Machine Learning, 5, 165–196 (1990)

Learning Nested Differences of Intersection-Closed Concept Classes

DAVID HELMBOLD (DPH@SATURN.UCSC.EDU)
Board of Studies in Computer and Information Sciences, University of California at Santa Cruz, Santa Cruz, CA 95064

ROBERT SLOAN (RSLOAN@ENDOR.HARVARD.EDU)
Aiken Computation Lab., Harvard University, Cambridge, MA 02138

MANFRED K. WARMUTH (MANFRED@SATURN.UCSC.EDU)
Board of Studies in Computer and Information Sciences, University of California at Santa Cruz, Santa Cruz, CA 95064

Abstract. This paper introduces a new framework for constructing learning algorithms. Our methods involve master algorithms which use learning algorithms for intersection-closed concept classes as subroutines. For example, we give a master algorithm capable of learning any concept class whose members can be expressed as nested differences (for example, $c_1 - (c_2 - (c_3 - (c_4 - c_5)))$) of concepts from an intersection-closed class. We show that our algorithms are optimal or nearly optimal with respect to several different criteria. These criteria include: the number of examples needed to produce a good hypothesis with high confidence, the worst case total number of mistakes made, and the expected number of mistakes made in the first t trials.

Keywords. concept learning, mistake bounds, exception handling, pac learning, nested difference, intersection-closed.

1. Introduction

We are interested in efficient algorithms for learning concepts from examples. Formally, *concepts* are subsets of some *instance domain* X from which instances are drawn and a *concept class* is a subset of 2^X, the power set of X. The instances are labeled consistently with a fixed *target concept* t which is in the concept class C to be learned; that is, an instance is labeled "+" if it lies in the target concept and "−" otherwise. Labeled instances are called *examples*.

There has been a surge of interest in learning from examples sparked by the introduction of a model of learning by Valiant (1984). This model accounts for both the performance of the learning algorithm as well as the computational resources and the number of examples used. Even though some practical learning algorithms have been found (Valiant, 1984; Rivest, 1987; Shvaytser, 1988; Littlestone, 1988; Blumer, Ehrenfeucht, Haussler & Warmuth, 1989; Haussler, 1989), and learnability of concept classes has been characterized (Blumer et al., 1989) using the Vapnik-Chervonenkis (VC) dimension (Vapnik and Chervonenkis, 1971), no practical algorithms have been found for many natural classes, such as DNFs, DFAs, and general decision trees. Recently strong evidence has been found that classes such as boolean formulae and DFAs are actually not efficiently learnable (Pitt and Warmuth, 1990b; Kearns and Valiant, 1989; Pitt and Warmuth, 1990a).

In this paper we give various schemes for composing known efficient learning algorithms to create provably efficient learning algorithms for more complicated problems. Thus we give *constructive* results for learning new classes of concepts for which efficient learning algorithms were not previously known. The composition technique consists of new master algorithms that use the algorithms for the "simpler" classes as subroutines. The master algorithms learn nontrivially more complicated classes which can be defined in terms of the simpler classes. The master algorithms do not need to know the specific simpler classes, since they only pass information among the various algorithms for the simpler classes.

The simple classes considered here are usually intersection-closed concept classes and the master algorithms learn various compositions of intersection-closed concept classes. (A concept class is *intersection-closed* if for any finite set contained in some concept the intersection of all concepts containing the finite set is also a concept in the class.) There is a canonical algorithm for learning intersection-closed classes which we call here the *Closure* algorithm: The hypothesis of this algorithm is always the smallest concept containing all of the positive examples (POS) seen so far (Natarajan, 1987). We denote this concept as CLOS(POS). This paper presents new algorithms, which use the Closure algorithm as a subroutine, for learning compositions of intersection-closed concept classes.

The simplest composition scheme we consider learns the concept class DIFF(C), which consists of all concepts of the form $c_1 - (c_2 - (c_3 - \cdots - (c_{p-1} - c_p) \cdots))$, where all c_i are in C, and p is a positive integer called the *depth* of the concept. It is easy to see that an instance x is in the concept $c_1 - (c_2 - (c_3 - \cdots - (c_{p-1} - c_p) \cdots))$ if and only if the lowest indexed c_i that does not contain x has an even index (assume for convenience that $c_{p+1} = \emptyset$).

A more involved scheme efficiently learns the class DIFF($C_1 \cup C_2 \cup \cdots \cup C_s$); that is, each c_i may be in any C_j, for $1 \leq j \leq s$. This scheme assumes that each class C_j is interesection-closed and that their Closure algorithms can be implemented efficiently.

Examples of intersection-closed classes with efficient Closure algorithms include orthogonal rectangles in R^n, monomials (i.e., orthogonal sub-rectangles of the boolean hypercube), vector sub-spaces of R^n (Shvaytser, 1988), and so forth. In Figures 1 and 2 we give examples of DIFF(C), when C is the class of orthogonal rectangles in R^2. In this case DIFF(C) contains staircase type objects and some restricted unions of orthogonal rectangles. If C is the class of initial segments on the real line (orthogonal rectangles in dimension one with the same left endpoint), then DIFF(C) is the class of unions of intervals. For finite domains, it has been shown (Natarajan, 1987) that a concept class is learnable with one-sided error (i.e., the error with respect to the negative distribution is zero)[1] if and only if it is intersection closed and the VC dimension of the class grows polynomially in the relevant parameters.

If a concept class B is not intersection closed one can always embed it into a larger class C that is. However, the VC dimension of C may be much larger than the VC dimension of the original class B. The Closure algorithm learns an intersection-closed class C from positive examples only. Note that DIFF(C) is not necessarily intersection-closed, and our master algorithms for learning DIFF(C) will use both positive and negative examples.

One can generalize the composition scheme for DIFF(C) by allowing the innermost concept, c_p, to be in an *arbitrary* polynomially learnable class B (the learning algorithm for B may use both positive and negative examples). Let DIFF(C, B) be all concepts of the

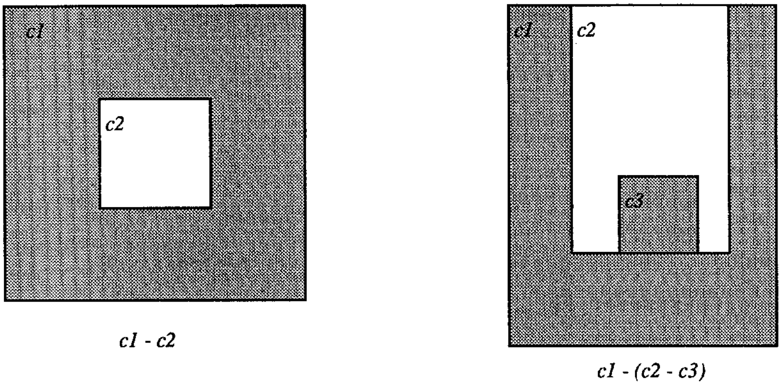

Figure 1. Two concepts in DIFF(Orthogonal Rectangles).

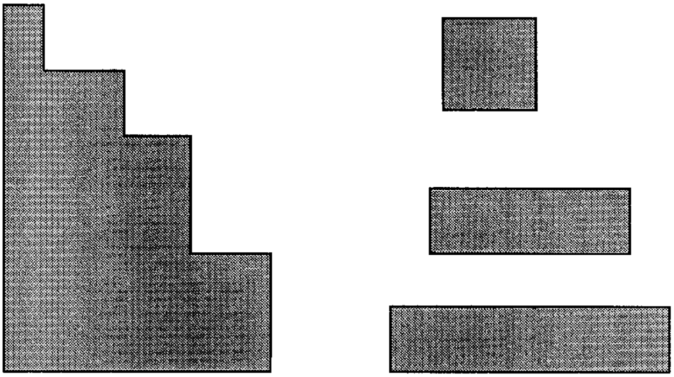

Figure 2. Another two concepts in DIFF(Orthogonal Rectangles).

form $c_1 - (c_2 - (c_3 - \cdots - (c_{p-1} - b) \cdots))$ where the c_j are in the intersection-closed class C and b is in B. The c_j are in some sense a filter formed with special concepts, while b is allowed to be more general.

Concepts in DIFF(C) and DIFF(C, B) of depth two were previously shown to be learnable in (Kearns, Li, Pitt and Valiant, 1987). In this paper, we consider the case of arbitrary depth. Observe the following closure properties: for two intersection-closed classes C_1 and C_2, the class $C_1 \wedge C_2 = \{c_1 \cap c_2 : c_1 \in C_1 \text{ and } c_2 \in C_2\}$ is intersection closed as well; the same holds for the class $C_1 \cap C_2 = \{c : c \in C_1 \text{ and } c \in C_2\}$, and dual results hold if intersection is replaced by union. Note also that C is intersection closed if and only if $\bar{C}$, which consists of the complements of the concepts of C, is "union closed."

In (Rivest, 1987) an algorithm is given for learning decision lists, which include nested differences of constant size monomials. In this paper we learn nested differences of intersection-closed classes by exploiting the combinatorial properties of intersection closedness. Since the class of monomials (of arbitrary size) is intersection closed, this leads to a learning algorithm for the nested differences of monomials. Constant size monomials are not intersection closed; however, this class of monomials is "simple" in the sense that it contains few concepts. The decision list algorithm performs an exhaustive search, thus relying on the smallness of its simple class.

We have developed two types of master algorithms, one that remembers all examples seen (the Total Recall algorithm), and one that remembers a number of examples bounded by the VC dimension (the Space Efficient algorithm).

We recently discovered that Steven Salzberg (1988) has independently developed a space efficient algorithm similar to ours for DIFF(C) (where C consists of orthogonal rectangles in R^n) as a subroutine in his algorithm for predicting, among other things, breast cancer data. In some cases, his algorithms outperform the best previously known prediction algorithms. However, those results are only empirical. The crux of the type of research presented here is that using the methodology of computational learning theory (Valiant, 1984; Haussler, Littlestone & Warmuth, 1988) (which is rooted in the earlier works of Vapnik and others in the area of pattern recognition (Vapnik and Chervonenkis, 1971; Vapnik, 1982)), we can give efficient algorithms and *prove* their optimality.

In a companion paper (Helmbold, Sloan & Warmuth, 1989a) we present an interesting application of our methods. We give a time and space efficient implementation of the Closure algorithm for a nontrivial intersection-closed class: the class C of subsets of $\mathbf{Z}^k$ that are closed under addition and subtraction. In algebraic terms C consists of all submodules[2] of the free $\mathbf{Z}$-module of rank k. The space efficient Closure algorithm for submodules can be used as a subroutine in the space efficient master algorithm, leading to learning algorithms for nested differences of submodules.

2. The inclusion-exclusion algorithms

In this section we present the Total Recall and Space Efficient algorithms for learning DIFF(C), nested differences of an intersection-closed class C. The first algorithm assumes *total recall*; that is, sufficient space is available to store all of the examples. Then we show how the algorithm can be modified for space efficiency (so that only a few examples are memorized).

A *batch learning algorithm* takes as input a set of labeled training examples and produces, as output, a hypothesis. The examples are labeled according to some unknown *target concept* from the concept class to be learned. Intuitively, the hypothesis is supposed to approximate the hidden target concept. (Note that the hypothesis is not required to be in the concept class to be learned.) An *on-line learning algorithm* interactively participates in a series of *trials*. On each trial, the algorithm gets an unlabeled instance and predicts what its label is. After predicting, the on-line algorithm is informed of the instance's true label. A *mistake* is a trial where the on-line algorithm makes an incorrect prediction. The distinction between batch and on-line algorithms is blurred by the fact that a batch algorithm can be used in an on-line setting and vice versa. To use a batch algorithm in an on-line setting, give it all the examples previously seen and predict with the resulting hypothesis on the next trial. An on-line algorithm can be run in batch mode by feeding it all of the training examples (ignoring its predictions) and using as its hypothesis the set of instances where, if seen on the next trial, the algorithm would predict "+".

We present the Total Recall algorithm as a batch algorithm and the Space Efficient algorithm as an on-line algorithm. From the above discussion it is easy to convert either algorithm to the other setting.

Before describing our two basic algorithms we give formal definitions of *closure* and *intersection closed.*

DEFINITION. For any concept class C and any subset S of the domain the *closure of S with respect to C*, denoted by CLOS(S), is the set $\bigcap \{c : c \in C \text{ and } S \subseteq c\}$. A concept class C is *intersection closed*[3] if C contains at least two concepts and whenever S is a finite subset of some concept in C then CLOS(S) is a concept of C.

Note that if C is intersection closed, then CLOS($\emptyset$) (i.e., the intersection of all concepts in C) is a member of C.

Description of the Total Recall algorithm: The algorithm first computes the closure of the positive examples. In general, this closure may contain some negative examples. These exceptions must be subtracted out of the hypothesis. The algorithm now focuses on those examples contained in the closure. By flipping their labels and computing the closure of the resulting positive examples (which were the original exceptions), the algorithm finds a suitable concept to subtract off. This concept may contain further exceptions—examples that were originally positive. However, by iterating this procedure, the Total Recall algorithm creates a nested difference consistent with the examples.

A detailed description of the Total Recall algorithm is given in Figure 3. For any intersection-closed class C it learns DIFF(C) assuming an efficient implementation of the Closure algorithm. The function POS takes a set of examples and returns those which are labeled positive. The function FLIP takes a set of examples and returns a new set of examples containing the same instances but with the labels flipped (i.e., "+" examples become "−" examples and vice versa).

The concept $h = h_1 - (h_2 - \cdots - (h_{p-1} - h_p) \cdots)$ is the hypothesis of the algorithm. If C contains the empty concept, then one possible hypothesis of depth one is $h = h_1 = \emptyset$. For syntactic purposes, we set h_{p+1} to $\emptyset$ (even if C does not contain the empty concept), ensuring that for any instance there is always some h_i which does not contain the instance.

When given a new instance x, the algorithm predicts its label according to h as follows (see Figure 4). Let l be the least index such that h_l does not contain x. If l is even then $h(x) = +$, and if l is odd then $h(x) = -$. In the on-line setting, the example is added to EX_1 even if the prediction was correct, and the Total Recall algorithm is executed to generate a new hypothesis.[4] This ensures that hypotheses produced by the Total Recall algorithm are consistent with all the examples that have been seen.

The Space Efficient algorithm differs from the Total Recall algorithm in that it keeps only a minimal number of instances for each h_i.

```
Algorithm Total Recall
Inputs: Examples of the target concept.
/* Computes hypothesis $h$ and depth $p$. */
$EX_1$ := all examples; $i := 0$;
repeat
        increment $i$;
        $h_i := CLOS(POS(EX_i))$;
        $EX_{i+1} := FLIP(h_i \cap EX_i)$;
        /* "FLIP" flips the labels of the examples */
until $POS(EX_{i+1}) = \emptyset$;
$p := i$; $h_{p+1} := \emptyset$;
$h := h_1 - (h_2 - \cdots - (h_{p-1} - h_p)$;
```

Figure 3. Total Recall algorithm.

```
Algorithm Predict
Inputs: Hypothesis $h$ and instance $x$.
/* Computes whether $x$ is in set represented by $h$. */
$l := 0$;
repeat $l := l + 1$ until $x \notin h_l$;
If $l$ is even then output +;
            else output −;
```

Figure 4. Prediction algorithm.

DEFINITION. Let S be a set of instances contained in some concept of C. A *spanning set* of S (with respect to some intersection-closed concept class C) is any $S' \subseteq S$ for which $\mathrm{CLOS}(S') = \mathrm{CLOS}(S)$.

Description of the Space Efficient algorithm: This algorithm (for a detailed description see Figure 5) represents each h_i by a minimal spanning set, S_i, and the hypothesis $h = h_1 - (h_2 - (h_{p-1} \cdots - h_p))$ by a sequence of minimal spanning sets, $S_1, \ldots, S_p$. Predicting is done as in the Total Recall algorithm. However, if there was a mistake made on an instance x and l is the least index such that $x \notin h_l$, then S_l is updated to a minimal spanning set of $S_l \cup \{x\}$ and thus h is modified by changing h_l to $\mathrm{CLOS}(S_l \cup \{x\})$. An illustration of the Space Efficient algorithm with domain R^2 and concept class DIFF(orthogonal rectangles) is given in Figures 6 and 7.

One would expect the Total Recall algorithm to perform well, since it produces a hypothesis in DIFF(C) of minimal depth that is consistent with the training examples. Because the

```
Algorithm Space Efficient
Inputs: Current hypothesis $h$, current depth $p$, and instance $x$.
/* On-line algorithm */
If first call then
        initialize $p := 1$; $S_1 := \emptyset$; $h_1 :=$ CLOS($\emptyset$); $h_2 := \emptyset$;
Call Predict($h, x$);
If mistake made then
        $l := 0$;
        repeat $l := l + 1$ until $x \notin h_l$;
        $S_l :=$ minimal spanning set($S_l \cup \{x\}$);
        $h_l :=$ CLOS($S_l$);
        if $l > p$ then $p := l$; $h_{p+1} := \emptyset$;
```

Figure 5. Space Efficient algorithm.

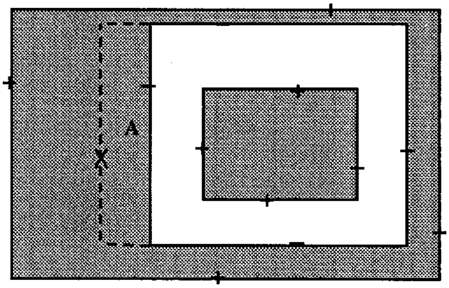

Figure 6. The minimal spanning sets stored by the Space Efficient algorithm and the corresponding hypothesis. The prediction for instance X is positive.

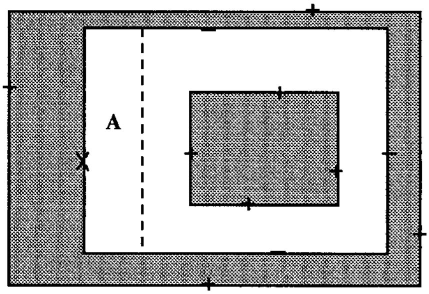

Figure 7. If a mistake was made on instance X, then the Space Efficient algorithm's new hypothesis is shown above. Note that the algorithm may have previously seen "+" points in region A. If it sees one of these points again, it will predict incorrectly.

hypothesis of the Space Efficient algorithm may not be consistent with the training examples (see Figures 6 and 7), the goodness of its performance is less clear. The remainder of the paper is devoted to proving performance bounds for these and related algorithms.

3. Performance criteria

We begin by enumerating several criteria by which we can judge how well a learning algorithm performs. In this enumeration each criteria is described informally. The definitions of the criteria are then formalized in the following subsections.

1. Bounds on the total number of mistakes made in any sequence of t trials, where an adversary chooses the examples and their order (Littlestone, 1988).
2. Bounds on the probability of making a mistake at trial t in a sequence of t trials, where the t instances are chosen by an adversary, but their order is picked at random from among the $t!$ possible permutations (Haussler et al. 1988).
3. Bounds on the probability of making a mistake at trial t in a sequence of t trials, where an adversary chooses a probability distribution over the instance domain, and the t instances are randomly drawn according to that distribution (Haussler et al. 1988).
4. Bounds on the expected total number of mistakes made in any sequence of t trials, where the examples are chosen by an adversary, but their order is picked at random from among the $t!$ possible permutations.
5. Bounds on the expected total number of mistakes made in a sequence of t trials where an adversary chooses a probability distribution over the instance domain, and the t instances are randomly drawn according to that distribution.
6. Bounds on the number of randomly drawn examples required for a batch algorithm to almost certainly produce a good hypothesis (Valiant, 1984).

The first five criteria, and indeed all mistake-based criteria, only make sense for on-line learning algorithms, while the last criteria is applicable only to batch algorithms.

We show in the next section that the Total Recall algorithm is optimal for criteria 2 through 6. Furthermore, if the Closure algorithm is optimal, then both the Total Recall and the Space Efficient algorithms are optimal under the first criteria. All of this paper's results are summarized in the conclusions.

3.1. *Worst case mistake bounds*

Our first performance criterion deals with the maximum number of mistakes made by the learning algorithm on any sequence of trials and was introduced by Littlestone (Littlestone, 1988).

For algorithm $\mathcal{A}$ and target concept c, define $\mathbf{M}_{\mathcal{A}}(c)$ to be the maximum number of mistakes made by algorithm $\mathcal{A}$ on any possible sequence of trials where the instances are labeled according to c. For any concept class C, define the *worst case mistake bound of* $\mathcal{A}$, $\mathbf{M}_{\mathcal{A}}(C) = \sup_{c \epsilon C} \mathbf{M}_{\mathcal{A}}(c)$. If the context makes it clear which algorithm's performance is being bounded, a "·" may be substituted for the subscript "$\mathcal{A}$".

3.2. *Instantaneous mistake bounds*

The following two instantaneous mistake bound measures were introduced by Haussler, Littlestone, and Warmuth (1988). Here they are called "instantaneous" because they bound the probability of a mistake on single trials.

The *instantaneous permutation mistake bound* for algorithm $\mathcal{A}$ and concept c is written $\hat{\hat{\mathbf{MI}}}_{\mathcal{A}}(c)(t)$. It is the supremum, over all multi-sets containing t labeled examples of concept c, of the probability that $\mathcal{A}$ makes a mistake on trial t where each of the $t!$ different orders in which the instances could be presented to $\mathcal{A}$ is equally likely.

The *instantaneous sample mistake bound* for algorithm $\mathcal{A}$ on concept c, written $\widehat{\mathbf{MI}}_{\mathcal{A}}(c)(t)$, is the supremum over all probability distributions P on the domain of the probability that algorithm $\mathcal{A}$ makes a mistake at trial t, when given t labeled examples of c which are generated by drawing independently at random from the instance domain according to P.

We generalize from concepts to concept classes in the same way as for worst case mistake bounds, so that $\hat{\hat{\mathbf{MI}}}_{\mathcal{A}}(C)(t) = \sup_{c \epsilon C}\mathbf{MI}_{\mathcal{A}}(c)(t)$, etc.

3.3. *Expected cumulative mistake bounds*

A simple variation of the instantaneous mistake bounds in (Haussler et al. 1988) is bounding the expected total number of mistakes made on a sequence of t trials rather than simply the probability of a mistake on the last trial.

We define the *cumulative permutation mistake bound* for algorithm $\mathcal{A}$ and concept c, written $\hat{\hat{\mathbf{MT}}}_{\mathcal{A}}(c)(t)$, as the supremum over all multi-sets of t examples labeled consistently with concept c of the expected total number of mistakes made by algorithm $\mathcal{A}$, where each of the $t!$ different orders in which the examples could be presented to $\mathcal{A}$ is equally likely.

The *cumulative sample mistake bound* for algorithm $\mathcal{A}$ and concept c, written $\widehat{\mathbf{MT}}_{\mathcal{A}}(c)(t)$, is the supremum, over all probability distributions P on the domain, of the expected total number of mistakes made by algorithm $\mathcal{A}$ over t trials generated by drawing independently at random (with respect to P) examples of c from the instance domain.

Again, we generalize from concepts to concept classes in the same way as for worst case mistake bounds. Although we are mainly interested in instantaneous and cumulative sample mistake bounds, the corresponding permutation mistake bounds are frequently easier to estimate. By the following lemma of (Haussler et al. 1988), the permutation mistake bounds are an upper limit on the sample mistake bounds.

LEMMA 1. For any on-line learning algorithm $\mathcal{A}$:

$$\begin{aligned} \hat{\hat{\mathbf{MI}}}_{\mathcal{A}}(C)(t) &\geq \widehat{\mathbf{MI}}_{\mathcal{A}}(C)(t); \\ \hat{\hat{\mathbf{MT}}}_{\mathcal{A}}(C)(t) &\geq \widehat{\mathbf{MT}}_{\mathcal{A}}(C)(t). \end{aligned}$$

It is not hard to see that several other relationships between the mistake bounds also hold, including:

$$\mathbf{M}_{\mathcal{A}}(C) \geq \hat{\hat{\mathbf{MT}}}_{\mathcal{A}}(C)(t),$$

$$\sum_{1 \leq i \leq t} \hat{\hat{\mathbf{MI}}}_{\mathcal{A}}(C)(i) \geq \hat{\hat{\mathbf{MT}}}_{\mathcal{A}}(C)(t), \text{ and}$$

$$\sum_{1 \leq i \leq t} \hat{\mathbf{MI}}_{\mathcal{A}}(C)(i) \geq \hat{\mathbf{MT}}_{\mathcal{A}}(C)(t).$$

3.4. *PAC style bounds*

The final criteria we consider is appropriate for batch learning algorithms and is due to Valiant (Valiant, 1984). This criteria deals with the minimum number of random examples needed by the learning algorithm in order to "almost always" produce a hypothesis "very close" to the target concept.

Let P be a probability distribution over the instance domain. The *error* of a hypothesis is the probability (with respect to P) of all instances in the symmetric difference between the hypothesis and the target concept. Given a batch learning algorithm $\mathcal{A}$ and probability distribution P, we define the function $t_{\mathcal{A}}(c, \epsilon, \delta, P)$ as the smallest number such that after seeing $t_{\mathcal{A}}(c, \epsilon, \delta, P)$ examples drawn independently at random and labeled according to c, $\mathcal{A}$ outputs, with probability at least $1 - \delta$, a hypothesis whose error (with respect to P) is at most ϵ. The function $t_{\mathcal{A}}(C, \epsilon, \delta)$ is the supremum over all $c \in C$ and all probability distributions P of $t_{\mathcal{A}}(c, \epsilon, \delta, P)$.

Note that the above definitions do not require that the hypotheses of algorithm $\mathcal{A}$ are members of the class being learned. However, the hypotheses of most of our algorithms are, in fact, members of C. We explicitly point out those algorithms using hypotheses which are not members of the concept class.

4. The performance of the Total Recall and the Space Efficient algorithms

The goal of this section is to evaluate both algorithms with the performance criteria introduced in the previous section. A very important combinatorial parameter used to estimate the complexity of learning a concept class is its Vapnik-Chervonenkis dimension (see (Vapnik and Chervonenkis, 1971; Haussler and Welzl, 1987; Pearl, 1978; Blumer et al. 1989)). For example, here it will help us bound the number of instances stored by the Space Efficient algorithm.

DEFINITION. A set of instances, S, is *shattered* (by the concept class C) if for each subset $S' \subseteq S$, there is a concept $c \in C$ which contains all of S', but none of the instances in $S - S'$. The *Vapnik-Chervonenkis dimension* of a concept class, denoted by $VCdim(C)$, is the highest cardinality such that there exists a set of that cardinality shattered by the concept class.

THEOREM 1. Given an intersection-closed concept class C and subset S of the domain, every minimal spanning set of S is shattered by C.

Proof. Let S' be a minimal spanning set of S. Since C is intersection closed, it suffices to show that some concept contains all of S', and for each $x \in S'$, some concept of C contains $S' - \{x\}$ but not x. The closure of S' is a concept in C containing S'. Since S' is minimal, if $x \in S'$ then CLOS(S') $\neq$ CLOS($S' - \{x\}$). Therefore, the closure of $S' - \{x\}$ contains $S' - \{x\}$ but not x. □

COROLLARY 1. All minimal spanning sets of a set S (with respect to C) have size at most *VCdim*(C).

Theorem 1 or statements equivalent to it have appeared in several places, including (Natarajan, 1987; Boucheron, 1988). Note that any shattered set is its own minimal spanning set.

An important fact about our algorithms is that in some sense they converge to the target concept from below. This idea is made more precise in the following lemma.

LEMMA 2. When given a set of examples consistent with some $c = c_1 - (c_2 - \cdots (c_{p-1} - c_p) \cdots)$ in DIFF(C), both the Total Recall and Space Efficient algorithms produce a hypothesis $h = h_1 - (h_2 - \cdots - (h_{p'-1} - h_{p'}) \cdots)$ where $p' \leq p$ and for all $1 \leq i \leq p'$, $c_i \supseteq h_i$. In addition, the Total Recall algorithm's hypothesis is consistent with the set of examples.

Proof. By induction on p' for the Total Recall algorithm and by induction on the number of mistakes made by the Space Efficient algorithm. □

Often we will want to focus our attention on concepts in DIFF(C) of restricted depth. To meet this need we define the concept class DIFF$^{\leq p}$(C) (for $p \geq 1$) as the set of all concepts in DIFF(C) whose depth is between 1 and p. The concept class DIFF$^{=p}$(C) is defined analogously.

It is easy to see that *VCdim*(DIFF$^{=p}$(C)) $= p \cdot$ *VCdim*(C) for at least some concept classes (for example, orthogonal rectangles). The following lemma shows that this is also a general upper bound on the VC dimension of DIFF$^{\leq p}$(C).

LEMMA 3. If C is intersection closed[5] and $p \geq 1$, then

$$VCdim(\mathrm{DIFF}^{=p}(C)) \leq VCdim(\mathrm{DIFF}^{\leq p}(C)) \leq p\ VCdim(C).$$

Proof. The first inequality is trivial since DIFF$^{=p}$(C) $\subseteq$ DIFF$^{\leq p}$(C). We prove the second inequality by induction on p. Let d = *VCdim*(C). The lemma trivially holds when $p = 1$ since by definition DIFF$^{\leq 1}$(C) = DIFF$^{=1}$(C) = C.

Assume to the contrary that DIFF$^{\leq p}$(C) shatters at most pd elements, but DIFF$^{\leq p+1}$(C) shatters a set S of size $(p + 1)d + 1$. This means that for every assignment of labels to instances in S, there is a concept $c_1 - (c_2 - (c_3 - \cdots - (c_{l-1} - c_l) \cdots)) \in$ DIFF$^{\leq p+1}$(C) (where $l \leq p + 1$) agreeing with the label assignment.

Let S' be a minimal spanning set of S with respect to C. By Theorem 1, $|S'| \leq d$. Consider a labeling of the instances in S where every instance in S' is labeled positive. If a concept $c = c_1 - (c_2 - (c_3 - \cdots - (c_{l-1} - c_l) \cdots)) \in \mathrm{DIFF}^{\leq p+1}(C)$ is consistent with this labeling, then c_1 must contain all of S' (and thus all of S). Therefore, for S to be shattered, the remaining c_i's (which form concepts in $\mathrm{DIFF}^{\leq p}(C)$) must be capable of shattering the remaining $pd + 1$ instances in $S - S'$; contradiction. □

Observe that (a) the hypothesis of the Total Recall algorithm is always consistent with the examples seen and (b) the hypothesis produced does not depend on the order in which the examples are seen. We can use (b) to obtain estimates of the permutation and sample mistake bounds described in Sections 3.3 and 3.2. Note that the Space Efficient algorithm satisfies neither (a) nor (b); however, we are still able to prove a worst case mistake bound for the Space Efficient algorithm (deferred to later) expressed as a function of the worst case mistake bound of the Closure algorithm for C.

Since (a) holds for the Total Recall algorithm, we can apply the following result of (Blumer et al. 1989):

THEOREM 2. Let B be a concept class over some instance domain X, and let P be a fixed well-behaved[6] probability distribution on X. Then for a sample S of size at least

$$\max\left(\frac{4}{\epsilon}\log_2\frac{4}{\delta}, \frac{8VCdim(B)}{\epsilon}\log_2\frac{13}{\epsilon}\right),$$

drawn independently at random according to P, and labeled consistently with some target concept $c \in B$, the probability that all $b \in B$ that are consistent with S have error at most ϵ is at least $1 - \delta$. This holds for any ϵ and δ between 0 and 1.

Using our notation, the above theorem shows that for any algorithm $\mathcal{B}$ which produces consistent hypotheses from $\mathcal{B}$,

$$t_{\mathcal{B}}(B, \epsilon, \delta) \leq \max\left(\frac{4}{\epsilon}\log_2\frac{4}{\delta}, \frac{8VCdim(B)}{\epsilon}\log_2\frac{13}{\epsilon}\right).$$

Applying this result with $B = \mathrm{DIFF}^{\leq p}(C)$ together with the fact that the Total Recall algorithm produces a consistent hypothesis in $\mathrm{DIFF}^{\leq p}(C)$ (Lemma 2) leads to the following:

THEOREM 3. For the Total Recall algorithm,

$$t(\mathrm{DIFF}^{\leq p}(C), \epsilon, \delta) \leq \max\left(\frac{4}{\epsilon}\log_2\frac{4}{\delta}, \frac{8p\ VCdim(B)}{\epsilon}\log_2\frac{13}{\epsilon}\right), \text{ for any } \epsilon, \delta \in [0, 1].$$

To bound both the probability of making a prediction mistake on the t-th instance and the expected total number of mistakes on the first t instances, we use the methodology developed in (Haussler et al. 1988), which bounds the probability of a mistake at trial t by averaging over all permutations of the input (Lemma 1).

DEFINITION. Let S be a sequence of t examples, $x \in S$, and $\mathcal{A}$ an on-line algorithm. The instance x is a *corner* of S (with respect to $\mathcal{A}$) if there is a permutation of S where both x occurs last and $\mathcal{A}$ makes a mistake on instance x when given the permuted sequence of examples.

The following lemma relates the maximum number of corners with respect to some algorithm to that algorithm's instantaneous permutation mistake bound.

LEMMA 4. Let $\mathcal{A}$ be an on-line learning algorithm for concept class C, and n be the maximum number of corners (with respect to $\mathcal{A}$) in any sequence of t examples labeled consistently with some $c \in C$. Then

$$\hat{\hat{\mathbf{MI}}}_{\mathcal{A}}(C)(t) \leq \frac{n}{t}.$$

Proof. Let S be any multi-set of t examples labeled consistently with some concept in C. Consider the $t!$ permutations of S. Each element of S occurs last in exactly $(t - 1)!$ of the permutations. Since S has at most n corners, there are at most $n(t - 1)!$ permutations of S where $\mathcal{A}$ makes a mistake on the t-th trial. Therefore,

$$\hat{\hat{\mathbf{MI}}}_{\mathcal{A}}(C)(t) \leq \frac{n(t-1)!}{t!} = \frac{n}{t}. \qquad \square$$

Note that an example x is a corner of S with respect to the Closure algorithm if and only if $x \in POS(S)$ and $x \notin CLOS(POS(S) - \{x\})$. Unless stated otherwise, corners are with respect to the closure algorithm. The corners of a set are in some sense the extremal instances of the set. Notice that some minimal spanning sets of S may contain instances that are not corners of S, and that the closure of the corners of S may not be equal to S (see Figure 8 for examples). However, the following lemma points out an important relationship between minimal spanning sets and corners.

LEMMA 5. Let S be a (multi-)set of positive instances in the domain. The set of corners of S (with respect to the Closure algorithm) is the intersection of all the (minimal) spanning sets of S.

Proof. Let x be a corner of S. Since x is not in the closure of $S - \{x\}$, x must be in every $S' \subseteq S$ for which $\mathrm{CLOS}(S') = \mathrm{CLOS}(S)$. Therefore x is in all minimal spanning sets of S. Conversely, if x is in every minimal spanning set of S, then no subset of $S - \{x\}$ (including $S - \{x\}$ itself) can have the same closure as S. Hence x must be a corner. □

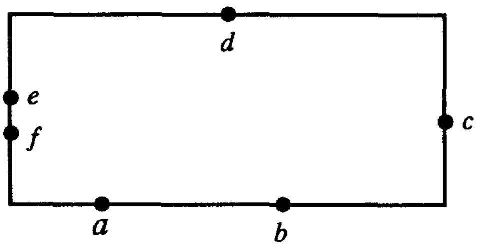

Figure 8. Only points c and d are corners. Both $\{a, c, d, e\}$ and $\{a, c, d, f\}$ are among the spanning sets.

It follows from Corollary 1 and Lemma 5 that, when S is a set of positive examples of a concept, the number of corners of S (with respect to the Closure algorithm) is no greater than the VC dimension of the concept class. The canonical Closure algorithm for learning intersection closed concept classes makes a mistake on the t-th instance, x_t, only when x_t is one of the corners of the set of all positive examples seen in the first t trials. By Lemma 4, the probability that the Closure algorithm makes a mistake on the t-th trial is at most

$$\frac{\text{\# of corners}}{t} \leq \frac{VCdim(C)}{t},$$

and the expected total number of mistakes in the first t trials is thus bounded by $VCdim(C) \cdot H_t$ (where H_t is the t-th harmonic number).

THEOREM 4. For the Total Recall algorithm:

1. $\widehat{\widehat{\mathbf{MI}}}.(\mathrm{DIFF}^{\leq p}(C))(t) \leq p\ VCdim(C)/t$,
2. $\widehat{\widehat{\mathbf{MT}}}.(\mathrm{DIFF}^{\leq p}(C))(t) \leq p\ VCdim(C)H_t$,
3. $\widehat{\mathbf{MI}}.(\mathrm{DIFF}^{\leq p}(C))(t) \leq p\ VCdim(C)/t$, and
4. $\widehat{\mathbf{MT}}.(\mathrm{DIFF}^{\leq p}(C))(t) \leq p\ VCdim(C)H_t$.

Proof. We prove the first part of the theorem here. The other statements follow from the relationships between performance measures in and following Lemma 1. The Total Recall algorithm for learning DIFF(C) makes a mistake only when the Closure algorithm makes a mistake learning some c_i at level i. If the Total Recall algorithm makes a mistake on x_t, then x_t is a corner (with respect to the Closure algorithm for C) of some EX_i. Therefore, given a sequence of t examples labeled consistently with some concept in $\mathrm{DIFF}^{\leq p}(C)$, the number of corners with respect to the Total Recall algorithm is at most p $VCdim(C)$. By Lemma 4, $\widehat{\widehat{\mathbf{MI}}}.(\mathrm{DIFF}^{\leq p}(C))(t) \leq p\ VCdim(C)/t$. □

Note that if the Closure algorithm is used when learning an intersection-closed class, no mistakes are made on the negative examples. Furthermore, negative examples do not cause changes to the hypothesis. Each mistake is caused by a positive example which increases the closure of all positive examples seen. Thus $\mathbf{M}_{\mathrm{CLOS}}(c)$ is the maximum number of mistakes made by the Closure algorithm on any sequence of *positive* instances of c.

Using $\mathbf{M}_{\mathrm{CLOS}}(C)$ we can obtain a worst case bound on the number of mistakes made by our master algorithms.

THEOREM 5. For both the Total Recall and the Space Efficient algorithm, $\mathbf{M}(\mathrm{DIFF}^{\leq p}(C)) \leq p\mathbf{M}_{\mathrm{CLOS}}(C)$. The Space Efficient algorithm stores at most p *VCdim*(C) examples when the target concept is in $\mathrm{DIFF}^{\leq p}(C)$.

Proof. Both master algorithms apply the Closure algorithm to a stream of positive examples at each level. The master algorithms make a mistake only when one of the Closure algorithms make an incorrect prediction, triggering an update to (at least) that closure. The Closure algorithm makes at most $\mathbf{M}_{\mathrm{CLOS}}(C)$ mistakes on any sequence of positive examples (contained in a concept of C), and, by Lemma 2, at most p levels are initiated when the target concept is in $\mathrm{DIFF}^{\leq p}(C)$. Therefore our master algorithms make at most $p\mathbf{M}_{\mathrm{CLOS}}(C)$ mistakes.

For the second part, observe that the Space Efficient algorithm stores a minimal spanning set for each level of the hypothesis. By Lemma 2, the number of levels is at most p when learning a concept in $\mathrm{DIFF}^{\leq p}(C)$. Since, by Corollary 1, each minimal spanning set has size at most *VCdim*(C), the second part of the theorem follows. □

5. Extensions of the inclusion/exclusion algorithms

In this section we extend our approach for learning $\mathrm{DIFF}(C)$ to two related nested difference classes defined below.

DEFINITION. Let the concept classes C_j, for $1 \leq j \leq s$, be classes over the same domain that are intersection closed. Then the class $\mathrm{DIFF}(\cup_{j=1}^{s} C_j)$ consists of all concepts of the form $c = c_1 - (c_2 - (c_3 - \cdots - (c_{p-1} - c_p) \cdots))$ where each c_i is in one of the s classes C_j.

The learning algorithms for $\mathrm{DIFF}(\cup_{j=1}^{s} C_j)$ will make use of the closure algorithms for the C_j's. However, a difficulty arises from the fact that the learning algorithms need to determine which of the s Closure algorithms to apply at each level.

The second nested class we consider is defined in terms of an intersection-closed class and a class that is not necessarily intersection-closed.

DEFINITION. Let C be an intersection-closed class and let B be any learnable class over the same domain. Then the class $\mathrm{DIFF}(C, B)$ consists of all concepts of the form $c_1 - (c_2 - (c_3 - \cdots - (c_{p-1} - b) \cdots))$, where the c_i are concepts in C and b is a concept of B.

Again we use superscripts after DIFF to denote depth restrictions on the concepts. For example, the class $\mathrm{DIFF}^{\leq p}(C, B)$ consists of all concepts of $\mathrm{DIFF}(C, B)$ of depth between 1 and p.

The learning algorithms for DIFF(C, B) will use the Closure algorithm for C and a given learning algorithm $\mathcal{B}$ for B as subroutines. We will present several learning algorithms for DIFF(C, B) that make different assumptions about $\mathcal{B}$. These assumptions include: $\mathcal{B}$ has a good instantaneous permutation mistake bound; the worst case mistake bound of $\mathcal{B}$ is bounded; $\mathcal{B}$ always produces a consistent hypothesis. Note that for various models of learning, a concept class is polynomially learnable if and only if there exists a polynomial time algorithm for finding a consistent hypothesis for a given sample (Pitt and Valiant, 1988; Haussler et al. 1990; Board and Pitt, 1990). Thus, if the Closure algorithm can be implemented in polynomial time, then, for these models, DIFF(C, B) is polynomially learnable if and only if the class B is polynomially learnable.

The ideas presented in the next two subsections can be used to solve other problems as well. For example, by combining the analysis of the DIFF($\cup_{j=1}^{s} C_j$) case with that for DIFF(C, B), we could design algorithms for learning concepts of the form $c = c_1 - (c_2 - (c_3 - \cdots - (c_{p-1} - b) \cdots))$ where each c_i, $1 \le i \le p - 1$, is in some intersection-closed C_j, $1 \le j \le s$, and b is in some concept class B for which there exist a learning algorithm $\mathcal{B}$ fulfilling the appropriate assumptions.

5.1. Learning algorithms for DIFF($\cup_{j=1}^{s} C_j$)

As mentioned above, when learning DIFF($\cup_{j=1}^{s} C_j$) the master algorithms need to decide which Closure algorithm to apply at each level. In the case where each class C_j contains the entire domain, which we call the *universal concept*, we can bypass this dilemma by applying all s closure algorithms at each level. In the resulting hypothesis, $h = h_1 - (h_2 - (h_3 - \cdots -(h_{p-1} - h_p) \cdots))$, each h_i is set to the intersection of all s closures at that level. Instead of learning DIFF($\cup_{j=1}^{s} C_j$), we actually learn a related class, DIFF($\wedge_{j=1}^{s} C_j$).

We first define DIFF($\wedge_{j=1}^{s} C_j$) and then develop the relationship between DIFF($\wedge_{j=1}^{s} C_j$) and DIFF($\cup_{j=1}^{s} C_j$). The resulting algorithm for DIFF($\cup_{j=1}^{s} C_j$) uses hypotheses from the class DIFF($\wedge_{j=1}^{s} C_j$) and assumes that each C_j contains the universal concept.

DEFINITION. Let C_j, for $1 \le j \le s$, be concept classes over the same domain. Then $\wedge_{j=1}^{s} C_j$ denotes the concept class $\{\cap_{j=1}^{s} c_j : c_j \in C_j\}$.

The closure with respect to the class C_j (for $1 \le j \le s$) is denoted by $\mathrm{CLOS}^{(j)}$, and the closure with respect to $\wedge_{j=1}^{s} C_j$ by $\mathrm{CLOS}^{(\wedge)}$.

LEMMA 6. If each class C_j, for $1 \le j \le s$, is intersection closed, then $\wedge_{j=1}^{s} C_j$ is also intersection closed, and for every subset S of any concept in $\wedge_{j=1}^{s} C_j$, we have $\mathrm{CLOS}^{(\wedge)}(S) = \cap_{j=1}^{s} \mathrm{CLOS}^{(j)}(S)$.

Proof. Let S be any subset of a concept in $\wedge_{j=1}^{s} C_j$, so at least one concept in each C_j contains S. Therefore each $\mathrm{CLOS}^{(j)}(S)$ is defined and the concept $\cap_{j=1}^{s} \mathrm{CLOS}^{(j)}(S)$ is in $\wedge_{j=1}^{s} C_j$ and contains S. Now we need only show that any concept in $\wedge_{j=1}^{s} C_j$ which contains S also contains $\cap_{j=1}^{s} \mathrm{CLOS}^{(j)}(S)$.

Let $c_1 \cap c_2 \cap \cdots \cap c_s$, where each $c_j \in C_j$, be any concept in $\wedge_{j=1}^{s} C_j$ containing S. Therefore each c_j contains S and, by the definition of closure, each $c_j \cap \mathrm{CLOS}^{(j)}(S)$. This implies that $c_1 \cap c_2 \cap \cdots \cap c_3$ contains $\cap_{j=1}^{s} \mathrm{CLOS}^{(j)}(S)$, completing the proof. □

The above lemma shows that if membership in $\mathrm{CLOS}^{(j)}(S)$ can be decided efficiently for each $1 \leq j \leq s$, then the same can be done for $\mathrm{CLOS}^{(\wedge)}(S)$. Thus, given efficient algorithms for deciding membership in $\mathrm{CLOS}^{(j)}(S)$, the Total Recall algorithm for learning $\mathrm{DIFF}(\wedge_{j=1}^{s} C_j)$ can be implemented efficiently.

LEMMA 7. Let S be any set of instances contained in a concept of $\wedge_{j=1}^{s} C_j$.

1. Let N_j, for $1 \leq j \leq s$, be a minimal spanning set of S with respect to C_j. Then $\mathrm{CLOS}^{(\wedge)}(\cup_{j=1}^{s} N_j) = \mathrm{CLOS}^{(\wedge)}(S)$, so that $\cup_{j=1}^{s} N_j$ is a (not necessarily minimal) spanning set of S with respect to $\wedge_{j=1}^{s} C_j$.
2. Let N be any minimal spanning set of S with respect to $\wedge_{j=1}^{s} C_j$. There are minimal spanning sets N_j with respect to C_j, such that $N = \cup_{j=1}^{s} N_j$.
3. Let N (respectively N_j) be the set of corners of S with respect to the closure algorithm for $\wedge_{j=1}^{s} C_j$ (respectively the closure algorithm for C_j). Then $N = \cup_{j=1}^{s} N_j$.

Proof of Part 1. By Lemma 6, $\mathrm{CLOS}^{(\wedge)}(\cup_{j=1}^{s} N_j) = \cap_{k=1}^{s} \mathrm{CLOS}^{(k)}(\cup_{j=1}^{s} N_j)$. Since $N_j \subseteq \cup_{j=1}^{s} N_j \subseteq S$ and each N_j is a spanning set of S with respect to $\mathrm{CLOS}^{(j)}$, we have $\mathrm{CLOS}^{(j)}(N_j) = \mathrm{CLOS}^{(j)}(\cup_{j=1}^{s} N_j) = \mathit{CLOS}^{(j)}(S)$. Therefore, $\cap_{k=1}^{s} \mathrm{CLOS}^{(k)}(\cup_{j=1}^{s} N_j) = \cap_{j=1}^{s} \mathrm{CLOS}^{(j)}(S) = \mathrm{CLOS}^{(\wedge)}(S)$.

Proof of Part 2. Since N is a spanning set of S with respect to $\wedge_{j=1}^{s} C_j$, $S \subseteq \mathrm{CLOS}^{(\wedge)}(N)$. By Lemma 6, $\mathrm{CLOS}^{(\wedge)}(N) = \cap_{j=1}^{s} \mathrm{CLOS}^{(j)}(N)$, so $\mathrm{CLOS}^{(j)}(N) \supseteq S$ for each j. Thus for each j, N contains a minimal spanning set N_j of S with respect to C_j. By Part 1, $\cup_{j=1}^{s} N_j$ contains a spanning set of S with respect to $\wedge_{j=1}^{s} C_j$. Because N is minimal, $N - \cup_{j=1}^{s} N_j$ must be empty, and $N = \cup_{j=1}^{s} N_j$.

Proof of Part 3. Observe that x is a corner of S with respect to some CLOS if and only if $\mathrm{CLOS}(S - \{x\}) \neq \mathrm{CLOS}(S)$. By Lemma 6, $\mathrm{CLOS}^{(\wedge)}(S - \{x\}) \neq \mathrm{CLOS}^{(\wedge)}(S)$ if and only if for some j, $\mathrm{CLOS}^{(j)}(S - \{x\}) \neq \mathrm{CLOS}^{(j)}(S)$. Thus x is a corner of S with respect to $\mathrm{CLOS}^{(\wedge)}$ if and only if x is a corner with respect to some $\mathrm{CLOS}^{(j)}$. □

By Part 1 of the preceding Lemma 7, if small spanning sets with respect to each C_j can be found efficiently, then small spanning sets can also be found with respect to $\wedge_{j=1}^{s} C_j$. This observation can be used in the implementation of the Space Efficient algorithm for $\mathrm{DIFF}(\wedge_{j=1}^{s} C_j)$. We next use Part 2 of Lemma 7 to bound $\mathit{VCdim}(\cup_{j=1}^{s} C_j)$.

LEMMA 8. Assuming each C_j contains the universal concept then:

1. $\cup_{j=1}^{s} C_j \subseteq \wedge_{j=1}^{s} C_j$ and $\mathrm{DIFF}^{\leq p}(\cup_{j=1}^{s} C_j) \subseteq \mathrm{DIFF}^{\leq p}(\wedge_{j=1}^{s} C_j)$.
2. $\max_{j=1}^{s}(\{\mathit{VCdim}(C_j)\}) \leq \mathit{VCdim}(\cup_{j=1}^{s} C_j) \leq \mathit{VCdim}(\wedge_{j=1}^{s} C_j) \leq \Sigma_{j=1}^{s} \mathit{VCdim}(C_j) \leq s \max_{j=1}^{s}(\{\mathit{VCdim}(C_j)\})$.[7]
3. $\mathit{VCdim}(\mathrm{DIFF}^{\leq p}(\cup_{j=1}^{s} C_j) \leq \mathit{VCdim}(\mathrm{DIFF}^{\leq p}(\wedge_{j=1}^{s} C_j) \leq p\, \Sigma_{j=1}^{s} \mathit{VCdim}(C_j) \leq ps \max_{j=1}^{s}(\{\mathit{VCdim}(C_j)\})$.

Proof. The first part follows from the definitions and the fact that each C_j contains the universal concept. Part 3 follows from Part 1 and Lemma 3. We now show the inequalities of Part 2. The first and the last inequalities are trivial and the second follows from Part 1.

The third inequality of Part 2 is the most interesting one. Let S be a maximum cardinality set that is shattered by $\wedge_{j=1}^{s} C_j$. The set S is its own minimum spanning set with respect to $\wedge_{j=1}^{s} C_j$ and, by Part 2 of Lemma 7, is at most as large as the sum of the sizes of minimal spanning sets of S with respect to each C_j. By Theorem 1, the sizes of the minimal spanning sets of S with respect to each C_j are bounded by $VCdim(C_j)$ and the inequality follows. □

By Part 1 of Lemma 8, we can learn the concept class $\mathrm{DIFF}(\cup_{j=1}^{s} C_j)$ by using the Closure algorithm for $\wedge_{j=1}^{s} C_j$ in our master algorithms. If the target concept from $\mathrm{DIFF}(\cup_{j=1}^{s} C_j)$ has depth p then the depth of the master algorithm's hypothesis from $\mathrm{DIFF}(\wedge_{j=1}^{s} C_j)$ will be at most p.

Theorem 6. Let each C_j, for $1 \le j \le s$, be an intersection-closed concept class containing the universal concept. The Total Recall algorithm, when using the Closure algorithm with respect to $\wedge_{j=1}^{s} C_j$, has the following performance bounds when learning a target concept in $\mathrm{DIFF}(\cup_{j=1}^{s} C_j)$:

$$t.(\mathrm{DIFF}^{\le p}(\bigcup_{j=1}^{s} C_j),\ \epsilon,\ \delta) \le \max\left(\frac{4}{\epsilon}\log_2\frac{4}{\delta},\ \frac{8p\ \Sigma_{j=1}^{s}\ VCdim(C_j)}{\epsilon}\log_2\frac{13}{\epsilon}\right).$$

$$\widehat{\mathbf{MI}}.(\mathrm{DIFF}^{\le p}(\bigcup_{j=1}^{s} C_j)) \le \widehat{\widehat{\mathbf{MI}}}.(\mathrm{DIFF}^{\le p}(\bigcup_{j=1}^{s} C_j)) \le \frac{p\ \Sigma_{j=1}^{s}\ VCdim(C_j)}{t}.$$

$$\widehat{\mathbf{MT}}.(\mathrm{DIFF}^{\le p}(\bigcup_{j=1}^{s} C_j)) \le \widehat{\widehat{\mathbf{MT}}}.(\mathrm{DIFF}^{\le p}(\bigcup_{j=1}^{s} C_j)) \le pH_t \sum_{j=1}^{s} VCdim(C_j).$$

Proof. By Lemma 6, $\wedge_{j=1}^{s} C_j$ is intersection closed and by Lemma 8, it has VC dimension at most $\Sigma_{j=1}^{s}\ VCdim(C_j)$. Now the theorem follows from the corresponding theorems in Section 4. □

Note that assuming each concept class C_j contains the universal concept does not significantly affect our performance bounds since $VCdim(C \cup \{\text{universal concept}\}) \le VCdim(C) + 1$.

Theorem 7. Let each C_j, for $1 \le j \le s$, be an intersection-closed concept class containing the universal concept. When given a sequence of examples labeled consistently with a concept from $\mathrm{DIFF}(\cup_{j=1}^{s} C_j)$, for both the Total Recall and the Space Efficient algorithms

$$\mathbf{M}.(\mathrm{DIFF}^{\le p}(\bigcup_{j=1}^{s} C_j)) \le p \sum_{j=1}^{s} \mathbf{M}_{\mathrm{CLOS}(j)}(C_j)$$

when the Closure algorithm with respect to $\wedge_{j=1}^{s} C_j$ is used. The Space Efficient algorithm stores at most $p \; \Sigma_{j=1}^{s} \; VCdim(C_j)$ examples when learning $\text{DIFF}^{\leq p}(\cup_{j=1}^{s} C_j)$.

Proof. By Lemma 8, $\cup_{j=1}^{s} C_j \subseteq \wedge_{j=1}^{s} C_j$ and by Lemma 6, $\wedge_{j=1}^{s} C_j$ is intersection closed. Thus (by Theorem 5) when the master algorithms use the Closure algorithm for $\wedge_{j=1}^{s} C_j$, $\mathbf{M}.(\text{DIFF}^{\leq p}(\cup_{j=1}^{s} C_j)) \leq p\mathbf{M}_{\text{CLOS}^{(\wedge)}}$, and the Space Efficient algorithm stores at most $pVCdim(\wedge_{j=1}^{s} C_j)$ many examples. By Lemma 8, $VCdim(\wedge_{j=1}^{s} C_j) \leq \Sigma_{j=1}^{s} \; VCdim(C_j)$.

By Lemma 6, the closure with respect to $\wedge_{j=1}^{s} C_j$ is a function of the closures with respect to the C_j's. Thus whenever the Closure algorithm for $\wedge_{j=1}^{s} C_j$ makes a mistake, at least one of the Closure algorithms for the C_j's makes a mistake, and $\mathbf{M}_{\text{CLOS}^{(\wedge)}}(\wedge_{j=1}^{s} C_j) \leq \Sigma_{j=1}^{s} \mathbf{M}_{\text{CLOS}^{(j)}}(C_j)$. □

5.2. *Learning* DIFF(C, B)

Since B is not necessarily intersection closed, we cannot rely on the closure algorithm for it. Therefore, we assume that a learning algorithm $\mathcal{B}$ for B is given to our master algorithms. Intuitively, master algorithms for DIFF(C, B) need to determine for each level whether to apply the Closure algorithm for C to produce the next portion of the hypothesis or whether the current level is the innermost one and the algorithm $\mathcal{B}$ should be applied. Thus we need criteria that tell the master algorithm when it has arrived at the innermost level. Various assumptions about $\mathcal{B}$ lead to such criteria.

Our first master algorithm for learning DIFF(C, B) is a variant of the Total Recall algorithm. It assumes that $\mathcal{B}$ produces a consistent hypothesis for a given sample whenever there is a concept $b \in B$ that is consistent with the sample. Given any set of examples labeled consistently with a target concept in DIFF(C, B), this master algorithm produces a consistent hypothesis $h_1 - (h_2 - (h_3 - \cdots - (h_{p-1} - h_p) \cdots))$ in DIFF(C, B) where the depth p of the hypothesis is at most the depth of the target.

The master works roughly as follows. Let EX_i be all examples remaining at the current level i. The master algorithm first attempts to find a consistent hypothesis in B for EX_i by feeding the examples to algorithm $\mathcal{B}$. If a consistent hypothesis is produced, then the master algorithm concludes that it has arrived at the innermost level and h_i is set to the concept output by $\mathcal{B}$. Otherwise, the master algorithm knows that the current level is not the innermost level of the target concept, and filters the examples with another concept from C. This is done by applying the Closure algorithm of C to the current example set, EX_i, and setting h_i to the concept of C output by the Closure algorithm. At this point all examples outside of h_i are classified correctly by the current hypothesis and we can restrict ourselves to the example set $\text{EX}_{i+1} = \text{FLIP}(h_i \cap \text{EX}_i)$ for the next level. See Figure 9 for a detailed description of the algorithm.

Our first step towards determining the performance of this algorithm is to bound the VC dimension of DIFF(C, B).

LEMMA 9. If C is intersection closed and $p \geq 1$, then

$$VCdim(\text{DIFF}^{=p}(C, B)) \leq VCdim(\text{DIFF}^{\leq p}(C, B)) \leq (p - 1) \; VCdim(C) + VCdim(B).$$

```
Algorithm Total Recall for DIFF$(C, B)$
Inputs: Examples of concept.
/* Computes hypothesis $h$ and depth $p$. */
/* Uses algorithm $\mathcal{B}$ for learning $B$ */
$\mathrm{EX}_1$ := all examples; $i := 1$
while $\mathcal{B}(\mathrm{EX}_i)$ fails to produce a consistent hypothesis do
    $h_i := \mathrm{CLOS}(\mathrm{POS}(\mathrm{EX}_i))$
    $\mathrm{EX}_{i+1} := \mathrm{FLIP}(h_i \cap \mathrm{EX}_i)$
    /* "FLIP" flips the labels of the examples */
    $i := i + 1$
end while;
$p := i$; $h_p := \mathcal{B}(\mathrm{EX}_i)$
$h_{p+1} := \emptyset$
$h := h_1 - (h_2 - \cdots - (h_{p-1} - h_p)_{\ldots})$
```

Figure 9. Total Recall algorithm for DIFF(C, B) when depth is not known.

Proof. Use the same arugment as the one in the proof of Lemma 3. □

Again it is easy to find concept classes C and B that show that the inequalities are tight. Applying this lemma together with Theorem 2, we can generalize Theorem 3 to: after

$$\max\left(\frac{4}{\epsilon}\log_2\frac{4}{\delta},\ \frac{8((p-1)VCdim(C)+VCdim(B))}{\epsilon}\log_2\frac{13}{\epsilon}\right)$$

examples drawn independently at random, the Total Recall algorithm in Figure 9 produces a hypothesis whose error is at most ϵ with probability at least $1 - \delta$ where p is the depth of the target concept in DIFF(C, B).

THEOREM 8. For the Total Recall algorithm in Figure 9,

$$t.(\mathrm{DIFF}^{\leq p}(C, B), \epsilon, \delta) \leq \max\left(\frac{4}{\epsilon}\log_2\frac{4}{\delta},\frac{8((p-1)VCdim(C)+VCdim(B))}{\epsilon}\log_2\frac{13}{\epsilon}\right).$$

Proof. Apply Theorem 2 using the bound in Lemma 9 on the VC dimension of $\mathrm{DIFF}^{\leq p}(C, B)$. □

Theorem 4 gives a bound on the probability that the Total Recall algorithm makes a mistake at trial t. The generalization of this theorem to the Total Recall algorithm in Figure 9 is not so straightforward. Our goal is to obtain a good estimate of the cumulative sample mistake bound as a function of the depth p of the target concept, the VC dimension of C, and the instantaneous permutation mistake bound for B. We need the following assumptions on the instantaneous permutation mistake bound for B:

DEFINITION. An algorithm $\mathcal{B}$ for concept class B is *suitable* if in addition to producing a consistent hypothesis from B (if such a hypothesis exists) for the given set of examples the following condition holds:

- It has an instantaneous permutation mistake bound, $\widehat{\widehat{\mathbf{MI}}}_{\mathcal{B}}(B)(t)$, such that $t\widehat{\widehat{\mathbf{MI}}}_{\mathcal{B}}(B)(t)$ is non decreasing.[8]

THEOREM 9. If $\mathcal{B}$ is a suitable algorithm for concept class B, then the Total Recall algorithm in Figure 9 has the following cumulative sample and cumulative permutation mistake bounds:

$$\begin{aligned}\widehat{\mathbf{MT}}.(\mathrm{DIFF}^{\leq p}(C, B))(t) &\leq \widehat{\widehat{\mathbf{MT}}}.(\mathrm{DIFF}^{\leq p}(C, B))(t)\\ &\leq (p - 1)\ VCdim(C)H_t + p - 1 + \sum_{i=1}^{t} \widehat{\widehat{\mathbf{MI}}}_{\mathcal{B}}(B)(i).\end{aligned}$$

Proof. The first inequality follows from Lemma 1. Thus it suffices to bound the cumulative permutation mistake bound.

Let S be any multi-set of examples labeled consistently with some concept c in $\mathrm{DIFF}^{\leq p}(C, B)$. Thus the depth of c is at most p. We will bound the expected total number of mistakes made by the algorithm over all permutations of S.

Note that whenever the Total Recall algorithm makes a mistake on some example, its hypothesis, h, is updated to some $h' \neq h$.[9] We divide the mistakes made by the Total Recall algorithm into three categories based on how the hypothesis gets updated and bound the expected total number of mistakes in each category separately.

If the depth of h' is greater than the depth of h then the mistake is a *level mistake*. If h and h' have the same depth k, but some $h_i \neq h_i'$ for $1 \leq i < k$, then the mistake is a *C-mistake*. Otherwise, h and h' differ in only the innermost concept, h_k, and the mistake is a *B-mistake*.

The depth of the Total Recall algorithm's hypothesis is initially 1. By induction it is easy to show that the depth of the hypothesis is non-decreasing and bounded by the depth of the target concept. Therefore, at most $p - 1$ level mistakes will be made on any sequence of examples consistent with a concept in $\mathrm{DIFF}^{\leq p}(C, B)$.

The Total Recall algorithm makes a C-mistake on the t-th example only when the t-th example is a corner of some EX_i $(1 \leq i \leq k)$, where k is the depth of the hypothesis. Since $k \leq p$, the chance that the t-th example is one of these corners is at most $(p - 1)\ VCdim(C)/t$. The argument of Theorem 4 shows that, over all permutations of S, the average total number of C-mistakes in the first t trials is bounded by $(p - 1)$ $VCdim(C)H_t$.

We now bound the number of permutations of S where the Total Recall algorithm makes a B-mistake at trial t. Let EX_k be the set of examples fed to $\mathcal{B}$ when building h', the hypothesis produced after trial t. If a B-mistake is made at trial t of a permutation of S, then both:

- An example of EX_k appears last in the permutation, and
- $\mathcal{B}$ made a mistake predicting the label of the last instance in EX_k.

The fraction of the permutations of S where the last example is in EX_k is EX_k/t. The fraction of the permutations of S where algorithm $\mathcal{B}$ makes a mistake predicting the label of the last instance in EX_k is at most $\hat{\hat{\mathbf{MI}}}_{\mathcal{B}}(B)(|EX_k|)$. Note that the relative ordering of the examples in EX_k is independent of whether one of them appears last in the permutation of S. Therefore, the fraction of the permutations of S where the Total Recall algorithm makes a B mistake at trial t is at most:

$$\frac{|EX_k|\hat{\hat{\mathbf{MI}}}_{\mathcal{B}}(B)(|EX_k|)}{t}.$$

Since $\mathcal{B}$ is suitable,

$$\frac{|EX_k|\hat{\hat{\mathbf{MI}}}_{\mathcal{B}}(B)(|EX_k|)}{t} \le \frac{t\hat{\hat{\mathbf{MI}}}_{\mathcal{B}}(B)(t)}{t} = \hat{\hat{\mathbf{MI}}}_{\mathcal{B}}(B)(t).$$

Summing over t, we see that the average, over all permutations of S, of the total number of B-mistakes in the first t trials is at most $\Sigma_{i=1}^{t}\hat{\hat{\mathbf{MI}}}_{\mathcal{B}}(B)(i)$.

Combining the bounds from the three cases gives us:

$$\mathrm{MT}.(\mathrm{DIFF}^{\le p}(C, B))(t) \le p - 1 + (p - 1)\, VCdim(C)H_t + \sum_{i=1}^{t} \hat{\hat{\mathbf{MI}}}_{\mathcal{B}}(B)(i). \qquad \square$$

Two related factors make it difficult, if not impossible, to obtain good instantaneous mistake bounds on the Total Recall algorithm's performance when learning *DIFF*(C, B). First, the hypothesis can suddenly change dramatically whenever its depth is increased (in addition to adding a level, an arbitrary concept in B is replaced by CLOS($\emptyset$)). In addition, an instantaneous mistake bound holds only if the domain is labeled consistently with a concept in the class being learned. Here, however, even though the particular examples given to algorithm $\mathcal{B}$ may be labeled consistently with a concept in B, the labeling of the domain from which these examples are drawn may be some nested difference.

To illustrate this point, we exhibit a concept class B' whose instantaneous sample mistake bound does not apply to other concept classes *even when* all of the examples seen are labeled consistently with a concept in B'. Let the domain contain four points and the concept class B' contain all pairs of instances. The trivial algorithm for this class keeps track of the (at most two) "+" examples previously seen, and predicts "+" on these instances and "−" on all others. The chance that this algorithm makes a mistake on the third trial is at most 8/27, achieved when each of the two "+" instances has probability 1/3. Consider now the situation when the four points are labeled "+, +, +, −" and the three plus points are given probability 1/5 each. The probability that both the first three trials contain at most two distinct "+" instances (and thus are consistent with a concept in B') *and* the algorithm makes a mistake on the third trial is $3(\frac{1}{5})\,((\frac{4}{5})^2 - 2(\frac{1}{5})^2) > \frac{1}{3}$.

In the remainder of this section we discuss space efficient algorithms for learning DIFF(C, B). These algorithms store only a small number of examples and are usually not consistent with previous examples seen. In the simplest case the depth p of the target is known. For levels 1 through $p - 1$ we simply run the space efficient algorithm of Figure 5 and at depth p we use the given space efficient algorithm $\mathfrak{B}$ for learning the class B. This master algorithm is called $\mathfrak{D}_p$ and is shown in Figure 10.

It is easy to see that Algorithm $\mathfrak{D}_p$ makes at most $(p - 1)\mathbf{M}_{\text{CLOS}}(C) + \mathbf{M}_{\mathfrak{B}}(B)$ mistakes on any sequence of examples consistent with a concept in DIFF(C, B) of depth p, giving us:

THEOREM 10. $\mathbf{M}_{\mathfrak{D}_p}(\text{DIFF}^{=p}(C, B)) \leq (p - 1)\mathbf{M}_{\text{CLOS}}(C) + \mathbf{M}_{\mathfrak{B}}(B)$.

If C contains the universal concept, then any target concept described by a nested difference can be syntactically changed by inserting two copies of the universal concept into adjacent levels without changing the target concept. This leads to the following corollary of the above theorem:

COROLLARY 2. If C contains the universal concept, then for any depth $\hat{p} \leq p$ which has the same parity as p, $M_{\mathfrak{D}_p}(\text{DIFF}^{=\hat{p}}(C, B)) \leq (p - 1)\mathbf{M}_{\text{CLOS}}(C) + \mathbf{M}_{\mathfrak{B}}(B)$.

```
Algorithm family $\mathcal{D}_p$, the Space Efficient algorithm for DIFF$(C,B)$
Parameters: The family has parameter $p$, the known depth of the target concept.
Inputs: Current hypothesis $h$ and instance $x$.
Uses: Closure algorithm for C, algorithm $\mathcal{B}$ for $B$ with initial hypothesis $\mathcal{B}_{\text{init}}$
/* On-line algorithm */
If first call then
        $h_p := \mathcal{B}_{\text{init}}$; $h_{p+1} := \emptyset$
        for $k := 1$ to $p - 1$
            initialize $S_k := \emptyset$; $h_k :=$ CLOS$(\emptyset)$
Call Predict$(h, x)$
If mistake made then
        $l := 0$
        repeat $l := l + 1$ until $x \notin h_l$
        if $l < p$ then
                /* update one of the outer shells */
                $h_l :=$ CLOS$(S_l \cup \{x\})$
                $S_l :=$ minimal spanning set$(S_l \cup \{x\})$
        else $l = p$
                /* update $h_p \in B$ */
                $h_p := \mathcal{B}(h_p, x)$
        end if /* $l < p$ */
end if /* mistake made */
```

Figure 10. Space Efficient algorithm for DIFF(C, B) when depth of target is known.

In the case where the depth of the target is not known, we can successively try various values for the depth p of the target. However for this to work we need a criterion for deciding whether the current choice of p is wrong, and which value for p to try next. We assume that the worst case total number of mistakes made by $\mathcal{B}$ on any sequence of examples labeled consistent with a concept of B, $\mathbf{M}_{\mathcal{B}}(B)$, is bounded and that the master algorithm knows $\mathbf{M}_{\mathcal{B}}(B)$ (or an upper bound thereof). Whenever the number of mistakes made on the innermost level exceeds $\mathbf{M}_{B}(B)$ (or its upper bound) then the master algorithm tries the next choice of p. The algorithm of Figure 11 implements this approach by iteratively trying $p = 1, 2, \ldots$. It is easy to see that it achieves the following worst case mistake bound.

THEOREM 11. The Space Efficient algorithm of Figure 11 which iteratively guesses depth 1, 2, ... for the target concept has a worst case mistake bound, $\mathbf{M}.(\mathrm{DIFF}^{=p}(C, B) = (p - 1)(\mathbf{M}_{\mathrm{CLOS}}(C) + 1) + p\mathbf{M}_{\mathcal{B}}(B)$.

```
Algorithm Space Efficient for DIFF($(C, B)$) when depth of target unknown.
Inputs: Current hypothesis of depth $p$, mistake count $m$, and instance $x$.
Uses: Closure alg. for C, algorithm $\mathcal{B}$ for $B$ with initial hypothesis $\mathcal{B}_{\mathrm{init}}$
/* On-line algorithm */
If first call then
        initialize $p := 1$; $S_1 := \emptyset$; $h_1 := \mathcal{B}_{\mathrm{init}}$; $h_2 := \emptyset$; $m := 0$
Call Predict($(h, x)$)
If mistake made then
        $l := 0$
        repeat $l := l + 1$ until $x \notin h_l$
        if $l < p$ then
             /* update one of the outer shells */
             $h_l :=$ CLOS($(S_l \cup \{x\})$)
             $S_l :=$ minimal spanning set($(S_l \cup \{x\})$)
        else $l \geq p$
             /* update $h_p$ */
             if $m + 1 > \mathbf{M}_{\mathcal{B}}(B)$ then
                          /* initialize new shell */
                          $S_p := \emptyset$; $h_p :=$ CLOS($(\emptyset)$)
                          $h_{p+1} := \mathcal{B}_{\mathrm{init}}$; $h_{p+2} := \emptyset$
                          $m := 0$; $p := p + 1$
             else $m < \mathbf{M}_{\mathcal{B}}(B)$
                          $h_p := \mathcal{B}(h_p, x)$
                          $m := m + 1$
             end if /* $m + 1 > \mathbf{M}_{\mathcal{B}}(B)$ */
        end if /* $l < p$ */
end if /* mistake made */
```

Figure 11. Space Efficient algorithm for DIFF(C, B) when depth of target is unknown.

Proof. For each level between 1 and $p - 1$, the algorithm makes at most $\mathbf{M}_{\mathrm{CLOS}}(C) + \mathbf{M}_{\mathcal{B}}(B) + 1$ mistakes. On the innermost level, the algorithm makes at most $\mathbf{M}_{\mathcal{B}}(B)$ mistakes. □

If C contains the universal concept then we can do better by using a doubling trick and by applying Corollary 2: use the sequence $1, 2, 3, 4, 7, 8, \ldots, 2^k - 1, 2^k, \ldots$ of guesses for the depth, and each time the total number of mistakes made on the innermost level p exceeds $\mathbf{M}_{\mathcal{B}}(B)$ go on to the next value $\hat{p}$ in the sequence. (This is done by setting $S_k := \emptyset$; $h_k := \mathrm{CLOS}(\emptyset)$, for $p \leq k < \hat{p}$; and $h_{\hat{p}} := \mathcal{B}_{\mathrm{init}}$ when a new shell is initialized.) The reason for trying a pair of consecutive guesses at each power of two is that the guessed depth must have the proper parity with respect to the depth of the target concept. Once the guessed depth reaches a value that has the same parity as, and is at least as large as, the depth of the target then the guessed depth will never be changed. The construction of the above sequence ensures that, if the depth of the target is p, then the guessed depth will never be larger than $2p - 2$. Thus by Corollary 2 the doubling trick improves the bound to

THEOREM 12. The Space Efficient algorithm of Figure 11 for learning $\mathrm{DIFF}(C, B)$, when modified to use the sequence of guesses $1, 2, 3, 4, 7, 8, \ldots, 2^k - 1, 2^k, \ldots$ for the depth of the target concept, has a worst case mistake bound, $\mathbf{M}(\mathrm{DIFF}^{\leq p}(C, B))$, which is $O(p\mathbf{M}_{\mathrm{CLOS}}(C) + \log p \cdot \mathbf{M}_{\mathcal{B}}(B))$.

Proof. See preceding paragraph. □

Although the doubling trick reduces the mistake bound, it may use a hypothesis almost twice as deep as that of the algorithm in Figure 11.

The space efficient algorithms presented so far in this section required that either the depth of the target (Theorem 10) or an upper bound on $\mathbf{M}_{\mathcal{B}}(B)$ (Theorems 11 and 12) be given to the master algorithm. Surprisingly, there is a reasonably space efficient master algorithm that assumes neither knowledge of the depth of the target nor of $\mathbf{M}_{\mathcal{B}}(B)$. The worst case mistake bound of this new algorithm is $O(p\mathbf{M}_{\mathrm{CLOS}}(C) + \mathbf{M}_{\mathcal{B}}(B))$, only a constant times larger than for the case when the depth is known (Theorem 10).

The new master algorithm runs, in parallel, multiple copies of the algorithm $\mathcal{D}_p$ (described in Figure 10) with different depth parameters. Whenever it is asked to predict on a new instance, the master first obtains the predictions made by each copy of $\mathcal{D}_p$ and then uses a weighted majority voting scheme to decide on its final prediction. The weights of the various copies are updated so that the influence of those copies making mistakes is decreased. When the final prediction is incorrect, new copies of $\mathcal{D}_p$ may be initiated. The resulting master algorithm for learning $\mathrm{DIFF}(C, B)$ is still space efficient in the sense that at most $O(\log(p\mathbf{M}_{\mathrm{CLOS}}(C) + \mathbf{M}_{\mathcal{B}}(B)))$ copies of $\mathcal{D}_p$ are run in parallel (i.e., logarithmic in the worst case mistake bound of the master). However, some copies may have depth guesses up to a constant times the total number of mistakes made.

The new master algorithm is an application of one of the Weighted Majority algorithms (called $\mathcal{WMG}$) introduced in (Littlestone and Warmuth, 1989). In the appendix, we give a description of $\mathcal{WMG}$ and prove the following (see Theorem 14) theorem.

With appropriate parameters, the algorithm $\mathcal{WMI}$ has the following properties:

1. The worst case mistake bound, $\mathbf{M}_{\mathcal{WMI}}(\mathrm{DIFF}^{=p}(C, B))$, is $O(p\mathbf{M}_{\mathrm{CLOS}}(C) + \mathbf{M}_{\mathcal{B}}(B))$.
2. At most $O(\log[\mathbf{M}_{\mathcal{WMI}}(\mathrm{DIFF}^{=p}(C, B))])$ copies of D_p are initiated.

6. Conclusions and open problems

In this section we discuss the optimality of our master algorithms for learning the concept classes DIFF(C), DIFF($\cup_{j=1}^{s} C_j$), and DIFF(C, B) with respect to the performance criteria introduced in Section 3 and in particular, discuss our results on space efficient master algorithms. A number of open problems are given.

All our performance bounds are a function of the VC dimension of the target class (or the hypothesis class if it is different from the target class). Thus the quality of our performance bounds depend on whether our estimates of the VC dimension of the target class are exact. The upper bounds for $VCdim$(DIFF(C)) and $VCdim$(DIFF(C, B)) given in Lemmas 3 and 9, respectively, are tight in the sense that there are many concept classes C and B where the upper bounds are reached.

Recall that we learned the class $\mathrm{DIFF}^{\leq p}(\cup_{j=1}^{s} C_j)$ using hypotheses from $\mathrm{DIFF}^{\leq p}(\wedge_{j=1}^{s} C_j)$. According to the bounds of Lemma 8, if $VCdim(\mathrm{DIFF}^{\leq p}(\cup_{j=1}^{s} C_j) \geq p \max_{j=1}^{s} VCdim(C_j)$ then $VCdim(\mathrm{DIFF}^{\leq p}(\wedge_{j=1}^{s} C_j))$ is at most a factor of s larger than $VCdim(\mathrm{DIFF}^{\leq p}(\cup_{j=1}^{s} C_j))$. In (Helmbold, Sloan, and Warmuth, 1989c) an alternate method was given where the VC dimension of the hypothesis class is at most a factor of $2s$ larger than $VCdim^{\leq p}(\cup_{j=1}^{s} C_j)$.

It is an open problem to find tight bounds for $VCdim(\cup_{j=1}^{s} C_j)$. (This is discussed in more detail in (Helmbold, Sloan & Warmuth, 1989b).)

All instantaneous and cumulative sample mistake bounds for learning DIFF(C) with the Total Recall algorithm are of the form d/t and dH_t, respectively, where d is the VC dimension of the target class. In the worst case, our bounds on d are tight (see previous paragraph) and for these two performance measures there are matching lower bounds. In (Haussler et al. 1988) it is shown that for any concept class of VC dimension d and large enough t, the instantaneous sample mistake bound of any algorithm is $\Omega(d/t)$, and for any d there are intersection-closed classes C of VC dimension d (containing the universal concept) such that the cumulative sample mistake bound for any algorithm is $\Omega(d \log t)$. Therefore the instantaneous and cumulative sample mistake bounds of the Total Recall algorithm for DIFF(C) are within a constant factor of optimal.

The VC dimension is a trivial lower bound on the total number of mistakes made in the worst case (Littlestone, 1988). Thus if $\mathbf{M}_{\mathrm{CLOS}}(C) = VCdim(C)$ and $VCdim(\mathrm{DIFF}^{\leq p}(C)) = pVCdim(C)$, then, under the worst case total number of mistakes performance criterion, both the Space Efficient algorithm and the Total Recall algorithm optimally learn DIFF(C), since by Theorem 5 they make at most $p\mathbf{M}_{\mathrm{CLOS}}(C)$ many mistakes[10].

Recall that (under reasonable assumptions) the VC dimension of the hypothesis class used when learning DIFF($\cup_{j=1}^{s} C_j$) could be up to a factor of s greater then the VC dimension of the target class. Because of this gap, the performance of the Total Recall algorithm for DIFF($\bigcup_{j=1}^{s} C_j$) may be up to an $O(s)$ factor worse than the optimum performance for the instantaneous and cumulative sample mistake bound measures. For similar reasons the worst case total mistake bounds of the Total Recall and Space Efficient algorithms for DIFF($\bigcup_{j=1}^{s} C_j$) might differ by an $O(s)$ factor from optimum.

It is much harder to design master algorithms for learning the class DIFF(C, B) with good mistake bounds. In the case of the instantaneous and cumulative sample mistake bounds, the main obstacle is that the distribution with which the innermost concept of B is learned does not stay fixed. In Theorem 9 we were still able to achieve an excellent cumulative sample mistake bound provided that an algorithm for B is given with a good instantaneous sample mistake bound. It is an open problem to find master algorithms for learning DIFF$^{\leq p}(C, B)$ with $O(d/t)$ instantaneous sample mistake bounds where $d = (p - 1)$ $VCdim(C) + VCdim(B)$.[11]

We next consider learning the class DIFF(C, B) space efficiently using the worst case total number of mistakes as the performance criterion. When $\mathbf{M}_{\mathrm{CLOS}}(C) = VCdim(C)$, $\mathbf{M}_{\mathcal{B}}(B) = VCdim(B)$, and $VCdim(\mathrm{DIFF}^{=p}(C, B)) = (p - 1)\ VCdim(C) + VCdim(B)$, the version of the Space Efficient algorithm for learning DIFF(C, B) (Figure 10) which is given the depth p of the target in advance is optimal (Theorem 10). When p is not given to the master algorithms, but a worst case mistake bound for learning B is given, various strategies for trying successive guesses for the depth p lead to reasonable algorithms, but their worst case number of mistakes is suboptimal (Theorems 11 and 12).

A powerful method based on weighted majority voting that is ideally suited to dealing with cases when a parameter (here the depth p) for an algorithm is not known was given in (Littlestone and Warmuth, 1989). An application of these methods leads to a reasonably space efficient master algorithm (given in the Appendix) for learning DIFF(C, B) which does not require knowledge of p and is optimal to within a constant factor. The space requirement for this algorithm might be roughly up to a $O(\log(p\mathbf{M}_{\mathrm{CLOS}}(C) + \mathbf{M}_{\mathcal{B}}(B)))$ factor larger than the space used by the Space Efficient algorithm which is given p in advance.

The major intriguing open problem is to analyze the Space Efficient algorithms with respect to the probabilistic performance measures. Although the Space Efficient algorithm is simple, its hypothesis is neither consistent with all of the examples previously seen, nor independent of their order. Therefore additional analysis techniques are needed to determine to what extent it is necessary to remember all previous examples for good probabilistic performance.

There are other approaches to designing space efficient algorithms for which one can prove reasonable probabilistic bounds. We did not present these algorithms in the main body of the paper because they are significantly more complicated than the Space Efficient algorithms presented here. We now sketch such an algorithm for learning DIFF(C) with respect to Valiant's criterion of producing with probability at least $1 - \delta$ a hypothesis of error at most ϵ: first draw a number of examples keeping track of a minimal spanning set of the positive examples; set the first shell h_1 to the closure of the spanning set; make the number of examples large enough so that with large enough probability the error of h_1 is small enough; do the same for the second shell by drawing enough further examples and by setting h_2 to a spanning set of all the negative examples falling into h_1; continue fixing further shells until the number of examples (with the proper label) falling into the innermost shell is small enough. The algorithm is space efficient since it stores at most p spanning sets, where p is the depth of the target.

As can be seen from the above high-level description, a lot of fine tuning needs to be done to determine how long each shell should be trained. Furthermore, this type of algorithm needs to know ϵ and δ.

Weighted majority voting can also be used to design master algorithms for DIFF(C, B) with a cumulative sample mistake bound of $O(d \log t)$, where $d = (p - 1)\ VCdim(C) + VCdim(B)$. Simply modify the variant of the Total Recall algorithm of Figure 9 for the case when the depth p is given as a parameter. A corner argument shows that this modified algorithm learns DIFF$^{=p}$ with an $O(d \log t)$ cumulative sample mistake bound. When the depth parameter is not given, use a weighted majority voting scheme on the following pool of algorithms: the i-th algorithm is the above variant of the Total Recall algorithm with depth parameter i. Surprisingly, this method leads to an algorithm for learning DIFF(C, B) for the case when the depth of the target is not known whose cumulative sample mistake bound is within a constant factor of optimal.

We conclude by summarizing our results for the performance criterion introduced by Valiant (Section 3.4). This criterion gives bounds on the number of examples required to produce, with probability at least $1 - \delta$, a hypothesis of error at most ϵ. We presented variants of the Total Recall algorithm for learning each of the classes DIFF(C), DIFF($\bigcup_{j=1}^{s} C_j$), and DIFF(C, B). The bounds on the number of examples required by each of these algorithms are proven using Theorem 2 of (Blumer et al. 1989). This theorem shows that any consistent algorithm needs only $O(\frac{d}{\epsilon} \log(\frac{1}{\epsilon}) + \frac{1}{\epsilon} \log(\frac{1}{\delta}))$ examples to produce, with probability at least $1 - \delta$, a hypothesis that has error at most ϵ. Note again that the bounds on the number of examples depend linearly on d, the VC dimension. In (Haussler et al. 1988) it has been shown that for some algorithms the bounds of this theorem are tight to within a constant factor. The theorem does not exclude the possibility that particular algorithms might require significantly fewer examples to produce, with probability at least $1 - \delta$, a hypothesis with error at most ϵ. However, a lower bound theorem of (Ehrenfeucht, Haussler, Kearns & Valiant 1989) shows that the savings can be at most a factor of $\log(\frac{1}{\epsilon})$: any algorithm requires $\Omega(\frac{d}{\epsilon} + \frac{1}{\epsilon} \log(\frac{1}{\delta}))$ examples to produce, with probability at least $1 - \delta$, a hypothesis with error at most ϵ. It is an open problem whether the performances of the Closure algorithm for intersection-closed classes and the Total Recall algorithm are within a constant factor of this lower bound.

Acknowledgments

This research was supported by ONR grant N00014-86-K-0454. In addition, Robert Sloan was also supported by an NSF graduate fellowship and by ARO grant 03-86-K-0171, and Manfred Warmuth was also supported by ONR grant N00014-85-K-0445.

We thank David Haussler and Nick Littlestone for valuable discussions, as well as Phil Long and Naoki Abe for their helpful comments. In particular, we would like to thank David Haussler for an early proof of the Closure algorithm's instantaneous mistake bound.

Notes

1. Learnability with one-sided error implies learnability from positive examples (Kearns et al., 1987). The opposite is not quite true.
2. Such submodules are also called integer lattices of Z^k.
3. If the concept class is finite (considered in (Natarajan, 1987)) this definition is equivalent to requiring that the intersection of any pair of concepts in the class is also in the class.
4. Some work can be saved by placing x directly into EX_l (the label of x must be flipped if l is even). Furthermore, the repeat loop need only be executed when a mistake occurs, and can be started with i initialized to $l - 1$.
5. The definition of intersection closed ensures that $VCdim(C) \geq 1$. If C contains only a single non-empty concept then $VCdim(C) = 0$, but $VCdim(\mathrm{DIFF}^{=2}(C)) = 1$.
6. Some relatively benign measure-theoretic assumptions are required (Blumer et al. 1989) on the probability distribution.
7. The inequalities $VCdim(\wedge_{j=1}^{s} C_j) \leq \Sigma_{j=1}^{s}\ VCdim(C_j)$ and $VCdim(\cup_{j=1}^{s} C_j) \leq \Sigma_{j=1}^{s}\ VCdim(C_j)$ do not hold for arbitrary concept classes C_j (Dudley, 1984). Also, if each C_j is intersection closed and contains the universal concept then the second inequality does not appear to be tight for large s. See (Helmbold et al. 1989b) for a more detailed discussion.
8. If $t\widehat{\widehat{\mathbf{MI}}}_{\mathcal{B}}(B)(t)$ is not nondecreasing, it may be possible to substitute a slight overestimate of $\widehat{\widehat{\mathbf{MI}}}_{\mathcal{B}}(B)(t)$ where t times the overestimate is nondecreasing.
9. The converse does not necessarily hold since $\mathcal{B}$ could change the innermost part of the hypothesis even when no mistake is made.
10. Note that (Littlestone, 1989) gives an optimal transformation from an algorithm making at most M mistakes to an algorithm producing with probability at least $1 - \delta$ a hypothesis of error at most ϵ from $O(\frac{M}{\epsilon} + \frac{1}{\epsilon} \log \frac{1}{\delta})$ random examples.
11. Note that by a theorem of (Haussler et al. 1988), any *consistent* algorithm for learning $\mathrm{DIFF}(C, B)$ (including the variant of the Total Recall algorithm given in Figure 9) achieves an instantaneous sample mistake bound of $O(\log(\frac{t}{d})\frac{d}{t})$, for large enough t.

Appendix: The weighted majority algorithm

This appendix describes one version of Weighted Majority Algorithm, $\mathcal{WMA}$, presented in (Littlestone and Warmuth, 1989). Algorithm $\mathcal{WMA}$ has the following inputs:

- Two parameters α and β in the interval (0, 1). For simplicity of presentation we set $\alpha = \beta = \frac{1}{2}$. Optimizing these parameters for particular applications appears to be nontrivial.
- A specification of a countably infinite sequence of algorithms $\mathcal{A}_1, \mathcal{A}_2, \ldots$
- Two computable functions, W and $\widehat{W}$ defined on the positive integers. The first function is used to determine the initial weights; that is, $W(i)$ is the initial weight of algorithm $\mathcal{A}_i$. The second function is used to bound the tails of the sequence $W(i)$. More precisely $\widehat{W}$ is any function satisfying $\widehat{W}\ (i) \geq \Sigma_{j=i}^{\infty} W(j)$.

Description of $\mathcal{WMA}$: At any point $\mathcal{WMA}$ runs an initial segment $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_\ell$ of the infinite sequence of algorithms in parallel. This segment is called the *active pool* and l the *active pool size*. Initially, l is set to 0; then, and whenever a mistake is made, it is increased (if necessary) until the inequality $\widehat{W}(l + 1) \leq \frac{1}{2}(\frac{7}{8})^m \widehat{W}(1)$ holds, where m is the number of mistakes so far. The initial weights of the algorithms that are newly added into the active pool are determined with the function W. Whenever $\mathcal{WMI}$ is asked to predict on a new unlabeled instance, it compares the total weight q_0 of the algorithms in the current active pool that predict 0 to the total weight q_1 of the algorithms predicting 1. Algorithm $\mathcal{WMI}$ predicts according to the larger total (or arbitrarily in case of a tie). When $\mathcal{WMI}$ makes a mistake then the weights of those algorithms in the active pool that agreed with the master algorithm (i.e., those that mislead the master) are cut in half.

The goal is to keep both the number of mistakes made by $\mathcal{WMI}$ and the active pool size small. The performance can be tuned by judiciously choosing the functions W and $\widehat{W}$. In the theorem below (very similar to Corollary 4.4 in (Littlestone and Warmuth, 1989)) we give a worst case mistake bound for $\mathcal{WMI}$ when the functions are chosen appropriately for our application.

We choose the initial weight function $W(i) = \frac{1}{2}^{2^{\lceil i/2 \rceil}}$, for $i \geq 1$, and bound the tails of the sequence $W(i)$ by $\widehat{W}(i) = \frac{1}{2}^{2^{\lceil i/2 \rceil}-2} \geq \Sigma_{j=i}^{\infty} W(j)$.

THEOREM 13. Let S be any sequence of examples and m_i (for $i \geq 1$) be the number of mistakes made by $\mathcal{A}_i$ on S. Let the function W and $\widehat{W}$ be chosen as above. Then the following holds for $\mathcal{WMI}$ when applied to the countably infinite pool $\mathcal{A}_1, \mathcal{A}_2, \ldots$ on the sequence S.

1. The total number of mistakes made by $\mathcal{WMI}$ on the sequence S is at most $\inf_{i \geq 1}(2^{\lceil i/2 \rceil} + m_i)/\log_2 \frac{8}{7}$.
2. After m mistakes have been made by $\mathcal{WMI}$ the size of the active pool is at most $\lceil 2 \log_2(m \log_2 \frac{8}{7}) \rceil + 5$.

We apply the above theorem for the case where $\mathcal{A}_i$ is the algorithm of Figure 10 with depth parameter $p = 2^{\lceil i/2 \rceil} - odd(i)$, where $odd(i) = 1$ if i is odd and 0 otherwise. In other words, each $\mathcal{A}_i$ is the algorithm $\mathcal{D}_{2^{\lceil i/2 \rceil} - odd(i)}$. Note that the sequence of depth parameters produced by the function $2^{\lceil i/2 \rceil} - odd(i)$ (for $i \geq 1$) is the same as the sequence used in Theorem 12.

THEOREM 14. The algorithm $\mathcal{WMI}$ using the pool $\mathcal{A}_i \gtrsim \mathcal{D}_{2^{\lceil i/2 \rceil} - odd(i)}$ $(i \geq 1)$ and the above weight functions W *and* $\widehat{W}$ has the following properties when the target concept is in $\mathrm{DIFF}^{=p}(C, B)$:

1. The worst case total number of mistakes $\mathbf{M}.(\mathrm{DIFF}^{=p}(C, B))$ is $O(p\mathbf{M}_{\mathrm{CLOS}}(C) + \mathbf{M}_{\mathcal{B}}(B))$.
2. The size of the active pool is at most $O(1) + 2 \log_2(\mathbf{M}.(\mathrm{DIFF}^{=p}(C, B)))$.
3. The largest depth parameter of all algorithms in the active pool is $O(\mathbf{M}.(\mathrm{DIFF}^{=p}(C, B)))$.
4. The total space used is $O(s_C\mathbf{M}.(\mathrm{DIFF}^{=p}(C, B)) + s_{\mathcal{B}}\log[\mathbf{M}.(\mathrm{DIFF}^{=p}(C, B))])$, where s_C and $s_{\mathcal{B}}$ bound the space used by the on-line Closure algorithm for C and the given on-line algorithm for B respectively.

Proof of Part 1. Let $\bar{i}$ be the smallest number such that $\bar{p} = 2^{\lceil \bar{i}/2 \rceil} - odd(\bar{i}) \geq p$ and $\bar{p}$ has the same parity as p. It is easy to see that $p \leq \bar{p} \leq 2p - 1$. Let $m_{\bar{i}} = \mathbf{M}_{\mathcal{A}_{\bar{i}}}(\mathrm{DIFF}^{=p}(C, B))$. By the above bounds on $\bar{p}$ and by Corollary 2, $m_{\bar{i}} \leq (2p - 2)$

$\mathbf{M}_{\text{CLOS}}(C) + \mathbf{M}_{\mathcal{B}}(B)$. Part 2 of Theorem 13 guarantees the following bound on the worst case total number of mistakes of $\mathcal{WMI}$ when applied with the specified inputs:

$$\mathbf{M}.(\text{DIFF}^{=p}(C, B)) \leq \frac{2^{\lceil \bar{i}/2 \rceil} + m_{\bar{i}}}{\log_2 \frac{8}{7}} = O(p\mathbf{M}_{\text{CLOS}}(C) + \mathbf{M}_{\mathcal{B}}(B)).$$

This proves Part 1 of the theorem.

Proof of the remaining parts. Part 2 follows from the first part and Part 1 of Theorem 13. Part 3 is derived from plugging the maximum pool size of Part 2 into the depth function $2^{\lceil i/2 \rceil} - odd(i)$. To prove Part 4, split the space used by all algorithms in the active pool into two portions: the space used for closure algorithms for C and the space used for simulating copies of $\mathcal{B}$ and for storing weights. Since the depth function grows doubly exponentially with i, the total space used for the first portion is at most $O(s_C \cdot p_{\max})$, where $p_{\max}$ is the maximum depth parameter of any call to $\mathfrak{D}_p$ that was bounded in Part 3. The space used in the second portion is on the order of s_B times the maximum pool size, which was bounded in Part 2. □

References

Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M.K. (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, *36*, 929–965.

Board, R. and Pitt, L. (1990). On the necessity of Occam algorithms. *Proceedings of the Twenty-Secnd Annual ACM Symposium on Theory of Computing*.

Boucheron, S. (1988). Learnability from positive examples in the Valiant framework. Unpublished manuscript.

Dudley, R.M. (1984). A course on empirical processes. *Lecture Notes in Mathematics No. 1097.* New York: Springer-Verlag.

Ehrenfeucht, A., Haussler, D., Kearns, M., and Valiant, L. (1989). A general lower bound on the number of examples needed for learning. *Information and Computation*, *82*, 247–261.

Haussler, D. (1989). Learning conjunctive concepts in structural domains. *Machine Learning*, *4*, 7–40.

Haussler, D., Kearns, M., Littlestone, N., and Warmuth, M.K. (1990). Equivalence of models for polynomial learnability. *Information and Computation*. To appear.

Haussler, D., Littlestone, N., and Warmuth, M.K. (1988). Predicting {0, 1}-functions on randomly drawn points. *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pp. 100–109. White Plains, NY: IEEE. Tech. Report, U.C. Santa Cruz. To appear (longer version).

Haussler, D. and Welzl, E. (1987). Epsilon-nets and simplex range queries. *Discrete Computational Geometry*, *2*, 127–151.

Helmbold, D., Sloan, R., and Warmuth, M.K. (1989a). *Learning lattices and reversible, commutative regular languages*. (Technical Report UCSC-CRL-89-23). Santa Cruz, CA: U.C. Santa Cruz, Computer Research Laboratory.

Helmbold, D., Sloan, R., and Warmuth, M.K. (1989b). *Learning nested differences of intersection-closed concept classes*. (Technical Report UCSC-CRL-89-19) Santa Cruz, CA: U.C. Santa Cruz, Computer Research Laboratory.

Helmbold, D., Sloan, R., and Warmuth, M.K. (1989c). Learning nested differences of intersection-closed concept classes. *Proceedings of the Second Workshop on Computational Learning Theory* (pp. 41–56). Santa Cruz, CA: Morgan Kaufmann.

Kearns, M., Li, M., Pitt, L., and Valiant, L. (1987). On the learnability of boolean formulae. *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing* (pp. 285–295). New York.

Kearns, M. and Valiant, L.G. (1989). Cryptographic limitations on learning boolean formulae and finite automata. *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing* (pp. 433–444). Seattle, Washington.

Littlestone, N. (1988). Learning when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, *2*, 285–318.

Littlestone, N. (1989). From on-line to batch learning. *Proceedings of the Second Workshop on Computational Learning Theory* (pp. 269–284). Santa Cruz, CA: Morgan Kaufmann.

Littlestone, N. and Warmuth, M.K. (1989). *The weighted majority algorithm* (Technical Report UCSC-CRL-89-19). Santa Cruz, CA: U.C. Santa Cruz, Computer Research Laboratory. An extended abstract is available in *Proceedings of the 30th Annual Symposium on Foundations of Computer Science* (pp. 256–261). Research Triangle, NC: October 1989.

Natarajan, B.K. (1987). On learning boolean functions. *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing* (pp. 296–304). New York.

Pearl, J. (1978). On the connection between the complexity and credibility of inferred models. *Journal of General Systems*, *4*, 255–264.

Pitt, L. and Valiant, L.G. (1988). Computational limitations on learning from examples. *Journal of the ACM*, *35*, 965–984.

Pitt, L. and Warmuth, M.K. (1990a). The minimum consistent DFA problem cannot be approximated within any polynomial. *Journal of the ACM.* To appear.

Pitt, L. and Warmuth, M.K. (1990b). Prediction preserving reducibility. *Journal of Computer and System Sciences.* To appear in special issue consisting of papers from the third annual IEEE Conference on Structures in Complexity Theory, 1988.

Rivest, R.L. (1987). Learning decision lists. *Machine Learning*, *2*, 229–246.

Salzberg, S. (1988). *Exemplar-based learning: theory and implementation.* (Technical Report TR-10-88) Cambridge, MA: Harvard University, Center for Research in Computing Technology.

Shvaytser, H. (1988). Linear manifolds are learnable from positive examples. Unpublished manuscript.

Valiant, L.G. (1984). A theory of the learnable. *Communications of the ACM*, *27*, 1134–1142.

Vapnik, V.N. (1982). *Estimation of Dependences Based on Empirical Data.* New York: Springer-Verlag.

Vapnik, V.N. and Chervonenkis, A.Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, *16*, 264–280.