Implementing Valiant's Learnability Theory Using Random Sets

E.M. OBLow

*Oak Ridge National Laboratory, Engineering Physics and Mathematics Division, P.O. Box 2008, Oak Ridge,
TN 37831-6364*

Editor: David Haussler

Abstract. A general learning framework which uses random sets is introduced for solving discrete-space classification problems. This framework is based on the pac-learning formalism introduced by Valiant (1984) and generalized in set-theoretic terms by Blumer, et al., (1989). The random set version of this theory is used to develop an algorithm which is a particularly efficient search scheme. This is accomplished by recasting the representational class and constructive proof presented in Valiant (1984) into random set terms and implementing it as an exhaustive search algorithm. The algorithm is a problem-specific incremental (psi) approach in that it satisfies learnability criteria for distribution-specific problems as examples are being sampled. Some theoretical and empirical analyses are presented to demonstrate the convergent pac-learnability and sample complexity of this psi-algorithm. Its performance is then tested on the multiplexor class of problems. This class has been analyzed by others as a benchmark for decision trees and genetic classifiers. Results from these test cases show that, despite using an exhaustive search, this random set implementation is computationally competitive with these more established methods (which use empirically proven heuristics). Conclusions are drawn about potential further improvements in the efficiency of this approach.

Keywords. Pac-learning, Valiant's learnability theory, Boolean classifiers, learning from examples

1. Introduction

This paper serves as an introduction to a random set approach to classification learning problems. The formalism to be discussed and demonstrated is based on the learning theory recently introduced by Valiant (1984, 1985) and generalized by Blumer, et al., (1989). The importance of Valiant's framework is that it supplies a general theoretical basis for probabilistically approximating classes of learnable concepts. For this reason, the theory has been termed pac-learning (Angluin, 1988), an acronym for probably approximately correct learning. The renewed interest in machine learning generated by Valiant's seminal theory is considerable, as evidenced by the attention it has received at recent meetings on computing and machine learning (see e.g., Haussler & Pitt 1988; Rivest, et al., 1989) and the wide range of learnability issues, including those related to neural nets (e.g., Baum & Haussler 1989), with which it can deal. It is precisely the probabilistic nature of this approach to learning which has motivated us to seek a random set interpretation of Valiant's theory.

In this theory's earliest published form, Valiant dealt primarily with the issue of defining learn ability in discrete sample spaces. The two most important problems addressed were:

1) estimating bounds on sample size needed to insure learning from examples to within a specified probabilistic error and 2) determining the classes of polynomial algorithms and representations that can use such bounded samples. Both of these problems are central to establishing, in a distribution-free manner, the complexity of a learning algorithm which can learn probabilistically. Valiant provided an analysis and constructive proof of learnability for the class of Boolean k -DNF and k -CNF functions.

While these two problems and its application to Boolean functions have generated a great deal of theoretical interest, direct set-theoretic implementations of Valiant's approach have not yet appeared. In practice, major discrete-space learning algorithms still rely heavily on decision tree (Quinlan, 1979; Quinlan, 1985), genetic (Holland, 1975), and stochastic schemes (Duda & Hart, 1973). Although this fact is in part due to the recent nature of the theory, it is we feel, more a result of the apparent inefficiency of such algorithms, as suggested by the worst-case bounds on sample sizes which have been developed for pac-algorithms. Since many practical problems are not expected to involve worst-case scenarios, these simpler cases need to be addressed in general as well. Blumer, et al., (1989) have pointed out that improving sample bounds in both general and specific cases and addressing learnability issues for the non-Boolean basis of other methodologies are both important outstanding research issues.

The developments presented in this paper will try to address these concerns by focusing on a random set approach to pac-learning. Since many in the machine learning community may not be familiar with this framework, we will try to introduce the random set formalism using the existing set-theoretic approach and notation developed in Blumer, et al., (1989). Within this context, we will describe a useful random set algorithm which is a generalization of Valiant's original constructive proof of learnability for k -DNF functions. A variety of other random set algorithms are also possible if heuristic approaches are used, but we will concentrate on this simple set-theoretic case in this paper.

Despite its limitations, we believe that the specific algorithm to be discussed serves as a stringent test of the potential of the random set approach in general. This is a result of its being based on an *exhaustive search* for finding consistent hypothesis sets. As such, it might be considered to be one of the more inefficient approaches to classification learning. It would certainly be difficult to scale this approach up to deal with many of the practical learning problems which need to be solved. All existing large-scale learning methodologies (e.g., decision tree and genetic algorithms), in fact, make liberal use of heuristics to limit the extent of search. While the development of heuristics within a random set framework is also possible, we will avoid this step for the sake of clarity in this paper. Instead, we feel that comparing an exhaustive algorithm with methodologies which use proven heuristics provides an even clearer test of the random set approach.

To meet this comparison objective, a test of the random set will be restricted to a Boolean learning problem which has received thorough investigation in the literature — the so-called multiplexor benchmark problem analyzed by Wilson (1987), Quinlan (1988) and Pagallo & Haussler (1990). These published analyses provide detailed results which can be used to test a large-scale random set implementation of Valiant's algorithm against more established learning methodologies.

In the following sections of this paper, the random set formulation of Valiant's theory will be presented and demonstrated. We will introduce the necessary random set concepts

first. The development of a specific random set algorithm very similar to Valiant's original constructive proof of learnability will then be given. A more practical approach to determining the degree of convergence of this algorithm will be covered next. This will provide the framework in which to define an incremental version of this algorithm which has problem-specific convergence properties. A successfully implemented computer code embodying this random set algorithm will then be outlined. Results generated by this code will be compared with other published work on the Boolean multiplexor which has served as a learning benchmark for decision tree and genetic algorithms.

2. Review of Valiant's approach

To begin, let us briefly review Valiant's pac-learning theory. Since this formalism has been generalized considerably in the years since its introduction, we need to cover both the approach of Valiant (1985) and the more recent work of Blumer, et al., (1989). In this exposition we will explicitly highlight the random set aspects of pac-learning theory, which are covered by Blumer, et al., but only implicitly. We will also spend some time on Valiant's Bernoulli trials approach to Boolean problems because of its additional usefulness in analyzing convergence properties of random set algorithms.

2.1. Random set formalism

Using the notation of Blumer, et al., (1989), we assume that we are trying to learn a target category T which is a member of a class of categories C . This class of categories is defined over a set of points $x \in X$, which constitute the space of examples from which learning will take place. The sample points which are to be classified by a learning algorithm are usually called instances or patterns or objects, depending on the learning problem being solved. We will use the term instances to describe such learning examples.

In the general set framework, C is a class of sets, with each class being some subset of the power set of X (i.e., $C \subseteq 2^X$). Any $T \in C$ will therefore be a particular subset of points $T \subseteq X$. The complement of T (i.e., $\bar{T}$), will then be the set of points $\bar{T} \equiv X - T$. A probability distribution $p(x)$ is assumed to be defined over X . A sample of m instances, randomly drawn from X using this distribution, is used to learn T or its complement $\bar{T}$. This random m -sample, denoted by $\bar{x} = (x_1, \dots, x_m)$, is thus an m -vector in the Cartesian product space X^m (i.e., $\bar{x} \in X^m$).

A learning algorithm $\mathcal{Q}$ is defined as a function $\mathcal{Q}(\bar{x})$

$$\mathcal{Q}(\bar{x}): [\bar{x} \in X^m] \rightarrow [H \in C], \quad (1)$$

which takes as its inputs the m -sample $\bar{x}$ and produces as its output a particular member of C called the hypothesis set H , an approximation to T . In the pac-learning framework, this hypothesis H probabilistically approximates T with error ε . This error is formally defined by the probability measure associated with the error set R , defined as the symmetric difference between H and T . That is

$$\mathcal{E}(\bar{x}) \equiv p(R) = p(H \triangle T). \quad (2)$$

The hypothesis set H and the error set R together are the fundamental sets of interest in the random set formulation of Valiant's learnability theory. H and R are clearly random sets since they both result from a mapping of random variables. That is, for a given value of m and a particular set T , each probabilistic realization of $\bar{x}$ gives rise to a particular H and R as a result of the mapping given in equation (1). The probability of realizing any $\bar{x}$ is determined solely by the probability ρ of picking any random m -sample of x 's from the probability distribution $p(x)$.

In this way, each m -sample is a point $\bar{x} \in X^m$ with an associated probability $\rho(\bar{x})$. Since both H and R are sets in X , they also have associated probability measures, $p(x \in H)$ and $p(x \in R)$, respectively. All the measures of interest are thus derived from $p(x)$ and are defined over domain and range of $\mathcal{Q}(\bar{x})$. The existence of such measures formally ties these sets to the random sets analyzed extensively by Kendall & Harding (1974) as part of the field of stochastic geometry.

To define learnability in Valiant's theory, the number of sampled instances m is usually specified to be $m(\epsilon, \delta)$, a function of two parameters ϵ and δ , the approximation *error* and its *confidence*, respectively. Both of these parameters have values in the range $[0, 1]$. Given fixed values for ϵ and δ , learnability is defined in Valiant's formalism to be the existence of an algorithm $\mathcal{Q}(\bar{x})$ which produces a hypothesis H that has an error $\mathcal{E}(\bar{x}) > \epsilon$ with probability $\leq \delta$. Stated another way, T is defined to be learnable under the distribution $p(x)$, if $\mathcal{Q}(\bar{x})$, using a series of randomly drawn instances of size $m(\epsilon, \delta)$, produces with probability $1 - \delta$, a hypothesis H with error no more than ϵ . The smallest sample size $m(\epsilon, \delta)$ which satisfies this criteria uniformly over all $T \in C$ in a worst-case scenario, denoted as L_m , is called the *sample complexity*.

2.2. Valiant's Boolean formalism

Since we will use a Boolean instance space X to test our approach and also because this case clarifies some of the concepts just defined, we need to briefly review the Boolean class discussed in Valiant (1985) at this point. In this latter work, Valiant takes X to be the discrete set of points defined by the vertices of an n -dimensional hypercube. Each point $x \in X$ is thus a binary bit-string of length n (an n -vector in $\{0, 1\}^n$).

The two most important aspects of learnability for the discrete-field problem discussed in detail by Valiant were: 1) the estimation of sample complexities for various learning problems and 2) the determination of classes of functions which could be called learnable in the sense defined above. One of the most important problems that he dealt with was learning Boolean functions $f(x) \in \{0, 1\}$, where x here is an n -dimensional instance vector of Boolean variables, a_i , $i = 1, \dots, n$. The variables a_i take on values in $\{0, 1\}$ so that in general $x \in \{0, 1\}^n$. This case was studied using target categories T from the class of either k -DNF and k -CNF Boolean functions.

Valiant defined the class k -DNF (k -CNF) to be a disjunctive (conjunctive) normal form which contains at most k literals per term. The term *normal form* was defined as a Boolean function consisting of a disjunction (conjunction) of terms, each of which is a conjunction

(disjunction) of literals. A literal, here, is any one of the n Boolean variables a_i or their complements $\tilde{a}_i$. The numbers of terms in both of these normal forms is $\binom{n}{k}$, the number of ways k literals can be formed from n attributes. These combinations are $O(n^k)$ with respect to the dimension n of the instance vector. This polynomial order is one of the properties that made this class of functions learnable with a deterministic polynomial algorithm.

For fixed k , just such a deterministic polynomial algorithm was presented by Valiant for learning approximations to target functions in the class of k -DNF or k -CNF functions. Such functions were learned from either positive or negative instances of the target function T . Positive instances were defined to be members of the set $T \equiv \{x \mid x \in X, f(x) = 1\}$ and negative instances were members of the complement set $\tilde{T} \equiv \{x \mid x \in X, f(x) = 0\}$. Valiant showed that target functions from the k -CNF class were learnable from positive instances chosen from $p_+(x) \propto p(x \in T)$, and functions from the k -DNF class were learnable from negative instances chosen from $p_-(x) \propto p(x \in \tilde{T})$. Any target function T in these classes was shown to be learnable (probabilistically approximated) to within an arbitrarily specified error ϵ with confidence $1 - \delta$. This convergence was proven by providing a distributed-free upper bound m_V , to the number of instances needed to achieve the stated error and confidence limits.

Valiant developed his general bound on sample complexity by analyzing a classic urn problem. In this problem, balls of at most s different types were drawn from an urn to obtain a representative sample of the types making up at least $1 - \epsilon$ of the total number of balls in the urn. The sampling process was formulated in terms of a succession of Bernoulli trials each with a probability of success of at least ϵ . Success here was defined to be the selection of a type of ball not previously seen before. The maximum number of successes was, therefore, less than or equal to s .

In particular, Valiant showed that L_m could be bounded from above with a sample size m_V given by

$$L_m < m_V = 2h(s + \ln h), \quad (3)$$

where s is $O(n^k)$ for k -DNF or k -CNF and $h = 1 / \epsilon = 1 / \delta$ with $h > 0$.

This result for discrete-space problems has been generalized and improved by Haussler (1988a) and Blumer, et al., (1989) giving

$$L_m < m_B = h \left(\ln |C| + \ln \frac{1}{\delta} \right), \quad (4)$$

where $|C|$ is the cardinality of the concept class.

While the class of Boolean functions Valiant discussed is quite general and useful in a machine learning context, the bound given above and the number of comparison steps needed in his constructive proof appear to be large for most practical implementations. This drawback makes Valiant's published algorithm seem uncompetitive with other alternative learning schemes. We will develop a random set algorithm from his constructive proof later in this paper and show that the drawbacks are not as serious as one might assume. To demonstrate this, we will first formally generalize Valiant's algorithm in explicit random

set terms and then construct an efficient computer implementation of it to solve general classification learning problems. New ways to establish learnability convergence, more suitable for practical implementation, will also be developed.

2.3. General random set approach

The computational algorithm to be developed is best introduced by generalizing Valiant (1985) using a specific representational class of categories in C . As a first step, we will extend the Boolean framework Valiant discussed to include the kinds of representations dealt with in genetic (Holland, 1975) and decision tree (Quinlan, 1985) algorithms. These representations fall between the two extremes of Boolean functions on the one hand and continuous functions on the other. They, therefore, serve as a vehicle for addressing the research area of non-Boolean representations highlighted by Blumer.

To begin, we expand the definition of an instance $x \in X$ to be a general discrete n -attribute vector $x = (a_1, a_2, \dots, a_n)$. Each attribute a_i is now allowed to have any number of values (not just two as in the Boolean case). These general v_i values will, for notational simplicity, be just the set of integers $\{1, \dots, v_i\}$. In a machine learning context, for example, the attributes can be defined to be colors, sizes, shapes or any other relevant properties used to describe an instance. These attributes embody all the information available to a learning algorithm to classify any instance into a category. The numbering scheme used here simply allows each attribute subcategory to be described by an integer. Thus, for instance, we could have 1 = green, 2 = red, etc., for a color attribute and 1 = small, 2 = large, etc., for a size attribute, and so on for all other attributes.

In general then, we have each $a_i \in \{1, \dots, v_i\}$ and cardinality of the instance space will be $|X| = \prod_{i=1}^n v_i$. In the Boolean case, for example, $v_i = 2, \forall i$ and shifting the values to $a_i \in \{0, 1\}$, we get $|X| = 2^n$. Since an arbitrary category $c \in C$ can, in general, be any subset of instances x in the set X , this class has exponential complexity. That is, $c \in 2^X$ since in general $C = 2^X$. To reduce this exponential complexity to a polynomial form, we will restrict the c 's to be members of the class of categories $C_k \subset C$ used in decision tree and genetic learning algorithms. This is accomplished by using the following vector notation to define the fundamental categories (schemata in genetic algorithm parlance) $b \in C_k$

$$b = (a'_1, a'_2, \dots, a'_n). \quad (5)$$

The unions of any combination of these basic categories will be used to define any allowable category $c \in C_k$.

In this vector form, each a'_i takes its "value" from an augmented value set $\{1, \dots, v_i, \#_i\}$. The $\#_i$ symbol used in this vector represents the entire *set* of v_i values, so that $\#_i \equiv \{1, \dots, v_i\}$. The appearance of the $\#_i$ symbol at the i th position in the vector means that the attribute it replaces takes on *all* the values from 1 to v_i . Such an attribute will be called *inactive*. The indices of these inactive attributes will be denoted by the set $I = \{i \mid a_i = \#_i\}$. In a similar fashion, those attributes *not* replaced by a $\#_i$ symbol will be called *active* and

the set of indices for them will be denoted by $A \equiv \{i \mid a_i \neq \#_i\}$. This allows any arbitrary b to be written equivalently as a set of instances

$$b = \{x \mid x = (a_1, \dots, a_n) \text{ with } a_i = 1, \dots, v_i, \forall i \in I\}. \quad (6)$$

The number of instances in this set (i.e., its cardinality $|b|$) is

$$|b| = \prod_{i \in I} v_i. \quad (7)$$

A simple example of one such category $b \in C_k$ for $n = 3$ and $v_2 = 3$ is $b = (2, \#, 1) \equiv \{(2, 1, 1), (2, 2, 1), (2, 3, 1)\}$.

From this example it should be clear that these categories (i.e., sets of instances) are simply conjunctions of the active attributes. The inactive attributes contain the information that can be ignored in defining the category. For example, if a category was given by the conjunction, $a_1 \wedge a_2 \wedge a_3$ in which each attribute had a specific integer value, then in vector notation the attributes a_i , $i = 4, \dots, n$ would be called inactive and have $\#$ symbols associated with them.

Since the number of ways l active attributes can be placed in a string of n attributes is given by the binomial coefficient $\binom{n}{l} = n!/(n-l)!l!$, the parameter l , called a *level*, can be used to define a natural ordering of the subclasses of C_k in terms of active attributes. In this way, we now define each of the $j = 1, \dots, \binom{n}{l}$ combinations of l active attributes to be a *partition of X using l attributes*. We denote these j -partitions by c_j^l . Each of these partitions represents a general conjunctive term of exactly l active attributes. We use the partition notation since it clearly denotes the disjoint subdivision of a space.

We will use the scripted notation A_j^l (or I_j^l) to denote the set of indices of the active (or inactive) attributes in the j th-partition level l . In this way, we can define the most basic categories in C_k , denoted by b_{vj}^l , which constitute the disjoint subsets of each partition as

$$b_{vj}^l = \{x \mid x = (a_1, \dots, a_n) \text{ with } a_i = 1, \dots, v_i, \forall i \in I_j^l\}, \nu = 1, \dots, v_j^l. \quad (8)$$

The maximum ν -index in this expression is given by

$$\nu_j^l = \prod_{i \in A_j^l} v_i \text{ with } \nu_1^0 \equiv 1. \quad (9)$$

In machine learning parlance, a basic category is thus a conjunctive of l active attributes in which each attribute takes on a specific value in its range. For example, take the general conjunction (partition) of interest to be $a_1 \wedge a_2 \wedge a_3$, where a_1 is a color attribute, a_2 is a size attribute, and a_3 is a shape attribute. One basic category might then be defined by the set of all instances which have attribute values of color-red *and* size-large *and* shape-square, another might be all those instances with color-green *and* size-small *and* shape-triangle, and so on over the whole range of colors, sizes, and shapes.

Using these definitions, the restricted polynomial class of categories $C_k \subset 2^X$ can now be constructed from unions of all possible combinations of the basic categories. This class is then

$$C_k = \{ c = \cup b_{\nu j}^l, \text{ for any } \nu = 1, \dots, \nu_j^l, j = 1, \dots, \binom{n}{l}, l = 0, 1, \dots, k \}. \quad (10)$$

The basic sets in any j -partition of X is then given by

$$c_j^l = \{ b_{\nu j}^l, \nu = 1, \dots, \nu_j^l \}. \quad (11)$$

Here, the order of this class is denoted by the maximum level k .

To clarify the notation just introduced for those more familiar with genetic algorithms and Boolean decision trees, for instance, the partitions and categories for an $n = 3$ Boolean case (i.e., 3-bit binary-string instances) with levels up to $k = 2$ (corresponding to a 2-DNF) are:

for $l = 0$,

$$c_1^0 = [b_{11}^0] = [(\#, \#, \#)], \quad (12)$$

for $l = 1$,

$$\begin{aligned} c_1^1 &= [b_{11}^1 \mid b_{21}^1] = [(0, \#, \#) \mid (1, \#, \#)] \\ c_2^1 &= [b_{12}^1 \mid b_{22}^1] = [(\#, 0, \#) \mid (\#, 1, \#)] \\ c_3^1 &= [b_{13}^1 \mid b_{23}^1] = [(\#, \#, 0) \mid (\#, \#, 1)], \end{aligned} \quad (13)$$

and for $l = 2$,

$$\begin{aligned} c_1^2 &= [b_{11}^2 \mid b_{21}^2 \mid b_{31}^2 \mid b_{41}^2] = [(0, 0, \#) \mid (0, 1, \#) \mid (1, 0, \#) \mid (1, 1, \#)] \\ c_2^2 &= [b_{12}^2 \mid b_{22}^2 \mid b_{32}^2 \mid b_{42}^2] = [(0, \#, 0) \mid (0, \#, 1) \mid (1, \#, 0) \mid (1, \#, 1)] \\ c_3^2 &= [b_{13}^2 \mid b_{23}^2 \mid b_{33}^2 \mid b_{43}^2] = [(\#, 0, 0) \mid (\#, 0, 1) \mid (\#, 1, 0) \mid (\#, 1, 1)]. \end{aligned} \quad (14)$$

At each level l , a series of j -partitions each divides the space X into 2^l disjoint categories $b_{\nu j}^l$. As such, these partitions represent all the possible conjunctive terms in a DNF representation of genetic algorithm or decision tree rules. The active attributes in each conjunctive term can be identified with the literals in these normal forms. The partitions also represent individual urn problems like the one Valiant used to estimate the bounds on sample complexity for target categories in a k -DNF or k -CNF representation.

Any target category T used in the learning algorithm to be developed will be assumed to be constructed from the unions of basic categories in C_k , augmented by the null set ϕ for the sake of completeness. In a machine learning context, what we are saying here is,

that T can be made up of arbitrary disjunctions of the conjunctions of attributes that define any partition. This construction results in a general disjunctive normal form for representing target elements in X .

The order of the class, denoted by k , is clearly related to the maximum number of attributes found in any single conjunctive combination of attributes in the class. For the Boolean case, this k value has a comparable usage in defining the class of k -DNF functions (i.e., the class of Boolean functions made up of disjunctions of conjunctions of literals with at most k literals per conjunction). In the general random set case, the k -DNF class would thus be equivalent to the class C_k if $X \equiv \{0, 1\}^n$. As defined, C_k is clearly equivalent to the schemata Holland used in formulating genetic algorithms (Holland, 1975) and classifier systems (Holland, 1986). It can also be interpreted as the representation used to specify rules or leaf nodes generated in the decision tree formalism of Quinlan (1985).

3. Bounds on sample size

Since learnability for general class of the C_k type have already been studied by Valiant (1985), Haussler (1988a), and Blumer, et al., (1989), we need not belabor this issue. While these studies provide distribution-free bounds on sample complexity, the proofs by and large are existential, making use of established inequality constraints. In this regard, they provide few clues as to how to actually construct worst-case distributions or how to take advantage of easier learning scenarios should they arise in practice. To make the random set formalism part of a more practical algorithm, we need to address these limitations.

The effort in this section will, therefore, focus on two issues. First, we will try to develop a suitably defined class of *hard* probability distributions which will allow us to test the limits of the random set approach we will use in practice. Second, we will explore the theory behind a problem-specific incremental (ψ) approach to bounding the number of instances needed for learning. Using this scheme, easier problems can be solved more quickly and with smaller numbers of instances. Both of these objectives will be achieved by reworking the Bernoulli trials problem Valiant proposed for establishing learnability bounds for the Boolean k -DNF class.

3.1. Bernoulli trials analysis

The first issue we will explore is finding a suitably defined *hard* class of probability distributions using the Bernoulli trials framework proposed by Valiant. In his analysis, Valiant defined a classic urn problem and a Bernoulli trials selection scenario. The urn was assumed to contain N balls of at most s different types, with $s \leq N$. In random set notation, the urn corresponds to a single j -partition with each type of ball corresponding to a particular basic category in such a partition. The types (categories) here, will be generically denoted by b and their total number can be interpreted as $s = \nu_j^1$.

To estimate the contents of the urn so defined, a random sample of balls was drawn from the distribution $p(b)$ in order to obtain a representative sample of at least $1 - \epsilon$ of the different types of balls. Here, the probability for drawing any ball was taken to be $1/N$.

The probability for drawing a particular type b is then $p(b) = n_b/N$, where n_b is the number of balls of type b .

This sampling process was described by Valiant as a series of Bernoulli trials, where a successful trial was defined to be the selection of a ball whose type had not been seen before. In Valiant's analysis, each successful trial j , with $1 \leq j \leq s$, was assumed to have a probability of success p_j of at least ϵ , the target accuracy for learning (i.e., $p_j \geq \epsilon$, $\forall j$). A bound on sample complexity was then simply estimated as the maximum number of instances needed to achieve all s possible successes to within a specified probability constraint.

Using this urn model, we are now in a position to construct a more realistic series of trials which will then be used to define a class of *hard* distributions. To accomplish this, two important facts need to be considered: 1) after each success the probability for the next success must necessarily *decrease* since a ball of type b with at least one representative is now known to reside in the urn and 2) the first selection is by definition a success. Since we are looking for hard learning scenarios, it is clear that slow rates of decrease in probability after each success and long waiting times between successes will be needed to generate problems requiring large sample sizes.

To meet these criteria, we propose a "semi-uniform" distribution for each type b_i , $i = 1, \dots, s$ in the urn with the following parameterized form

$$p(b_i) = \begin{cases} 1 - \alpha & \text{if } i = 1; \\ \alpha/(s - 1) & \text{if } 1 < i \leq s, \end{cases} \quad (15)$$

where α is such that $p(b_1) > p(b_i)$, $\forall i$ and $0 < \alpha \leq 1 - 1/s$. A specific member of this class of distributions has also been proposed by Haussler (1988b) and was found to be useful in bounding sample complexities from below. The class should, therefore, provide useful hard learning problems for testing purposes.

With this distribution, we have one type of ball b_1 which is drawn with probability $p(b_1) = 1 - \alpha$ and all the $s - 1$ other types are uniformly distributed. If we were now to *assume* that the most probable ball b_1 was the first one chosen, then the Bernoulli trial success probabilities, p_j , $j = 1, \dots, s$ for such a distribution would be

$$p_j = \begin{cases} 1 & \text{if } j = 1; \\ \alpha(s - j + 1)/(s - 1) & \text{if } 1 < j \leq s. \end{cases} \quad (16)$$

Here, after selecting type b_1 , the success probabilities would decrease by the constant $\alpha/(s - 1)$ after each future success is recorded. The second success probability is thus α and no more than s successes are possible since $p_{s+1} = 0$. Since ball type b_1 is the most probable first selection, we can use this success distribution to obtain useful *mean* value results.

To construct a *mean* value estimation problem, we only need to note the following facts about the proposed distribution. First, *on average* in a series of Bernoulli trials, the first success (choosing type b_1 with success probability $p_1 = 1$) will decrease the success probability to α . While α remains arbitrary for the moment, it is clear that smaller α 's give

rise to longer waiting times for future successes. Second, the decrease in success probability p_j at each subsequent successful trial $1 < j \leq s$, is minimized by having a uniform distribution for the remaining different types of balls.

To find the distribution in this class which is hardest to learn (as far as mean sample sizes are concerned), we have to determine the maximum number of instances m needed to reach a remaining success probability level of ϵ . The tradeoff between reaching ϵ too quickly and decreasing probability too slowly, on average, with the first success is what must be determined here.

A solution for this problem can be found most easily by dealing with the expected (mean) sample size m , denoted by $E[m]$, which first achieves the ϵ cutoff. The maximum value of $E[m]$, taken with respect to α , will thus define a worst-mean scenario. The traditional approach to estimating waiting times in Bernoulli trials with variable success probabilities given in Feller (1968) is the needed basis for solving this problem. This analysis uses the geometric distribution to describe the expected waiting times in terms of numbers of failures before the next success.

Following the analysis of Feller (1968), we find that in a sequence of Bernoulli trials whose probability for success decreases uniformly by $\alpha/(s - 1)$ after the first trial, the expected number of trials before J successes is achieved is given by

$$E[m] = \sum_{j=1}^J p_j^{-1} = 1 + \sum_{j=2}^J \frac{s - 1}{\alpha(s - j + 1)}, \quad (17)$$

where J is such that

$$p_J = \frac{\alpha(s - J + 1)}{s - 1} = \epsilon. \quad (18)$$

This latter condition defines the J th success to be the one in which the success probability p_J just equals ϵ .

The sum in equation (17) can be approximated by an integral for large s to give

$$E[m] \approx \frac{s - 1}{\alpha} \ln \left(\frac{\alpha}{\epsilon} \right). \quad (19)$$

Differentiating this expression with respect to α and setting it equal to zero allows the value of α giving a maximal value of $E[m]$ to be found as

$$\alpha_{\max} = e\epsilon. \quad (20)$$

This result gives a good approximation to the value of α which defines the hardest distribution for learning in the class defined by equation (15). While this analysis is only approximate, and based on an expected sample size rather than distribution-free considerations,

producing a hard probability distribution which can be used for testing purposes, we feel, more than makes up for this deficiency.

Substituting $\alpha_{\max}$ into equation (19) gives the mean sample size which needed to meet the error constraint (i.e., the mean sample complexity)

$$E_{\max} [m] \equiv L_{E_m} \approx \frac{s-1}{e\epsilon}. \quad (21)$$

This value with $\epsilon = h^{-1}$, $\delta = 0.5$, and $|C| = 2^s$ should be compared to Valiant's bound given in equation (3) and Blumer's bound given in equation (4). A more rigorous derivation of this mean error and the worst-mean distribution that generated it can be found in Oblow & Uppuluri (1991).

It should be clear here that Valiant's analysis is consistent with the one just presented, if we were to assume that $\alpha_{\max} = \epsilon$ and that no decrease in probability occurred after any success (i.e., $p_j = \epsilon$ for $j > 1$). These conditions reduce the success probability immediately to ϵ after the first selection and hold it constant (albeit unrealistically) for all future successes to ensure a worst-case bound.

Having thus established the hardest distribution in the class proposed, we now only need to formally tie this distribution to the random set pac-learning framework. We can make this connection by noting that the random hypothesis set H in Valiant's analysis is defined as the set of types of balls in the urn which have *not* yet been seen in a sample of size m . These are the ones that have not been ruled out of the hypothesis set as being inconsistent with the counterexamples seen so far. The random hypothesis set of interest here is thus $H \equiv U$, the set of balls whose types have not yet been seen. Since U is a random set (arising from a random m -sample), its measure $p(U)$ is a random measure.

The importance of this measure is that it effectively defines the probability of making a classification error for any randomly selected instance [see equation (2)]. The worst-case scenario for reducing the *expectation* of the random error measure $p(U)$ to less than ϵ can thus be found using the class of distributions given in equation (15). This will define a worst-mean error distribution and a mean-error sample complexity which will be an approximation to a pac-learning problem with confidence parameter $\delta = 0.5$ (which defines a median error). Although the mean and the median errors are not equivalent in general, for the class of distributions and sets we are studying this approximation is reasonable. It will be tested empirically later, in any event, and should, therefore, pose no problems at this point.

For the mean value estimation problem at hand then, the expected (mean) error is by definition given by

$$e[p(U)] \equiv \sum_r \rho(\bar{x}_r) p(U(\bar{x}_r)). \quad (22)$$

Here, $\rho_r(\bar{x}_r)$ is the probability of realizing m -vector $\bar{x}_r$ in m instances sampled independently from $p(b)$, and $U(\bar{x}_r)$ is the random set of unseen types of balls which results from this realization.

Since the distribution of realizations of size m is analytically given by the multinomial distribution for choosing m balls with s possible types, this definition is equal to

$$E[p(U)] = \sum_{r(k_1, \dots, k_s)} \binom{s}{k_1, \dots, k_s} p(b_1)^{k_1} \dots p(b_s)^{k_s} \sum_{b_i \in U(\bar{x}_r)} p(b_i). \quad (23)$$

The order of this double sum can be exchanged to give the general result that

$$E[p(U)] = \sum_{i=1}^s p(b_i)(1 - p(b_i))^m. \quad (24)$$

To bound this expected error we need only to take the maximum of each individual term in this sum. Since each term achieves a maximum when

$$p(b_i) = \frac{1}{m+1}, \quad \forall i, \quad (25)$$

the desired bound is thus

$$E_{\max}[p(U)] < \sum_{i=1}^s \frac{1}{m+1} \left(1 - \frac{1}{m+1}\right)^m < \frac{s}{em}. \quad (26)$$

It should be clear here, that any probability distribution which is suitably normalized so that *all but one* of the terms in equation (24) achieves a maximum value, is a member of the class proposed in equation (15). For the important case of $m > s$, such a distribution can be used to find that the worst-mean error can be approximated to $O[(s/m)^m]$ as

$$E_{\max}[p(U)] \approx \frac{s-1}{em}. \quad (27)$$

If we set this expected error equal to ϵ , then we find that the distribution that gives rise to a worst-mean error equal to ϵ is

$$p(b_i) = \begin{cases} 1 - e\epsilon = 1 - \alpha_{\max} & \text{if } i = 1; \\ e\epsilon/(s-1) = \alpha_{\max}/(s-1) & \text{if } 1 < i \leq s. \end{cases} \quad (28)$$

The distribution that gives the worst-mean error and the one that gives the worst-mean sample complexity are thus equivalent, as expected. This distribution is also a member of the class proposed in equation (15). Its usefulness in providing fairly tight lower bounds on sample complexity and, therefore, its suitability for testing the random set approach to be developed, must now be established.

3.2. Empirical tests

Using the *hard* distributions defined by equation (15), a simple computational algorithm was written to test equation (27) and also estimate L_m for this class given s , ϵ , and δ . A sample problem with $T = \phi$, the null set, was chosen as the hardest urn problem target for this algorithm to learn from negative examples. The algorithm uses stochastic samples from $p(x)$ to find L_m from its definition as the smallest value of m which satisfies the condition that the remaining probability for success be more than ϵ with confidence less than δ . The program steps through m and α values and integrates the actual distribution to arrive at its results. For each α value, each m -sample is repeated several thousand times to get adequate statistics (i.e., accuracy to within a few percent).

Results for several selected urn problems with balls of s different types and $|C| = 2^s$ are given in Tables 1 and 2. They cover a range of s values from 16 to 256 and a large case of 2048. These numbers are suitable for use in k -DNF Boolean problems with n -attribute strings and k values from 4 to 8 and 11. Both tables give the bounds achieved for a target error of $\epsilon = 0.1$. Since the maximum sample sizes for all runs were achieved at α values within a few percent of $\alpha_{\max}$, the primary reported results are for this latter value. The real maximum (occurring at a slightly different α value) is given in parenthesis. A Blumer bound m_B with $|C| = 2^s$ for each case is also given for comparison purposes. Results in Table 1 were generated using a confidence value of $\delta = 0.5$, which effectively defines a median error. As such, these results are a reasonable test of the analytic L_{E_m} defined by equation (27). The results in Table 2 used a confidence value of $\delta = 0.1$ and they, therefore, provide a better test of Blumer's bounds. The L_{E_m} bounds in this table are not strictly applicable for this confidence limit, but are listed anyway for comparison purposes.

Table 1. Bounds for urn problems with $\epsilon = 0.1$ and $\delta = 0.5$.

s	L_m	L_{E_m}	m_B
16	55 (60)	55	118
32	113 (119)	114	229
64	227 (237)	232	451
128	467 (472)	468	894
256	942 (945)	939	1,781
2048	7,542 (7,557)	7,532	14,203

Table 2. Bounds for urn problems with $\epsilon = 0.1$ and $\delta = 0.1$.

s	L_m	L_{E_m}	m_B
16	79 (88)	55	134
32	147 (154)	114	245
64	280 (289)	232	467
128	536 (538)	468	910
256	1,037 (1,042)	939	1,797
2048	7,953 (7,965)	7,532	14,219

The results in Table 1 show clearly that the worst-mean errors computed from equation (27) are an excellent approximation to the $\delta = 0.5$ bound for the worst distribution in the class. Even in the worst case ($s = 16$), the largest sample size, 60 achieved at $\alpha = 0.250$, is still within 10% of the worst-mean value of 55 at $\alpha = 0.2718$. The analytic expression for L_{E_m} should, therefore, provide useful information in practical problem solving. Since the Blumer's results are about a factor of two higher than these estimated bounds in all of these cases, L_{E_m} appears to be a useful lower bound to sample complexity.

The Table 2 results provide a good direct comparison test of Blumer's bound. The actual sample complexity and the worst-mean distribution results are again very close and just about a factor of two different from the Blumer's upper bound. The largest difference in the sample size at its maximum from the one at $\alpha_{\max}$ was again found at $s = 16$. Here, a result of 88 was achieved at $\alpha = 0.250$ compared with a value of 79 for the worst-mean distribution.

What one can conclude from these comparisons is that Blumer's bound is only about a factor of two higher than the actual maximum sample sizes generated from the worst-mean distribution. In this sense, the class of distributions proposed in equation (15) clearly gives rise to *hard* learning problems. The worst distribution in this class produces a lower bound on sample complexity which is fairly close to the established upper bound. In addition, this class of distributions certainly yields the hardest case in the worst-mean error sense. These results thus establish the class as being capable of generating some of the most difficult sampling distributions for testing the random set methodology.

3.3. Practical bounds

Despite being able to provide analytic mean-error bounds, in actual practice many problems can be expected to require much smaller numbers of instances to meet learnability goals because of simpler distributions of learning examples. The usefulness of worst-case bounds in these cases is, therefore, limited. Although the bounds for such cases are problem-specific and not distribution-free, as intended by Valiant and others, it is important to have a general method for finding specific bounds for any learning problem. Even more important is being able to estimate learnability incrementally (i.e., as an algorithm is actually learning). Some of the ideas proposed by Linial, et al., (1988), introducing the concept of dynamic sampling, form the basis for such a problem-specific incremental (psi) approach. However, the Bernoulli trials framework and the results derived in Section 3.1 can be used to construct a more general variation on this dynamic sampling theme.

The basis for a random set psi-algorithm, is the fact that after the $(j - 1)^{\text{st}}$ success, the probability $h(p_j)$ of having a string of m failures in future selections from any probability distribution is

$$h(p_j) \equiv (1 - p_j)^m. \quad (29)$$

If the remaining success probability p_j were *actually* ϵ then this m -failure probability would be $h(\epsilon)$. In this latter case, if no new successes were recorded in m additional trials and the selection process were halted at this point, the final remaining success probability would be ϵ .

Clearly then, $h(\epsilon)$ represents the probability of halting with remaining success probability ϵ and this probability is, according to equation (29), less than $h(\epsilon)$ if $p_j > \epsilon$. This fact allows us to find the number of additional instances needed to ensure the convergence criteria that the probability that the remaining success probability be greater than ϵ will be less than δ . This number is found to be

$$m = \frac{\ln \left(\frac{1}{\delta} \right)}{\ln \left(\frac{1}{1 - \epsilon} \right)} \approx \frac{1}{\epsilon} \ln \left(\frac{1}{\delta} \right). \quad (30)$$

Using this test, if no additional successes are found in m future selections, the selection process can be halted with the assurance that both the ϵ and δ conditions for convergence of this learning algorithm would be met.

To implement this halting test, however, we need to address one further concern. That is, the test is not intended to be used only once in the algorithm. It must be used in sequence after each batch of m instances is sampled. The probability of passing the test after a whole sequence of checks needs to be less than δ if the desired convergence criteria is to be achieved for the whole algorithm. This added problem can be dealt with in an incremental way as well, by changing the batch size at each stage to insure convergence.

A simple justification of such a class of modified halting conditions can also be developed using the Bernoulli trials framework. In such an analysis, we see that the probability Q_J of halting after applying a series of J probabilistic halting tests h_j is given by

$$Q_J = \sum_{j=1}^J h_j \prod_{i=0}^{j-1} (1 - h_i) \quad \text{with } h_j \equiv (1 - p_j)^{m_j} \quad \text{and } h_0 = 0. \quad (31)$$

Here, p_j is the success probability and m_j is the now variable batch size at the j -th trial, $J + 1$ denotes the trial at which $p_{J+1} \leq \epsilon$ and since $p_0 \equiv 1$, we have $h_0 = 0$. The summation in this expression is carried out over the region in which $p_j > \epsilon$ in order to obtain the probability of halting with a remaining success probability greater than ϵ .

If we now define a halting test for variable batch size m_j , parameterized by η and β , as

$$h_j = (1 - \epsilon)^{m_j} \equiv \left(\frac{\eta}{j} \right)^\beta \quad \text{with } \beta > 1, \quad (32)$$

we see that Q_J can be bounded from above by

$$Q_J < \sum_{j=1}^J \left(\frac{\eta}{j} \right)^\beta < \int_{1/2}^{J+1/2} \left(\frac{\eta}{x} \right)^\beta dx, \quad (33)$$

which gives rise to the convergence criteria

$$Q_J < \frac{\eta^\beta}{\beta - 1} \left(2^{\beta-1} - \frac{1}{(J + 1/2)^{\beta-1}} \right) = \delta. \quad (34)$$

The largest batch size m_j which just achieves this confidence criteria is found by letting $J \rightarrow \infty$. Solving for η in this case and substituting it back into equation (32) gives,

$$m_j \approx \frac{1}{\epsilon} \ln \left(\frac{2^{\beta-1} j^\beta}{(\beta - 1) \delta} \right). \quad (35)$$

This result with $\beta = 2$ was previously derived by Lineal, et al., (1988).

By taking the derivative of m_j with respect to β , this general form can be seen to give minimum batch sizes as a function of β when

$$\beta = 1 + \left(\frac{1}{\ln(2j)} \right), \quad (36)$$

This relation makes β a particularly useful control parameter in a psi-algorithm. Thus, for discrete space problems where J can be estimated, the batch size can be minimized in the region $j = J$ of expected convergence.

For problems with large J , it is useful to make β as close as possible to unity, giving rise to the following simplified test

$$m_j = \frac{1}{\epsilon} \ln \left(\frac{j}{\delta^*} \right) \quad \text{for } \beta \approx 1, \quad (37)$$

where $\delta^* = (\beta - 1)\delta/2^{\beta-1}$.

In practice then, equation (35) can be used to define a general halting test batch size that achieves convergence for each specific learning problem encountered. Any algorithm which uses such an approach will, effectively, be a batchwise, incremental learning scheme (a psi-algorithm) which guarantees learnability when it halts.

3.4. Implementation

To implement this formalism in a random set psi-algorithm, two halting tests are required: 1) a test on unseen categories and 2) a test on performance. The first test uses a halting criteria with success defined in terms of sampling a new category in a sequence of Bernoulli trials. This test, based on the random hypothesis set of unseen categories $H = U$, will be needed to converge the algorithm at any level l . The second test defines success in terms of performance error and is based on the random error set R . This version will be used to determine whether a finer partition of the space X is needed to meet the performance goal of the algorithm (i.e., whether additional levels need to be searched).

Together, both these tests make it unnecessary to specify that a target set T be a member of the class C_k being searched. If the target set is a member of C_k , the performance test will halt the algorithm at the appropriate level l . If it is not, the algorithm will halt at the

Table 3. Halting results for worst-mean distribution ($\epsilon = 0.1$, $\delta = 0.5$).

s	L_m	$\bar{m}$	$\bar{\epsilon}$	$\bar{\delta}$
16	55	163	0.013	0.015
32	113	282	0.022	0.015
64	227	527	0.026	0.010
128	467	986	0.033	0.025
256	942	1,872	0.037	0.010
2048	7,542	13,857	0.046	0.005

Table 4. Halting results for uniform distribution ($\epsilon = 0.1$, $\delta = 0.5$).

s	L_m	$\bar{m}$	$\bar{\epsilon}$	$\bar{\delta}$
16	34	99	0.001	0.010
32	69	169	0.005	0.005
64	143	292	0.007	0.010
128	297	514	0.018	0.005
256	588	987	0.027	0.005
2048	4,728	7,075	0.033	0.005

maximum specified level k having converged on unseen categories. This latter case gives an approximation to $T \in C_k$ which does not meet the performance goal (though it is the best the algorithm can do within this representation class). To test this halting scheme under the difficult circumstances that can be expected in practice, we give in Tables 3 and 4 some representative results of using this approach for urn problems with examples drawn from the class of distributions given in equation (15). These tables again cover a range of s values from 16 to 256 and a large case of 2048. They were generated using equation (37) with $\beta = 1.2$, $\epsilon = 0.1$ and $\delta^* = 0.087$ ($\delta = 0.5$). The initial batch size was $m_1 = 25$ for these parameter limits.

The results in the tables all represent averages over 200 repeated runs to convergence as determined by the halting criteria. For each set of repeated runs until the algorithm halted, an average sample size $\bar{m}$ was calculated. Using the known probability distribution, an average error $\bar{\epsilon}$ was also computed along with a record of the number of times the ϵ criteria was violated, denoted by $\bar{\delta}$. These results are presented along with a stochastic estimate of L_m for comparison purposes. The estimated L_m was generated with the same program that was used to get the results presented in Tables 1 and 2. The two cases considered in these tables are a worst-mean case distribution scenario (Table 3) and a simpler uniform distribution learning problem (Table 4).

The results given in Table 3 show clearly that approximations to within a factor of two of the L_m for the worst-mean case distribution (i.e., $\alpha = e\epsilon$) are obtained using the halting scheme. In general the halting condition requires larger numbers of instances than are actually needed, but the errors and confidence values they produce are almost an order of magnitude better. This result is primarily due to the upper bound approach used to derive equation (35).

For the easier uniform distribution problem given in Table 4, the mean-error values have clearly decreased substantially. These values are again within a factor of two of the stochastically estimated values of L_m . Due to the conservative nature of the halting condition, we again see that this approach produces order of magnitude better convergence. It should be clear from these results that only a psi-approach to bound estimation can take advantage of this easier learnability to achieve converge with sample sizes smaller than those required by worst-case estimates. Since even a uniform distribution does not result in an easy learning problem, much more dramatic improvements can be expected in practice for much simpler distributions.

What one should conclude from these discussions, is that bounds on sample complexity can greatly overestimate the sample sizes needed in practice. Such bounds, while theoretically useful for guaranteeing the existence of learnability (an essential consideration), are impractical for use in actual pac-learning algorithms. It appears that a much more pragmatic approach to computer implementation of Valiant's learnability theory is the use of the halting approach in a psi-algorithm. This is indeed the course taken in the GPAC algorithm presented next.

4. Random set algorithm

To demonstrate the usefulness of the random set formulation of Valiant's theory, we chose to implement Valiant's constructive proof of learnability as an exhaustive search of the sets in C_k . This specific algorithm is by no means the best implementation of the random set approach, since no use of heuristics will be employed. But, as an exhaustive search, it should serve as a very stringent test of the overall methodology. In fact, what we intend to show is that this inefficient implementation using halting tests, is already a practical alternative to other more conventional approaches which use heuristics quite effectively. This algorithm can easily be parallelized for use in advanced computing environments and is subject to a host of improvements using information-theoretic, stochastic, and search space pruning heuristics.

4.1. GPAC algorithm

The random set algorithm developed is a C-language implementation of Valiant's original constructive proof of learnability for k -DNF's. This proof has been generalized to search the space C_k . The algorithm, called GPAC, uses halting tests to achieve convergence and, therefore, represents a psi-approach to classification learning.

In the general code, the instances $x \in X$ are the n -attribute instance vectors with arbitrary numbers of values per attribute described in Section 2.1. The search space consists of all basic categories b_{vj}^l of the class C_k , which has proven learnability in Valiant's formalism. Instances from X are generated randomly from $p(x)$ and are labelled T and called positive instances if $x \in T$ and labelled $\bar{T}$ and called negative instances if $x \in \bar{T}$. This labelling is done by an *oracle* subroutine.

Two initial hypothesis sets H are defined and searched simultaneously. The first is generated by labelling all basic categories b_{vj}^l with the target label T and the second by labelling

all basic categories with the complement $\tilde{T}$. As learning instances are chosen, the algorithm updates both of these hypothesis spaces by eliminating all categories with labels that are inconsistent with the instances chosen. The updating phase of this algorithm is carried out within a series of $\binom{n}{l}$ subproblems corresponding to the j -partitions at each successive level $l \leq k$. Each of these subproblems is solved independently (in parallel if desired) using the same set of randomly chosen instances. The category structure of these partitions are those defined previously in equation (8).

Learning is accomplished using a scheme similar to the one described in Valiant (1985). This amounts to finding all categories consistent with all the instances seen so far. Both T and $\tilde{T}$ are learned simultaneously from counterexamples. Each of these learning tasks invokes the following procedures: 1) select random instances x from either $p_+(x)$, $p_-(x)$ or $p(x)$, 2) invoke an oracle to label each instance with either a T or a $\tilde{T}$, 3) find all the categories which the chosen instances have membership in, and 4) eliminate from the labelled hypothesis space any such categories that are inconsistent with the labels of the instances chosen. The code reports results for the set of instances (positive or negative) which first satisfies both halting tests. This set is used to create an approximation to T (or $\tilde{T}$, as the case may be).

The resulting C-language algorithm is given schematically as follows:

```

LEARN ( ) {
  while ( $l \leq k$ ) {
    s: Select  $m_j$  instances from examples( $x$ );
    while ( $j \leq \binom{n}{l}$ ) {
      Put instances into hypothesis categories  $b_{vj}^l$ ;
      Eliminate inconsistent categories from  $H$ ;
      Check for new unseen categories;
      Increment  $j$ ;
    }
    Check halt_conditions ( $h_U$ ,  $h_R$ ,  $\epsilon$ ,  $\delta$ );
    if ( $h_U = \text{false}$ )
      goto s;
    elseif ( $h_R = \text{false}$ )
      Increment  $l$ ;
    }
     $H = \cup \text{consistent\_categories}(b_{vj}^l)$ ;
    Output  $H$ ;
  }
}

```

Here, the basic subproblems given in equation (11) are solved in a loop over levels up to $l = k$. At each level, the batch of m_j instances defined in equation (35) with a chosen β is partitioned into categories b_{vj}^l . Each category that contains instances in the batch that are inconsistent with the oracle's label are recorded for elimination from the hypothesis set H . No comparisons are needed in partitioning a batch in this procedure. Only a category address for each instance needs to be calculated using full integer word arithmetic (i.e., no bit-string algebra shortcuts are used). The elimination step in this particular implementation

is immediate removal from the hypothesis set once an inconsistent instance is found. We have thus assumed a noise-free, perfect learning mode. This specialization can, however, be easily removed since both the number of positive and negative instances in each basic category are available for more sophisticated elimination criteria.

After all m_j instances have been processed and any inconsistent categories in the hypothesis set(s) have been eliminated for all subproblems at a given level l , the halting condition for unseen categories is tested. In this test, if no new (inconsistent) categories were seen in the m_j samples, then the halting condition given by equations (32) and (35) is considered to be satisfied for unseen categories (i.e., the probability of finding another inconsistent category is less than ϵ with confidence δ). If the halting condition is not satisfied, a new batch of m_j instances is chosen and the entire learning procedure at the current level is repeated.

If the unseen category halting test is satisfied in all the subproblems at the current level, then the performance halting condition is tested. In this test, if none of the m_j samples were misclassified, then the halting condition given by equations (32) and (35) is considered to be satisfied for performance (i.e., the probability of making a classification error is less than ϵ with confidence δ). If this test fails, then learning at the current level is terminated and the next level is processed. When the performance halting condition has been satisfied at some level $l < k$ or $l = k$ (whichever comes first), the hypothesis consisting of the union of all remaining consistent categories is output. The algorithm will thus solve the perfect learner problem if T (or $\tilde{T}$) is contained in C_k . Otherwise, it will terminate with some approximation to this solution (which does not satisfy the performance criteria) within C_k .

In the series of c_j^l subproblems, a scoring table, which records the number of instances n_ν which are members of each category $b_\nu \equiv b_{\nu j}^l$, will be generated. This table is used by a consistency testing routine to eliminate categories from the hypothesis set H . A typical perfect learner table will have the form shown in Figure 1.

In Valiant's terms, the top row of this figure represents a partition of the class C_k into categories b_ν which have label T . This row serves as a starting consistent hypothesis set for categories labelled T to be used with negative instances. A similar interpretation can be made for the bottom row in terms of a starting hypothesis set for categories labelled $\tilde{T}$ to be used with positive instances.

In this framework, each instance in a batch will be a counterexample of some category b_ν in one of the hypothesis sets and can thereby be eliminated from the appropriate consistent hypothesis set of categories. There will be only one such category in each partition

	b_1	b_2	b_3	$\dots$	$\dots$	b_ν
T	n_1	0	0	$\dots$	$\dots$	n_ν
$\tilde{T}$	0	n_2	n_3	$\dots$	$\dots$	0

Figure 1. Hypothesis categories for C_k .

for each instance. The union of all the categories b_i , having no counterexamples will then be the random set $H = U$ of unselected categories. As such, this set represents a maximal covering approximation to the target category T or $\bar{T}$ derived from counterexamples (be they negative or positive instances, respectively). For the Boolean problem considered by Valiant, this procedure can be considered equivalent to learning a k -DNF (or k -CNF) from negative (or positive) instances.

Both of these procedures are carried out together if both positive and negative instances are present. Since some set of these labelled columns are complements of each other at appropriate levels for the perfect learner problems we are considering, only one of these sets of instances is needed to completely specify the target (or its complement). The algorithm chooses the hypothesis set that converges first as its representation of the solution. The algorithm can (and does) use both sets, but the halting test will be satisfied first by only one of them. Only this resulting set of instances and its consistent hypothesis set needs to be reported.

During the final stages of the GPAC algorithm at any level l , all the consistent categories forming the final hypothesis H are checked to see whether any redundant categories exist. At this point, if any $b_{ij}^l \subset b_{ij}^{l'}$ for $l' < l$ can be identified, then these redundant subsets are eliminated from H and any future processing. This latter procedure allows a minimal representation to be found in set form and reduces storage requirements considerably.

Using both halting tests for the whole learning procedure up to $l = k$, we find that the algorithm either halts and returns the approximate category H given by the union of all unique consistent categories or it halts on a specified maximum k . At each stage in this process, the algorithm uses at most $O(m_j n^k)$ operations to partition and test all m_j instances.

In this form, the algorithm is equivalent to an exhaustive search of the category space for all the b_{ij}^l categories that are consistent with the instance data. In this manner, it is similar to the version space approach of Mitchell (1982) for finding a consistent hypothesis set and to Valiant's constructive proof of learnability for Boolean problems. It is also equivalent, in the decision tree format, to finding *all* consistent trees, not just the one with the highest information content. The practicality of such an exhaustive search scheme, compared with methodologies that limit search using heuristics, is what we must establish next.

5. Multiplexor problem

To test the general algorithm just described, a classification benchmark problem studied by Quinlan (1988), Wilson (1987), and Pagallo & Haussler (1990) was used. This example provides a somewhat difficult nonlinear Boolean function which must be learned in order to classify all instances. The function is nonlinear in that no single hyperplane separates all positive and negative instances. It can be computationally difficult to learn since it is formulated in terms of a parameter which scales up quickly enough to easily tax the powers of even the largest supercomputers. It certainly provides an extreme test of an exhaustive search algorithm since the rule is deep and no heuristics are used to eliminate portions of the search space.

5.1. Problem definition

The so-called *multiplexor* family of problems which Wilson (1987) described, can be generically defined by a Boolean function $f(x)$ which embodies the rule

$$f(x) = \begin{cases} 0 & \text{if } a_{r+j+1} = 0; \\ 1 & \text{if } a_{r+j+1} = 1, \end{cases} \quad \text{for } j \in \{0, 1, \dots, 2^r - 1\} \quad (38)$$

where the a_i 's are the attributes of an n -attribute binary bit string x and $n = r + 2^r$ for a given fixed value of r .

In this form, the instances x from which this function must be learned define two target categories *negative* (0) and *positive* (1), corresponding to the function values $f \in \{0, 1\}$. The first r bits of these instances represent a binary address $j \in \{0, 1, \dots, 2^r - 1\}$ which acts as pointer to the $(r + j + 1)^{\text{th}}$ bit of the instance. The class the instance belongs to is determined by the value of this addressed bit. That is, the Boolean function value is taken to be the value of the $(r + j + 1)^{\text{th}}$ bit. For fixed r , this scheme uses instances represented by a bit-string of total length $n = r + 2^r$. The order of the multiplexor class of problems is then denoted by this parameterized length n and each member of the class is called an *n-multiplexor*.

This class of learning problems has been explored by both Quinlan (1987) and Pagallo & Haussler (1990) using a decision tree formalism (DT) and Wilson (1987) using a genetic algorithm (GA) classifier system. In Quinlan's DT methodology, the instances containing representatives of both positive and negative members of a multiplexor class are subdivided recursively by attribute until all subsets of the class contain only instances of a single type. The record of these subdivisions is stored as a decision tree which can be used to classify any new instances encountered (Quinlan 1986). To avoid an exhaustive tree search, Quinlan's methodology uses an information-theoretic measure to prune the search tree.

The code Quinlan used in his studies was *C4* (Quinlan 1987), which is a member of a larger class of decision tree codes termed the TDIDT family (Quinlan 1986). The *C4* version of this code class allows production rules to be generated from a decision tree by removing all redundant information from the tree. Each rule corresponds to a single term in a k -DNF representation of the Boolean function to be learned. Pagallo & Haussler use a variation of this algorithm which uses Boolean conjuncts as nodes in the tree, not just single attributes. This improves the performance of a DT algorithm for Boolean problems by eliminating the unnecessary subdivision of the sample space into very small subsets. Several algorithms are proposed by Pagallo along these lines, each using different heuristics to improve performance.

In Wilson's GA classifier scheme, the rules representing sets of instances and their class are learned directly using the genetic learning model of Holland (1986). Here, an initial set of rules is tested recursively against the instances seen. This results in probabilistic weights being assigned to each rule based on its classification performance. These rules are then modified by the genetic heuristic of eliminating those with low weights and forming new ones by *mutation* and *crossover* operations similar to those found in biologically reproducing systems. The resulting set of rules learned are thus those with the highest classification performance for the learning instances seen.

In their published results, Quinlan and Wilson both report the total number of instances needed to learn the classification rules of each of the standard multiplexor problems to various error levels. Pagallo & Haussler report numbers for only a single set of instances whose size was computed from worst-case learning arguments. They solve a non-standard class of multiplexors, constructed by adding additional redundant attributes to the standard problems. The instances in all these published cases were chosen randomly from a uniform distribution of n -attribute vectors. For comparison purposes then, we will also use uniform distributions of instances to test the random set approach.

5.2. Multiplexor results

Tables 5 and 6 summarize the results of several typical runs of the GPAC code for standard multiplexor problems of varying complexity. The similarity of the rules in either a DNF representation of T or $\bar{T}$, together with the use of a uniform distribution, allowed GPAC to solve the problem equally well using either positive or negative instances alone. Since statistically insignificant differences were found between the results of using either set of instances, only the negative instance results, corresponding to a k -DNF solution for T , were reported. In addition, since all the multiplexor problems studied were members of the class C_k , with $k = r + 1$, the GPAC algorithm always converged on performance at level $l = r + 1$ using the halting criteria approach. Due to the conservative nature of these tests, some of the results found the exact representations of the target functions in C_k .

For comparison purposes, performance statistics testing converged results were generated from a batch of 5000 additional randomly drawn instances. These statistics were recorded at various batch stages prior to reaching the tightest convergence criteria. Similar results for the GA and DT analyses were generated by Wilson and Quinlan and are thus tabulated in Tables 5 and 6 for comparison purposes. Both positive and negative instances were used, however, in the learning modes for these latter two methodologies. The sample sizes needed to get their performance levels reflect this difference with the random set approach. DT results are, in addition, given for both the simple decision tree algorithm and the more elaborate rule generation approach. Table 5 summarizes the comparisons for an 11-multiplexor problem while Table 6 covers the more complex 20-multiplexor.

Tables 7 and 8 present comparisons for the augmented 6- and 11-multiplexor problems studied by Pagallo & Haussler (1990). Both of these problems were solved using a uniform distribution of positive and negative examples. In the 6-multiplexor case, the attribute vector

Table 5. Results for 11-multiplexor

Method-Computer	Examples	Accuracy %
C4-Sun 3/50	100	63 (72)
Decision tree (Rule)	200	78 (98)
± examples	400	93 (100)
GPAC-286/PC	107	83
– examples	187	98
	219	100

Table 6 Results for 20-multiplexor

Method-Computer	Examples	Accuracy (%)	Time (min)
GA-Sun 3/50 ± examples	70,000	90	55
C4-Sun 3/50	200	69 (69)	n.a.
Decision tree (Rule)	400	82 (88)	n.a.
± examples	600	87 (97)	n.a.
	800	92 (98)	13 (63)
GPAC-286/PC	361	79	10
– examples	369	88	12
	449	98	17
	638	100	20

Table 7. Results for augmented 6-multiplexor

Method	Examples	Accuracy (%)
FRINGE-first(last)	480 (720)	100 (100)
GREEDY3-no prune (prune)	480 (720)	99.6 (100)
GROVE-no prune (prune)	480 (720)	99.1 (98.8)
GPAC-negative examples	118	98
	165	100

Table 8. Results for augmented 11-multiplexor

Method	Examples	Accuracy (%)
FRINGE-first(last)	1,067 (1,600)	86.9 (100)
GREEDY3-no prune (prune)	1,067 (1,600)	98.7 (99.5)
GROVE-no prune (prune)	1,067 (1,600)	96.6 (96.1)
GPAC-negative examples	251	94
	357	100

had 58 added redundant attributes (for a total of 64). (Note here that the Pagallo paper misprinted this value as 16). The 11-multiplexor was augmented with 21 redundant attributes (for a total of 32). Random set results for both these cases were generated using a uniform distribution. Negative example results alone were reported because of the insignificant differences with positive only results. Comparisons were made with three of Pagallo's algorithm. In each case an initial and final result are presented. The initial case either reflected a first iteration (for FRINGE), or a DT that had no pruning (for the other two methods). The last case, listed in parenthesis, reflected the last iteration (for FRINGE), or a DT that had been pruned (for the other two methods).

5.3. Discussion of results

The results in Table 5 show that the C4 methodology and GPAC produce comparable results for similar sample sizes. C4 requires about 200 positive and negative instances to find a decision tree with 78% predictive accuracy. Further pruning of this tree gives rise to production rules that increase the accuracy to 98%. A comparable GPAC sample size of 187 negative instances also gives 98% predictive accuracy. An increase in the sample size to 219 allows the GPAC methodology to achieve a perfect 100% accuracy.

The more complete set of comparisons given in Table 6 show again that C4 and GPAC achieve comparable accuracies with sample sizes that are within a factor of two of each other. For this 20-multiplexor case, C4 needs about 800 positive and negative instances to produce a tree which has predictive capabilities of about 92%. Pruning to obtain rules again increases the predictive capabilities to about 98%. Similar results are achieved with about 449 negative instances using GPAC. The GA algorithm of Wilson on the other hand needs about 70,000 instances to produce about 90% accuracy.

Comparisons with Pagallo and Haussler's augmented 6- and 11-multiplexor results given in Tables 7 and 8 show similar trends to those found for Quinlan's DT case. Although the three algorithms Pagallo and Haussler propose give excellent results for reasonably small sample sizes, the GPAC runs for this uniform distribution case required smaller numbers of samples. This is the case even after eliminating the factor of two advantages GPAC has by using only negative instances. Performance statistics in these cases are comparable as well.

5.4. Algorithmic efficiency

As far as sample size comparisons are concerned, the GPAC results appear to give numbers which are comparable to or smaller than those generated by the other methodologies. The only advantage of the GPA approach highlighted here is the ability to learn from positive or negative instances separately. This capability could prove useful in practice. The most important differences observed in these test problems, however, was related to algorithmic efficiency. In this area the exhaustive GPAC search seems to be exceptionally efficient in comparison with the other methodologies, despite their use of proven heuristics to prune the search space.

The timing estimates provided by Quinlan (1987) for the 20-multiplexor problem (given in Table 6), indicate that in order to achieve comparable 90% accuracies, Wilson's GA requires about one hour and a C4 decision tree requires about 13 minutes of Sun 3/50 cpu time. To increase the accuracy of the C4 results to about 98% with tree pruning, requires an increase in run time by about a factor of five. This latter timing was estimated from the results Quinlan reported for the 6-multiplexor (not shown). GPAC appears to be much faster in these comparisons. Accuracy of better than 98% is achieved in 17 minutes on a 286/PC compared to the estimated 70 minutes on the Sun 3/50 for a C4 rule based run. Since the 286/PC is about a factor of 3 slower than the Sun 3/50 used by Quinlan, the comparable GPAC results thus appear to require possibly an order a magnitude less time.

While these timing comparisons are somewhat rough, it still appears that GPAC has computational advantages over the already well established speed and versatility of the TDIDT

class of codes and Wilson's classifier. This result is achieved *despite* the exhaustive nature of the GPAC algorithm and the established heuristics used in the other approaches. Some of this advantage is surely a result of GPAC requiring only negative instances as opposed to the use of both positive and negative instances in C4 and the GA. Since a uniform distribution was used in these problems, half the instances were positive and the other half were negative. Although a factor of two differences might explain some of the GPAC speed increase, more of it is a result of the inherent efficiency of the partition structure used. No function evaluations or comparison steps are needed to place an instance into a category.

In comparing computational efficiencies with the faster DT methodology, it should also be pointed out that while the DT scheme produces rules from a single decision tree, GPAC produces the rules from *all* trees as its solution. The GPAC algorithm again uses an exhaustive search of all possible rules to produce ones that are consistent with the learning instances seen. For example, C4 generates an exact set of 8 rules to learn the 11-multiplexor with 100% accuracy. GPAC produces 21 rules comprising all trees for this same problem and accuracy. Some of these alternative but somewhat redundant rules were not even anticipated before the problem was solved. The exhaustive search in GPAC is again seen to be quite efficient using no heuristics. This efficiency should prove quite useful in generating heuristic versions of GPAC for harder problems which are not amenable to exhaustive search.

As a final note, we must be fair to Wilson's GA classifier by pointing out that the genetic algorithm is iterative in nature, in that it cannot immediately generate a proper classification for instances it has already seen. Proper classification is only achieved after having seen the same instance many times. This approach is better suited for problems which changing classifications over time (an evolutionary approach) or noisy sample problems. A strictly comparable problem for all three methodologies would thus involve a noisy environment. In this case we might see an order of magnitude more instances required by GPAC and C4 to achieve comparable performance levels. Some of the GA classifier's poorer performance might be overcome in such a comparison. Also, the multiplexor problem class has been studied using improved classifier schemes (see e.g., Parodi, et al., 1990) with much improved results (although still not nearly as good as those reported using GPAC or DTs). In any event, the GA appears to be fairly efficient, considering the number of instances it must see to learn. Sample complexity appears to be its weakness.

6. Conclusions

The major conclusion that can be drawn from this work is that the implementation of Valiant's theory in random set terms is both computationally feasible and competitive with other existing approaches. The exhaustive search version of this methodology, which might have been its weakness, does not seem to hurt its practical problem solving abilities. GPAC should thus be considered to be an extremely efficient implementation of the random set pac-learning methodology—thus realizing one of the major objectives of our research. Scaling this approach up for larger problems will clearly require heuristics, since even the 20-multiplexor already taxes GPAC's computational resources on a PC to their fullest. The random set methodology, however, is amenable to heuristics and such an extension poses no significant limitation to its applicability.

Several additional benefits of the random set approach were also highlighted in this paper.

- We found that the convergence criteria required by pac-learning theory can be estimated incrementally to establish a psi-approach to learning. That is, the halting tests proposed allow bounds to be determined in a problem-specific incremental (psi) manner, as opposed to being analytically estimated from worst-case or distribution-free arguments. This improves practical problem solving capabilities by orders-of-magnitude in many cases while still retaining theoretical rigor.
- Running times for GPAC were found to be either shorter than (or certainly comparable to) those of the other methods tested even without much in the way of C-code optimization. Future implementations should allow significant improvements in both speed and storage requirements, making this methodology quite competitive in practical learning problems even with an exhaustive search format.
- The Bernoulli trials analysis of random set success probabilities allowed us to derive both a halting test to assure algorithmic convergence and a hard class of probability distributions to test this scheme. This hard class of distributions allowed stringent tests to be made of the psi-approach to convergence by providing slowly decreasing success rates. Since these tests pushed the halting test to its limits, success here indicates that this approach is well suited for solving more practical problems with simpler distributions.

Acknowledgments

The author wishes to thank Oscar Manley of the Department of Energy's Office of Basic Energy Sciences for his continuing financial support of random set research.

References

- Angluin, D. (1988). Queries and concept learning. *Machine Learning*, 2, 319-342.
- Baum, E.B. & Haussler, D. (1989). What size net gives valid generalization? *Neural Computation*, 1, 151-160.
- Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M.K. (1989). Learnability and the Vapnik-Chervonenkis dimension. *JACM*, 6, 929-965.
- Duda, R.O. & Hart, P.E. (1973). *Pattern classification and scene analysis*. New York: John Wiley & Sons.
- Feller, W. (1968). *An introduction to probability theory and its applications*, Volume I, Third Edition. New York: John Wiley & Sons.
- Haussler, D. (1988a). Quantifying inductive bias. *Artificial Intelligence*, 36, 177-221.
- Haussler, D., et al., (1988b). Predicting 0-1-functions on randomly drawn points. *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*.
- Haussler, D. & Pitt, L. (Eds.) (1988). *Proceedings of the 1988 Workshop on Computational Learning Theory*. Boston, MA: Morgan Kaufmann.
- Holland, J.H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.
- Holland, J.H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R.S. Michalski, et al., (Eds.), *Machine learning: An artificial intelligence approach*. Los Altos, CA: Morgan Kaufmann.
- Kendall, D.G. (1974). Foundations of a theory of random sets. In E.F. Harding, & D.G., Kendall, (Eds.), *Stochastic geometry*. New York: John Wiley & Sons.

- Linial, N., Mansour, Y., & Rivest, R.L. (1988). Results on learnability and the Vapnik-Chervonenkis dimension. *Proceedings of the 1988 Workshop on Computational Learning Theory* (pp. 56–68). Boston, MA: Morgan Kaufmann.
- Mitchell, T.M. (1982). Generalization as search. *Artificial Intelligence*, 18, 203–226.
- Oblow, E.M. & Uppuluri, V.R.R. (1991). *Analysis of Complexity Bounds for pac-Learning with Random Sets* (ORNL/TM-11844). Oak Ridge, TN: Oak Ridge National Laboratory, Engineering Physics and Mathematics Division.
- Pagallo, G. & Haussler, D. (1990). Boolean feature discovery in empirical learning, *Machine Learning*, 5, 71–99.
- Parodi, A., Bonelli, P., Sen, S., & Wilson, S.W. (1990). Newboole: A fast GBML system. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 153–159). Palo Alto, CA: Morgan Kaufmann.
- Quinlan, J.R. (1979). Discovering rules by induction from large collections of examples. In D. Mitchie (Ed.), *Expert systems in the micro electronic age*. Edinburgh University Press.
- Quinlan, J.R. (1986). Induction of decision trees. *Machine Learning*, 1, 81–106.
- Quinlan, J.R. (1988). An empirical comparison of genetic and decision-tree classifiers. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 135–141). Ann Arbor, MI: Morgan Kaufmann.
- Rivest, R., Haussler, D., & Warmuth, M.K. (Eds.) (1989). *Proceedings of the 1989 Workshop on Computational Learning Theory*. Santa Cruz, CA: Morgan Kaufmann.
- Valiant, L.G. (1984). A theory of the learnable. *Comm. ACM*, 27, 1134–1142.
- Valiant, L.G. (1985). Learning disjunctions and conjunctions. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 560–566). Los Angeles, CA: Morgan Kaufmann.
- Wilson, S.W. (1987). Quasi-Darwinian learning in a classifier system. *Proceedings of the Fourth International Workshop on Machine Learning*. Irvine, CA: Morgan Kaufmann.