Report Number: WUCS-93-46

1993

# Learning Unions of Boxes with Membership and Equivalence Queries

Paul W. Goldberg, Sally A. Goldman, and H. David Matthias

We present two algorithms that use membership and equivalence queries to exactly identify the concepts given by the union of s discretized axis-parallel boxes in d-dimensional discretized Euclidean space where each coordinate can have n discrete values. The first algorithm receives at most s*d counterexamples and uses time and membership queries polynomial in s and logn for d any constant. Further, all equivalence queries made can be formulated as the union of O(s*d*log(s)) axis-parallel boxes. Next, we introduce a new complexity measure that better captures the complexity of a union of boxes than simply the number of boxes and... **Read complete abstract on page 2.**

# Learning Unions of Boxes with Membership and Equivalence Queries

Paul W. Goldberg, Sally A. Goldman, and H. David Matthias

Complete Abstract:

We present two algorithms that use membership and equivalence queries to exactly identify the concepts given by the union of s discretized axis-parallel boxes in d-dimensional discretized Euclidean space where each coordinate can have n discrete values. The first algorithm receives at most s*d counterexamples and uses time and membership queries polynomial in s and logn for d any constant. Further, all equivalence queries made can be formulated as the union of O(s*d*log(s)) axis-parallel boxes. Next, we introduce a new complexity measure that better captures the complexity of a union of boxes than simply the number of boxes and dimensions. Our new measure, [sigma], is the number of segments in the target polyhedron where a segment is a maximum portion of one of the sides of the polyhedron that lies entirely inside or entirely outside each of the other halfspaces defining the polyhedron. We then present an improvement of our first algorithm that uses time and queries polynomial in [sigma] and log(n). The hypothesis class used here is the decision trees of height at most 2*s*d. Further we can show that the time and queries used by this algorithm are polynomial in d and log(n) for s any constant thus generalizing the exact learnability of DNF formulas with a constant number of terms. In fact, this single algorithm is efficient for either s or d constant.

# Learning Unions of Boxes with Membership and Equivalence Queries

**Paul W. Goldberg, Sally A. Goldman and H. David Mathias**

**WUCS-93-46**

**revised May 1994**

**Department of Computer Science**
**Washington University**
**Campus Box 1045**
**One Brookings Drive**
**St. Louis MO 63130-4899**

***Supported in part by the U.S. Department of Energy under contract DE-AC04-76AL85000, NSF Grant CCR-9110108 and NSF NYI Grant CCR-9357707.***

# Learning Unions of Boxes with Membership and Equivalence Queries

Paul W. Goldberg*
Department 1423
Sandia National Laboratories, MS 1110
P.O. Box 5800
Albuquerque, NM 87185-1110
pwgoldb@cs.sandia.gov

Sally A. Goldman†
Dept. of Computer Science
Washington University
St. Louis, MO 63130
sg@cs.wustl.edu

H. David Mathias
Dept. of Computer Science
Washington University
St. Louis, MO 63130
dmath@cs.wustl.edu

May 20, 1994

**Abstract**

We present two algorithms that use membership and equivalence queries to exactly identify the concepts given by the union of $s$ discretized axis-parallel boxes in $d$-dimensional discretized Euclidean space where each coordinate can have $n$ discrete values. The first algorithm receives at most $sd$ counterexamples and uses time and membership queries polynomial in $s$ and $\log n$ for $d$ any constant. Further, all equivalence queries made can be formulated as the union of $O(sd \log s)$ axis-parallel boxes.

Next, we introduce a new complexity measure that better captures the complexity of a union of boxes than simply the number of boxes and dimensions. Our new measure, $\sigma$, is the number of segments in the target polyhedron where a segment is a maximum portion of one of the sides of the polyhedron that lies entirely inside or entirely outside each of the other halfspaces defining the polyhedron. We then present an improvement of our first algorithm that uses time and queries polynomial in $\sigma$ and $\log n$. The hypothesis class used here is decision trees of height at most $2sd$. Further we can show that the time and queries used by this algorithm are polynomial in $d$ and $\log n$ for $s$ any constant thus generalizing the exact learnability of DNF formulas with a constant number of terms. In fact, this single algorithm is efficient for either $s$ or $d$ constant.

*This research was performed while visiting Washington University. Currently supported by the U.S. Department of Energy under contract DE-AC04-76AL85000.

†Supported in part by NSF Grant CCR-9110108 and an NSF NYI Grant CCR-9357707.

# 1 Introduction

Recently, learning geometric concepts in $d$-dimensional Euclidean space has been the subject of much research. One such class of geometric concepts is unions of boxes. (By a "box", we mean an axis-aligned hypercuboid. So a box is the set of all points whose Cartesian coordinates satisfy a given set of univariate linear inequalities.) We study this problem under the model of learning with queries [1] in which the learner is required to output a final hypothesis that correctly classifies *every* point in the domain as to whether or not it is inside of one of the target boxes. To apply such a learning model to a domain such as learning boxes (or unions of boxes) in $d$-dimensional Euclidean space, it is necessary to look at a discretized (or digitalized) version of the domain. We use $\text{BOX}_n^d$ to denote the class of axis-parallel boxes over $\{1,\ldots,n\}^d$. So $d$ represents the number of dimensions and $n$ represents the number of discrete values that exist in each dimension. Let $[i,j]$ denote the set $\{m \in \mathbf{N} |\ i \leq m \leq j\}$. Then, $\text{BOX}_n^d = \{\times_{k=1}^d [i_k, j_k] \mid 1 \leq i_k \leq j_k \leq n\}$. So $i_k$ and $j_k$ are the minimum and maximum positive values of the $k$-th coordinate of a box. Note that by allowing equality of $i_k$ and $j_k$ we include in $\text{BOX}_n^d$ boxes with zero size in dimension $k$. Finally, let $\bigcup_s \text{BOX}_n^d$ denote the class of the union of at most $s$ concepts from $\text{BOX}_n^d$. We note that it is easy to show that this class is a generalization of disjunctive normal form (DNF) formulas and a special case of the class of unions of intersections of half-spaces over $\{1,\ldots,n\}^d$.

In this paper, we first present an algorithm that uses membership and equivalence queries to exactly learn the concepts given by the union of $s$ axis-parallel boxes over $\{1,\ldots,n\}^d$. This algorithm receives at most $sd$ counterexamples, makes $O((4s)^d + sd\log n)$ membership queries, and uses $O((4s)^d + sd\log n)$ time. Thus our algorithm is the first algorithm to exactly learn the union of $s$ discretized boxes in $d$-dimensional discretized Euclidean space in polynomial time in $s$ and $\log n$ for any constant $d$.

The hypothesis class used by this algorithm, selected to keep the algorithm simple, can be evaluated in time $O(d\log s)$[1]. However, in $O((2s)^{2d})$ time we can transform our hypothesis to the union of at most $O(sd\log s)$ boxes for $\text{BOX}_n^d$. Thus we obtain the even stronger result that our algorithm can exactly learn the union of $s$ axis-parallel boxes over $\{1,\ldots,n\}^d$ while

[1]The hypothesis essentially partitions $\{1,\ldots,n\}^d$ into at most $(4s+1)^d$ regions where all points in any region are classified as either positive or negative. The classifications for the regions are stored in a bit matrix, and we use a set of $d$ balanced binary search trees to efficiently find the region in which a point is contained. (See Section 4 for more details.)

making at most $sd+1$ equivalence queries[2] where each equivalence query is simply the union of $O(sd \log s)$ concepts from $\text{BOX}_n^d$, making $O((4s)^d + sd \log n)$ membership queries, and using $O(sd \cdot (2s)^{2d} + sd \log n)$ computation time. Thus for any constant $d$, this algorithm still uses time and queries polynomial in $s$ and $\log n$.

Then, in the second half of this paper, we introduce a new complexity measure that better captures the complexity of a union of boxes than simply the number of boxes and dimensions. More specifically, our new measure, $\sigma$, is the number of segments in the target concept (or polyhedron) where a segment is a maximum portion of one of the sides of the polyhedron that lies entirely inside or entirely outside each of the other halfspaces defining the polyhedron. It is easily seen that $\sigma \leq (2s)^d$. In this half of the paper we then present an improvement of our first algorithm that uses time and queries polynomial in $\sigma$ and $\log n$. Thus, this algorithm also uses time and queries polynomial in $s$ and $\log n$ for $d$ constant. More importantly, the complexity is polynomial in $\log n$ and our complexity measure of the number of segments in the target polygon. The hypothesis class used by this modified algorithm is decision trees of height at most $2sd$. Thus, observe that the hypothesis output (and the intermediate hypotheses) can be evaluated in time polynomial without any restrictions on $s$ or $d$. We then show that the time and queries used by this new algorithm are polynomial in $d$ and $\log n$ for $s$ any constant thus generalizing the exact learnability of DNF formulas with a constant number of terms. Combining these two methods of analysis, we get the interesting result that this single algorithm is efficient for either $s$ or $d$ constant.

## 2 Previous Work

The problem of learning geometric concepts over a discrete domain was extensively studied by Maass and Turan [14, 15, 16]. One of the geometric concepts that they studied was the class $\text{BOX}_n^d$. They showed that if the learner was restricted to only make equivalence queries in which each hypothesis was drawn from $\text{BOX}_n^d$ then $\Omega(d \log n)$ queries are needed to achieve exact identification [11, 16]. Auer [2] improves this lower bound to $\Omega(\frac{d^2}{\log d} \log n)$.

If one always makes an equivalence query using the simple hypothesis that produces the smallest box consistent with the previously seen examples, then the resulting algorithm makes $O(dn)$ equivalence queries. An algorithm making $O(2^d \log n)$ equivalence queries was given by

[2]The final equivalence query is the correct hypothesis, and thus at most $sd$ counterexamples are received.

Maass and Turan [13, 15]. The best known result for learning the class $\text{BOX}_n^d$ was provided by the work of Chen and Maass [7] in which they gave an algorithm making $O(d^2 \log n)$ equivalence queries. They also provide an algorithm to learn the union of two axis-parallel rectangles in the discretized space $\{1,\ldots,n\} \times \{1,\ldots,m\}$ in time polynomial in $\log n$ and $\log m$, where one rectangle has a corner at $(0,m)$ and the other has a corner at $(n,0)$. Auer [2] investigates exact learning of boxes where some of the counterexamples, given in response to equivalence queries, are noisy. Auer shows that $\text{BOX}_n^d$ is learnable if and only if the fraction of noisy examples is less than $1/(d+1)$ and presents an efficient algorithm that handles a noise rate of $1/(2d+1)$. More recently, Chen [5] gave an algorithm that used equivalence queries to learn general unions of two boxes in the (discretized) plane. The algorithm uses $O(\log^2 n)$ equivalence queries, and involves a detailed case analysis of the shapes formed by the two rectangles. It does not appear to generalize easily to higher numbers of boxes or dimensions.

In work independent of ours, Chen and Homer [10] have given an algorithm to learn the union of $s$ rectangles in the plane using $O(s^3 \log n)$ queries (both membership and equivalence) and $O(s^5 \log n)$ time. The hypothesis class of their algorithm is the union of $8s^2-2$ rectangles. In work subsequent to that presented here, Chen and Homer [6] have improved upon their earlier result by giving an algorithm that learns the union of $s$ boxes in $d$ dimensions using $O(k^{2(d+1)} d^2 \log^{2d+1} n)$ equivalence queries by applying techniques from recursive function theory.

Closely related to the problem of learning the union of discretized boxes, is the problem of learning the union of non-discretized boxes in the PAC model [18]. Blumer et al. [4] present an algorithm to PAC-learn an $s$-fold union of boxes in $E^d$ by drawing a sufficiently large sample of size $m = poly\left(\frac{1}{\epsilon}, \lg\frac{1}{\delta}, s, d\right)$, and then performing a greedy covering over the at most $\left(\frac{em}{2d}\right)^{2d}$ boxes defined by the sample. Thus for $d$ constant this algorithm runs in polynomial time. Long and Warmuth [12] present an algorithm to PAC-learn this same class by again drawing a sufficiently large sample and constructing a hypothesis that consists of at most $s(2d)^s$ boxes consistent with the sample. Thus both the time and sample complexity of their algorithm depend polynomially on $s, d^s, \frac{1}{\epsilon}$, and $\lg\frac{1}{\delta}$. So for $s$ constant this yields an efficient PAC algorithm.

We note that either of these PAC algorithms can be applied to the class $\bigcup_s \text{BOX}_n^d$ giving efficient PAC algorithms for this class for either $d$ constant or $s$ constant. As discussed by Maass and Turàn [16] the task of a concept learning algorithm is to provide a "smart" hy-

pothesis based on the data available. The results from Blumer et. al [4] show that under the PAC model any concise hypothesis that is consistent with the data is "smart enough". In other words, the PAC model provides no suitable basis for distinction among different consistent hypotheses. On the other hand, a method for defining a "smart" hypothesis is implicitly contained within the exact learning model. One must select hypotheses for the equivalence queries so that sufficient progress is made with each counterexample. This requirement of selecting a "smart" hypothesis makes the problem of obtaining an efficient algorithm to *exactly* learn the class $\bigcup_s \text{BOX}_n^d$ significantly harder than obtaining the corresponding PAC result.

Finally, under a variation of the PAC model in which membership queries can be made, Frazier et al. [9] have given an algorithm to PAC-learn the $s$-fold union of boxes in $E^d$ for which each box is entirely contained within the positive quadrant *and* contains the origin. Furthermore, their algorithm learns this subclass of general unions of boxes in time polynomial in both $s$ and $d$. Recall that since $\bigcup_s \text{BOX}_n^d$ generalizes DNF, a polynomial-time algorithm for arbitrary $d$ and $s$ would solve the problem of learning DNF. Observe that the class considered by Frazier et al. is a generalization of the class of DNF formulas in which all variables only appear negated.

# 3 Definitions

The learning model we use in this paper is that of learning with queries developed by Angluin [1]. In this model the learner's goal is to learn *exactly* how an unknown, Boolean-valued target function $f$, drawn from *concept class* $C$, classifies as positive or negative, all instances from *instance space* $\mathcal{X}$. It is often convenient to consider a concept as the set of instances that it classifies as positive. Thus, for each concept $f \in C$, $f \subseteq \mathcal{X}$. We say that $x$ is a positive instance for target concept $f$ if $x \in f$ (also denoted $f(x) = 1$) and say that $x$ is a negative instance otherwise (also denoted $f(x) = 0$). A *hypothesis* is a polynomial-time algorithm that, given any $x \in \mathcal{X}$, outputs a prediction for $f(x)$.

As mentioned above, the learning criterion in this paper is that of *exact identification*. In order to achieve exact identification, the learner's final hyptothesis, $h$, must be such that $h(x) = f(x)$ for all instances $x \in \mathcal{X}$. To achieve this goal the learner is provided with queries with which to learn about $f$. Our algorithms for learning $\bigcup_s \text{BOX}_n^d$ use membership queries and equivalence queries. A *membership query*, $\text{MQ}(x)$, returns "yes" if $f(x) = 1$ and returns "no" if $f(x) = 0$. An *equivalence query*, $\text{EQ}(h)$, returns "yes" if $h$ is logically equivalent to

$f$ or returns a counterexample otherwise. A *positive counterexample* $x$ is an instance such that $f(x) = 1$ and $h(x) = 0$. Similarly, a *negative counterexample* is such that $f(x) = 0$ and $h(x) = 1$.

Another important learning model is the PAC model introduced by Valiant [18]. In this model the learner is presented with labeled examples chosen at random according to an unknown, arbitrary distribution $\mathcal{D}$ over the instance space. The learner's goal is to output a hypothesis that with high probability, at least $(1-\delta)$, correctly classifies most of the instance space. That is, the weight, under $\mathcal{D}$, of misclassified instances must be at most $\epsilon$. The learner is permitted time polynomial in $1/\epsilon$, $1/\delta$ and relevant size measures to formulate a hypothesis.

Throughout this paper we use $y_1, \ldots, y_d$ to denote the $d$ dimension variables. An alternate view of a concept $f \in \bigcup_s \text{BOX}_n^d$ that we use is to treat it as an axis-aligned $d$-dimensional discretized polyhedron. For the remainder of this paper, when using the word "polyhedron" we refer to such an axis-parallel discretized polyhedron. Observe that each box is defined by the intersection of at most $2d$ halfspaces, two in each dimension. Thus, it follows that there are at most $2sd$ halfspaces that define the target polyhedron. We introduce the following definition:

**Definition 1** *For a concept $f \in \bigcup_s \text{BOX}_n^d$, we define a dimension $i$, $+/-$ pair to be a positive point $p_+ = (x_1, \ldots, x_i, \ldots, x_d)$, where $y_j = x_j$ for $1 \leq x_j \leq n$, paired with a negative point $p_-$ where $p_- = (x_1, \ldots, x_i + 1, \ldots x_d)$ or $p_- = (x_1, \ldots, x_i - 1, \ldots, x_d)$ where we implicitly assume that any point outside of $\{1, \ldots, n\}^d$ is a negative point. We use $+/^i-$ pair to denote a dimension $i$, $+/-$ pair.*

The following observation, while intuitively obvious, is useful.

**Observation 1** *A straight line between any two differently classified instances must cross at least one side of the target concept.*

We define the halfspace associated with a given $+/-$ pair to be the unique, axis-aligned halfspace $H$ that contains the positive but not the negative point. So for a $+/^i-$ pair where the positive point's $i$th coordinate is $c$, if the negative point's $i$th coordinate is $c+1$ then $H$ is given by $y_i \leq c$. Similarly, if the negative point's $i$th coordinate is $c-1$ then $H$ is given by $y_i \geq c$. We define the associated *hyperplane* to be the set of all points satisfying

$y_i = c$. Throughout Section 4 it is best to think about our algorithm finding the hyperplanes corresponding to the halfspaces that define the target polyhedron. Then in Section 5 we return to focusing on the halfspaces that define the target polyhedron.

# 4 An Initial Algorithm

In this section we present an algorithm that exactly identifies any concept from $\bigcup_s \text{BOX}_n^d$ while receiving at most $sd$ counterexamples, and using $O((4s)^d + sd \log n)$ membership queries and processing time.

For each of the $d$ dimensions, we will maintain a set of halfspaces, defined by the sides of the target polyhedron, that have been identified by the existence of $+/-$ pairs for the given dimension. For $1 \leq i \leq d$, let $p_i$ be the number of hyperplanes that are associated with the halfspaces defined by $+/^i-$ pairs. As we have already observed, $p_i \leq 2s$. For $1 \leq j \leq p_i$, suppose that the $p_i$ hyperplanes found are $y_i = x_j$ where $1 \leq x_j \leq n$. Thus in dimension $i$ we have decomposed $\{1, \ldots, n\}^d$ into up to $2p_i + 1$ regions: $p_i$ corresponding to the hyperplanes themselves, and $p_i + 1$ corresponding to the "strips" obtained when $\{1, \ldots, n\}^d$ is cut by each of the $p_i$ hyperplanes.

For our hypothesis, we divide $\{1, \ldots, n\}^d$ into

$$\Pi_{i=1}^d (2p_i + 1) \leq \Pi_{i=1}^d (4s + 1) = (4s + 1)^d$$

regions, and then just classify all points in a given region as positive (or as negative). For $1 \leq i \leq d$, our hypothesis maintains a balanced binary search tree $T_i$ for dimension $i$ where each internal node of the tree corresponds to one of the hyperplanes (with $x_j$ used for the key), and each leaf node corresponds to one of the regions defined by the hyperplanes. For each leaf node $v$ and dimension $i$ we keep a pair $(min_v^i, max_v^i)$, where $min_v^i$ (respectively, $max_v^i$) holds the minimum (respectively, maximum) $x_j$ such that $y_i = x_j$ is a point in the region corresponding to leaf $v$. (For the internal nodes, the key itself serves the role of both $min_v^i$ and $max_v^i$.)

Let $S_i = \{[min_v^i, max_v^i] \mid min_v^i \text{ and } max_v^i \text{ are the range for node } v \text{ in the tree } T_i\}$ and let $\mathcal{T} = \{T_1, \ldots, T_d\}$. Observe that the hyperplanes stored in the internal nodes of $\mathcal{T}$ partition $\{1, \ldots, n\}^d$ into a set of regions $R_\mathcal{T}$ given by the cross product $R_\mathcal{T} = S_1 \times S_2 \times \cdots \times S_d$. For $r \in R_\mathcal{T}$, corresponding to node $v$, we shall refer to the point $(min_v^1, \ldots, min_v^d)$ as the *lower* corner of $r$ and $(max_v^1, \ldots, max_v^d)$ as the *upper* corner of $r$. Thus upper and lower form two opposing corners of region $r$.

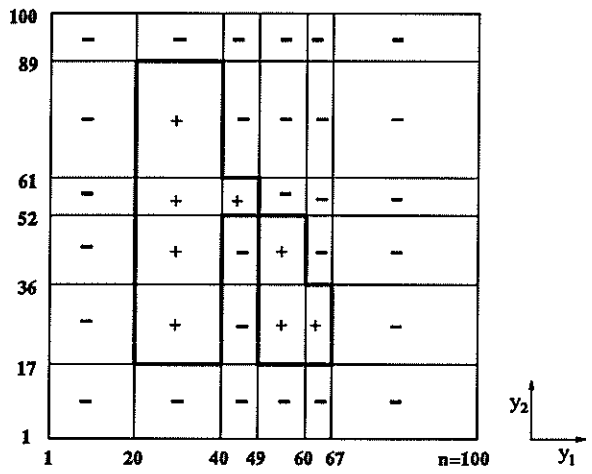


Figure 1: This shows the final set of regions corresponding to the target polyhedron that is outlined in bold. The classification of each two-dimensional region is shown inside the region. (The classifications of the one-dimensional and zero-dimensional regions are not shown but are stored in the prediction matrix $A$.)

In addition to the trees $T_1, \ldots, T_d$, our hypothesis also maintains a prediction array $A$ with $|R_T|$ entries where for $r \in R_T$, $A[r]$ is either 0 (indicating that for any point in $r$ the hypothesis will predict negative), or 1 (indicating that for any point in $r$ the hypothesis will predict positive) or contains a pointer to an element of a queue (to be explained in a moment) that indicates that the classification of the points in region $r$ is not well defined. We use $h_{R_T,A}$ to denote the hypothesis defined by the regions in $R_T$ with the classifications given in $A$. Figure 1 shows the set of regions defined by a target concept once all hyperplanes are discovered, and the classifications of all of the regions (as stored in $A$).

Given a hypothesis $h_{R_T,A}$ and a point $x = (x_1, \ldots, x_d)$ we can compute $h(x)$, the prediction made by hypothesis $h$ on point $x$, as follows. For $1 \leq i \leq d$ we perform a search for $x_i$ in tree $T_i$ to find the node having $x_i$ in its range. Combining the ranges of the $d$ nodes found defines the region $r \in R_T$ that contains $x$. Finally $h(x) = A[r]$.

We define a region $r \in R_T$ to be *valid* if the upper and lower corner points of $r$ have the same classification. We define hypothesis $h_{R_T,A}$ to be *valid* if each region in $R_T$ is valid. A key step of our algorithm is to build a valid hypothesis that incorporates all known halfspaces. We first prove that given a set of hyperplanes (represented in the $d$ binary trees) the learner

can efficiently construct a valid hypothesis. To help in the process of making a hypothesis valid, we maintain a queue $Q$ of invalid regions. In addition, for each region $r \in Q$ we store a bit $q.lower$ (respectively, $q.upper$) giving the classification of the lower (respectively, upper) corner of $r$. Observe that by the definition of an invalid region, exactly one of these bits will be 1.

### 4.1 Building a Valid Hypothesis

In this section we provide a procedure that takes an invalid hypothesis $h_{R_T,A}$ and the queue $Q$ of invalid regions from $R_T$, and refines $h_{R_T,A}$ so that it is valid. In refining $h_{R_T,A}$ our procedure uses membership queries to find *new* hyperplanes with which to modify the hypothesis. Our procedure to build a valid hypothesis never removes any hyperplane from any tree in $\mathcal{T}$, and only searches for a new hyperplane in such a way that we are certain that an existing hyperplane will not be rediscovered in the process. We also maintain the invariant that $Q$ always contains exactly one entry for each invalid region of $R_T$.

Our procedure to build a valid hypothesis repeatedly does the following until $Q$ is empty (and thus the hypothesis is valid). Let $r$ be the region at the front of the queue. Since $r$ is not a valid region its two opposing corners are known to have different classifications. As we saw in Observation 1, there must be a side of the target concept between these two corners. Thus we can use these two points to perform a binary search (where the comparisons are replaced by membership queries) for a $+/-$ pair contained within region $r$. We are guaranteed to find a $+/-$ pair for which both points in the pair are contained within $r$ while using only $\lceil \log n \rceil$ membership queries and $O(\log n)$ time. Furthermore, the hyperplane defined by this $+/-$ pair is guaranteed to be a hyperplane that has not yet been discovered (by the definition of a region). For the remainder of this paper, we shall just speak of performing a binary search between a positive and negative point to find a hyperplane.

We now describe the procedure ADD-HYPERPLANE that modifies our hypothesis to incorporate the new hyperplane found. Without loss of generality, we assume that the hyperplane found by the binary search is $y_i = c$. Let $v$ be the leaf in $T_i$ with the range $[min_v^i, max_v^i]$ such that $min_v^i \leq c \leq max_v^i$. We begin by using the standard tree insertion procedures for a balanced search tree to update tree $T_i$ so that $v$ becomes an internal node with a key of $c$, it has left child $v_{left}$ with the range $[min_v^i, c-1]$, and it has right child $v_{right}$ with the range

$[c+1, max_v^i]$. Thus each region in

$$R_{delete} = S_1 \times \cdots \times S_{i-1} \times \{[min_v^i, max_v^i]\} \times S_{i+1} \times \cdots \times S_d$$

is replaced by three regions, giving us the new regions

$$R_{add} = S_1 \times \cdots \times S_{i-1} \times \{[min_v^i, c-1], [c, c], [c+1, max_v^i]\} \times S_{i+1} \times \cdots \times S_d.$$

Since all regions in $R_{delete}$ no longer exist, we remove any that are in $Q$ by using the pointer provided in $A$. For each region $r \in R_{add}$ we make a membership query on the lower and/or upper opposing corners if those queries have not already been made. If the classification of these two corners are the same then the classification is entered in $A$, otherwise the region is enqueued. Once the queue is empty, we know that we have a valid hypothesis and thus have completed the process. The algorithm is shown in Figure 2.

### 4.2 Putting it Together

Our algorithm LEARN-BOXES1 works as follows. For ease of exposition we artificially extend the instance space from $\{1, \ldots, n\}^d$ to $\{0, 1, \ldots, n, n+1\}^d$ where it is known a priori that any example with a coordinate of 0 or $n+1$ in any dimension is a negative example. (The pseudocode does not explicitly make this check, but one could imagine replacing the calls to MQ by a procedure that first checks for such cases.) Initially, $R_T$ just contains the single region corresponding to the entire instance space. Since all of the corners of this region are negative, the initial hypothesis predicts 0 for all instances.

We then repeat the following process until a successful equivalence query is made. Let $x$ be the counterexample received from an equivalence query made with a valid hypothesis. We now discuss how to use membership queries (in the form of a binary search) to find two new hyperplanes defined by the target concept. Without loss of generality, we assume that $x$ is a positive counterexample. (Negative counterexamples are handled similarly.) Every counterexample is within one of the regions, say region $r$, in $S_1 \times \cdots \times S_d$. Since the hypothesis was valid and $x$ is a positive counterexample, we know that the upper and lower corners of $r$ are classified as negative. Thus we can use these corners of $r$ (with $x$) as the endpoints for binary searches to discover two new hyperplanes. The hypothesis is updated using ADD-HYPERPLANE to incorporate these two hyperplanes. Finally, we call MAKE-VALID-HYPOTHESIS to further refine any invalid regions. Figure 3 gives the complete algorithm.

ADD-HYPERPLANE($h_{R_T,A}, Q, i, c$)

Let $v$ be the leaf of $T_i$ for which $min_v^i \leq c \leq max_v^i$

Using a standard balanced tree insertion procedure, update $T_i$ so that

$v$ is an internal node with key $c$

$v$ has a left child with range $[min_v^i, c-1]$

$v$ has a right child with range $[c+1, max_v^i]$

Let $R_{delete} = S_1 \times \cdots \times S_{i-1} \times \{[min_v^i, max_v^i]\} \times S_{i+1} \times \cdots \times S_d$

Let $R_{add} = S_1 \times \cdots \times S_{i-1} \times \{[min_v^i, c-1], [c,c], [c+1, max_v^i]\} \times S_{i+1} \times \cdots \times S_d$

For each $r \in R_{delete}$

Let $r_=$, $r_<$ and $r_>$ be the regions in $R_{add}$ for which $(r_= \cup r_< \cup r_>) = r$

If $A[r] = b$ (for $b = 0$ or $b = 1$)

Let $A[r_=] = A[r_<] = A[r_>] = b$

Else (so $A[r]$ is a pointer to element $q$ of $Q$)

Generate a new queue node for $r_=$, $r_<$ and $r_>$

Set the corresponding entries of $A$ to point to these new nodes

Copy $q.lower$ and $q.upper$ in the appropriate queue entries for $r_<$ and $r_>$

Remove $q$ from $Q$

For each $r' \in R_{add}$

Make a membership query to determine $q.lower$ and $q.upper$ if not already known

If $A[r']$ points to $q$ (versus being 0 or 1)

If $q.lower = q.upper$ then let $A[r'] = q.lower$

Else ($r$ is invalid)

$Q$.ENQUEUE($q$)

Figure 2: Our subroutine to update $h_{R_T,A}$ to incorporate the newly discovered hyperplane $y_i = c$. The new hyperplane is added to tree $T_i$. Then all regions in $R_T$ that are split are removed from $Q$. Finally this procedure initializes the new entries of $R_T$ in the prediction matrix $A$.

LEARN-BOXES1:

Let $Q \leftarrow \emptyset$

For $1 \leq i \leq d$

  Initialize $T_i$ to be a single leaf covering the range 0 to $n+1$

For $r$ be the single region of $R_{\mathcal{T}}$, let $A[r] = 0$

While Equiv$(h_{R_{\mathcal{T}},A}) \neq$ "yes"

  Let $x$ be the counterexample where $x$ is in region $r$ of $h_{R_{\mathcal{T}},A}$

  Let $z_1$ and $z_2$ be the lower and upper opposing corners of $r$

  For $1 \leq \ell \leq 2$

    Perform binary search between $x$ and $z_\ell$ to find hyperplane $y_i = c$

    ADD-HYPERPLANE$(h_{R_{\mathcal{T}},A}, Q, i, c)$

  $h_{R_{\mathcal{T}},A} \leftarrow$MAKE-VALID-HYPOTHESIS$(h_{R_{\mathcal{T}},A}, Q)$

Return $h_{R_{\mathcal{T}},A}$

MAKE-VALID-HYPOTHESIS$(h_{R_{\mathcal{T}},A}, Q)$

While $Q \neq \emptyset$

  $q \leftarrow Q$.DEQUEUE

  Perform binary search between $q.lower$ and $q.upper$

    to find the hyperplane $y_i = c$

  ADD-HYPERPLANE$(h_{R_{\mathcal{T}},A}, Q, i, c)$

Figure 3: Algorithm for learning unions of $d$-dimensional axis-parallel boxes.

### 4.3 Analysis

We now analyze the time and query complexity of LEARN-BOXES1. As part of this analysis we use the following lemma.

**Lemma 1** *Every counterexample can be used to discover (at least) two distinct new hyperplanes of the target concept.*

**Proof Sketch:** Let $f$ be the target concept, let $x = (x_1, \ldots, x_d)$ be the counterexample and $r \in R_T$ be the region containing $x$. Since $h_{R_T,A}$ is a valid hypothesis, we know that the upper and lower corners of $r$ are classified opposite $x$ and all points in $r$ are classified opposite $x$ by the hypothesis. Since a positive point and a negative point must be separated by some hyperplane of the target polyhedron, searches between $x$ and each of upper and lower will find some $+/-$ pair. These will be distinct since the two searches move away from each other in all dimensions. □

We now prove that our first algorithm has the stated complexity.

**Theorem 1** *Given any target concept $f \in \bigcup_s \text{BOX}_n^d$,* LEARN-BOXES1 *achieves exact identification of $f$ making at most $sd + 1$ equivalence queries, and using $O((4s)^d + sd \log n)$ time and membership queries.*

**Proof Sketch:** The correctness of LEARN-BOXES1 is trivial. Since the algorithm only returns a hypothesis $h_{R_T,A}$ for which Equiv($h_{R_T,A}$) returns "yes", the algorithm is correct upon returning a hypothesis.

We now analyze the query and time complexity of LEARN-BOXES1. Recall that since there are only $s$ boxes in the target concept, there are at most $2sd$ hyperplanes in the final hypothesis. Furthermore, since no hyperplane is ever rediscovered and every binary search (which uses $O(\log n)$ membership queries) discovers a hyperplane, we know that $O(sd \log n)$ membership queries are used during all of the binary searches made by the algorithm. Also, since there are at most $(4s + 1)^d$ regions in the final hypothesis, the number of membership queries used for querying the upper and lower opposing corners is at most $2 \cdot (4s + 1)^d = O((4s)^d)$. Since these are the only two places in which membership queries are performed, the total number of membership queries made by our algorithm is $O((4s)^d + sd \log n)$.

From Lemma 1 we know that each counterexample enables LEARN-BOXES1 to find at least two new distinct hyperplanes of the target concept. Thus since there are at most $2sd$

hyperplanes comprising the $s$ boxes, at most $sd$ counterexamples can be received and thus at most $sd+1$ equivalence queries will be made.

The time needed to evaluate $h_{R_T,A}(x)$ for an unlabeled example $x$ is $O(d \log s)$ since the key operation is performing $d$ searches in balanced search trees of depth $O(\log s)$. Thus, it is easily seen that the time complexity of this algorithm is $O((4s)^d + sd \log n)$. □

We note that, if desired, the use of membership queries in our algorithm can be reduced to only their use within the binary searches. Instead of querying opposing corners of new regions created, we can instead use the classification of the single corner known or otherwise a default of negative for the classification of the region. Then the counterexamples from the equivalence queries can be used to obtain a positive and negative point in a region that can be used for the binary search. This method dramatically increases the number of equivalence queries used.

### 4.4 Using a Hypothesis Class of Unions of Boxes

We now describe how a valid hypothesis can be converted to the union of $O(sd \log s)$ boxes from $\text{BOX}_n^d$. Since all equivalence queries are made with valid hypotheses, such a conversion enables our algorithm to learn the union of $s$ boxes from $\text{BOX}_n^d$ using as a hypothesis class the union of $O(sd \log s)$ boxes from $\text{BOX}_n^d$.

Recall that a valid hypothesis $h$ essentially encodes the set of positive regions. Thus our goal is to find the union of as few boxes as possible that "cover" all of the positive regions. We now describe how to formulate this problem as a set covering problem for which we can then use the standard greedy set covering heuristic [8] to perform the conversion. The set $X$ of objects to cover will simply contain all positive regions in $h$. Thus $|X| \leq (4s+1)^d$. Then the set $\mathcal{F}$ of subsets of $X$ will be made as follows. Consider the set $B$ of boxes where each box in $B$ is formed by picking a minimum and maximum coordinate in each dimension, from the hyperplanes represented in $h$ for that dimension. For any $b \in B$, if $b$ contains any negative region, then throw it out. Otherwise, place in $\mathcal{F}$ the set of regions contained within $b$. Thus $|\mathcal{F}| \leq (2s)^{2d}$ since there are at most $2s$ values in each dimension that can form the two sides of the box. Furthermore, $\mathcal{F}$ contains a subset of size $s$ that covers all items in $X$. Finally, we can apply the greedy set covering heuristic to find a set of at most $s(\ln |X| + 1) = s(d \ln(4s+1) + 1) = O(sd \log s)$ boxes that cover all positive regions. The time to perform the conversion is $O((2s)^{2d})$. Thus, since at most $sd+1$ equivalence queries

are made, the total time spent in converting the internal hypotheses into hypotheses that are unions of boxes is at most $O(sd \cdot (2s)^{2d})$.

# 5 An Improved Algorithm

Observe that by extending the hyperplane defined by a $+/-$ pair across the entire domain, the algorithm LEARN-BOXES1 may unnecessarily split a valid region into a large number of smaller regions all of which make the same prediction. The algorithm we present here is motivated by the goal of reducing this unnecessary splitting by only splitting the region in which the counterexample is contained.

Before presenting our improved algorithm, we briefly examine how one might measure the complexity of a concept from $\bigcup_s \text{BOX}_n^d$. Observe that the number of boxes $s$ used to form the target concept is not a good measure of the complexity of the target concept. For example, consider the two examples shown Figure 4. While both targets are composed of 6 boxes, the first is clearly more complex than the second. Thus the complexity of an algorithm should depend on some quantity other than just the number of boxes and dimensions of the target polyhedron. We now introduce such a new complexity measure, $\sigma$, to better capture the complexity of the target concept. We define a *segment* of the target polyhedron $f$ as a maximum portion of one of the sides of $f$ that lies either entirely inside or entirely outside of each of the other halfspaces defining the polyhedron. For example the target concept shown in Figure 1 has 20 segments. For a concept $f \in \bigcup_s \text{BOX}_n^d$, we let $\sigma$ denote the number of segments in the polyhedron corresponding to $f$. Observe that $s \leq \sigma \leq (2s)^d$, where the first inequality is based on the assumption that none of the boxes are contained within the union of the other $s-1$ boxes.

The hypothesis class we use in this algorithm is a decision tree over the halfspaces defining the target polyhedron. Namely, each hypothesis $T$ is a rooted binary tree where each internal node is labeled with a halfspace and whose leaves are labeled from $\{0,1\}$. We evaluate $T$ recursively by starting at the root and evaluating the left subtree if the root's halfspace does not contain the point, and right subtree otherwise. When a leaf is reached its label is output. Observe that each node of $T$ corresponds to a sub-region of the domain, with the root corresponding to the entire domain. The halfspace $H$ associated with each internal node divides its region $r$ into two sub-regions, with the left child being the sub-region given by $\overline{H} \cap r$, and the right child being the sub-region given by $H \cap r$. The leaves correspond to a

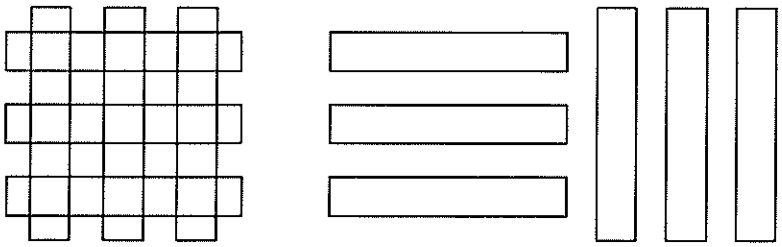


Figure 4: These concepts illustrate why number of boxes is an inadequate measure of complexity. (a) is more complex than (b) because of the large number of intersections of the sides of the boxes.

set of nonoverlapping boxes that cover the entire region where the label for a given region is given by the label for the corresponding leaf. Observation 2 shows that the height of the final decision tree will be at most $2sd$. Thus the hypothesis can be evaluated in time polynomial in both $s$ and $d$.

**Observation 2** *The height of the final decision tree constructed by* LEARN-BOXES2 *is at most* $2sd$ *since each of the at most* $2sd$ *halfspaces defining the target polyhedron can appear at most once on any path from the root to any leaf.*

We now describe our new algorithm. We initialize $T$ to be a single 0 leaf node. (As in the previous section we use the instance space $\{0, 1, \ldots, n, n+1\}^d$.) When a counterexample is received, we first search in $T$ to find the leaf $v$ containing it. Let $r$ be the sub-region corresponding to $v$. Then as in LEARN-BOXES1 we use a binary search to find a $+/-$ pair contained in $r$ that defines a halfspace $H$. We replace $v$ with an internal node labeled with $H$, having left child $v_L$ corresponding to the region given by $\overline{H} \cap r$ and right child $v_R$ corresponding to the region given by $H \cap r$. At this point we call a procedure that recursively visits all newly created leaves in a depth-first manner and checks if the corresponding region is a valid region. If the region $r'$ associated with leaf $v'$ is valid then the classification field is filled, otherwise we use a binary search to obtain a halfspace $H'$ for $r'$ and replace $v'$ by an internal node labeled with $H'$. We generate two new leaves: $v'_L$ corresponding to the region $\overline{H'} \cap r'$ and $v'_R$ corresponding to the region $H' \cap r'$. Then a recursive call is made to validate (if necessary) each of these new regions. The algorithm is shown in Figure 5. One possible final hypothesis that could be constructed by this algorithm, for the target polyhedron shown in Figure 1, is

shown in Figure 6. Figure 7 shows the decomposition of $\{1,\ldots,n\}^d$ that corresponds to the decision tree shown in Figure 6.

# 6 Analysis

We now give two separate techniques for analyzing this algorithm. The first method of analysis gives that this algorithm uses queries and time polynomial in $\sigma$ and $\log n$ (and thus polynomial in $s$ and $\log n$ for $d$ constant). The second method of analysis shows that our algorithm uses queries and time polynomial in $d$ and $\log n$ for $s$ any constant.

**Theorem 2** *Given any target concept $f \in \bigcup_s \text{BOX}_n^d$, the algorithm* LEARN-BOXES2 *achieves exact identification of $f$ making at most $(\sigma/2+1)$ equivalence queries, and using $O(\sigma \log n)$ time and membership queries.*

**Proof:** Observe that each segment of the target polyhedron will cause at most one region to be split. Thus the number of leaves in the decision tree created will be at most $\sigma+1$.

By Lemma 1 we get that two segments are found from the counterexample to each equivalence query (here the second halfspace is implicitly found by the call to SPLIT-REGION). Thus at most $\frac{\sigma}{2}+1$ equivalence queries will be made.

Furthermore, since there are at most 2 membership queries made to query the upper and lower corners of each leaf, and $\log n$ membership queries used in the binary searches for the $\sigma$ halfspaces, it follows that the number of membership queries made is at most $2\sigma + \sigma\lceil \lg n \rceil = O(\sigma \log n)$.

Observe that the depth of $T$ is at most $2sd$ since any of the $2sd$ hyperplanes defined by the $s$ boxes in the target concept will appear at most once on any path from a root to the leaf. Thus the time to locate the region to split is $O(sd)$ and it immediately follows that the time complexity is $O(\sigma \log n)$. □

**Corollary 3** *The algorithm* LEARN-BOXES2 *achieves exact identification for any $f \in \bigcup_s \text{BOX}_n^d$ using time and queries polynomial in $s$ and $\log n$ for $d$ constant.*

**Proof:** This follows immediately from Theorem 2 and the observation that $\sigma \leq (2s)^d$. □

Finally, observe that just as we described in Section 4.4 our final hypothesis can be converted to the union of $O(sd \log s)$ boxes from $\text{BOX}_n^d$. Recall that the time to perform the conversion is $O((2s)^{2d})$ and thus this will be efficient only if $d$ is constant.

LEARN-BOXES2:

Initialize $T$ to be the single 0-leaf

While Equiv($T$) $\neq$ "yes"

    Let $x$ be the counterexample

    Search in $T$ to find the leaf $v$ corresponding to region containing $x$

    $T \leftarrow$SPLIT-REGION$(T, v, x)$

Return $T$

SPLIT-REGION$(T, v, x)$:

Perform binary search between $x$ and corner of region $r$ corresponding to $v$

Let $H$ be the hyperplane found

Let $v_L$, and $v_R$ correspond to the new regions created

Make $v$ an internal node labeled with $H$ and having left child $v_L$, and right child $v_R$

Let $r_L$ be the region $\overline{H} \cap r$ corresponding to $v_L$

Let $r_R$ be the region $H \cap r$ corresponding to $v_R$

For each $r' \in \{r_L, r_R\}$

    Let $v'$ be the leaf corresponding to region $r'$

    Perform a membership query on the upper and lower opposing corners of $r'$

    If both corners have classification $b \in \{0, 1\}$ let $v'$ be a $b$-leaf of $T$

        Let $v'.prediction \leftarrow v.prediction$

    Else

        SPLIT-REGION$(h_T, upper, lower)$

Figure 5: Alternate algorithm for learning unions of $d$-dimensional axis-parallel boxes.

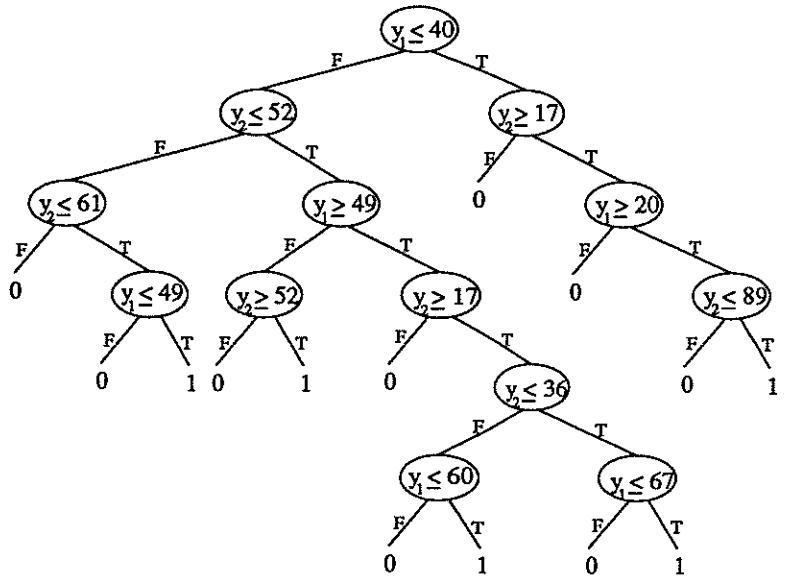


Figure 6: This shows the final decision tree that could be constructed by LEARN-BOXES2 for the target polyhedron shown in Figure 1.

We now use a different method of analysis to show that LEARN-BOXES2 uses time and queries polynomial in $d$ and $\log n$ for $s$ constant.

We begin by examining the number of regions created by this algorithm. Each region is represented by a single leaf in the hypothesis decision tree. Thus, we can find the number of regions by finding the number of leaves in our hypothesis. We now derive a recurrence relation for the number of leaves in the final decision tree. Let $f \in \bigcup_s \text{BOX}_n^d$ be the target concept and $T$ be the final decision tree output by our algorithm. For each internal node $T$ there is an associated region of $\{1,\ldots,n\}^d$. For any node $r$ in $T$, let $h_r$ denote the height of the subtree of $T$ rooted at $r$ and let $s_r$ be the minimum number of boxes needed to cover the region of $\{1,\ldots,n\}^d$ associated with $r$. So for the region $r$ corresponding to the root of $T$ we have that $s_r \leq s$ and $h_r \leq 2sd$ since, by Observation 2, the height of $T$ is at most $2sd$.

Let $L(s,h)$ denote the maximum number of leaves in a decision tree rooted at a node $r$ with $s_r = s$ and $h_r = h$. Then we have that $L(s,h) = L(s,h-1) + L(s-1,h-1)$ where for all $s \geq 0$, $L(s,0) = 1$ and for all $h \geq 1$, $T(0,h) = 1$. To see this, observe that when we find, while building the hypothesis, a hyperplane that splits a node, the two subproblems that correspond to the left and right children both must have at most $h-1$ hyperplanes left

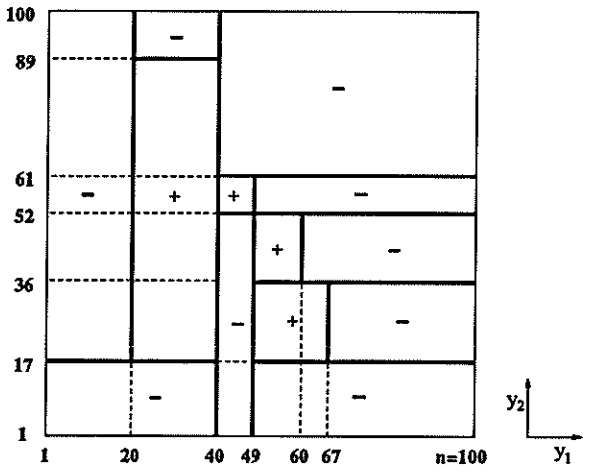


Figure 7: This shows the decomposition of $\{1,\ldots,n\}^d$ corresponding to the decision tree shown in Figure 6.

to find since in the worst case, all other hyperplanes are split by this one and thus appear on both sides. Finally, since the hyperplane just found must be the side of one of the $s$ boxes, that box will not appear in one of the recursive calls. (In the worst case all other boxes will be split).

Notice that there are no leaves in the decision tree on levels 0 to $s-1$. On each level from $s$ to $h$ there are leaves caused by the base case $s=0$ of the recurrence. The number of these leaves at level $j$ is given by

$$\sum_{j=s}^{h} \binom{j-1}{s-1}$$

since the number of nodes, in the recursion tree, with $s=0$ at level $j$ is equal to the number of nodes with $s=1$ at level $j-1$. There are also leaf nodes caused by the other base case of the recurence, $h=0$. Note that this is the last level of the tree. The number of leaves here is given by

$$\sum_{j=0}^{s-1} \binom{h}{j}$$

since this gives the number of nodes for each non-zero value of $s$ (the $s=0$ nodes on this level were already counted in the previous expression). Thus, the total number of leaves is:

$$\sum_{j=s}^{h} \binom{j-1}{s-1} + \sum_{j=0}^{s-1} \binom{h}{j} \quad = \quad \sum_{j=s-1}^{h-1} \binom{j}{s-1} + \sum_{j=0}^{s-1} \binom{h}{j}$$

$$
\begin{aligned}
&= \sum_{j=0}^{h-1} \binom{j}{s-1} + \sum_{j=0}^{s-1} \binom{h}{j} \\
&= \binom{h}{s} + \sum_{j=0}^{s-1} \binom{h}{j} \\
&= \sum_{j=0}^{s} \binom{h}{j} \qquad (1)
\end{aligned}
$$

Recall that $h \leq 2sd$. In the following Lemma we show that $(2sd)^s$ is an upper bound on the summation in Equation 1.

**Lemma 2** *The number of leaves in the hypothesis decision tree constructed by* LEARN-BOXES2 *is bounded above by* $(2sd)^s$ *for* $s > 1$ *and is* $2d+1$ *for* $s = 1$.

**Proof:** Let $n = sd$ and $k = s$ for $s > 1$. Then the expression we have derived for the number of leaves is $\sum_{j=0}^{k} \binom{2n}{j}$. It is easily shown by induction on $k$ that $\sum_{i=0}^{k} \binom{2n}{i} \leq (2n)^k$ for $n \geq k > 1$ and thus the result follows for $s > 1$. Finally, for $s = 1$ the number of leaves in the hypothesis is $2d + 1$. □

We are now ready to prove the running time of our algorithm using this method of analysis. For ease of exposition we assume $s > 1$ in the remainder of this paper.

**Theorem 4** *Given any target concept* $f \in \bigcup_s \text{BOX}_n^d$, *the algorithm* LEARN-BOXES2 *achieves exact identification of* $f$ *making at most* $(2sd)^s$ *equivalence queries, making* $O((2sd)^s \cdot \log n)$ *membership queries, and using* $O((2sd)^{s+1} \cdot \log n)$ *time.*

**Proof:** Observe that the number of counterexamples received by LEARN-BOXES2 is at most the number of internal nodes in our final decision tree. Thus the number of equivalence queries made by LEARN-BOXES2 is at most the number of leaves in the final decision tree.

Equation 1 shows that $L(s,h) = \sum_{j=0}^{s} \binom{h}{j}$ and Lemma 2 proves a bound for the number of leaves of $T$ of at most $\sum_{j=0}^{s} \binom{2sd}{s} \leq (2sd)^s$. Thus it immediately follows that at most $(2sd)^s$ equivalence queries are made. Since at most $\lceil \lg n \rceil$ membership queries are used by the binary search procedure when splitting a node, it follows that the number of membership queries made by LEARN-BOXES2 is $O((2sd)^s \cdot \log n)$. Finally, since it takes $O(sd)$ time to find the node corresponding to the region containing the counterexample and at most $O(\log n)$ time for each binary search, it follows that the time complexity of LEARN-BOXES2 is $O((2sd)^{s+1} \cdot \log n)$. □

Thus LEARN-BOXES2 achieves exact identification for any $f \in \bigcup_s \text{BOX}_n^d$ using time and queries polynomial in $d$ and $\log n$ for $s$ constant. We note that for $s \geq 6$ we can remove a factor of $2^s$ in the complexity by using the tighter upperbound that $\sum_{j=0}^{s} \binom{2sd}{s} \leq (sd)^s$. Finally, it is easily seen that if this algorithm is run in parallel with $(2sd)^s$ processors then the number of parallel steps needed is at most $2sd$.

# 7 Concluding Remarks

In this paper we presented the first known algorithm to exactly learn the union of $s$ boxes from $\{1,\ldots,n\}^d$ that runs in time polynomial in $s$ and $\log n$ for any constant $d$. Furthermore, we can show that $\bigcup_s \text{BOX}_n^d$ is learnable with time and queries polynomial in $s$ and $\log n$ for $d$ any constant even when the hypotheses used for all equivalence queries are a union of $O(sd \log s)$ boxes from $\text{BOX}_n^d$.

Next we introduced a new complexity measure, $\sigma$, which is the number of segments in the target polyhedron, that better captures the complexity of a union of boxes than simply the number of boxes and dimensions. We have given an improvement to our initial algorithm that runs in time polynomial in $\log n$ and $\sigma$. Furthermore, this same algorithm can be shown to use time and queries polynomial in $d$ and $\log n$ for $s$ constant thus generalizing the exact learnability of DNF formulas with a constant number of terms. Combining these two results for our second algorithm we get the result that we have a single algorithm for learning the union of $s$ boxes from $\{1,\ldots,n\}^d$ that is efficient for either $s$ or $d$ constant.

A number of important open questions, that we have not answered, concern the necessity of membership queries to exactly learn the class $\bigcup_s \text{BOX}_n^d$ in both the situation in which the hypotheses (used for the equivalence queries) can be any polynomially evaluatable program [3], and the situation in which the hypotheses must come from $\bigcup_s \text{BOX}_n^d$ (or at least a union of a "small" number of boxes from $\text{BOX}_n^d$) [4]. To illustrate such a distinction, consider the class of $k$-term DNF formulas where $k$ is constant. (This is the special case of $\bigcup_s \text{BOX}_n^d$ where $n = 2$ and $s$ is constant.) Pitt and Valiant [17] have shown that the class of $k$-term DNF formulas are not exactly learnable in polynomial time without membership queries when all hypothesis must

[3] For the case of $d$ constant the recent result of Chen and Homer shows that membership queries are not needed when the need not be from the class.

[4] For general $d$ and constant $s$ it is not known if membership queries are needed even if the hypothesis class is not restricted.

be $k$-term DNF formulas. In fact, they prove the even stronger result that for $k \geq 4$ the class of $k$-term DNF formulas are not exactly learnable in polynomial time without membership queries even when all hypothesis come from the class of $(2k-3)$-term DNF formulas. On the other hand, if we just require that the hypotheses are polynomially evaluatable, then it is well known that this class is efficiently learnable without using membership queries by just using the class of $k$-CNF formulas as the hypothesis class. It seems likely to us that a similar behavior will be seen when studying the issue of the necessity of membership queries for learning $\bigcup_s \text{BOX}_n^d$ for either $s$ or $d$ constant.

Another interesting direction is to explore other complexity measures, besides $\sigma$, that better capture the complexity of the target polyhedron. In particular, we feel that the number of sides of the target polyhedron is a good measure of the complexity of this class. For example the target concept shown in Figure 1 has 14 sides.

Finally, it would be interesting to see if $\bigcup_s \text{BOX}_n^d$ can be efficiently learned in time polynomial in $s$ and $\log n$ for $d = O(\log s)$ or in time polynomial in $d$ and $\log n$ for $s = O(\log d)$ (i.e. a generalization of the Blum and Rudich [3] result that $O(\log n)$-term DNF formulas are exactly learnable). Of course, since $\bigcup_s \text{BOX}_n^d$ generalizes the class of DNF formulas, it seems very unlikely that one could develop an algorithm that is polynomial in $s$, $\log n$, and $d$.

## Acknowledgements

We thank Peter Auer for suggesting the refinement discussed in Section 4.3. We thank Nader Bshouty for suggesting the analysis that we used in the proof of Theorem 4. We also thank the COLT committee for their very valuable comments.

## References

[1] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[2] Peter Auer. On-line learning of rectangles in noisy environments. In *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory*, pages 253–261, July 1993.

[3] Avrim Blum and Steven Rudich. Fast learning of $k$-term DNF formulas with queries. In *Proceedings of the Twenty Fourth Annual ACM Symposium on Theory of Computing*, pages 382–389, May 1992.

[4] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(4):929–965, October 1989.

[5] Zhixiang Chen. Learning unions of two rectangles in the plane with equivalence queries. In *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory*, pages 243–252. ACM Press, July 1993.

[6] Zhixiang Chen and Steven Homer. The bounded injury priority method and the learnability of unions of rectangles. Unpublished manuscript, May 1994.

[7] Zhixiang Chen and Wolfgang Maass. On-line learning of rectangles. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 16–27. ACM Press, July 1992.

[8] V. Chvatal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[9] Mike Frazier, Sally Goldman, Nina Mishra, and Leonard Pitt. Learning from a consistently ignorant teacher. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, July 1994.

[10] Steven Homer and Zhixiang Chen. Fast learning unions of rectangles with queries. Unpublished manuscript, July 1993.

[11] Nick Littlestone. Learning when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[12] Philip M. Long and Manfred K. Warmuth. Composite geometric concepts and polynmomial predictability. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 273–287. Morgan Kaufmann, August 1990.

[13] Wolfgang Maass and György Turán. On the complexity of learning from counterexamples. In *30th Annual Symposium on Foundations of Computer Science*, pages 262–267, October 1989.

[14] Wolfgang Maass and György Turán. On the complexity of learning from counterexamples and membership queries. In *31st Annual Symposium on Foundations of Computer Science*, pages 203–210, October 1990.

[15] Wolfgang Maass and György Turán. Algorithms and lower bounds for on-line learning of geometrical concepts. Technical Report IIG-Report 316, Technische Universität Graz, TU Graz, Austria, October 1991.

[16] Wolfgang Maass and György Turán. Lower bound methods and separation results for on-line learning models. *Machine Learning*, 9:107–145, 1992.

[17] Leonard Pitt and Leslie G. Valiant. Computational limitations on learning from examples. *Journal of the Association for Computing Machinery*, 35(4):965–984, 1988.

[18] Leslie Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.