(x,curved-solid)) }.

If the next instance requested by both systems is a **large bronze brick**, and it is negative, both systems will make a prediction error, which triggers bias weakening.

CL-Q responds to the prediction error by using its bias weakening queries. It requests one instance to retest the abstraction to "any size" - a **small bronze brick**, which is positive. Since the class is different than that of the **large bronze brick**, CL-Q decides the abstraction to "any size" is faulty and removes it. CL-Q then requests a **large aluminum brick**, a **large copper brick**, a **large brass brick**, and a **large steel brick** to retest the abstraction to "any material". Since these instances have the same class as the **large bronze brick**, this abstraction is considered permissible. CL-Q now minimally weakens the bias to obtain the hypotheses:

POS HYP: { x |
(size(x,small) & shape(x,brick)) }

NEG HYP: { x |
((size(x,large) & shape(x,brick))
v
(size(x,large) & shape(x,curved-solid))) }.

The system now needs one more instance, a **small curved-solid** of any material, to converge on the target concept and its negation precisely. The odds are high that a random choice will request this instance before all instances have been seen. (Note that if the bias strengthening queries were used, this would be the next instance requested.)

After making a prediction error on the large **bronze brick**, CL behaves differently. It does not make the extra five queries that CL-Q made. Neither does it have the extra information CL-Q obtained. Rather than *minimally* weakening the bias, CL weakens the bias to the instance language. CL, therefore, requires *all* remaining 27 instances to precisely learn the target concept and its negation. In this case, it is clear that the information gained by CL-Q from the queries have offset their cost.

5 Related work

Our approach is related to theoretical research on irrelevance (e.g., Subramanian 1989) and relevance (Grosof & Russell 1989). Subramanian's definition of irrelevance is similar to ours. However, her definition is tailored for reformulating a problem solver's language to increase the problem solver's efficiency. Our definition, on the other hand, is tailored for incremental concept learning. Grosof and Russell have created a theory of shifting bias as nonmonotonic reasoning. They use the notion of relevance to motivate bias shifts. Prior to learning, biases are ordered from stronger (i.e., having fewer relevant components) to weaker (i.e., having more relevant components). Learning begins with a strong bias and shifts to weaker biases as needed. Unlike the bias shifts described here, Grosof and Russell's bias shifts are not motivated by an analysis of biasing errors and therefore they are unable to guarantee that the bias can be minimally weakened.

Our approach is also related to approaches that form abstractions based on equivalence classes. If cohesion holds for a value a of feature f, then a forms an equivalence class in terms of the target concept membership of instances having this value of feature f. Kokar's COPER is an example of another system that uses equivalence classes for concept learning (Kokar 1990). COPER uses the concept of invariance and an expectation of equivalence classes to indicate when constructive induction is needed. Constructive induction is the dynamic generation of new features. PREDICTOR could use a similar approach to COPER's to decide when to invent new abstractions. For example, if cohesion does not hold for some abstraction a in a value tree, this could be considered an indication of the need to split a into two separate abstractions for which cohesion does hold.

The use of active learning in our approach is related to literature on queries for concept learning, both theoretical (e.g., Angluin 1988) and experimental (e.g., Sammut & Banerji 1986; Muggleton 1987). Our approach is most similar to that of the few systems that query an oracle *and* shift the bias. Notable examples include the MARVIN system of Sammut & Banerji (1986), Gross's CAT (Gross 1991), Muggleton's Duce (1987), and the CLINT system described in De Raedt & Bruynooghe (1990). MARVIN shifts the bias by learning the definitions of new user-selected terms and then uses these new terms for further learning. This system also queries an oracle to test generalizations within the term definitions. These queries involve a form of perturbation similar to that used here. Nevertheless, MARVIN's queries are not for the purpose of deciding how to shift the bias.

CAT, Duce, and CLINT are all systems that query an oracle to make bias shifting decisions. Of all these approaches, our approach is most similar to that of CLINT, which uses irrelevance queries for bias strengthening. Nevertheless, the latter system does not use irrelevance queries to select a weaker bias. Furthermore, our approach is unlike that of CLINT and all other systems because our choice of a weaker bias is determined by bias tests that diagnose errors in such a way as to guarantee that the bias can be minimally weakened.

6 Summary

This paper presents a unique approach to bias shifting. Rather than performing unjustified bias shifts, as most concept learning systems do, we first test assumptions about the relationship between the bias and the concept being learned. We also *re*-test these assumptions in light of new, possibly contradictory evidence, i.e., a prediction error. These tests are performed with queries to an oracle. By using this approach, a system can both strengthen and minimally weaken the bias.

In addition to presenting our method for bias testing, this paper also summarizes empirical results. The empirical results are then explained by a cost/benefit analysis. Both the empirical and analytical results indicate that when bias strengthening is desirable, the query costs are lower and the benefits of a stronger bias, namely, a reduced sample complexity, are increased. Likewise, when bias strengthening is not appropriate, the query costs are increased and the benefits are reduced. The low costs and increased benefits have produced a large performance gain over other systems when bias strengthening is appropriate, and the high costs and decreased benefits have produced a performance loss with respect to other systems

when bias strengthening is inappropriate.

Acknowledgements

I thank Bill Spears for his constant help, encouragement, and insightful suggestions throughout this entire project. Also, thanks to Jaime Carbonell, Don Perlis, and Chinoo Srinivasan for their guidance, and to Alan Schultz for his helpful comments on previous drafts.

References

Angluin, D. (1988). Queries and concept learning. *Machine Learning, 2*, 319-342.

De Raedt, L. & Bruynooghe, M. (1990). Indirect relevance and bias in inductive concept learning. *Knowledge Acquisition, 2*, 365-390.

Gordon, D. (1990). *Active bias selection for incremental, supervised concept learning.* Ph.D. thesis, University of Maryland, College Park. Also (Technical Report UMIACS-TR-90-60 CS-TR-2464) University of Maryland, Department of Computer Science.

Gordon, D. (1992). *Actively testing and minimally weakening the inductive bias.* (NCARAI Technical Report AIC-91-011). Also, submitted to *Machine Learning.*

Grosof, B. & Russell, S. (1989). *Shift of bias as non-monotonic reasoning.* (Technical Report 14620) IBM, Yorktown Heights.

Gross, K. (1991). *Concept acquisition through attribute evolution and experiment selection.* Ph.D. thesis, Carnegie-Mellon University, Pittsburgh.

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

Iba, G. (1979). *Learning disjunctive concepts from examples.* (A.I. Lab Memo 548). Massachusetts Institute of Technology.

Kokar, M. (1990). Semantic equivalence in concept discovery. In D. Benjamin (Ed.), *Change of Representation and Inductive Bias.* Boston: Kluwer.

Michalski, R., Mozetic, I., Hong, J. & Lavrac, N. (1986). *The AQ15 inductive learning system: An overview and experiments.* (Technical Report UIUCDCS-R-86-1260). University of Illinois, Department of Computer Science.

Mitchell, T. (1980). *The need for biases in learning generalizations.* (Technical Report TR CBM-TR-117). Rutgers University, Department of Computer Science.

Muggleton, S. (1987). Duce, an oracle based approach to constructive induction. In *Proceedings of the Tenth International Conference on Artificial Intelligence.* Milan, Italy: Morgan Kaufmann.

Porter, B. & Kibler, D. (1986). Experimental goal regression: A method for learning problem-solving heuristics. *Machine Learning, 1*, 249-286.

Provost, F. & Buchanan, B. (1992). Inductive policy. In *Proceedings of the Tenth National Conference on Artificial Intelligence.* San Jose: Morgan Kaufmann.

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

Rendell, L. (1990). Feature construction for concept learning. In D. Benjamin (Ed.) *Change of Representation and Inductive Bias.* Boston: Kluwer.

Sammut, C. & Banerji, R. (1986). Learning concepts by asking questions. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Spears, W. & Gordon, D. (1991). Adaptive strategy selection for concept learning. *Proceedings of the First International Workshop on Multistrategy Learning.* Harpers Ferry: George Mason University.

Subramanian, D. (1989). *A theory of justified reformulations.* Ph.D. thesis, Stanford University, Stanford.

Utgoff, P. (1986). Shift of bias for inductive concept learning. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos: Morgan Kaufmann.