An Investigation of Bias Shifting and Bias Shifting Methods

Shawn Robert Wolfe

CIS-TR-92-23
November 1992

Abstract

Algorithms that learn by means of induction typically employ some sort of bias to make the task of learning computationally feasible. We examine in depth one inductive learning algorithms, the candidate elimination algorithm, and examine the effects of the choice of bias has upon performance. We consider the difficulties that occur when a poor bias is chosen, and present a framework from which we can study biases. We present bias shifting as one method of overcoming these difficulties, and present several bias shifting methods.

Department of Computer and Information Science
University of Oregon

AN INVESTIGATION OF BIAS SHIFTING AND BIAS SHIFTING METHODS

by

SHAWN ROBERT WOLFE

A THESIS

Presented to the Department of Computer and Information Science
and the Graduate School of the University of Oregon
in partial fufillment of the requirements
for the degree of
Master of Science

August 1992

An Abstract of the Thesis of

Shawn Robert Wolfe for the degree of Master of Science in the Department of Computer and Information Science to be taken August 1992

Title: AN INVESTIGATION OF BIAS SHIFTING AND BIAS SHIFTING METHODS

Approved: _____
Dr. Arthur M. Farley

Algorithms that learn by means of induction typically employ some sort of bias to make the task of learning computationally feasble. We examine in depth one inductive learning algorithm, the candidate elimination algorithm, and examine the effects of the choice of bias has upon performance. We consider the difficulties that occur when a poor bias is chosen, and present a framework from which we can study biases. We present bias shifting as one method of overcoming these difficulties, and present several bias shifting methods.

VITA

NAME OF AUTHOR: Shawn Robert Wolfe

PLACE OF BIRTH: Cleveland, Ohio

DATE OF BIRTH: May 24, 1968

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon
Iowa State University

DEGREES AWARDED:

Master of Science, 1992, University of Oregon
Bachelor of Science, 1990, Iowa State University

AREAS OF SPECIAL INTEREST:

Artificial Intelligence, Machine Learning, Expert Systems

PROFESSIONAL EXPERIENCE:

Research Specialist, AI Research Branch, NASA Ames Research Center, August 1992 to current

ACKNOWLEDGMENTS

The author gratefully acknowledges the guidance and suggestions of Dr. Arthur Farley, whose useful contributions helped shape this research project. Honorable mention also goes to the dedicated faculty and staff at the University of Oregon who's zeal for their own work inspired the research herein. Last but in no way least, gratitude goes to Christina O'Guinn, whose undying patience and gentle understanding helped keep the fires of ambition and creativity alive through the duration of this project.

TABLE OF CONTENTS

CHAPTER I

BIAS AND THE CANDIDATE ELIMINATION ALGORITHM

Machine learning is an important area in artificial intelligence. It is a broad field, covering discovery, speed-up learning, and inductive learning. We will consider only one paradigm of learning, inductive learning, which is concept learning from a series of pre-classified examples. It is the goal of the learner to be able to classify any instance within the domain as positive or negative instances of the concept. The learning occurs by means of induction, because the concept is learned by examining the concrete examples, rather than some more abstract knowledge.

In our paradigm, the learner will be given a series of pre-classified examples, or *training instances*, in the domain of the concept to be learned. Each example will be described in terms of a pre-determined set of attributes. That is, the instances are described by a set of features, each with a set of acceptable values; the method of describing these instances remains invariant. No knowledge of the domain, outside from the examples given, will be available for use by the learning program. Therefore the method used must rely solely on the similarities between instances to learn the target concept, and is therefore called a *similarity-based method*.

There are several significant similarity-based methods in existence today. Among these are the decision-tree family of algorithms, founded by J.

Ross Quinlan [8]; the star methodology, by Ryszard Michalski[4]; the family of neural net learning algorithms, which began with perceptron of F. Rosenblatt [9]; and the candidate elimination algorithm, as developed by Tom Mitchell [5]. Each differs in its approach. The decision-tree algorithms use statistical methods to create a tree which is traversed in order to classify an instance. The star methodology creates a description of the target concept by grouping positive instances of the concept into increasingly large groupings. The perceptron is particular type of neural net that can learn any linearly separable concept. The candidate elimination algorithm eliminates competing hypotheses from a set of plausible hypotheses called a *version space*.

We will only examine the candidate elimination algorithm in our discussion of bias. The major advantage the candidate elimination algorithm has over the other similarity based method is that it explicitly handles bias, and so it facilitates a study of bias by allowing the bias to be manipulated. The candidate elimination algorithm is also unlike several other well-known similarity-based methods because it learns incrementally, i.e., one instance at a time. This makes it possible to make claims about the goal concept before all the training instances have been processed. Furthermore, it is appealing from a theoretical standpoint because of its use of version spaces, a concept linked to bias which shall be defined later.

Generalization and Bias

It becomes evident from studying similarity-based learning methods that the power to learn effectively comes from the ability to generalize [6]. That is, the goal of the learning system is to come up with a reasonable

representation of the concept, which is an accurate generalization of the given training instances, i.e., the generalization does not contradict any information known about the target concept. What is meant by reasonable may change from situation to situation, but typically a reasonable representation must be cheap to store and cheap to match against training instances. It is not satisfactory to merely record the training instances, but rather some sort of conclusion, or generalization, must be drawn from these instances.

Bias is what allows a similarity-based learning method to generalize effectively. Simply put, bias is any sort of method for choosing one generalization over another. Though at first it may seem desirable to consider all possible generalizations, closer examination reveals that this is often computationally intractable. We can reformulate the concept learning problem as the search for a partitioning of the learning domain that satisfies certain properties. Let D be the domain that the training examples will be drawn from, let T be the set of training instances considered. Assume that T is a subset of D, which is certainly true in any normal learning situation. Define U to be D - T, which corresponds to the set of unseen instances. The goal of learning can be restated as finding a partitioning of D that partitions D into two partitions, P and N, such that all positive instances are contained within P and all negative instances are contained within N. Define a hypothesis H to be any partitioning of D into two partitions, P_H and N_H. Note that since T is a subset of D, H also partitions T into a partition of positive instances and a partition of negative instances (though either partition may be empty). Let C be the hypothesis that partitions D correctly, i.e., P_C is exactly the set of positive instances within D and N_C is exactly the set of negative instances within D.

H is said to be *consistent with respect to* T (or simply consistent) when H partitions T just as C partitions T. More formally, H is consistent w.r.t T iff $(P_H \cap T) \equiv (P_C \cap T)$ and $(N_H \cap T) \equiv (N_C \cap T)$. Every consistent hypothesis is indistinguishable from C by examination of T. It follows from the definition of consistency that all consistent hypotheses must partition T in exactly the same way. However, there are no constraints on how to partition U, since nothing is known about the classification of any of the instances contained in U. Therefore, there exists a consistent hypothesis H for every partitioning of U into P_H and N_H. Each instance in U can be placed either into P_H or N_H. Therefore there are $2^{|U|}$ possible partitionings, which results in $2^{|U|}$ consistent hypotheses w.r.t T. The impact of this result is strengthened by the realization that the cardinality of many learning domains is infinite, and compounded by the fact that a single generalization (partitioning) may be expressed in several ways.

Let us consider a simple domain. Our task will be to learn the class of "tasty mushrooms" from the domain of mushrooms. We will use a highly simplified description of mushrooms, describing each in terms of *weight* (in grams) and *color*. Note that *weight* is a numerically valued attribute, and that *color* is a nominally valued attribute. That is, *weight* is expressed by a number, which implies a total ordering of the possible values of *weight*, whereas *color* is expressed in terms of a name, and in this case there is no ordering defined over the possible values[1]. Let us make the simplifying assumptions that *weight* is given in multiples of 0.1 and ranges from 4.0 g to 8.0 g, and color is one of the set {red, green, blue, yellow, purple, orange, brown, gray, white, black}. It is easy to see that there are 40 possible values for

[1]Of course, an ordering could be established in terms of the color spectrum, but we will assume no ordering is defined in our example.

weight, 10 possible values for colors, and 400 unique descriptions of mushrooms. This gives a candidate set of 2^{400}, which is on the order of magnitude of 10^{120}, many times the number of atomic particles in the universe. Clearly, this makes the size of the unbiased candidate set intractable. Considering that the description scheme of mushrooms given above is not particularly complete, it becomes obvious that bias is necessary to make the size of the candidate set tractable for most realistic learning domains.

As stated earlier, bias is any method of choosing certain generalizations over another. Nonetheless, two sorts of biases predominate in the existing literature [6] [8] [10]: *preference biases* and *restricted hypothesis space biases*. A preference bias is an ordering over the set of consistent hypotheses. This ordering may be total or partial. A learning program with a preference bias would consider only certain generalizations by choosing only the highest ordered generalizations. A restricted hypothesis space bias limits the generalizations to be considered before learning takes place. A typical method of creating a restricted hypothesis space is by choosing a concept description language that is itself biased. That is, if it is not possible to represent certain subsets from the domain with the given concept description language, some subsets will be immediately eliminated from the set of possible competing subsets. When a concept description language is biased in such a way that it cannot express all possible partitionings of D into P and N, it is said to be *incomplete*. This is the type of bias used by Mitchell's candidate elimination algorithm. A bias that severely limits the hypothesis space is said to be a strong bias, and a bias that does not significantly limit the hypothesis space is a said to be a weak bias. If bias A has a smaller hypothesis space than bias B, bias A is stronger than bias B and bias B is weaker than bias A. Note that this

does not imply that the hypothesis space induced by bias A is a subset of the hypothesis space induced by bias B, although that may be the case.

We return to the tasty mushroom domain. We consider a simple concept description language. All hypotheses will be described in terms of a single value for color and as a continuous range of values for weight. That is, a hypothesis will be of the form [**color**: c; **weight**: i to j], where $c \in$ {red, green, blue, yellow, purple, orange, brown, gray, white, black}, i is some multiple of 0.1 in the range [4.0..8.0], and j is some multiple of 0.1 in the range of [i..8.0]. This gives a 10 possible values for color and $\sum_{m=1}^{40} m = 820$ possible values for weight, yielding 8,200 hypotheses in the candidate set. Since this is of order of magnitude 10^3, we have reduced the number of hypotheses in the candidate by a factor of 10^{117}, a considerable reduction!

Version Spaces and the Candidate Elimination Algorithm

Now that we have defined bias and consistency with respect to a set of training instances, we can concisely state the definition of a version space. Given a biased concept description language and a set of training instances T, the version space with respect to T are all hypotheses that are consistent w.r.t. T and are expressible in the biased description language[2]. The candidate elimination algorithm maintains a representation of the version space, and updates it appropriately for every newly encountered training instance. Any hypothesis that does not properly classify the newest instance is eliminated

[2]For the remainder of this thesis, it will be a unstated assumption that only hypotheses that are expressible in the concept description language will be considered.

from the version space. A very high-level description of the algorithm follows.

```
function candidate_elimination

begin
    let T = {}
    let VS = {the set of hypotheses consistent with T}
    for each new training instance t do
        T = T ∪ {t}
        for each h ∈ VS do
            if h is not consistent with T
                then VS = VS - {h}
            fi
        od
    od
    return VS
end
```

Note that when T is empty, the version space contains every hypothesis over the domain that is expressible in the concept description language. Even with a biased concept description language, this can quite a large set, and can be infinite in certain domains (e.g., it is infinite in any domain that allows one or more continuously valued attributes).

Fortunately, it is not necessary to represent each consistent hypothesis within the candidate set explicitly. It is sufficient to keep track of only the most general and the most specific consistent hypotheses in the version space.

A hypothesis h_i is more general than h_j iff $P_i \supset P_j$, where P_i is the partition of positive instances on the domain imposed by h_i, and P_j is the partition of positive instances on the domain imposed by h_j (as before). The specificity relation is defined similarly; a hypothesis h_i is more specific than h_j iff $P_j \supset P_i$, where P_i and P_j are as defined previously. For notational convenience, we will represent the relation h_i is more general than h_j as $h_i >_g h_j$, and h_i is more specialized than h_j as $h_i <_s h_j$.[3] The more general relation and the more specific relation are logical inverses, i.e., $h_i <_g h_j$ iff $h_i >_s h_j$. It is worth noting that the more general and more specific relations impose a partial ordering upon the hypotheses in the hypotheses space, but not necessarily a total ordering.

We can now redefine the version space in terms of these most general consistent hypotheses and the most specific consistent hypotheses. Let G be the set of most general consistent hypotheses w.r.t T, and let S be the set of most specific w.r.t T. Let I be the set of hypotheses such that for each $h \in I$, there exists some $g \in G$ and some $s \in S$ such that $h <_g g$ and $h >_s s$ (note that we do not insure that these hypotheses are consistent, as that will follow). We can define the version space w.r.t T to be $G \cup S \cup I$.

It is easy to prove that all hypotheses in G , S , and I are consistent w.r.t T. By definition, this is true for all hypotheses in G and S. Note that this implies that every hypothesis in G and S partition T correctly, which is to say that each hypothesis in G and S partition T exactly as C does, where C is the target concept. Since every hypothesis in I is less specific than some hypothesis in S, any hypothesis in I must partition all positive instances in T as some hypothesis in S partitions T. Since every hypothesis in I is less

---

[3]Please note that the letter of the subscript is actually meaningless, i.e., $<_g$ and $<_s$ are equivalent relations, as are $>_g$ and $>_s$.

general than some hypothesis in G, any hypothesis in I must classify no more instances in T positive than some hypothesis in G. Since each S and G partition T identically, and since I must lie between these, by the squeezing lemma, any hypothesis in I must also partition T in the same way.

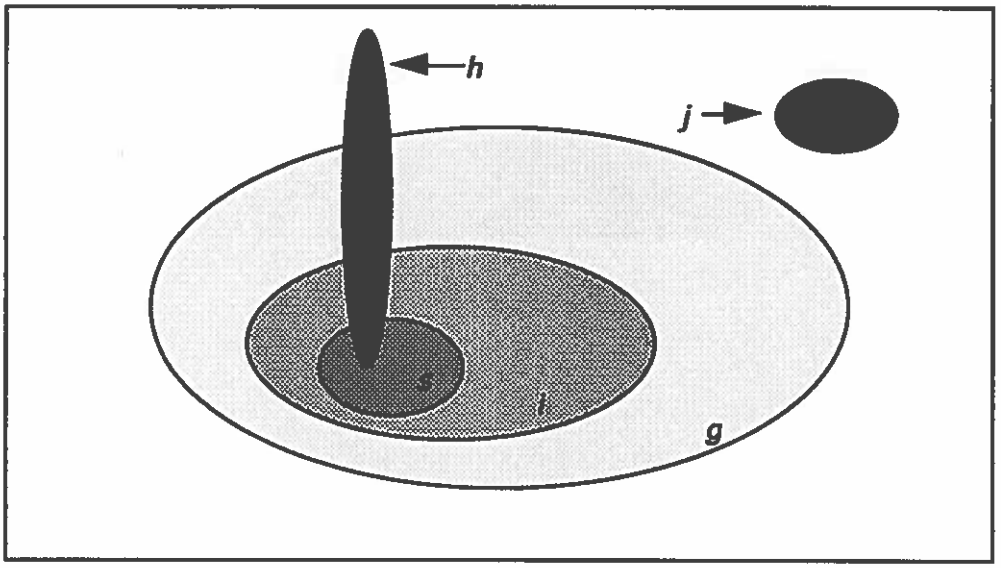


Figure 1.1: Relation of hypotheses in the hypothesis space. The area on the graph represents instances accepted by the hypothesis. Hypothesis g in more general than i, whereas i is more general than s. Hypotheses h and j are incomparable to the other hypotheses.

We still need to prove that no consistent hypothesis is excluded from $G \cup S \cup I$. Let us assume that there is a consistent hypothesis h such that $h \notin G \cup S \cup I$. Since I contains all hypotheses less general than some hypothesis in G and less specific than some hypothesis in S, h must either be more general than all hypotheses in G, more specific than all hypotheses in S, or simply incomparable to any hypothesis in G or S. Clearly, h cannot satisfy

either of the first conditions, for that would violate the definition of G and S. However, if h is incomparable to any hypothesis in G and S, it would be in both G and S, since it would be both most general and most specific. Therefore, h is self-contradictory, and does not exist. Therefore the version space w.r.t T corresponds exactly to $G \cup S \cup I$, as previously stated.

Since I is defined solely in terms of G and S, it need not be represented explicitly. Therefore we can represent a version space in terms of G and S, adopting the notation of Mellish[3], as $<G, S>$. We can now modify our description of the candidate elimination algorithm to reflect these improvements. Let us assume that the description language allows a single hypothesis that classifies all instances as positive, denoted by $\top$, and a single hypothesis that classifies all instances as negative, denoted by $\bot$. The candidate elimination algorithm follows on the next page.

```
function candidate_elimination

begin
    let G = { ⊤ }
    let S = { ⊥ }
    for each new training instance t do
        T = T ∪ {t}
        if classification(t) ≡ positive
            then G = validate(G, T)
                 S = generalize(S, T)
            else G = specialize(G, T)
                 S = validate(S, T)
        fi
    od
    return <G, S>
end
```

The functions **validate**, **generalize**, and **specialize** will be defined in more detail in chapter three. Function **validate** removes all hypotheses from the set given that are not consistent with $\mathcal{T}$. Function **generalize** returns the set of hypotheses that are consistent with $\mathcal{T}$, no more general than any of the hypotheses in G, and no more specific than any of the hypotheses in S. Function **specialize** is analogous to **generalize** except it specializes its argument rather than generalizing it.

A Sample Run of the Candidate Elimination Algorithm

We now return once again to our mushroom domain, to consider an example of the version space algorithm at work. We will use the notation $(\langle V,G\rangle, i) \Rightarrow \langle V', G'\rangle$ to denote that version space $\langle V, G\rangle$ is transformed into version space $\langle V', G'\rangle$ after encountering training instance i. The comments interspersed with the example refer to the event immediately above the comment.

(<{(**color**: ⊤; **weight**: ⊤)}, {(**color**: ⊥; **weight**: ⊥)}>, [**color**: brown; **weight**: 4.5; **class**: +])
⇒ <{(**color**: ⊤; **weight**: ⊤)}, {(**color**: brown; **weight**: [4.5..4.5])}>

Since the first instance is positive, the *G* set remains unchanged, for its sole hypothesis is consistent with the training data. The most specific hypothesis, though, must at least accept this single positive instance, and so the *S* set contains the single hypothesis that excepts only this instance.

(<{(**color**: ⊤; **weight**: ⊤)}, {(**color**: brown; **weight**: [4.5..4.5])}>, [**color**: brown; **weight**: 7.2; **class**: -])
⇒ <{(**color**: ⊤; **weight**: [4.0..7.1])}, {(**color**: brown; **weight**: [4.5..4.5])}>

Having encountered a negative instance, the single hypothesis in the *G* set must be replaced by those immediately less general than it that are consistent with the training data. Though hypotheses that accept any instance that is not brown would be less general than (**color**: ⊤;

weight: ⊤) and would properly reject the newest instance, they would not be less general than any hypothesis in the *S* set, and therefore they are not in the new *G* set. Similarly, (**color**: ⊤; **weight**: [7.3..8.0]) is not more general than any hypothesis in the *S* set, and therefore the single element of the new *G* set is (**color**: ⊤; **weight**: [4.0..7.1]). As before, since the single element of the *S* set is consistent with the new instance, the *S* set remains unchanged.

(<{(**color**: ⊤; **weight**: [4.0..7.1])}, {(**color**: brown; **weight**: [4.5..4.5])}>,
[**color**: brown; **weight**: 5.1; **class**: +])
⇒ <{(**color**: ⊤; **weight**: [4.0..7.1])}, {(**color**: brown; **weight**: [4.5..5.1])}>

(<{(**color**: ⊤; **weight**: [4.0..7.1])}, {(**color**: brown; **weight**: [4.5..5.1])}>,
[**color**: gray; **weight**: 5.1; **class**: -])
⇒ <{(**color**: brown; **weight**: [4.0..7.1])}, {(**color**: brown; **weight**: [4.5..5.1])}>

This is the first mushroom encountered that is not brown. Since it is a negative instance, the *G* set must specialize its members such that no hypothesis accepts a non-brown mushroom (due to our bias's inability to express disjunction). Had the instance been positive, the *S* set would have to accept any color mushroom, and therefore the single element would be (**color**: ⊤; **weight**: [4.5..5.1])

(<{(**color**: brown; **weight**: [4.0..7.1])}, {(**color**: brown; **weight**: [4.5..5.1])}>,
[**color**: brown; **weight**: 5.2; **class**: -])
⇒ <{(**color**: brown; **weight**: [4.0..5.1])}, {(**color**: brown; **weight**: [4.5..5.1])}>

(<{(**color**: brown; **weight**: [4.0..7.1])}, {(**color**: brown; **weight**: [4.5..5.1])}>,
[**color**: brown; **weight**: 4.4; **class**: -])
⇒ <{(**color**: brown; **weight**: [4.5..5.1])}, {(**color**: brown; **weight**: [4.5..5.1])}>

We note that at this point the *S* and *G* sets have converged upon a single hypotheses, and so the candidate set has been reduced to a single hypothesis. From this point on, no learning can occur, regardless of subsequent instances considered. Each new instance will either be consistent with the current *S* and *G* sets, causing no change in the representation, or the bias will be revealed as insufficient, in which case learning will fail.

Difficulties with the Candidate Elimination Algorithm

Nonetheless, there are difficulties with the candidate learning algorithm that arise from preset assumptions, i.e., the bias, which can cause learning to fail completely. Careful examination of the candidate elimination algorithm reveals that the *G* set and the *S* set can become empty. When this happens, the version space is empty, and the candidate elimination algorithm reports that it cannot learn the target concept. There are two reasons why this failure can occur. Since the candidate elimination algorithm guarantees that all of the hypotheses in <*G*,*S*> are consistent with the training data, severe problems can result if the data is itself inconsistent. Noisy data can cause the algorithm to incorrectly remove a hypothesis from the version space, which will in the very least result in the algorithm eliminating a legitimate representation of the target concept, and in the worse case will result in the algorithm failing to learn completely. For similar reasons, the candidate

elimination algorithm is not well structured for training data with instances which have unknown attribute values.

But by far the most serious obstacle to the candidate elimination algorithm is selecting the proper bias. The dominating factor of the time and space required to run the version space is the strength of the bias: the stronger the bias, the more efficient the algorithm, and the better the resulting hypotheses. However, this is true only when the target concept is expressible in the concept description language. When the target concept is not expressible in the concept description language, given a sufficient number of unique training instances, the version space will become empty (because no concept description exists in the concept description language that is consistent with the training data). When the choice of bias excludes the target concept from the version space, the bias is said to be *incorrect*, and learning will fail, given an adequate number of unique training instances. Conversely, when the target concept is expressible in the concept description language, the bias is said to be *correct*.

In the mushroom example above, let us assume that the target concept is actually all brown *or* yellow mushrooms that have weight from 4.5 to 5.1 grams. This is not expressible in the concept description language. Indeed, in the example run of the candidate elimination algorithm given previously, if the next training example is [**color**: yellow; **weight**: 4.7; **class**: +], the version space will become empty and learning will fail.

As seen previously, choosing a bias that is too weak will cause the learning problem to be intractable, whereas the most recent example shows that an incorrect bias can also thwart learning. Therefore, the goal of the agent employing the candidate elimination algorithm is then to choose the strongest correct bias possible. Not surprisingly, this is mostly guesswork,

since typically little intuition can be gained from the training data without perhaps running another learning algorithm on it. Sadly, experience shows that a first guess is most likely wrong, and all effort spent in learning is therefore wasted. The fact that seemingly small increases in bias cause a very severe degradation in performance worsens the problem, because it makes it unacceptable to err on the side of a weak bias.

CHAPTER II

PREVIOUS RESEARCH

Theoretical Results Concerning Bias and Version Spaces

In order to better understand inductive bias, we will examine the formalism and results of Valiant [11], Haussler [2], and Vapnik and Chervonenkis [12]. It would be useful to be able to estimate the number of instances needed to reveal a bias as correct or incorrect. Not surprisingly, it is not possible to accurately estimate a reasonable number of instances required in every case, but we can make estimates with a high degree of probability.

Let D be our learning domain, T be a set of training instances from that domain, and H the restricted hypothesis space. Clearly, there are $2^{|T|}$ different ways of partitioning T into positive set P and negative set N. If H is capable of expressing each of these partitionings, H *shatters* T. It is easy to see that whenever H shatters T, the current bias is sufficient to express a hypothesis consistent with the training data. This can give us a lower bound on the number of instances required to reveal a bias a incorrect.

Let us consider an example. We return to the examples given in the previous chapter. We allowed each hypotheses to be described in terms of [**color:** *c*; **weight:** *i* to *j*] as before. A simple bias of this type is known as a *purely conjunctive bias*, because the concept description language is only

capable of expressing concepts in terms of conjunctions of acceptable attribute values. We expand upon this definition somewhat by permitting *weight* to be expressed as a contiguous range rather than a single numerical value.

Let T = {[**color**: brown; **weight**: 4.5], [**color**: yellow; **weight**: 5.5]}. It is easy to see that regardless of which mushrooms in T are tasty, a corresponding hypothesis exists in H. Let us add one more instance to T such that T = {[**color**: brown; **weight**: 4.5], [**color**: yellow; **weight**: 5.5], [**color**: brown; **weight**: 5.5]}. H no longer shatters T, because no hypothesis consistent with {[**color**: brown; **weight**: 4.5; **class**: +], [**color**: yellow; **weight**: 5.5; **class**: +], [**color**: brown; **weight**: 5.5; **class**: -]} exists within H. As this example illustrates, common biases can fail to shatter relative small sets of instances. The reason the H no longer shatters I is because it is forced to generalize over the third instance, i.e., after seeing the first two instances, the bias forces the third instance to be classified as '+' as well.

Though this result shows us how quickly a bias can be revealed to be incorrect, it does not lend us much intuition on how many instances can be expected to reveal a bias as insufficient. Define the *Vapnik-Chervonenkis dimension* of H, denoted as VCdim(H), to be $|T|$, where T is the largest subset of D that is shattered by H [2] [12]. In the example above, the initial set {[**color**: brown; **weight**: 4.5], [**color**: yellow; **weight**: 5.5]} is the largest (not necessarily unique) subset of D that is shattered by $H_{,,}$ and so VCdim(H) = 2. Our experience indicates that most useful biases produce relatively small values for VCdim(H).

We now introduce the notion of exhausted and ε-exhausted version spaces. A version space is said to be *exhausted* when either it has been reduced to a single hypothesis consisting of the target concept, or when it is empty. An exhausted version space is therefore one where no further

learning can occur, because either the concept has been learned or learning has failed. Unfortunately, this concept is not particularly useful, for in most situations it is difficult to insure that a version-space will be exhausted. A version space is said to be *ε-exhausted* when every hypothesis remaining within the version space misclassifies less than ε of the domain [11]. So whereas an ε-exhausted version space may not be exhausted, at least it consists of hypothesis that are arbitrarily close to the target concept (depending on ε) or is empty.

It has been shown[4] that for a hypothesis space H and a set of instances T, the corresponding version space will be ε-exhausted with probability at least 1-δ when

$$|T| \geq \frac{4\log(2/\delta) + 8\mathrm{VCdim}(H)\log(13/\varepsilon)}{\varepsilon}$$

Though obviously the right-hand side can be arbitrarily large depending on VCdim(H) and error parameters ε and δ, we can claim that the version space is nearly exhausted with high confidence after considering relatively few instances.

However, this result is not quite enough for us to reasonably estimate when an incorrect bias will be revealed as incorrect, because of the definition of an ε-exhausted version space. In particular, since the remaining hypothesis in an ε-exhausted version space are allowed to differ from the target concept, the target concept may not be in the version space and the bias may still be proven incorrect in subsequent trials. Let ω be the smallest α such that some $h \in H$ differs from the target concept C on exactly α of D. This is the

---

[4]See [Haussler 88] for a formal proof.

probability that any given training instance will eliminate the hypothesis differs the least from C from the version space. Let h be the hypothesis that differs from C on ω of D. Therefore, we can expect that h will be eliminated from the version space in $\log(\delta)/\log(1 - \omega)$ trials with at least probability $1-\delta$. Since no hypothesis differs from C on less $t \in D$ than h, we can state that with probability at least $1-\delta$, an incorrect bias will be revealed after

$$\max\left(\left\lceil\frac{\log(\delta)}{\log(1-\omega)}\right\rceil, \left\lceil\frac{4\log(2/\delta) + 8VCdim(H)\log(13/\varepsilon)}{\varepsilon}\right\rceil\right)$$

independent trials. This result shows that if there is a significant difference between the target concept and the most similar hypothesis in H, that the bias will be revealed as incorrect with relatively few trials with a high probability.

Introducing STABB, a Bias Shifting Program

Somewhat surprisingly, little research has been done in the area of bias shifting for the candidate elimination algorithm. However, there does exist one significant work, Paul Utgoff's STABB program [10]. STABB (*Shift To A Better Bias*) is a bias-shifting program for the candidate elimination algorithm that is built into Mitchell's LEX program. To understand STABB fully, we must examine the LEX system.

LEX is a self-contained learning system that learns heuristics in the domain of integral calculus. It is composed of four inter-related parts: a problem generator, a problem solver, a critic, and a generalization engine which uses the candidate elimination algorithm. The problem generators takes the version spaces of the generalization engine as input; from this input, it creates new problems to be solved by the problem solver. These

problems are then given to the problem solver, which uses various techniques to reduce the integrals. The paths of the solutions produced by the problem solver are recorded, and passed to the critic as input. The critic then analyzes every application of a technique; an application that is on a minimum cost solution path is labeled as a positive instance, and the rest are classified as negative. These classification are then passed to the generalizer as training instances. These training instances are incorporated into the version spaces, which are passed to the problem generator, starting the process anew.

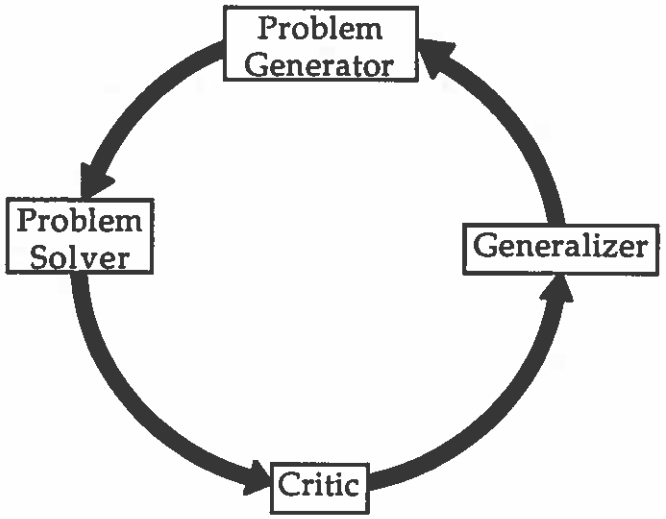


Figure 2.1: Overview of LEX

The domain of LEX was that of integral calculus problems. An instance of that domain is some unsolved integral. The concept description language (i.e., the bias) was chosen to be a grammar that could express such integral calculus problems. It is crucial to realize what effects such a decision has on the hypothesis space and the bias. Since a grammar is used to describe the

instances, the attributes are said to be *tree-structured*[5]. That is, possible attribute values can be arranged in a tree (a parse tree in this case), rather than in a total ordering as with linear structured attributes. An interesting consequence of this structure is that a disjunction of attribute values may be expressible by an attribute value that is a common parent, a property exploited by STABB. We also note that as LEX was learning heuristics for multiple techniques, it maintained multiple version spaces, one for each technique.

The initial concept description language chosen for LEX proved to be insufficient for the task of describing the target concept for all the techniques. STABB was created to enable the generalizer to shift its bias, and was added to LEX as a subroutine for the generalizer.

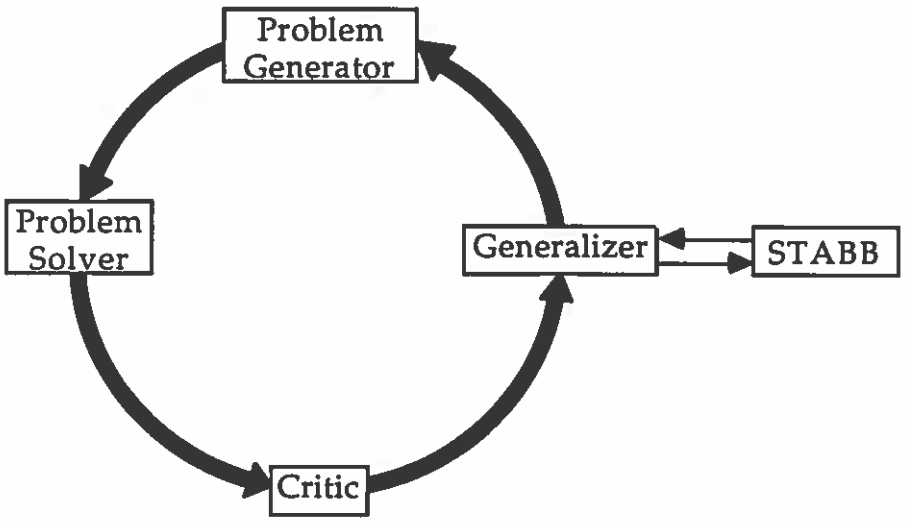


Figure 2.2: Interaction between LEX and STABB

[5]Presumably, instances in LEX's domain were specified with a single attribute, the description of the integral, though this is not stated by Utgoff.

Overview of the Bias-shifting Method Used by STABB

Utgoff divided the bias-shifting process into three phases: recommending new constructs to be added to the concept description language, translating these recommendations into a new concept description language, and assimilating newly expressible hypotheses into the version space. The last phase can be thought of as creating the appropriate version space in the new bias.

The distinction between the first two phases can be best illustrated by means of an simplified example. Recall that the description of the integral is expressed in single terms of a grammar, and so the attribute is tree structured. Assume that we have seen but one positive instance, which is sin. This is graphed on the tree structure below, the classification of the instance below the instance name, where n/a indicates a composite instance (whose classification is not meaningful, since they will never be encountered as a training instance).

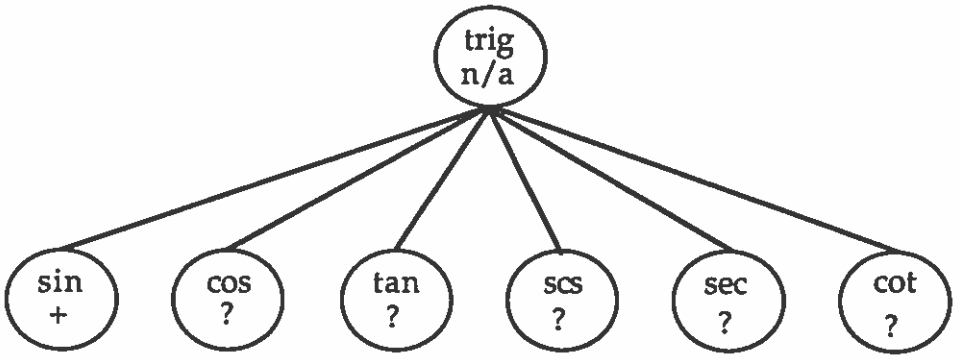


Figure 2.3: Tree-structured Attribute, One Positive Instance

Currently, the version space is *<{trig},{sin}>*, using the *<G,S>* notation. Now assume we see cos, another positive instance. The version space now contains but a single hypothesis, being *<{trig},{trig}>*. Let us assume that the next training instance is a negative instance, and that instance is tan. At this point, the version space becomes empty, since there is no symbol currently in the grammar that would allow sin and cos, but not tan. Therefore, a shift of bias must occur to enable incorporation of the new instance.

In the first phase, recommendation, the problem is identified, and a solution is proposed. In this case, the concept description language needs to be able to accept sin and cos without accepting tan. Though accepting any subset of the children of trig that includes sin and cos and not tan could be used, we will assume that the recommendation produce the recommendation, "Enrich the language such that the disjunction of sin and cos can be expressed." This marks the end of the first phase.

However, though the problem has been identified and a solution proposed, there is still no way to incorporate the newest instance into the current (unenriched) concept description language. This goal of the translation phase is to facilitate this incorporation. The recommendation produced by the first stage is translated into symbols to be added to the grammar. The fact concept description language type (i.e., a grammar) remains the same, but is only increasing in complexity (number of symbols) is one of the advantage of tree-structured attributes. We can add a new symbol to the grammar that represents the disjunction of sin and cos, which is equivalent to allowing a concept to be described as a disjunction of sin and cos, but is simpler to implement. The resulting grammar is shown in Fig. 2.4. The symbol is added to the grammar, and the translation phase is completed.

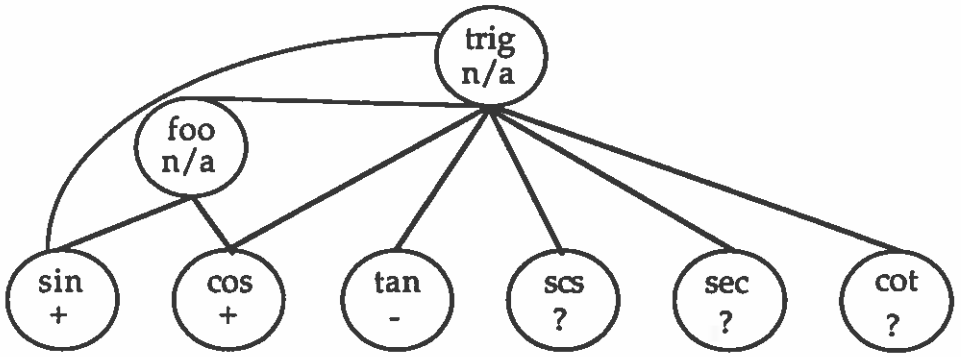


Figure 2.4: Translating the New Symbol into the Grammar

Implementation of the Three Phases of STABB

In the previous section, we have hinted on how the translation step might be implemented, as it is fairly straightforward given the tree-structured attributes and the recommendation produced in the first stage. The recommendation stage is not so obvious, however. Utgoff outlines two competing methods used by STABB: the least-disjunctions method, a goal-free method; and constraint back-propagation, a goal-sensitive method. A goal-free method is one that incorporates no knowledge of how the trials were acquired or how the data is to be used, and so is independent of the particular goal of learning. A goal-sensitive method makes use of this sort of domain knowledge. Since in our paradigm we assume no domain knowledge, the constraint back-propagation is not applicable and will not be covered here.

The least-disjunctions method is similar to Michalski's STAR methodology. The goal of this method, as its name suggests, is to produce the a recommendation with the least number of disjunctions necessary to sufficiently enrich the language. The process starts by creating an initial

disjunction of the set of all positive instances, a worst-case disjunction. Then, the size of the disjunctions are increased by combining disjunctions. Any disjunction that contains a negative instance is discarded, and any disjunction that is a subset of another disjunction is also discarded. This process continues until the set of disjunctions is quiescent. Then, any common features are removed from the remaining disjunctions. As an example, let us consider the tasty mushroom domain used previously. Assume that the least-disjunction returned was (**color**: brown; **weight**: [4.5..5.1]) $\cup$ (**color**: yellow; **weight**: [4.5..5.1]). Since **weight**: [4.5..5.1] is a common context for both disjuncts, it is removed from the recommendation, and the recommendation returned is "Enrich the language such that the union of brown and yellow over color is expressible". The advantage of removing this context is that the other concepts, for instance (**color**: brown or yellow; **weight**: $\top$) would also be expressible in the language.

Finally, it remains to be seen how to implement the final phase, creating a new version space in the enriched bias that is consistent w.r.t previously seen trials. Utgoff suggests a simple and effective method. Though the candidate elimination algorithm may safely discard a training instance after updating the version space, the STABB program must maintain a history of past instances to run its least-disjunction method. Since all the previous instances are still available, it would be easy to simply restart the generalizer with a version space of $<\{\top\},\{\bot\}>$, and have it reprocess the previous instances.

CHAPTER III

IMPLEMENTATION ISSUES FOR THE CANDIDATE ELIMINATION ALGORITHM

We now return to our discussion of the implementation of the candidate elimination algorithm. We expand upon our earliest formulation into an equivalent formulation that is more efficient in terms of both time and space, although it is not as straightforward. We will show the relationship between revision as they are introduced.

We noted previously that it is not necessary to keep track of all members of the candidate set explicitly, since a most general set of consistent hypotheses and a most specific set of hypotheses can be maintained, which will uniquely define the candidate set. This is by far the most beneficial in our series of refinements, since it greatly lowers the space requirements (since only a fraction of the candidate set is represented). This also greatly reduces the time spent, because no processing is ever done to the implicitly represented candidate hypotheses. Of course, processing of the explicitly represented candidate hypotheses (the G and S set, as defined previously) becomes considerably more complicated. It will be necessary to generate a new G and S set that is consistent with $T \cup t$ from the old sets consistent with T. We introduce three new functions for this purpose: **validate**, **generalize**, and **specialize**. Function **validate**(Set, T) will return a new set Set′ that contains all hypotheses in Set that are consistent with T.

Function **generalize**(S, T) will return S′, which is the set of most specific candidate hypotheses consistent with T. Function **specialize**(G, T) is analogous to **generalize**(S, T) except that it returns the set of most general candidate hypotheses consistent with T. A functional definition of these functions are follows.

```
function validate(Set, T)

begin
    let Set' = {}
    for each h ∈ Set do
        consistent = true
        for each t ∈ T do
            if h misclassifies t
                then consistent = false
            fi
        od
        if consistent
            then Set' = Set' ∪ {h}
        fi
    od
    return Set'
end
```

```
function generalize(S, T)

begin
    let S' = {}
    for each h ∈ S do
        consistent = true
        for each t ∈ T do
            if h misclassifies t
                then consistent = false
            fi
        od
        if consistent
            then S' = S' ∪ {h}
            else S' = S' ∪ generalize_theory(h, T)
        fi
    od
    return S'
end
```

```
function specialize(G, T)

begin
    let G' = {}
    for each h ∈ G do
        consistent = true
        for each t ∈ T do
            if h misclassifies t
                then consistent = false
            fi
        od
        if consistent
            then G' = G' ∪ {h}
            else G = G ∪ specialize_theory(h, T)
        fi
    od
    return G'
end
```

The implementations of **generalize_theory(h, T)** and **specialize_theory(h, T)** are bias-dependent. Function **generalize_theory(h, T)** returns the set of all most specialized hypotheses that are more general than **h** and are consistent w.r.t. *T*. Function **specialize_theory(h, T)** returns the set of all most generalized hypotheses that are more specific than **h** and are consistent w.r.t. *T*.

We can see that the previously stated properties of *G* and *S* are maintained by this algorithm. Consistency is maintained since every

inconsistent hypothesis is removed from the G and the S sets. Also, since a hypothesis is replaced with the most general or most specific consistent hypotheses directly below (or above) the invalid hypothesis **h** in the partial ordering, the sets will contain the proper members. This leads to the observation that the hypotheses in G never become more general, but only more specific. That is, there does not exist a hypothesis in G_T that is more specific than some hypothesis in $G_{T \cup \{t\}}$. The contrary is true for S, i.e., there foes not exist a hypothesis in S_T that is more general than some hypothesis in $S_{T \cup \{t\}}$. This corresponds to our intuition that as more trials are considered, our knowledge should be monotonically increasing, and so the cardinality of the version space should be decreasing (although this does not imply that our version space *representation* as $<G,S>$ in terms of storage costs is strictly decreasing).

Our candidate elimination algorithm implementation is still not as efficient as it could be. Incorporating the information represented by a new trial takes $\Theta(T(|G| + |S|))$ time, neglecting the time required to perform the **generalize_theory(h, T)** and **specialize_theory(h, T)** operations. Since this must be done once as each trial is added to T, we have a lower bound of $\Omega(T^2)$ time. We can improve on this by making some observation about the relationship between G and S. Recall the our previous formulation of the version space w.r.t T to be $G \cup S \cup I.$, where I was the set of all hypotheses h such that there exists some $g \in G$ and $s \in S$ such that $h <_g g$ and $s >_s h$. We can state a similar relation between S and G. For every $g \in G$, there exists some $s \in S$ such that either $g >_g s$ or $g \equiv s$. Similarly, for every $s \in S$ there exists some $g \in G$ such that either $s <_s g$ or $s \equiv g$. Our argument follows:

Recall that G is defined to be the set of consistent hypotheses w.r.t T that are not less general than some other consistent hypothesis w.r.t T, and S is defined to be the set of consistent hypotheses w.r.t T that are not less specific than some other consistent hypothesis w.r.t T. It follows that for every $g \in G$ there exists some $s \in S$ such that either $g >_g s$ or $g \equiv s$. We can prove this by contradiction. Assume that there exists a consistent hypothesis h such that $g >_g h$, but $h \notin S$, and there does not exist some $s \in S$ such that $h >_s s$. This contradicts the definition of S, because h meets the criterion for inclusion in S, since it is both consistent and not less specific than any other consistent hypothesis. Therefore there is no such h. Assume that there exists an consistent hypothesis h such that $g \equiv h$, but $h \notin S$ and there does not exist some $s \in S$ such that $h >_s s$. Again, this contradicts the definition of S, because h meets the criterion for inclusion in S. Therefore there is no such h, and the claim is proven. By a similar argument, it can be proven that for every $s \in S$ there exists some $g \in G$ such that either $s <_s g$ or $s \equiv g$.

We can use this relation between S and G to perform our consistency checks and therefore make it unnecessary to check for consistency with previous trials. Let t be the newest training instance, T be the set of training instances prior to t, and define T' to be $T \cup \{t\}$. Let $<G_T,S_T>$ be the version space w.r.t T and $<G_{T'},S_{T'}>$ be the version space w.r.t T'. Consider the **validate(***Set*, t**)** function. Since $<G_T,S_T> \supseteq Set$, we know that every hypothesis $h \in Set$ is consistent w.r.t T. Therefore, we need only eliminate $h \in Set$ from Set that do not properly classify t. The rewritten **validate(***Set*, t**)** is given on the following page.

```
function validate(Set, t)

begin
    let Set = {}
    for each h ∈ Set do
        consistent = true
        if h misclassifies t
            then consistent = false
        fi
        if consistent
            then Set' = Set' ∪ {h}
        fi
    od
    return Set'
end
```

For similar reasons, we are assured that every hypothesis $h \in G_t$ is consistent w. r. t. T, and hypothesis $h \in S_t$ is consistent w. r. t. T. We would like to be able to generalize and specialize hypotheses without considering T. Let G_{t-} be the subset of G that misclassifies t. Let us change the definition of **specialize_theory(h, t)** so that it returns the set of hypotheses that correctly classify t that are immediately less general than h. That is, the set returned by **specialize_theory(h, t)** will be the set of hypotheses such that each member hypothesis h' satisfies the following conditions; $h' <_g h$, h' is consistent w.r.t t, and there does not exist a hypothesis h'' such that $h'' <_g h$, $h' <_g h''$, and h'' is consistent w.r.t. t. Unfortunately, this no longer guarantees that the set returned by **specialize_theory(h, t)** is consistent

with T, but merely that it is consistent with t. However, as noted previously, for every $g \in G$ there exists some $s \in S$ such that either $g >_g s$ or $g \equiv s$. We can use this property to prune the set returned by **specialize_theory(h, t)**. Fortunately, this pruning also insures that only hypotheses consistent with T are returned. Let G' be the set returned from this process. Let $h \in G'$ be such that $h \notin G_{T'}$. We have proved above that there is no hypothesis in $G_{T'}$ that is more general than h, so we must assume that h is not consistent w. r. t. T'. Since h is consistent with respect to t, h must not be consistent w.r.t T. That is equivalent to claiming that there is some training instance in T that is misclassified by h. Let t' be that instance. Assume that t' is a positive instance. Since there exists an $s \in S$ such that either $h >_g s$ or $h \equiv s$ (because of our pruning), s must also misclassify t', by the definition of more general and equivalence relations. This would violate the definition of S, producing a contradiction. So it must be that t' is a negative instance. However, we are guaranteed that there exists a $g \in G$ such that $h <_g g$. This implies that g must also misclassify t', by the definition of the less general relation. However, since G is consistent w.r.t. T, this is a contradiction. Therefore, no such h exists, and $G' \equiv G_{T'}$.

A similar argument can be generated for the construction of $S_{T'}$. The pseudocode for the revised functions **generalize(S, t)** and **specialize(G, t)** are given on the following pages.

```
function generalize(S, G, t)

begin
    let S' = {}
    for each h ∈ S do
        if h misclassifies t
            then begin
                S+ = generalize_theory(h)
                for each s ∈ S+ do
                    prune = true
                    for each g ∈ G do
                        if s <s g then
                            prune = false
                        fi
                    od
                    if prune
                        then S+ = S+ - {s}
                    fi
                od
            S' = S' ∪ S+
            end
            else S' = S' ∪ {h}
        fi
    od
    return S'
end
```

```
function specialize(G, S, t)

begin
    let G' = {}
    for each h ∈ G do
        if h misclassifies t
            then begin
                G+ = specialize_theory(g)
                for each g ∈ G+ do
                    prune = true
                    for each s ∈ S do
                        if g >g s then
                            prune = false
                        fi
                    od
                    if prune
                        then G+ = G+ - {g}
                    fi
                od
                G' = G' ∪ G+
            end
        else G' = G' ∪ {h}
    fi
    od
    return G'
end
```

Finally, we incorporate these new functions into the candidate elimination algorithm. For efficiency, we introduce one more function, **eliminate_redundancies(*Set*)**, which eliminates redundant representations from a set. Multiple representations is a by-product from the **generalize(*S*, *G*, t)** and **specialize(*G*, *S*, t)** functions, which may produce the same hypothesis by generalizing or specializing different hypothesis.

```
function candidate_elimination

begin
    let G = {T}
    let S = {⊥}
    for each new training instance t do
        if classification(t) ≡ positive
            then G = validate(G, t)
                 S = generalize(S, G, t)
                 S = eliminate_redundancies(S)
            else S = validate(S, t)
                 G = specialize(G, S, t)
                 G = eliminate_redundancies(G)
        fi
    od
    return <G, S>
end
```

We note several interesting properties about the candidate elimination algorithm and version spaces. Perhaps the most elegant feature of the

candidate algorithm is that once a training instance has been processed, it can be discarded. The relation between the *S* and *G* sets and the training instances also deserves some mention. Though both are constrained to correctly classify all previously seen instances, the *S* set can be thought of as an acceptor set, since it is formed to cover all positive instances and as little of anything else as possible. By contrast, the *G* set can be thought of as a discriminator set, since it is formed to reject all negative instances, and reject as little of anything else as possible. Finally, since the candidate elimination algorithm works with version spaces, it has several advantages. It can guarantee that if the target concept is expressible in the concept description language, it will successfully learn that concept. Also, if the *G* and *S* sets do not converge on a single hypothesis, the algorithm can return all consistent hypotheses. The intelligent agent can then choose the most beneficial representation.

CHAPTER IV

A CLOSER LOOK AT BIAS

A Generalized Framework for Examining Bias Shifting

In order to better understand what is involved in bias shifting, we propose a general framework that should be sufficient for any learning problem in our paradigm. Although it would certainly be possible to achieve better results in particular domains if a more specialized model was used, we opt for a more general approach as to eliminate any advantages that may be gained from particular learning formulations between the methods we will examine.

We formalize our idea of a generalized framework by stating two necessary and sufficient requirements of a generalized framework; it must be possible to express arbitrarily complicated descriptions of training instances, and a hierarchy of biases must be stated that will allow the bias-shifting program to arrive at the correct bias after a finite number of bias shifts.

Let us consider the first requirement. It is necessary to be able to describe any completely specified training instance. In the framework we propose, this is facilitated by allowing an arbitrary large number of attributes to be used to describe the instances of any learning domain. Note that we do not require that there is a method of describing partially described instances (i.e., instances who have unspecified values for certain attributes)-- indeed,

the candidate elimination algorithm makes no provision for such instances and we also do not state how they should be handled. Another approach would be to limit the number of attributes but make the attribute values arbitrarily complex, as meta-attribute approach. However, we feel that our approach is more natural and easier to implement. For similar reasons, we require that all attributes be either nominally-valued (with an optional ordering defined, partial or total) or numerically-valued (continuous or discrete) but not tree-structured. The tree-structured approach certainly has some advantages over ours; however, it complicates implementation, makes bias more difficult to quantify, is not applicable to many domains, and is not consistent with our policy of avoidance of meta-value attributes values (since an attribute parent value represents several attribute values, namely its children). Since we can express any meta-value attribute value with a disjunction of leaf attribute values, we apply Occam's razor and do not allow tree-structured attributes. For simplicity, however, we do enrich the set of possible attribute values by adding one meta-value attribute value, namely $\top$, and one new attribute value representing no value, namely $\perp$.

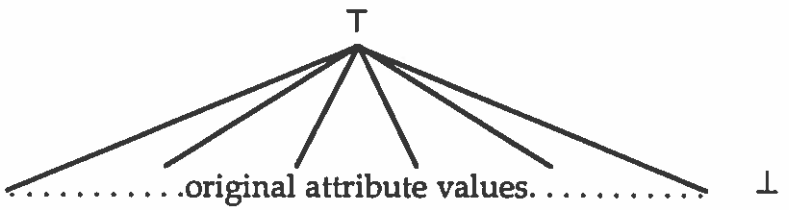


Figure 4.1: Enriched Set of Possible Attribute Values

Before we consider an appropriate hierarchy of biases, we shall consider what is meant by a hierarchy of biases. Recall that biases differ in there ability

to partition a domain, which is how their strength is measured. For a given domain, an ordering can be defined over the set of biases B where each bias is mapped into an equivalence class according to the value of VCdim. If the size of the domain is infinite, there is an infinite number of equivalence classes and an infinite number of biases.

Though we have defined an ordering over the (possibly infinite) set B, our equivalence relation is not quite intuitive, since biases in the same class are not necessarily able to represent the same partitionings of the domain. We define a partial ordering over instances in the domain D where $a > b$ iff VCdim(a) > VCdim(b) and all concepts expressible in b are also expressible in a. This ensures that there are no tradeoffs in expressive power, and so that if shifts in bias are made in terms of strictly increasing in terms of our partial ordering, then there is never a chance of switching from a bias that could express the target concept less accurately then the previous bias. This serves as our hierarchy of biases.

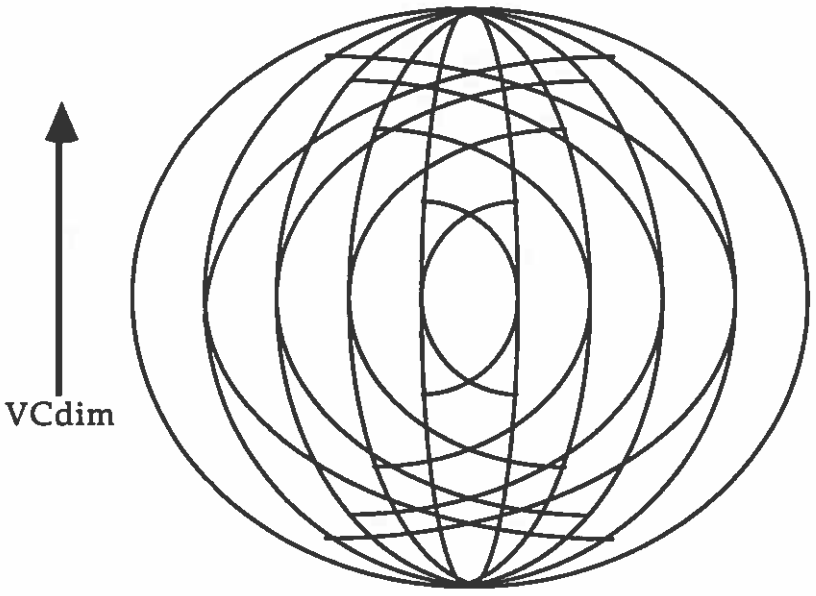


Figure 4.2: A Representation of a Partial Ordering of B

Our reason for wanting to define such an ordering is simple; though it would be possible to move effectively through the space of biases in such a way that biases are not strictly increasing in terms of our partial ordering, such movement would surely require more information than we allow in our learning paradigm. One can imagine a version-space approach through the space of biases, a meta-candidate elimination algorithm where insufficient biases are eliminated from the candidate set as their own candidate sets become empty. Though such a formulation may be of theoretical interest, it would be computationally intractable for all but the most trivial of domains and therefore is not a practical bias shifting method.

It is therefore necessary to choose a path through the hierarchy rather than considering multiple paths[6]. However, we want to avoid the difficulties associated with backtracking (i.e., switching to a bias that has a lesser VCdim value), so it is necessary to choose a path through the hierarchy that is guaranteed to contain some bias capable of expressing a hypothesis consistent with the training data. This is equivalent to stating that for every possible subset of the domain (i.e., possible training instances), there exists a bias on our path through the hierarchy that is capable of expressing a hypothesis consistent with that subset.

To simplify matters, we predefine a path through the hierarchy and use that for all of our tests. We shall start with the purely conjunctive bias as described earlier as our base bias. For each bias shift, we will enrich our concept description language so that it can express an additional disjunct. Since the purely conjunctive bias does not support any disjunction, it is equivalently the bias of one disjunct (or 1-d). The next bias in our list of

---

[6]This two approaches are analogous to a depth-first and a breadth-first search through the space of biases.

biases is 2-d, which means that any hypotheses can be a disjunction of two conjunctions. In the case of 2-d, a hypotheses can be thought of as a disjunction of 1-d hypotheses (i.e., a disjunction of purely conjunctive hypotheses). This should not be confused with internal disjunction, where each attribute value is a can contain disjunctions, but the overall hypotheses is still a conjunction of acceptable values for each attribute.

We return to our previous examples in the mushroom domain. Assume that we have seen the following instances:

{ [**color**: brown; **weight**: 4.5; class: +], [**color**: brown; **weight**: 5.1; class: +],
[**color**: brown; **weight**: 4.7;
class: -]}

This is not expressible with just one conjunction. However, with the 2-d bias, the hypothesis (**color**: brown; **weight**: [4.5..4.5] ∪ **color**: brown; **weight**: [5.1..5.1]) can be expressed, which is consistent with the training data.

For bias n-d, the next highest bias will be (n+1)-d. It is easy to show that a correct bias lies on this infinite sequence of biases. Let T be the set of training instances considered. The bias $|T|$ is a correct bias, because it is possible to express a consistent hypothesis as a disjunction of all the positive instances in T, which is at most $|T|$.

Consequences of Our Choice of Paths Through the Hierarchy

Any purely ascending path through the hierarchy of biases would serve as a legitimate path, provided that it contained some bias capable of

expressing a hypotheses consistent with the training data, for every possible set of training data. In particular, it would not be necessary to require the concept description languages (and therefore the bias) to be as similar as in our sequence of biases. The strengthening technique in our sequence is an explicit use of disjunction; we will examine the increase in ability to shatter a subset of a domain given by explicit disjunction and the effect disjunctions have on the size of the *S* and *G* sets, which in turn have a direct effect upon space and time resources expended by the algorithm.

The Base Bias

We will begin with the base bias, the purely conjunctive bias(1-d), which allows no disjunctions. We will denote the hypothesis space as defined by bias *B* over *D* as *H(B)*. As noted earlier, the VCdim(*H*(1-d)) is 2, because there does not exist a subset of *D* which is shattered by *H*(1-d) that contains more than two instances. Therefore, 1-d is a relatively strong bias[7]. An interesting property of the 1-d bias is that the *S* set is always a singleton. We consider a graph of a domain with two attributes on the following page, with positive instance represented as '+'. Our domain consists of two totally ordered attributes with quantized values. This differs from the mushroom domain described previously, because the color attribute of the mushroom domain was not ordered. The mushroom domain could also be graphed, but a range on the graph in the color dimension would not be meaningful.

[7]Recall that the strength of a bias refers to the number of candidate hypotheses that are inexpressible; therefore a weaker bias will be able to express more distinct hypotheses.

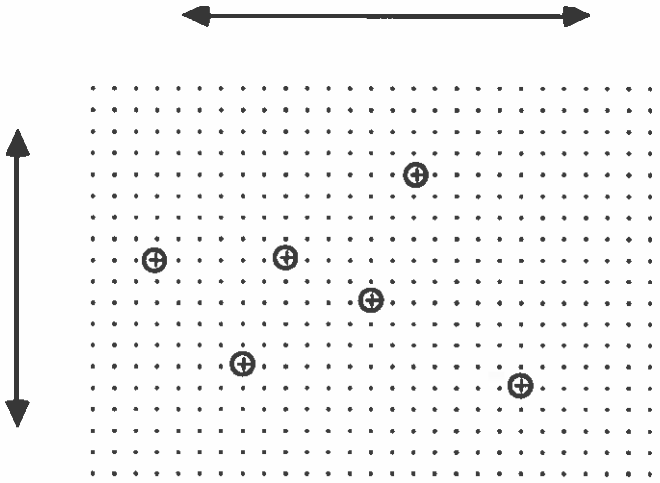


Figure 4.3: Positive instances in a domain with two attributes.

It is easy to see that there is only one most specific hypothesis that is consistent with the training data. The reason for this is simple; any hypothesis that is consistent with the training data must extend to the instances at the extremes for each attribute, and there is only one minimal way of doing so.

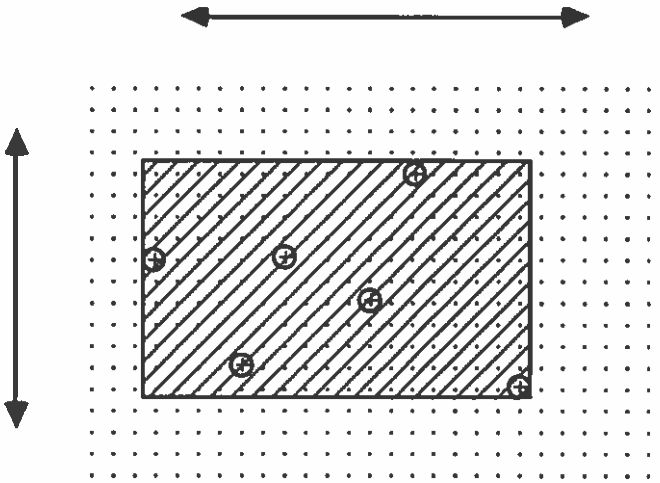


Figure 4.4: The Unique Most Specific Hypothesis.

Upon examining Fig. 4.4, we make another observation; the three instances of the "borders" of the hypothesis determine the shape of the hypothesis, while the "internal" instances have no affect on the hypothesis. Due to the conjunctive bias, we are forced to classify the internal instances as positive, and hence encountering them does not affect the composition of the *S* set. It is reasonable to consider such a hypothesis to be defined by a set of instances; we will call the instances that have determined the shape of a hypothesis *h* to be the *defining instances of* **h**. For any most specific hypothesis that contains more than one possible training instance from the domain, there must have been anywhere from 2 to 2 * (number of attributes) that has defined the hypothesis. In our example, this would be from two to four instances; in this case, it is three.

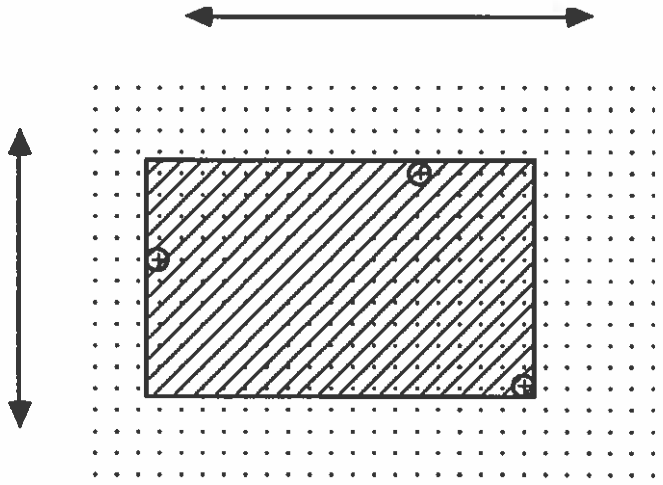


Figure 4.5: Set of Instances Defining the Most Specific Set.

Similar statements can be made about the members of *G*. Sadly, *G* is not nearly as well behaved; it can grow to exponentially large sizes (in the number of attributes). Let us consider a similar example, but with negative instances rather than positive ones. As figure 4.5 shows, even with only

three instances, the situation becomes very complicated. For clarity, the figure has been divided into two parts to make the individual hypotheses more clear; the darkest shaded regions are where two hypotheses overlap.

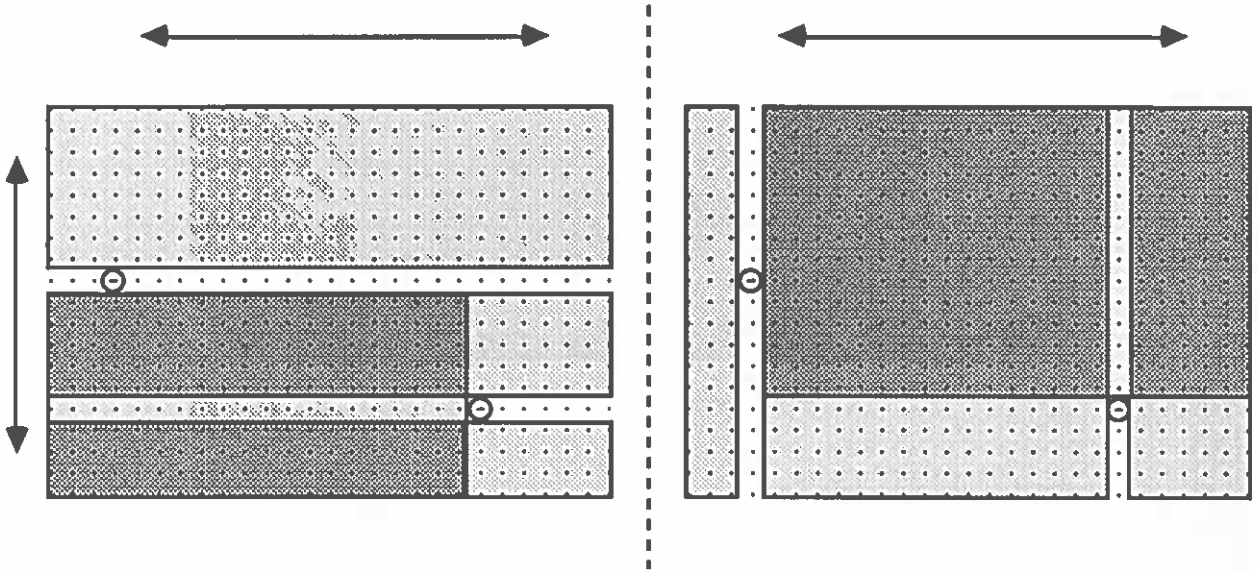


Figure 4.6: The *G* set.

There are eight hypothesis in the *G* set shown above. This is actually slightly worse than would occur in practice, because the candidate elimination algorithm as formulated by Mitchell states that learning must begin with a positive instance. Nonetheless, the *G* set can be very large in relation to the number of instances; each subsequent negative trial can "split" each hypothesis in *G* into two hypothesis for each attribute in the domain; though at most one of these two hypothesis may be consistent per attribute, one may be consistent per attribute, which means that the size of the *G* set can increase by $O(|G|^{|A|})$, where *A* is the set of attributes. Consider the following set of instances;

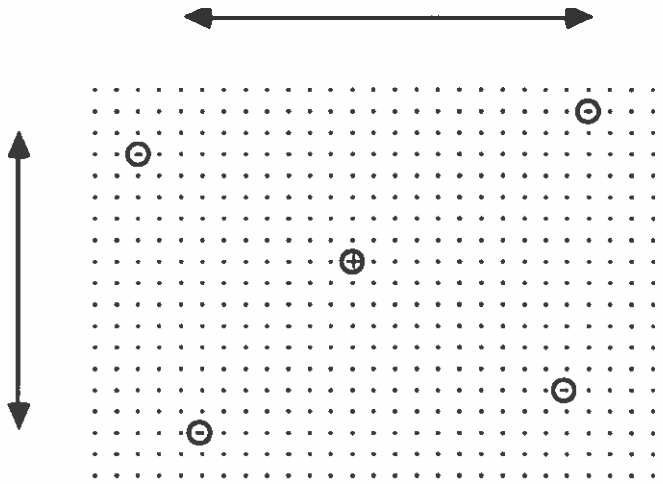


Figure 4.7: A Few Training Examples Leading to a Very Large *G* set.

We leave it to the reader to count the number of hypothesis that would be in the *G* set after encountering the training instances above.

Similar to the most specific theories of the *S* set, the most general theories of the *G* set have a set of defining instances, as well as other instances that do not affect the hypothesis. Since a most general hypothesis is as general as possible without accepting any negative instances, it is the negative instances just beyond the boundary of the most general hypothesis that determine its composition. This can be anywhere from one instance to $2 * |A|$. Similarly, those negative instances do not lie on the boundary of acceptance for a most general hypothesis do not affect it.

Biases with an Arbitrary Number of Disjunctions

Now that we have an understanding of how the candidate elimination algorithm behaves in the base bias, we consider biases with a higher number

of disjunctions. The observations we made about hypotheses in the base bias will apply to the individual disjuncts of hypotheses in these biases with a larger number of disjuncts. It is not immediately clear what the interaction between these disjuncts might be. We consider the value of VCdim(H(2-d)). It is easy to see that any subset of D containing four or less instances is shattered by H(2-d); however, for reasons similar as with 1-d, H(2-d) does not shatter any subset of D that contains five or more instances. Indeed, we realize that for k-d, VCdim(k-d) = 2k. However, the decrease in bias strength is quite large for the switch from 1-d to the 2-d. How does this affect the performance of the candidate elimination algorithm?

We will examine the effect of disjunction on the S set. Unfortunately, the property that the S set contains only one most specific hypothesis is not true for biases with an arbitrary number of disjunctions. Indeed, we shall see that the S set can grow to be quite large. There are eight most specific hypotheses consistent with the training data used in the previous example of six positive instances. A list of these are given in Appendix 1. Sadly, this is not even a worse case example. The G set also suffers a similar increase in size relative to that of the 1-d bias. Though the difference between subsequent biases are not as great as between 1-d and 2-d, a similar decrease in performance (in terms of time and space required to process an instance) is encountered for each increase, since the amount of weakening is the same.

Disjunctive Bias in Practice

Our experience has been that for many real databases, even the 2-d bias can become unusable, due to the exponential explosion of the G and S sets.

This makes the need to perform learning in as strong a bias as possible even more apparent. The amount of time spent in an invalid bias is negligible to the time spent in the next higher bias in most cases. This makes bias shifting even more attractive, since the wasted time in invalid biases is proportionally very small, and well worth examining the possibility that one of the weaker biases may actually be correct.

One interesting feature of the size of the *S* and *G* set is that they tend to grow exponentially at first, taper off at some large size, and then gradually contract. It is therefore the learning that occurs immediately after a bias shift that is the most costly.

CHAPTER V

A REVIEW OF BIAS-SHIFTING TECHNIQUES

Utgoff's Approach and Our Generalized Framework

We recall that Utgoff's STABB program was devised for use by a particular learning system, LEX, which performed learning in a particular domain, the domain of integral calculus. Can the techniques used by STABB be of value in our generalized framework? We address this question presently.

Utgoff divided the process of bias shifting into three phases: recommending a new bias, translating the recommendation into the concept description language, and assimilating the training data previously considered into a new version space in the new bias. In our new formulation, the second phase becomes trivial, for we need only change the maximum number of disjuncts allowed. We therefore present a simplified model of bias-shifting, based on Utgoff's model that consists only of two phases: recommending a new bias (in our framework, a number of allowed disjunctions), and assimilating the previous learning data into the version space in the new bias. Utgoff's method for assimilating previous training data, which is simply restarting the candidate elimination algorithm in the new bias on the previously seen data, easily translates into our framework.

We now consider the bias recommendation phase. Utgoff presented two methods of recommending a new bias; a least-disjunctive method and constraint back-propagation. Since constrain back-propagation utilizes domain knowledge, it is not applicable to our learning paradigm. There is no apparent use of domain knowledge in the least disjunction method, and so we will consider its use within our framework. Although Utgoff never states the algorithm formally, he suggests an algorithm that combines two or more disjuncts at a time, eliminating inconsistent generalizations, until no two disjuncts can be combined without including a negative instance.

An efficient algorithm based on this vague description can be devised for a learning domain that has only one attribute (which is totally ordered). Begin the first disjunction with the first positive instance, and consider strictly increasing training instances, adding each positive instance to the disjunct until a negative instance is encountered. At this point, continue examining consecutive instances until another positive instance is encountered. Begin the next disjunct with this positive instance, and continue the process, until the greatest instance is considered. The result will be a least-disjunction of the positive instances. It is easy to see that the time complexity of this algorithm is linear in the number of instances.

Sadly, there does not seem to be way of generalizing this algorithm into an efficient algorithm for an arbitrary number of dimensions. Indeed, we were unable to come up with any least-disjunction algorithm that had a running time less than $O(2^{|T|})$, which is clearly not acceptable. Another method of recommending a new bias must be used. We opt for a simple yet sufficient method; if k disjunctions proves to be insufficient, try k+1 disjunctions. This guarantees that we will arrive at the strongest bias in our sequence of biases that is sufficient for the learning task, provided that we

assimilate previously seen instances. We are not guaranteed that the bias we initially recommend is sufficient, unlike in the least-disjunction method, but we instead may have to shift bias again before we have finished assimilating the training instances. Nonetheless, we are guaranteed to have arrived at strongest correct bias once assimilation is complete.

Two Methods for Shifting Bias

We are now prepared to state two methods of bias shifting within in our generalized framework. Since we have divided the bias shifting process into two phases, a bias recommendation phase and training data assimilation phase, we need only state how each method will perform in each phase to fully specify the method. We shall call our first method the fully consistent method, which is closest to Utgoff's original method. For the first phase, bias recommendation, we will use the heuristic stated earlier, and simply switch to the next highest bias in the sequence. For the second phase, we shall restart the candidate elimination in the new bias with the version space $\langle \{\top\}, \{\bot\} \rangle$, and assimilate the previously seen instances. Once this assimilation is completed, the resulting version space is guaranteed to be the version space consistent with all previously seen trials.

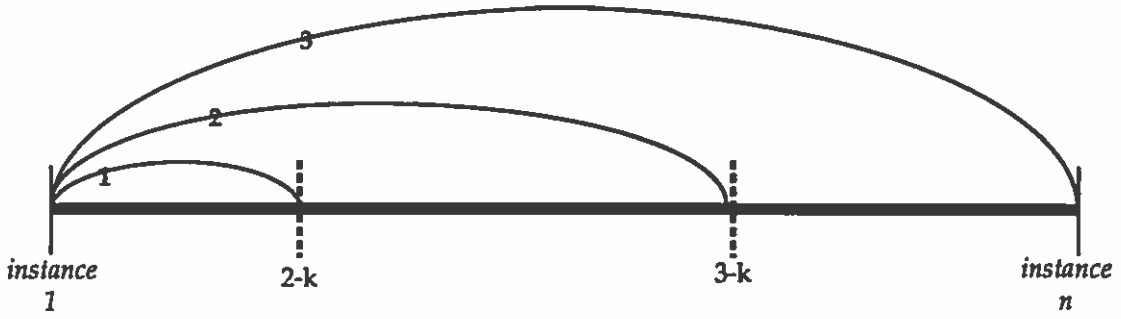


Figure 5.1: Order of instances considered for the fully consistent method. The dashed lines represent bias shifts, whereas the numbers show the order in which the instances are considered.

It is questionable if this assimilation of previous training instances is really necessary. Our theoretical results show us that in many cases, an incorrect bias will be discovered in relatively few trials. Similar results show us that a version space will be ε-exhausted with high probability after relatively few trials. Furthermore, since we do not need previous instances except for such replays (since we have abandoned the least disjunction method), we could lessen our space requirements not replaying instances and discarding previous instances.

These observations led to the formulation of the base method. For the bias recommendation phase, we will use the same heuristic as in the fully consistent method, i.e., simply increasing the bias by one. However, in the assimilation phase, we shall not recapture lost training data, but instead start with the version space $\langle \{\top\}, \{\bot\} \rangle$ on the newest training instance. No assimilation takes place. Our hope is that we shall arrive at the same version space as the fully consistent method when the training data is exhausted, but will have saved memory overhead having not kept every training instance.

Note that in contrast to the fully consistent method, the base method does not guarantee consistency with past training instances once a bias shift occurs.

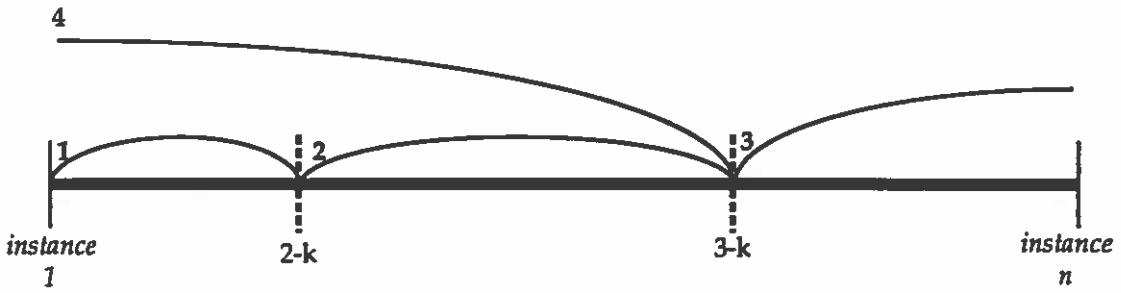


Figure 5.2: Order of instances considered for the base method. The dashed lines represent bias shifts, whereas the numbers show the order in which the instances are considered.

BEANHEAD: An Aggressive Heuristic Method for Shifting Bias

We notice one great failing of all the bias shifting methods that we have considered so far; none make any attempt to utilize the information represented by the learning prior to the bias shift. That is, all learning in the incorrect bias (i.e., the previous version space) is simply discarded-- no benefit is received from running the candidate elimination algorithm on the previous training instances with the incorrect bias. The question arises: can no useful knowledge be gleaned from the old version space? The BEANHEAD system is one system meant to answer this question by using information from the old version space.

We will note two types of information that could possibly be derived from the previous version space: information about instances seen in the old

version space, and information about the hypothesis in the revised version space. What information can be gleaned about the previously seen instances?

Sadly, it is impossible in most case to identify what instances have been seen previously, because many previously seen instances will not be defining instances in the current version space. We will simplify matters by only considering what can be learned by analyzing a single hypotheses rather than a set of hypotheses. This is a reasonable simplification, considering that the candidate set is usually very small before a bias shift. Determining which instances have been seen previously from the invalid version space is difficult because of two reasons. It is ambigious which instances have defined a hypothesis, and instances which are in the "interior" of a hypothesis are not represented since they are generalized.

We shall make these statements more clear with an example. Let us consider the version space

{(**color**: ⊤; **weight**: [4.0..7.1])}, {(**color**: brown; **weight**: [4.5..5.1])}

What instances have been previously encountered? In this particular domain, we are assisted by the fact that color in nominally valued. We can deduce from the *S* set that we have seen [**color**: brown; **weight**: 4.5; **class**: +] and [**color**: brown; **weight**: 5.1; **class**: +], which are the instances on the "edge" of the sole hypothesis in *S*. However, we do not know if we have seen instances on the "interior" of this single hypothesis, i.e., we do not know if we have seen any tasty brown mushrooms whose weight is greater than 4.5 grams but less than 5.1 grams. The reason we do not know is because encountering them would not change the version space; they are already classified and therefore do not represent any new knowledge. Similarly, we

can deduce from the G set that we have seen [**color**: brown; **weight**: 3.9; **class**: -] and [**color**: brown; **weight**: 7.2; **class**: -], but we cannot determine if we have seen any negative instances beyond these boundaries, nor if we have seen any negative instances that are not brown.

It would be possible to run the candidate elimination algorithm on every possible set of defining instances for the invalid version space. This would be another meta-candidate elimination approach. Again, though it is interesting in a theoretical sense, it would translate into a very poor algorithm. Considering that it also would much more costly then simply retaining the previous training instances, and offer no advantage over that approach, we abandon this particular source of information.

Can anything be said about the composition of the new version space directly? This is the tactic we take with the BEANHEAD system. Let t be the bias-breaking instance, and let T be the training instances prior to the bias-breaking instances. We make one simple yet compelling observation about the version space prior to t, that it is consistent with all but one trial in $T \cup t$. Furthermore, we know the sole instance that the prior version space is not consistent with; it is t. Can we use this information to generate the new version space without considering any past instance other than t?

One approach would be use the principles of the candidate elimination algorithm to determine what the new version space should be. We can enumerate from the current information every possible S and G set in the new bias that would be consistent with $T \cup t$. This would require creating a most general and most specific version of *both* the G and S set, because we would be running the candidate elimination algorithm on each to find the real G and S set. Let these be sets be G_g, G_s, S_g, and S_s. G_g would correspond to the most general possible G sets, whereas G_s would be the most specific

possible G sets, based on what the composition of T might actually be. As with the standard candidate elimination algorithm, we would be guaranteed that the corresponding G set, had we run through all the trials with new bias, will lie between these most general and most specific boundaries. S_g and S_s are defined similarly.

Though correct, the requirements of this approach in terms of space and time are prohibitive. So we adopt a heuristic for our BEANHEAD system; if a disjunct was valid throughout past learning, it may be valid in the new version space. This allows us to "repair" the previous version space, rather than generating a completely new one, by making small changes to the existing hypotheses. Our algorithm is simple; for each hypothesis, and for each disjunct within the hypothesis, replace the disjunct with a new disjunct that represents no information if that disjunct caused a misclassification, otherwise do not modify the disjunct. Once this process in completed, we weaken the bias, and the result is a version space in the new bias that can incorporate the bias breaking instance without needing a further switch in bias. Pseudocode for our algorithm follows. Please note that *VS* is the version space prior to encountering t.

```
Function BEANHEAD(VS, t)

begin
    for each hypothesis h in G or S do
        for each disjunct d in h do
            if d misclassifies t
                if classification(t) = positive
                    then d = ⊥
                    else d = ⊤
                fi
            fi
        od
    od
    VS = incorporate(VS, t)
    return VS
end
```

BEANHEAD's aggressive strategy gives it two advantages over the other methods. The obvious advantage is that the assimilation phase is much faster, depending only on the size of the *S* and *G* sets, rather than also begin dependent on the number of trials previously encountered. The other benefit is that once assimilation has taken place, learning the BEANHEAD way will continue to be faster than in the other methods. This is because the version space generated by BEANHEAD will be more restrictive in most cases and therefore be closer to exhaustion.

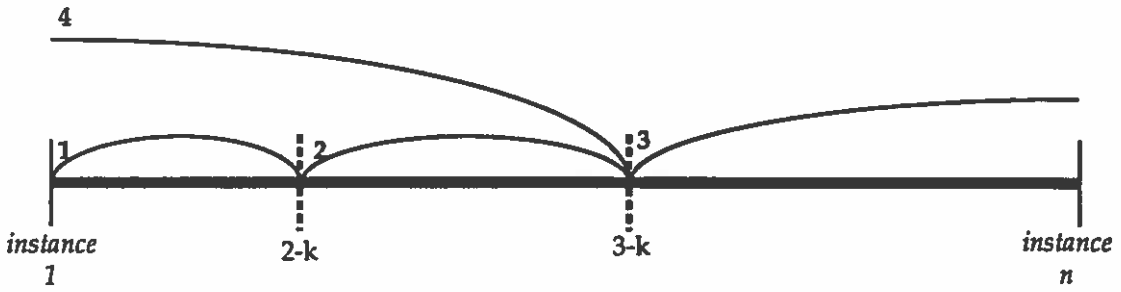


Figure 5.3: Order of instances considered for the BEANHEAD method. The dashed lines represent bias shifts, whereas the numbers show the order in which the instances are considered.

Of course, such an aggressive strategy is not foolproof. The main hazard of the BEANHEAD's heuristic is that it may cause an erroneous increase in bias. This is because the goal concept may not be in the version space returned by BEANHEAD, even though it is expressible in the current bias. During subsequent learning, when the version space becomes empty, BEANHEAD will mistakenly switch to a higher bias, instead of realizing that its previous version space estimate was incorrect.

Let us consider an example of BEANHEAD's version space reparation. We will show the *G* set and the *S* set separately for clarity. The following figures show BEANHEAD translating an inadequate hypothesis in the 2-k bias to one that does not misclassify any instances in the 3-k bias (although it may fail to classify some previously encountered instances).

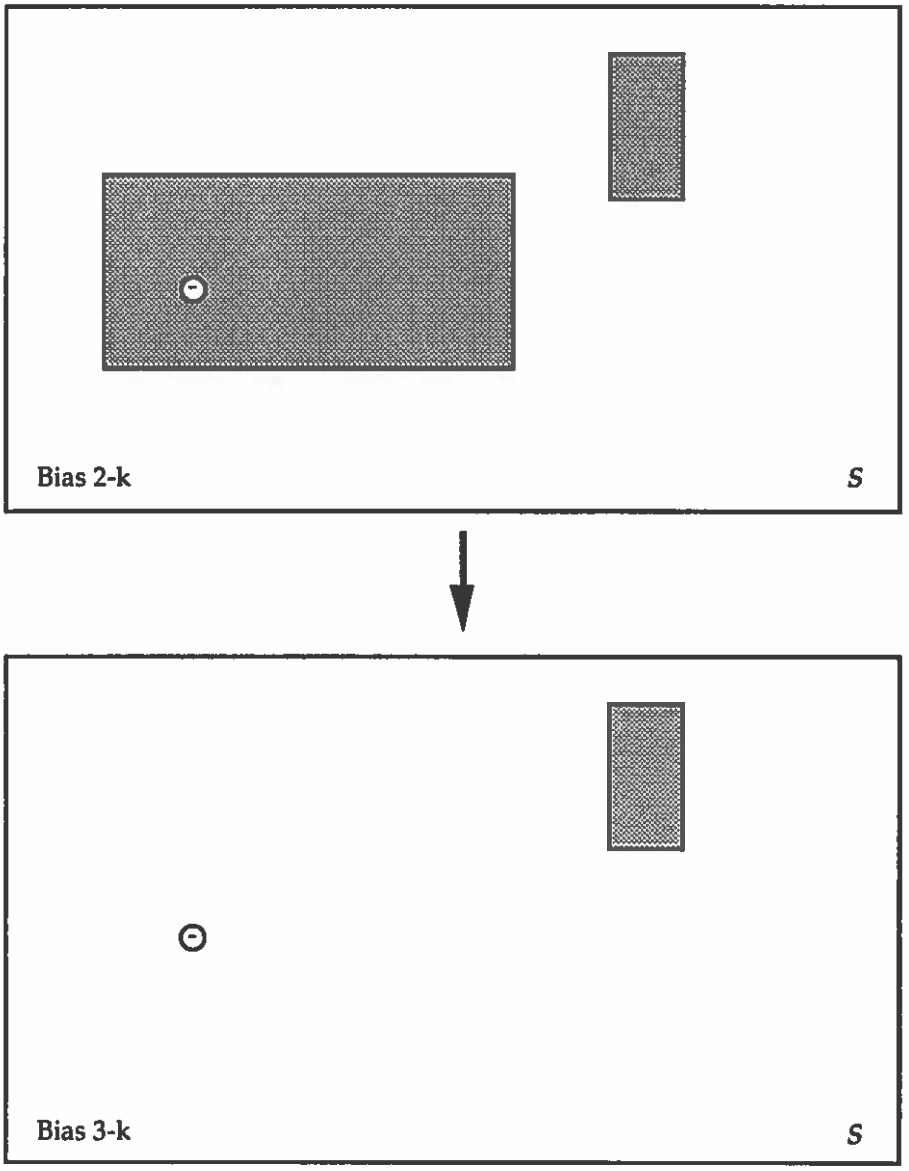


Figure 5.4: BEANHEAD translates a single hypothesis in the *S* set in 2-k to a new hypothesis in 3-k that is consistent with the negative bias breaking instance. Note that instances that fell within the bottom left disjunct in the hypothesis in the 2-k bias are now not covered in the new hypothesis in the 3-k bias.

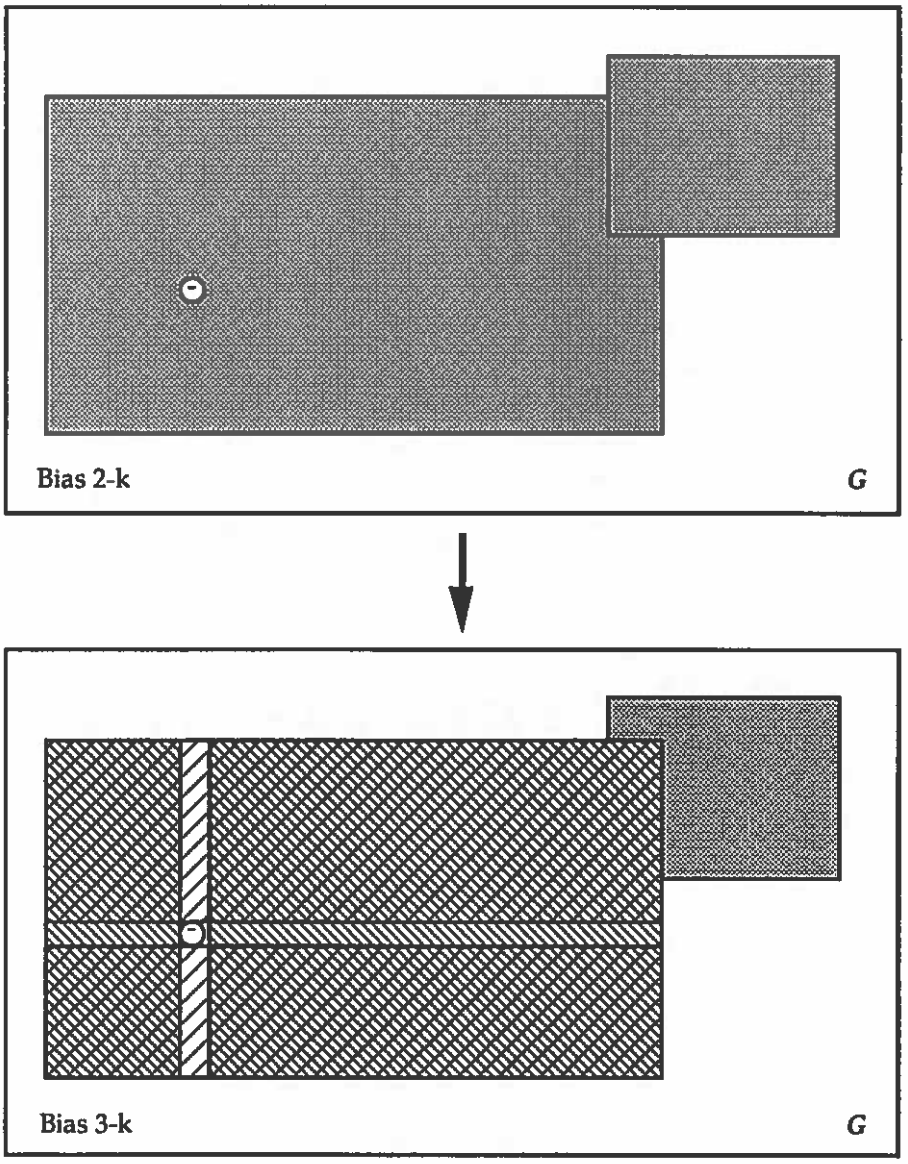


Figure 5.5: BEANHEAD translates a single hypothesis in the *G* set in 2-k to a new set of hypotheses in 3-k that is consistent with the negative bias breaking instance. The diagram is complicated by the fact that the old leftmost disjunct can be specialized in several ways, and so the four overlapping regions can be combined in 4*3 = 12 different combinations, each a valid hypothesis in the new *G* set.

CHAPTER VI

AN EXPERIMENTAL COMPARISON

Due to the heuristic nature of the method we have proposed, it is difficult to compare it to the others with strictly theoretical means. For this reason, we will evaluate the methods solely by their comparative performance. We ran several the candidate elimination on several databases with each of the following methods. Our desire was to see how the bias-shifting methods would perform on real problems. Sadly, we encountered a serious impediment; either the concept could be described in 1-d, requiring no bias-shifting, or all three methods performed so poorly that comparison was impractical. This poor performance was due to the increased complexity when disjunctions added.

Luckily, one database was simple enough to allow comparison. This database was originally used by J. Cendrowska [1], the domain being the fitting of contact lenses. This domain has four attributes and 24 possible instances, all which are present in the database. There are three distinct concepts in the database; patients that should be fitted with hard contact lenses, patients that should be fitted with soft contact lenses, and patients that did not need contact lenses. The methods were used to learn each class. Two of the concepts required a 2-d bias, whereas the third required a 3-d bias. The fully consistent method was required to consider each instance in the set only once; the other methods, since they do not insure consistency, were required to consider each

instance in the database after the last bias shift. This affect was accomplished by feeding the instances prior to the last bias shift back to the algorithm after the last instance had been read.

The results of all our tests are given in terms of trials considered[8], which is a good measure of the time required. The results for the three concepts are given below.

Concept #1

Fully consistent	BEANHEAD	Base
39	39	39

Concept #2

Fully consistent	BEANHEAD	Base
44	44	44

Concept #3

Fully consistent	BEANHEAD	Base
42	38	38

Table 6.1: Results using the LENSES database.

[8]Trials considered in this case include trails "reconsidered" during the assimilation phase.

As the data shows, the three methods performed identically on concept #1 and concept #2, but the base method and BEANHEAD outperformed the fully consistent method. This is because concept #3 is a 3-d concept, which requires two bias shifts. Since the fully consistent method requires that all previous trials be assimilated into the new version space after a bias shift, the first trials were assimilated three times, the initial time and one for each bias shift. However, the other methods delay assimilating past information; since both had switched to the 3-d bias (the correct bias) before the initial trials were exhausted, they avoided assimilating the earliest instances one extra time.

Though encouraging, these results gave us little insight on how successful the version space repair method of BEANHEAD was. It become apparent that it would be necessary to use databases that would require a several shifts of bias to see a noticeable difference between the methods. Due to our inability to find usable databases from real domains, we created a set of artificial databases with a very small number of instances. Each database contained every instance from the domain; the exact composition of all artificial databases are given in Appendix 2. Each database was a n x n matrix (two attributes), and was named according to the n and the required number of disjuncts. Our results follow;

Database 3-3

Fully consistent	BEANHEAD	Base
22	22	22

Database 3-4

Fully consistent	BEANHEAD	Base
29	23	29

Database 3-5

Fully consistent	BEANHEAD	Base
35	28	34

Table 6.2: Results using artificial databases with 3 x 3 attributes, requiring three to five disjuncts

Database 4-3

Fully consistent	BEANHEAD	Base
43	39	39

Database 4-4

Fully consistent	BEANHEAD	Base
54	44	51

Database 4-5

Fully consistent	BEANHEAD	Base
57	53	56

Table 6.3: Results using artificial databases with 4 x 4 attributes, requiring three to five disjuncts

Database 4-6

Fully consistent	BEANHEAD	Base
74	61	73

Database 4-7

Fully consistent	BEANHEAD	Base
85	55	84

Database 4-8

Fully consistent	BEANHEAD	Base
78	68	92

Table 6.4: Results using artificial databases with 4 x 4 attributes, requiring six to eight disjuncts

The results on this set of databases were in BEANHEAD's favor. For all the databases given above, BEANHEAD had the best performance of the three methods. Also, the base method out-performed the fully consistent method on most of the databases. The reason is the same as stated for the lenses database; since the base method delays reconsidering past instances until all instances have been considered, a savings can occur. Interestingly, this savings appears to be minimal in the above trials, with the base and fully

consistent method giving roughly the same performance. In one case, the 4-8 database, the base method performed considerably worse. This is because important instances (in terms of bias shifting) were encountered early in learning; since the base method does not reconsider such instances immediately, it took significantly longer to detect necessary bias shifts.

The fact that BEANHEAD also outperformed the base method, especially on the 4-8 database (where the base method's performance was so poor) is significant, because it indicates that the reason for its success on these databases is linked to its version-space reparations and not because it does not immediately reconsider the previous instances (the same method used by the base method).

Unfortunately, the results on these databases are somewhat biased, for all databases share the same characteristics; they all are complete, and they all have positive disjuncts that contain only one positive instance. It could be that BEANHEAD performs best on these sorts of databases, but poorly on others. Though we were limited by experimental difficulties previously mentioned, were able to devise several usable databases that do not share these characteristics. We defined three databases that had at least one positive disjunct that contained multiple positive instances, and two databases that were incomplete. We follow the nomenclature used earlier, although we add a "n" to the complete databases and an "i" to the incomplete databases. Our results follow.

Database n4-3

Fully consistent	BEANHEAD	Base
32	28	36

Database n4-4

Fully consistent	BEANHEAD	Base
54	46	49

Database n4-5

Fully consistent	BEANHEAD	Base
44	42	42

Table 6.5: Results using artificial databases with 4 x 4 attributes, requiring three to five disjuncts, some disjuncts nontrivial.

Database i10-3.1

Fully consistent	BEANHEAD	Base
41	36	36

Database i10-3.2

Fully consistent	BEANHEAD	Base
26	34	29

Table 6.6: Results using artificial databases with 10 x 10 attributes, requiring three disjuncts, with approximately 20% of possible instances represented.

The results on the complete databases with larger positive disjuncts is very similar to the results on the previous databases; BEANHEAD outperforms the others by approximately the same amount. However, we note that the incomplete databases the results are not the same as with the other databases; BEANHEAD has the best performance on one database, but had the worst on the other. Why the difference? Upon examining the trace of the BEANHEAD run on the i10-3.2 database, we realize BEANHEAD had switched to the 4-k bias, which is unnecessary for this database. This was a possible flaw with BEANHEAD's approach that we noted in the previous chapter. Since BEANHEAD aggressively re-uses the old version space, it generates a new version space that is usually more restrictive than necessary; in some cases it will cause BEANHEAD to switch to a bias that is too weak.

There are two reasons why BEANHEAD may have an empty version space (unlike with the other methods, which would only have the first of the following two reasons: the bias is insufficient; or the bias in not insufficient, but in a past reparation of the version space BEANHEAD arrived at a bad representation of the version space. Unfortunately, BEANHEAD does not know the reason it has arrived at an empty version space, and so it always assumes the first reason and shifts its bias. Presumably this hazard would be lessened for databases that require a weaker bias, because there would be less difference between consecutively weaker biases according to out hierarchy. Also, since the base method also performed worse than the fully consistent method on database i10-3.2, we realize that BEANHEAD also suffered because of the order it considers instances.

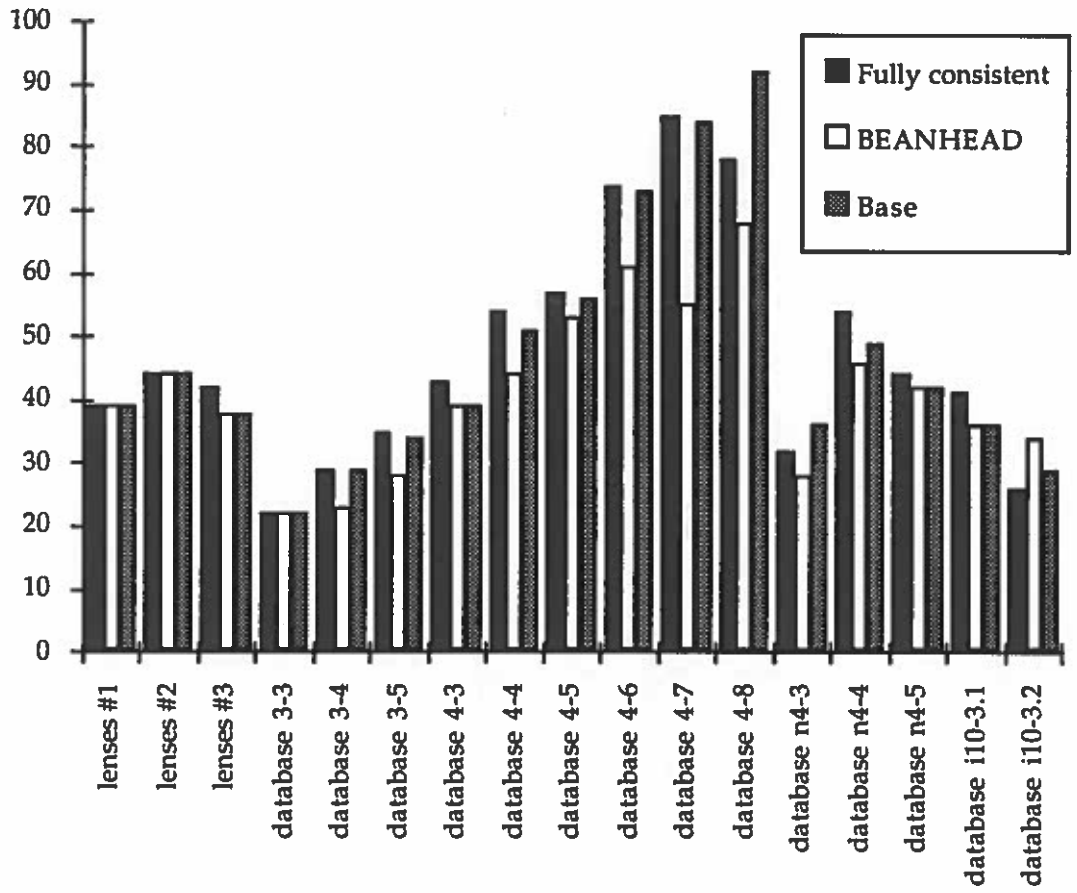


Figure 6.7: Graph of results on all databases.

CHAPTER VII

CONCLUSION

This thesis was motivated by a desire to better understand the intricacies of bias shifting, and to evaluate several methods of bias shifting. We used the candidate elimination algorithm as our method for studying bias, as it allows a direct encoding of arbitrary biases, and represents every hypothesis expressible in the given bias that is consistent with the training data. We studied the candidate elimination from a theoretical point of view, from which we made two salient observations; a relatively few number of trials are required to learn the concept with a small amount of error in most cases, and a relatively few number of trials are required to reveal an inadequate bias in most cases.

In order to perform an experiment that was impartial, we developed a generalized framework for learning an arbitrary concept with an arbitrary bias. In particular, we defined a hierarchy of biases, which is an ordering over the domain of possible biases, from the strongest bias to the weakest bias. Our model of bias shifting is rephrased in terms of traversing this hierarchy. However, an unguided traversal through this hierarchy would be difficult to implement and analyze; for this reason, we deemed it necessary to choose a subset of this hierarchy, a totally ordered set of biases where each successively weaker bias could express every concept expressible in the weaker biases. We

chose the class of disjunctive biases, with the purely conjunctive bias serving as our base case.

We presented three methods for shifting bias. The first was the fully-consistent method, which guaranteed that its version space was consistent at all times with previous training data, similar to the bias shifting method STABB. This consistency was guaranteed by maintaining a list of previous instances, and restarting the version space on the previous instances before considering the next instance. The second was the base method, which started the version space anew after a bias shift without immediately reconsidering past instances. Once the instances were exhausted, all instances encountered before the last bias shift were reconsidered, in order to guarantee consistency. The final method was BEANHEAD, an aggressive heuristic method that attempts to repair a version space and translate it into the weaker bias rather than completely disregard previous learning.

Unfortunately our experiment was somewhat thwarted by the lack of appropriate databases. In most cases, the databases we considered either required no bias shifting, or were intractable with our bias. One natural database did provide results, though there was not enough difference in performance to draw any conclusions. Therefore, we created a series of very small databases that required several bias shifts. Our data showed little difference between the fully-consistent method and the base method. However, BEANHEAD significantly outperformed the others in most cases, the improvement in performance was more pronounced as the number of bias shifts increased for the databases we considered.

Nonetheless, we recognize that the BEANHEAD method will not perform optimally in every case. In particular, because of its heuristic nature, it may "repair" the version space in such a way that improperly excludes the

target concept, even though it is expressible in the current bias. The result of this improper exclusion is that BEANHEAD will erroneously switch to a weaker bias than is necessary with the other methods in order to express the target concept. For the databases we used, this only occurred once; however, we realize that our evaluation was limited by difficulties with finding training data usable with our implementation and therefore the impact of our results is somewhat lessened.

Areas for Future Research

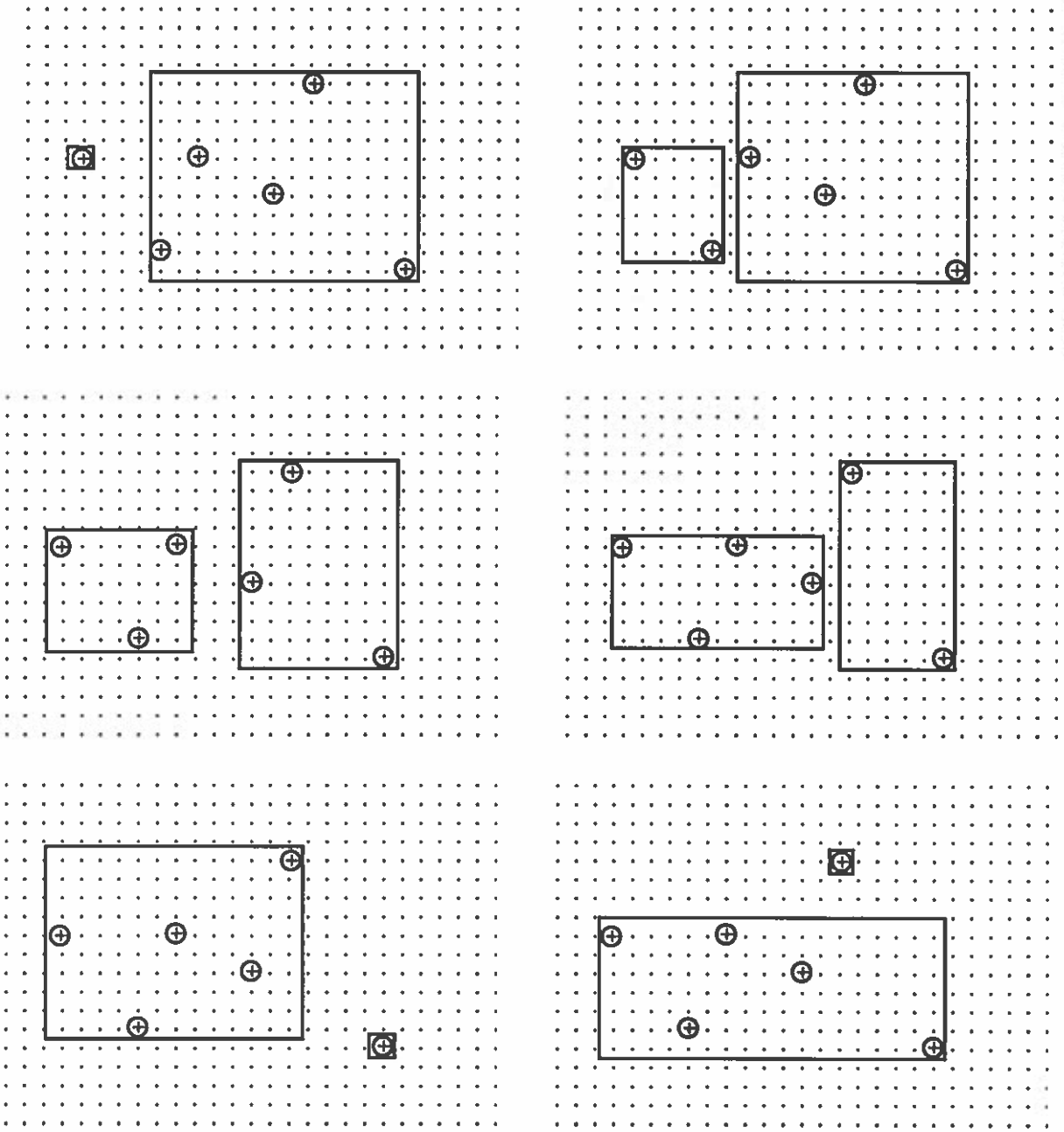
We were interested in exploring the possibility of using learning in an inadequate bias to guide learning in a correct bias. The success of BEANHEAD shows that such information can successfully enhance learning in the correct bias. BEANHEAD is but one possible method for re-using the learning in the incorrect bias. Various hybrid methods could be proposed, such as a method that uses more than just the most recent instance to repair the version space, or a method that could combine the three methods that we presented, deciding at run-time which method was likely to meet with the most success per bias-shift.

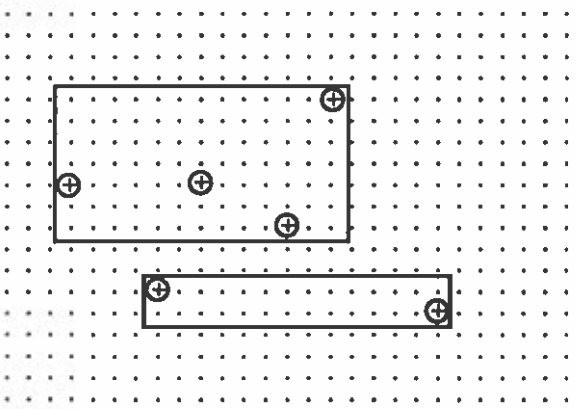
Further evaluation of BEANHEAD is also necessary to determine if it is a good heuristic for all learning problems or just those of the type considered. In particular, it would be illustrative to evaluate BEANHEAD using a different set of biases. Though our biases were straightforward to implement and understand, the granularity may have been too coarse for many real world domains. It would be revealing to test BEANHEAD's performance with a sequence of biases that allowed a much finer weakening of bias, such as with a tree-structured bias.

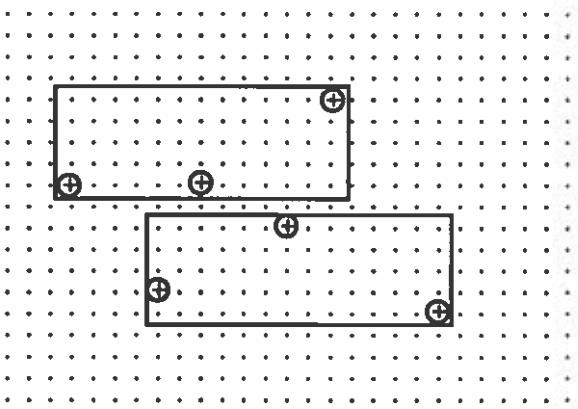
Finally, we would feel more confident with the results had they been acquired with databases that simulated real learning problems. It is an open question what the typical learning problem might be; therefore, it was hard to create databases that would assuredly give a fair test of the bias-shifting methods. Nonetheless, these initial results show that BEANHEAD is indeed a viable method in at least some domains, and that further research in making use of learning in an invalid bias may well prove profitable.

APPENDIX A

HYPOTHESIS IN THE *S* SET WITH THE 2-D BIAS







APPENDIX B

COMPOSITION OF ARTIFICIAL DATABASES

The following are a graphical representation of the artificial database used in this thesis to evaluate the competing bias shifting methods. Each database has only two attributes; '+' will be used to denote a positive instance, and '-' will be used to denote a negative instance.

3-3

```
+  -  -
-  +  -
-  -  +
```

3-4

```
+  -  -
-  +  -
+  -  +
```

3-5

```
+  -  -
-  +  -
+  -  +
```

4-3

```
+  -  -  -
-  +  -  -
-  -  +  -
-  -  -  -
```

4-4

```
+  -  -  -
-  +  -  -
-  -  +  -
-  -  -  +
```

4-5

```
+  -  -  -
-  +  -  -
+  -  +  -
-  -  -  +
```

4-6

+	-	-	-
-	+	-	+
+	-	+	-
-	-	-	+

4-7

+	-	-	-
-	+	-	+
+	-	+	-
-	+	-	+

4-8

+	-	-	+
-	+	-	+
+	-	+	-
-	+	-	+

BIBLIOGRAPHY

1. Cendrowska, J., Prism: An Algorithm for Inducing Modular Rules, *International Journal of Man-Machine Studies* **27** (1987) 349-370

2. Haussler, D., Quantifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework, *Artificial Intelligence* **36** (1988), 177-221

3. Mellish, C., The Description Identification Problem, *Artificial Intelligence* **52** (1991), 151-167

4. Michalski, R. S., A Theory and Methodology of Inductive Learning, *Artificial Intelligence* **20** (1983), 111-116

5. Mitchell, T. M., Version Spaces: an Approach to Concept Learning, Stanford CS Report STAN-CS-78-711, HPP-79-2 (1978)

6. Mitchell, T. M., The Need for Biases in Learning Generalizations, in: J. Shavlik, T. Dietterich (Eds.),*Readings in Machine Learning*, (Morgan Kaufmann Publishers, San Mateo, CA, 1978) 184-191

7. Mitchell, T. M., Generalization as Search, *Artificial Intelligence* **18** (1982), 203-206

8. Quinlan, J. R., Induction of Decision Trees, *Machine Learning* **1** (1986), 81-106

9. Rosenblatt, F., The Perceptron: a Probabilistic Model for Information Storage and Organization in the Brain, *Psychological Review* **65**, (6) (1958)

10. Utgoff, P. E., Shift of Bias for Inductive Concept Learning, in: Michalski, R. S., Carbonell, J. S., Mitchell, T. M. (Eds.), *Machine Learning: an Artificial Intelligence Approach, Volume II* (Morgan Kaufmann Publishers, San Mateo, CA, 1986)

11. Valiant, L. G., A Theory of the Learnable, *Communications of the ACM* **27** (1984) 1134-1142

12. Vapnik, V. N., and Chervonenkis, A. Y., On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities, *Theoretical Probability Applications* **16** (2) (1971) 264-280