SIAM J. COMPUT.
Vol. 28, No. 2, pp. 674–699

EXACT LEARNING OF DISCRETIZED GEOMETRIC CONCEPTS[*]

NADER H. BSHOUTY[†], PAUL W. GOLDBERG[‡], SALLY A. GOLDMAN[§], AND H. DAVID MATHIAS[¶]

Abstract. We first present an algorithm that uses membership and equivalence queries to exactly identify a discretized geometric concept defined by the union of m axis-parallel boxes in d-dimensional discretized Euclidean space where each coordinate can have n discrete values. This algorithm receives at most md counterexamples and uses time and membership queries polynomial in m and $\log n$ for any constant d. Furthermore, all equivalence queries can be formulated as the union of $O(md \log m)$ axis-parallel boxes.

Next, we show how to extend our algorithm to efficiently learn, from *only* equivalence queries, any discretized geometric concept generated from any number of halfspaces with any number of known (to the learner) slopes in a constant dimensional space. In particular, our algorithm exactly learns (from equivalence queries *only*) unions of discretized axis-parallel boxes in constant dimensional space in polynomial time. Furthermore, this equivalence query only algorithm can be modified to handle a polynomial number of lies in the counterexamples provided by the environment.

Finally, we introduce a new complexity measure that better captures the complexity of the union of m boxes than simply the number of boxes and the dimension. Our new measure, σ, is the number of segments in the target, where a segment is a maximum portion of one of the sides of the target that lies entirely inside or entirely outside each of the other halfspaces defining the target. We present a modification of our first algorithm that uses time and queries polynomial in σ and $\log n$. In fact, the time and queries (both membership and equivalence) used by this single algorithm are polynomial for *either* m or d constant.

Key words. computational learning, geometric concepts, exact learning, membership and equivalence queries

AMS subject classifications. 68Q25, 68T05

PII. S0097539794274246

1. Introduction.

Recently, learning geometric concepts in d-dimensional Euclidean space has been the subject of much research [6, 15, 17, 28, 30, 31, 32, 34]. We study the problem of learning geometric concepts 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. To apply such a learning model to a geometric domain, it is necessary to look at a discretized (or digitalized) version of the domain. We use d to denote the number of dimensions and n to denote the number of discrete values that exist in each dimension. Thus a discretized geometric concept G is a set of

[*]Received by the editors September 6, 1994; accepted for publication (in revised form) January 12, 1997; published electronically August 4, 1998. Portions of this paper appear in preliminary form in [22] and [12].

http://www.siam.org/journals/sicomp/28-2/27424.html

[†]Department of Computer Science, University of Calgary, Calgary, AB, Canada T2N 1N4 and Department of Computer Science, Technion 32000, Haifa, Israel (bshouty@csa.technion.ac.il). The research of this author was supported in part by the NSERC of Canada.

[‡]Department of Computer Science, University of Warwick, Coventry CV47AL, UK (pwg@dcs.warwick.ac.uk). The research of this author was performed while visiting Washington University with support from NSF NYI grant CCR-9357707, and while at Sandia National Lab with support from U.S. Department of Energy contract DE-AC04-76AL85000.

[§]Department of Computer Science, Washington University, St. Louis, MO 63130 (sg@cs.wustl.edu). The research of this author was supported in part by NSF grant CCR-9110108 and NSF NYI grant CCR-9357707 with matching funds provided by Xerox PARC and WUTA.

[¶]Ohio State University, Columbus, OH 43210 (dmath@cis.ohio-state.edu). The research of this author was performed while visiting Washington University with support from NSF NYI grant CCR-9357707 with matching funds provided by Xerox PARC and WUTA.

integer points $G \subseteq \mathcal{N}_n^d$, where $\mathcal{N}_n = \{1, \ldots, n\}$. In this paper we consider discretized geometric concepts whose boundaries are defined by hyperplanes of known slope.

We begin by studying a special case: the well-studied class of unions of axis-parallel boxes. (By a "box," we mean an axis-aligned hypercuboid.) The algorithm for this special case is easily extended to learn discretized geometric concepts defined by axis-parallel hyperplanes. We use BOX_n^d to denote the class of axis-parallel boxes[1] over $\mathcal{N}_n^d$, and $\bigcup_{\leq m} \text{BOX}_n^d$ to denote the class of the union of at most m concepts from BOX_n^d. Let c be a concept in $\bigcup_{\leq m} \text{BOX}_n^d$. We say that c is defined by $s \leq m$ boxes from BOX_n^d if s is the minimum number of boxes whose union is equivalent to c. 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 halfspaces over $\mathcal{N}_n^d$.

We first present a query algorithm that exactly learns $\bigcup_{\leq m} \text{BOX}_n^d$ with at most $md + 1$ equivalence queries[2] and $O((4m)^d + md(\log n + d))$ membership queries and computation time. Thus our algorithm exactly learns the union of m discretized axis-parallel boxes over $\mathcal{N}_n^d$ in polynomial time for any constant d.

The hypothesis class used by this algorithm can be evaluated in time $O(d \log m \log n)$. Furthermore, in $O((2m)^{2d})$ time we can transform our hypothesis to the union of at most $O(md \log m)$ boxes. Thus we obtain the stronger result that our algorithm can exactly learn the union of m axis-parallel boxes over $\mathcal{N}_n^d$ while making at most $md+1$ equivalence queries, where each equivalence query is simply the union of $O(md \log m)$ concepts from BOX_n^d; making $O((4m)^d + md(\log n + d))$ membership queries, and using $O((md)^2 \log m \cdot (2m)^{2d} + md \log n)$ computation time. Thus for any constant d, this algorithm still uses time and queries polynomial in m and $\log n$. We also describe a variation of this basic algorithm that uses *only* equivalence queries and still has complexity polynomial in m for time and queries and $\log n$ for d constant. Our algorithm uses $O((8d^2 m \log n)^d)$ equivalence queries and computation time.

Next we study the problem of learning with only equivalence queries the class of discretized geometric concepts in which the hyperplanes defining the boundaries of the concept need *not* be axis parallel but rather can have any known slopes. That is, the geometric discretized concepts we study here are those whose boundaries lie on hyperplanes $\{x = (x_1, \ldots, x_d) \mid \sum_{j=1}^d a_{ij} x_j = b\}$ for $i = 1, \ldots, |S|$, where S is the set of slopes of the hyperplanes. The possible slopes of those hyperplanes, i.e., $a_i = (a_{i1}, \ldots, a_{id})$, are known to the learner, but the same slope with different shifts b can be used for many hyperplanes in the target concept g. Note that if we choose the slopes $S = \{e_i\}$, the standard basis, then we get the special case in which all hyperplanes are axis parallel.

Let $S \subset \mathcal{Z}^d$ where $\mathcal{Z}$ is the set of integers, and let $\|S\|$ denote the representation size of S (the sum of the logarithms of the absolute values of the integers in S). Let g be a geometric concept whose boundaries lie on m hyperplanes in $\mathcal{N}_n^d$ with slopes from S. A key result of this paper is that for any constant d, any such geometric concept is exactly learnable in $poly(l, m, \|S\|, \log n)$ time and equivalence queries even if the equivalence oracle lies on l counterexamples. So for example, if the space is the plane $\mathcal{N}_n^2$ and $S = \{(0,1), (1,0), (1,1), (1,-1), (1,2), (2,1), (1,-2), (2,-1)\}$, then our algorithm can efficiently learn the geometric concepts generated from lines that make angles $0, 90, 135, 45, 120,$ $150, 30,$ and 60, respectively, with the x-axis, in polynomial

[1]Note that we include in BOX_n^d boxes with zero size in any dimension.

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

time. (As this example illustrates, the representation that we use for a slope is that derived from the formula, given above, defining a hyperplane.) In higher constant dimensional space our algorithm can efficiently learn any geometric concept whose boundary slopes are known. Another generalization of this result is an algorithm to exactly learn polynomially sized decision trees over the basis "Is $x_i \succ c$," where $\succ \in \{>, <, \geq, \leq\}$ in constant dimensional space.

Finally, we reexamine our first algorithm for learning the union of m discretized boxes. 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, σ, is the number of segments in the target concept, where a segment is a maximum portion of one of the defining hyperplanes of the target that lies entirely inside or entirely outside each of the other defining hyperplanes. We show that $\sigma \leq (2m)^d$. We present an improvement of our first algorithm that uses time and queries polynomial in σ and $\log n$. The hypothesis class used by this modified algorithm is that of decision trees of height at most $2md$. Thus, observe that the hypothesis output (and the intermediate hypotheses) can be evaluated in polynomial time without any restrictions on m or d. We then use an alternate analysis of this algorithm to show that the time and queries used are polynomial in d and $\log n$ for any constant m, 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* m or d constant.

The paper is organized as follows. In section 2 we describe the learning model that we use. Next, in section 3 we summarize the previous work on learning geometric concepts. Then in section 4 we give some preliminary definitions. Section 5 describes our results for learning unions of boxes with membership and equivalence queries. We present this algorithm, in part, because it introduces the approach used to obtain our other results and also because it uses very few equivalence queries, which is of interest if one's goal is to minimize the number of prediction errors made by the learner [13]. Next, in section 6 we describe a modification of this algorithm that efficiently learns the union of boxes in constant dimensional space with only equivalence queries. In section 7 we present our extensions to learning the class of geometric concepts defined by any hyperplanes of known slopes using only equivalence queries. In section 8 we describe how to modify this algorithm to handle the situation in which there are lies in the answers to the equivalence queries. In section 9 we present our new complexity measure and describe a modification of our first algorithm that runs in polynomial time with respect to this complexity measure. Finally, in section 10 we conclude with some open problems. This paper subsumes the results presented by Goldberg, Goldman, and Mathias [22] and includes several results given by Bshouty [11].

2. Learning model.

The learning model we use in this paper is that of learning with queries developed by Angluin [1]. When applied to our class of discretized geometric concepts, the learner's goal is to learn *exactly* how an unknown target concept, g, drawn from the *concept class* $\mathcal{G} \subseteq 2^{\mathcal{N}_n^d}$, classifies as positive or negative all instances from the *instance space* $\mathcal{N}_n^d$. Thus each concept is the set of instances from $\mathcal{N}_n^d$ that it classifies as positive. We say that $y \in \mathcal{N}_n^d$ is a positive instance for target concept g if $y \in g$ (also denoted $g(y) = 1$) and say that y is a negative instance otherwise (also denoted $g(y) = 0$). It is often convenient to view the target concept g as the Boolean function $g : \mathcal{N}_n^d \rightarrow \{0, 1\}$. A *hypothesis* h is a polynomial-time algorithm that, given any $y \in \mathcal{N}_n^d$, outputs a prediction for $g(y)$. Throughout we use y to denote an instance (i.e., a point in $\mathcal{N}_n^d$) and $x = (x_1, \ldots, x_d)$ to denote the variables associated with the d axes. For example, for $d = 2$, $2x_1 + 3x_2 = 7$ defines a two-dimensional hyperplane. For the point $y = (2, 1)$, y is on this hyperplane.

As mentioned above, the learning criterion in this paper is that of *exact identification.* In order to achieve exact identification, the learner's final hypothesis, h, must be such that $h(y) = g(y)$ for all instances $y \in \mathcal{N}_n^d$. To achieve this goal the learner is provided with two types of queries with which to learn about g. A *membership query*, MQ(y), returns "yes" if $g(y) = 1$ and returns "no" if $g(y) = 0$. We note that learning using only membership queries is quite difficult for many concept classes. For example, to learn a single positive point in an n^d discrete grid requires $O(n^d)$ membership queries. An *equivalence query*, EQ(h), returns "yes" if h is logically equivalent to g or returns a counterexample otherwise. A *positive counterexample* y is an instance such that $g(y) = 1$ and $h(y) = 0$. Similarly, a *negative counterexample* is such that $g(y) = 0$ and $h(y) = 1$. Equivalence queries are answered by a computationally unbounded adversary with knowledge of the target concept and the learning algorithm. Several of the algorithms we present use only equivalence queries. The others use *both* membership and equivalence queries. In some domains, presenting the learner with a single positive instance (as a counterexample to an equivalence query) makes learning with membership queries feasible (see the discussion of *geometric probing* in section 3).

An exact learning algorithm is said to run in polynomial time if the computation time and the number of queries are polynomial in both the size[3] of an example (whether it is for a membership query or a counterexample from an equivalence query) and the size of the target concept. Here an example is one of the n^d points in $\mathcal{N}_n^d$ and thus it can be represented with $d\lceil \lg n \rceil$ bits. Each box in the target concept can be encoded with $2d\lceil \lg n \rceil$ bits, and thus encoding the entire target concept uses $2dm\lceil \lg n \rceil$ bits. Thus for the problems studied here, a polynomial-time algorithm must use time and queries polynomial in $d, \log n$, and m.

An *l-liar* equivalence oracle is an oracle that is allowed to lie at most l times during the learning session when providing counterexamples to equivalence queries. The teacher is allowed at most l lies *total* even if multiple lies are for the same instance. This definition seeks to capture a notion of erroneous data and robust, error-tolerant algorithms. We seek a more efficient approach than testing each counterexample we receive using a membership query, especially in view of the fact that we are interested in *algorithms that may use only equivalence queries.*

Another important learning model is the PAC model introduced by Valiant [38]. 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. Given values for parameters ϵ and δ, 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 ϵ. This is in contrast to the query learning model in which the learner is required to classify correctly every instance in the instance space. The learner is permitted time polynomial in $1/\epsilon$, $1/\delta$, the size of an example, and the size of the target concept to formulate a hypothesis. The relationship between the PAC model and the query model is well understood. Angluin [1] showed that any class that is learnable using only equivalence queries is also PAC learnable. The relationship is unchanged by the addition of membership queries to each model. Blum [8] showed that PAC learnability does not imply query learnability. The concept class studied here has also been considered in the PAC

[3]By size we mean the number of bits to encode the example.

model, as summarized in the next section.

3. Previous work. The problem of learning geometric concepts over a discrete domain was extensively studied by Maass and Turán [31, 32]. One of the geometric concepts that they studied was the class BOX_n^d. They showed that if the learner is restricted to make only equivalence queries in which each hypothesis was drawn from BOX_n^d then $\Omega(d \log n)$ queries are needed to achieve exact identification [28, 31]. Auer [6] 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 Maass and Turán [30]. The best result known for learning the class BOX_n^d was provided by Chen and Maass [17]. 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 $(n, 0)$. More recently, Chen [15] 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.

Homer and Chen [25] presented an algorithm to learn the union of m rectangles *in the plane* using $O(m^3 \log n)$ queries (both membership and equivalence) and $O(m^5 \log n)$ time. The hypothesis class of their algorithm is the union of $8m^2 - 2$ rectangles. In work independent of ours, Chen and Homer [16] have improved upon their earlier result by giving an algorithm that learns any concept from $\bigcup_{\leq m} \mathrm{BOX}_n^d$ using $O(m^{2(d+1)} d^2 \log^{2d+1} n)$ equivalence queries by efficiently applying the bounded injury method from recursive function theory. This algorithm appears in [11] along with the equivalence-query algorithms presented here in sections 6 and 7. While the Chen and Homer result is very similar to our result of section 6, they use a very different technique to obtain the result. Also, our algorithm uses only $O((8d^2 m \log n)^d)$ equivalence queries.

Finally, in other independent work, Maass and Warmuth [34] have developed, as part of a more general result, an algorithm to learn any concept from $\bigcup_{\leq m} \mathrm{BOX}_n^d$ using $O(md \log n)$ equivalence queries and $O\left((md \log n)^{O(d)}\right)$ computation time. In addition their technique enables them to efficiently learn a single box in a constant dimensional space that is not axis parallel but uses slopes from a known set of slopes. Their algorithm improves upon our result of section 6, in that the number of equivalence queries has polynomial dependence on m, d, and $\log n$. However, their work does *not* give results comparable to the other results presented in this paper. Most notably, in section 7 we give an algorithm that can learn the union of non-axis-parallel boxes (using slopes from a known set of slopes) versus just learning a single box.

Closely related to the problem of learning the union of discretized boxes, is the problem of learning the union of nondiscretized boxes in the PAC model [38]. Blumer et al. [10] present an algorithm to PAC-learn an m-fold union of boxes in E^d by drawing a sufficiently large sample of size $m' = poly\left(\frac{1}{\epsilon}, \lg \frac{1}{\delta}, m, d\right)$ and then performing a greedy covering over the at most $(\frac{em'}{2d})^{2d}$ boxes defined by the sample. Thus for d constant this algorithm runs in polynomial time. Long and Warmuth [29] 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 $m(2d)^m$ boxes consistent with the sample. Thus both the time and the sample complexity of their algorithm depend polynomially on m, d^m, $\frac{1}{\epsilon}$, and $\lg \frac{1}{\delta}$. For m constant this yields an efficient PAC algorithm.

We note that either of these PAC algorithms can be applied to the class $\bigcup_{\leq m} \text{BOX}_n^d$ giving efficient PAC algorithms for this class for either d constant or m constant. As discussed by Maass and Turán [31], the task of a concept learning algorithm is to provide a "smart" hypothesis based on the data available. In other words, the hypothesis must be carefully chosen so that as much information as possible is obtained from each counterexample. To illustrate this point consider the task of learning a concept of a half interval over $\{1, \ldots n\}$ (so an example $x \in \{1, \ldots, n\}$ is positive iff $x \leq r$ where $0 \leq r \leq n$ is not known). Within the PAC model a satisfactory hypothesis would be $r = h$, where h is the maximum positive example (0 if no positive examples have been seen). However, in the exact learning model when using *only equivalence queries* this hypothesis performs very poorly in that each counterexample could just be one to the right of the last one. Thus n counterexamples may be needed. However, if one uses the "smarter" hypothesis of $r = (h+g)/2$, where h is the maximum positive example seen and g is the minimum negative example seen ($n+1$ if no negative examples have been seen), then at most $\lceil \log n \rceil$ counterexamples are needed. More generally, the results from Blumer et al. [10] show that under the PAC model any concise hypothesis that is consistent with the data is satisfactory. In other words, the PAC model provides no suitable basis for distinction among different consistent hypotheses. On the other hand, a criterion for defining a "smart" hypothesis is implicitly contained within the query 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 learn *exactly* the class $\bigcup_{\leq m} \text{BOX}_n^d$ significantly harder than obtaining the corresponding PAC result. Also Blum [8] has shown that if one-way functions exist, then there exist functions that are PAC-learnable but not exactly learnable.

Finally, under a variation of the PAC model in which membership queries can be made, Frazier et al. [21] have given an algorithm to PAC-learn the m-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 m and d. Recall that since $\bigcup_{\leq m} \text{BOX}_n^d$ generalizes DNF, a polynomial-time algorithm for arbitrary d and m 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.

While there has been some work addressing the general issue of mislabeled training examples in the PAC model [3, 27, 37, 26], there has been little research on learning geometric concepts with noise. Auer [6] investigates exact learning of boxes where some of the counterexamples, given in response to equivalence queries, are noisy. Auer shows that BOX_n^d is learnable using hypotheses from BOX_n^d 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 $1/(2d+1)$. In the query learning model, Angluin and Kriķis [2] examine the case in which membership queries can be answered incorrectly under adversarial control.

There has also been some work on learning discretized geometric concepts defined by non–axis-parallel hyperplanes. Maass and Turán [33] study the problem of learning a single discretized halfspace using only equivalence queries. They give an efficient algorithm using $O(d^2(\log d + \log n))$ queries and give an information theoretic lower bound of $\binom{d}{2}$ on the number of queries when all hypotheses are discretized halfspaces. There has also been work on learning non–axis-parallel discretized rectangles with only equivalence queries. Maass and Turán [32] show an $\Omega(n)$ information theoretic lower bound on the number of equivalence queries when the hypotheses must be drawn from the concept class. Contrasting this lower bound, Bultman and Maass [14] give an efficient algorithm that uses membership and equivalence queries to learn this class using $O(\log n)$ equivalence queries. Their algorithm returns a hypothesis consisting of a description of the vertices and edges of the polygon.

Computational geometry researchers have looked at the slightly related problem of *geometric probing* (for example, see [36]). Geometric probing studies how to identify, verify, or determine some property of an unknown geometrical object using a measuring device known as a probe. In one special case of geometric probing the aim is to construct (or learn) an unknown convex polygon given a point inside the polygon along with the ability to make a probe in which the algorithm can "shoot" a ray in a specified direction to find out the location where the ray hits the polygon. Note that using a binary search technique, such a probe can be simulated for this problem with a polynomial number of membership queries (for discretized domains). Thus such work can be thought of as learning a convex polygon from only membership queries along with a single positive example. However, when working with nonconvex objects, the probe used in such work is more powerful than a membership query.

Geometric testing (for example, see [35, 4]) is a subarea of geometric probing involved with solving verification problems. The work that has been done on this problem [5, 7, 23, 24] involves determining which of a finite set of models is being probed, and seems to be the most closely related to this paper among the geometric probing literature. The restriction to a finite set of models effectively discretizes the domain of geometric points, as is done here by considering points with bounded integer coordinates.

4. Preliminaries.

Let $\mathcal{N}, \mathcal{Z}$, and $\mathcal{R}$ be the set of nonnegative integers, integers, and reals, respectively. Recall, as discussed in section 2, that we use $x_1, \ldots, x_d$ to denote the variables associated with the d axes. Let $\mathcal{N}_n = \{1, \ldots, n\}$ and $S = \{a_1, \ldots, a_s\} \subset \mathcal{Z}^d$ be a set of slopes. A d-dimensional *hyperplane* is $\{x = (x_1, \ldots, x_d) \mid \sum_{j=1}^{d} a_{ij}x_j = b\}$ for some $a_{ij} \in \mathcal{Z}, b \in \mathcal{R}$. A *halfspace* is $\{x = (x_1, \ldots, x_d) \mid \sum_{j=1}^{d} a_{ij}x_j \succ b\}$, where $\succ \in \{>, \geq, <, \leq\}$. A *discretized hyperplane* (respectively, halfspace) is $H \cap \mathcal{N}_n^d$ for some hyperplane (respectively, halfspace) H. A *geometric concept generated from hyperplanes* with slopes from $S \subset \mathcal{Z}^d$ is a set $\hat{g} \subset \mathcal{R}^d$ whose boundaries are defined by hyperplanes with slopes from S. A *discretized geometric concept generated from hyperplanes* with slopes from $S \subset \mathcal{Z}^d$ is $g = \hat{g} \cap \mathcal{N}_n^d$.

The *complexity* $C_S(\hat{g})$ of a geometric concept $\hat{g}$ is the minimum number of hyperplanes with slopes from S such that their union contains the boundary of $\hat{g}$. The complexity $C_S(g)$ of a discretized geometric concept g is the minimum $C_S(\hat{g})$ over all geometric concepts $\hat{g}$ that satisfy $g = \hat{g} \cap \mathcal{N}_n^d$. By a simple information theoretic argument, it follows that any exact learning algorithm for a nontrivial discretized geometric concept g cannot run in time less than the complexity $C_S(g)$. Also, the complexity of learning one box in $\mathcal{N}_n^d$ is at least $\Omega(d \log n)$. The input for our algorithms is S, together with n and d. We use $\|S\|$ to denote the sum of the logarithms of the absolute values of the integers in S and call this the *size* of S. Thus, a learning algorithm, for the class of discretized geometric concepts with complexity measure $C_S(g)$, is a polynomial-time algorithm if the time and query complexities are $poly(\|S\|, C_S(g), \log n, d)$.

Observe that if we choose the slopes $S = \{e_i\}$, the standard basis, then for any discretized geometric concept g defined by the union of m boxes in d dimensional space, $C_S(g) \leq 2md$ since at most $2d$ halfspaces are needed to define the boundaries of each box.

5. Learning unions of boxes with membership and equivalence queries.

In this section we present an algorithm that exactly identifies any concept from $\bigcup_{\leq m} \text{BOX}_n^d$ (so $S = \{e_i\}$, the standard basis) while receiving at most md counterexamples and using time and membership queries that are polynomial in m and $\log n$ for any constant d. This section serves two purposes: (1) the other algorithms presented build upon this basic algorithm, and thus for ease of exposition we present it here, and (2) it uses very few equivalence queries, which is of interest if one's goal is to minimize the number of prediction errors made by the learner [13].

The following definition is used throughout this section.

DEFINITION 1. *For a discretized geometric concept g, we define a $+/^i-$ pair to be a positive point $y_+ = (y_1, \ldots, y_i, \ldots, y_d)$ paired with a negative point y_- where $y_- = (y_1, \ldots, y_i+1, \ldots y_d)$ or $y_- = (y_1, \ldots, y_i-1, \ldots, y_d)$, where we implicitly assume that any point outside $\mathcal{N}_n^d$ is a negative point.*

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 ith coordinate is c, if the negative point's ith coordinate is $c+1$, then H is given by $x_i \leq c$. Similarly, if the negative point's ith coordinate is $c-1$, then H is given by $x_i \geq c$. We define the associated *hyperplane* to be the set of all points satisfying $x_i = c$.

For each of the d dimensions, we maintain a set $\mathcal{H}_i$ of hyperplanes discovered for that dimension that define the boundaries of g. We let $\mathcal{H} = \bigcup_{i=1}^d \mathcal{H}_i$. Observe that in dimension i we have decomposed $\mathcal{N}_n^d$ into up to $2|\mathcal{H}_i|+1$ regions: $|\mathcal{H}_i|$ corresponding to the hyperplanes themselves and $|\mathcal{H}_i|+1$ corresponding to the "strips" obtained when $\mathcal{N}_n^d$ is cut by each of the $|\mathcal{H}_i|$ hyperplanes. Recall that $|\mathcal{H}_i| \leq 2m$, and thus our final hypothesis divides $\mathcal{N}_n^d$ into a *grid* $G_{\mathcal{H}}$ of

$$\prod_{i=1}^{d}(2|\mathcal{H}_i|+1) \leq \prod_{i=1}^{d}(4m+1) = (4m+1)^d \tag{5.1}$$

regions (or connected components). We say that $G_{\mathcal{H}}$ is *consistent* if for any two points, with known classification, in any region of $G_{\mathcal{H}}$, the points have the same classification. Given a consistent $G_{\mathcal{H}}$, a hypothesis is obtained by simply classifying all points according to the unique classification of all known points in that region (with negative used as a default).

We now demonstrate that we can represent such a hypothesis so that it is very efficient to evaluate. For each dimension i we maintain a balanced binary search tree T_i where each internal node corresponds to one of the hyperplanes in $\mathcal{H}_i$ and each leaf node corresponds to one of the strips created. The ith coordinate of the points in a hyperplane in $\mathcal{H}_i$ is used as the key for the associated node in T_i. For each leaf node v of T_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 such that x is the ith coordinate of some point in the region corresponding to leaf v. (For the internal nodes, the key itself serves the role of both $\min_v^i$ and $\max_v^i$.)

We define $R_i = \{[\min_v^i, \max_v^i] \mid v \text{ is a node of } T_i \text{ and } \min_v^i \text{ and } \max_v^i \text{ are the minimum and maximum values of the } i\text{th coordinates of points in } v\}$. Let $\mathcal{T} =$

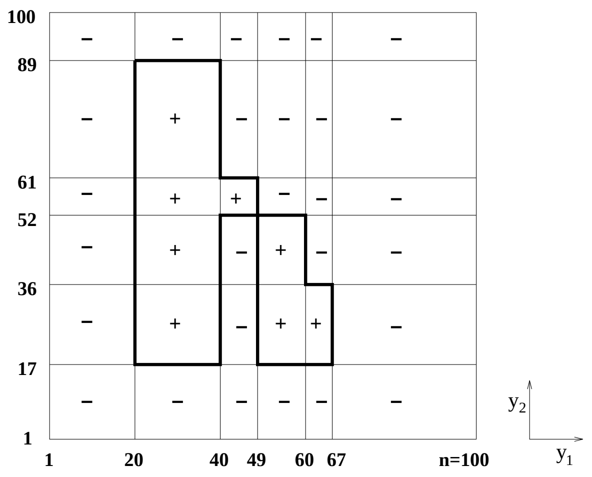

FIG. 1. *This shows the final set of regions corresponding to the target concept, formed by a union of* $m = 4$ *boxes, outlined in bold. The classification of each two-dimensional region is shown inside the region. (The classifications of the one-dimensional and zero-dimensional regions are not shown but are stored in the prediction matrix A.) The boundary of the target concept is segmented to illustrate a new complexity measure introduced in section* 9.

$\{T_1, \ldots, T_d\}$. Thus $G_{\mathcal{H}} = R_1 \times R_2 \times \cdots \times R_d$. For $r \in G_{\mathcal{H}}$, corresponding to nodes $v_i \in T_i$, we shall refer to the point $(\min^1_{v_1}, \ldots, \min^d_{v_d})$ as the *lower corner* of r and $(\max^1_{v_1}, \ldots, \max^d_{v_d})$ as the *upper corner* of r.

In addition to the trees $T_1, \ldots, T_d$, our hypothesis also maintains a prediction array A with $|G_{\mathcal{H}}|$ entries where, for $r \in G_{\mathcal{H}}$, $A[r]$ gives the classification for region r. For a consistent region (i.e., all known points have the same classification) A will contain either a 0 (for negative) or a 1 (for positive). However, for regions in which there is an inconsistency, there will be a pointer into a queue Q of inconsistent regions. In addition, for each region $r \in Q$ we store points y_+ and y_- giving a pair of inconsistent points in r. We use $h(G_{\mathcal{H}}, A)$ to denote the hypothesis defined by the regions in $G_{\mathcal{H}}$ with the classifications given in A. Note that the hypothesis is well-defined only if all regions are consistent. 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 $h(G_{\mathcal{H}}, A)$ (we sometimes denote this simply as h) for which the queue Q of inconsistent regions is empty, and a point y, we can compute $h(y)$, the prediction made by hypothesis h on point y, as follows. For $1 \le i \le d$ we perform a search for y_i in tree T_i to find the node having y_i in its range. Combining the ranges of the d nodes found defines the region $r \in G_{\mathcal{H}}$ that contains y. Finally $h(y) = A[j_1, \ldots, j_d]$, where j_i is contained in the node found in T_i during the search for y and denotes the dimension i index for A corresponding to the region that contains y. Then combining the indices obtained from each of the trees provides an index into A. Since we know an upper bound on the number of nodes in each tree, we can initially allocate enough space for A.

5.1. The membership and equivalence query algorithm. Our algorithm works by repeatedly building a consistent hypothesis that incorporates all known halfspaces. The hypothesis is represented as a decision tree rather than as a union of boxes; subsequently we show how to express it as a union of (not too many) boxes.

Given an inconsistent hypothesis $h(G_{\mathcal{H}}, A)$ and the queue Q of inconsistent regions from $G_{\mathcal{H}}$, we refine $h(G_{\mathcal{H}}, A)$ so that it is consistent using the following procedure that uses membership queries to find *new* hyperplanes with which to modify the hypothesis. We never remove any hyperplane from $\mathcal{H}$ and search for a new hyperplane only 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 inconsistent region of $G_{\mathcal{H}}$. Note that we never make an equivalence query using an inconsistent hypothesis.

Our procedure to build a consistent hypothesis repeatedly does the following until Q is empty (and thus the hypothesis is consistent). Let r be the region at the front of the queue. Since r is not a consistent region, we know that there must be some unknown hyperplane of g that goes between y_+ (a known positive point in r) and y_- (a known negative point in r). Thus we can perform a binary search between y_+ and y_- (where the comparisons are replaced by membership queries) to find a $+/-$ pair contained within region r using only $d + \lceil \log n \rceil$ membership queries. We use $\lceil \log n \rceil$ queries to find a positive point and a negative point whose positions differ by at most one in each coordinate by performing a single binary search on all d dimensions. (Query points are chosen by bisecting the range in each dimension; noninteger coordinates may be rounded arbitrarily.) We then use d additional queries to determine the slope of the separating hyperplane. 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).

The full details of procedure ADD-HYPERPLANE, which modifies $h(G_{\mathcal{H}}, A)$ to incorporate the new hyperplane found, is given in Figure 2. The learner begins by using a standard tree insertion procedure to insert the new hyperplane into the search tree for the appropriate dimension. Then the set of regions that have been split by the hyperplane are deleted from $\mathcal{H}$, and each is replaced by three new regions (one of them being a degenerate region corresponding to the hyperplane itself). For each new region of r we make a membership query on the lower and/or upper opposing corners if those queries have not already been made. As we shall discuss in section 6, this step can be replaced by just using 0 as the default value for $A[r]$. If the classifications of these two corners are the same, then the classification is entered in $A[r]$; otherwise the region is placed in Q with these corners used for y_+ and y_-.

Our algorithm LEARN-WITH-MQS works as follows. For ease of exposition we artificially extend the instance space from $\mathcal{N}_n^d$ 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 with a procedure that first checks for such cases.) Initially, $G_{\mathcal{H}}$ just contains the single region corresponding to the entire instance space. Since the upper and lower 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 y be the counterexample received from an equivalence query made with a consistent hypothesis. Using membership queries (in the form of a binary search) we can find two new hyperplanes of the target concept. Without loss of generality, we assume that y is a positive counterexample in region r of $G_{\mathcal{H}}$. Since the hypothesis was consistent and y 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 y) as the endpoints for binary searches to discover two new hyperplanes. We find a hyper-

```
ADD-HYPERPLANE($h(G_{\mathcal{H}}, A), Q, i, c$)
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 $R_{delete} = R_1 \times \cdots \times R_{i-1} \times \{[\min_v^i, \max_v^i]\} \times R_{i+1} \times \cdots \times R_d$
Let $R_{add} = R_1 \times \cdots \times R_{i-1} \times \{[\min_v^i, c-1], [c,c], [c+1, \max_v^i]\} \times R_{i+1} \times \cdots \times R_d$

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 new queue nodes for $r_=$, $r_<$, and $r_>$
        Set the corresponding entries of $A$ to point to these new nodes
        Copy $y_-$ and $y_+$ in the appropriate queue entries for $r_<$ and $r_>$
        Remove $q$ from $Q$
    For each $r' \in \{r_=, r_<, r_>\}$
        Let $y_u$ (respectively, $y_\ell$) be the upper (respectively, lower) corner of $r'$
        Make a membership query to determine $y_u$ and $y_\ell$ if not already known
        If $g(y_u) = g(y_\ell)$ then let $A[r'] = g(y_u)$
        Else ($r'$ is inconsistent)
            Assign $y_u$ and $y_\ell$ to $y_-$ and $y_+$ as appropriate
            $Q$.ENQUEUE($r'$)
```

FIG. 2. *Our subroutine to update $h(G_{\mathcal{H}}, A)$ to incorporate the newly discovered hyperplane $x_i = c$. The new hyperplane is added to tree T_i. Then all regions in $G_{\mathcal{H}}$ that are split are removed from Q. Finally this procedure initializes the new entries of $G_{\mathcal{H}}$ in the prediction matrix A. Note that in the for loop that adds new regions, we could choose not to perform membership queries on the upper and lower corners of r, as is done in section* 6. *Instead, we could just use the known points in $r_=$, $r_<$, and $r_>$ to determine their labels (with* 0 *as a default if there are no points in the region). The advantage of doing this step is that the number of equivalence queries is reduced by a factor of* 2 *since two hyperplanes of the target are guaranteed to be found for each counterexample (Lemma* 1*). If we find that the opposing corners have different classifications then we place that newly created region on Q, which in turn will cause a binary search to be done to find a new hyperplane and split that region further.*

plane by doing a binary search until we find a $+/-$ pair (two points that are labeled differently and differ by at most one in each dimension). We know that a side of a target box must pass between these two points (or through one of the points). We then use d additional queries to discover the slope of this hyperplane. The hypothesis is updated using ADD-HYPERPLANE to incorporate these two hyperplanes. Finally, we call MAKE-CONSISTENT-HYPOTHESIS to refine any inconsistent regions. Figure 3 gives the complete algorithm.

5.2. Analysis. We now analyze the time and query complexity of LEARN-WITH-MQS. 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. Let y be the counterexample and $r \in G_{\mathcal{H}}$ be the region containing y. Since $h(G_{\mathcal{H}}, A)$ is a consistent hypothesis, we know that the upper and lower corners of r

```
LEARN-WITH-MQS:
 Let $Q \leftarrow \emptyset$
 Let $(0,\ldots,0)$ and $(n+1,\ldots,n+1)$ be negative corners of single region $r$
 For $1 \le i \le d$
     Initialize $T_i$ to be a single leaf covering the range 0 to $n+1$
 For $r$ the single region of $G_{\mathcal{H}}$, let $A[r] = 0$

 While Equiv($h(G_{\mathcal{H}}, A)$) $\neq$ "yes"
     Let $y$ be the counterexample where $y$ is in region $r$ of $h(G_{\mathcal{H}}, A)$
     Let $z_1$ and $z_2$ be the lower and upper opposing corners of $r$
     For $1 \le \ell \le 2$
         Perform binary search between $y$ and $z_\ell$ to find hyperplane $x_i = c$
         ADD-HYPERPLANE($h(G_{\mathcal{H}}, A), Q, i, c$)
     $h(G_{\mathcal{H}}, A) \leftarrow$MAKE-CONSISTENT-HYPOTHESIS($h(G_{\mathcal{H}}, A), Q$)
 Return $h(G_{\mathcal{H}}, A)$

MAKE-CONSISTENT-HYPOTHESIS($h(G_{\mathcal{H}}, A), Q$)
 While $Q \neq \emptyset$
     $r \leftarrow Q$.DEQUEUE
     Perform binary search between $y_-$ and $y_+$ from $r$
        to find the hyperplane $x_i = c$
     ADD-HYPERPLANE($h(G_{\mathcal{H}}, A), Q, i, c$)
```

FIG. 3. *Algorithm for learning unions of d-dimensional axis-parallel boxes using membership and equivalence queries. Note that i, in the calls to* ADD-HYPERPLANE, *is the dimension separated by the hyperplane found in the binary search.*

are classified opposite y and all points in r are classified opposite y by the hypothesis. Since a positive point and a negative point must be separated by some hyperplane of the target concept, searches between y and each of the upper and lower corners 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* $g \in \bigcup_{\le m} \text{BOX}_n^d$, LEARN-WITH-MQS *achieves exact identification of* g *making at most* $md+1$ *equivalence queries, and using* $O((4m)^d + md(\log n + d))$ *time and membership queries.*

Proof. The correctness of LEARN-WITH-MQS is trivial. Since the algorithm only returns a hypothesis $h(G_{\mathcal{H}}, A)$ for which Equiv($h(G_{\mathcal{H}}, A)$) returns "yes," the algorithm is correct upon returning a hypothesis.

We now analyze the query and time complexity of LEARN-WITH-MQS. Recall that since there are only m boxes in the target concept, there are at most $2md$ hyperplanes in the final hypothesis. Clearly, any box can be subdivided into a union of smaller boxes, unnecessarily increasing the complexity of the target. However, our algorithm adds hyperplanes only where there is evidence for the existence of a side of a target box (a $+/-$ pair) and, therefore, does not *unnecessarily* subdivide boxes (subdividing does occur due to extending hyperplanes through the entire domain, but this is already accounted for in our analysis). Furthermore, since no hyperplane is ever rediscovered and every binary search (which uses $O(\log n + d)$ membership queries) discovers a hyperplane, we know that $O(md(\log n + d))$ membership queries are used during all of the binary searches made by the algorithm. Also, as given in (5.1), there are at most $(4m+1)^d$ regions in the final hypothesis, and thus the number of membership queries used for querying the upper and lower opposing corners is at most $2 \cdot (4m+1)^d = O((4m)^d)$. Since these are the only two places in which membership queries are performed, the total number of membership queries made by our algorithm is $O((4m)^d + md(\log n + d))$.

From Lemma 1 we know that each counterexample enables LEARN-WITH-MQS to find at least two new, distinct hyperplanes of the target concept. Since there are at most $2md$ hyperplanes comprising the m boxes, at most md counterexamples can be received and thus at most $md + 1$ equivalence queries are made.

The time needed to evaluate $h(G_{\mathcal{H}}, A)(x)$ for an unlabeled example y is $O(d \log m)$ since the key operation is performing d searches in balanced search trees of depth $O(\log m)$. Thus, the time complexity of this algorithm is found to be $O((4m)^d + md(\log n + d))$. $\square$

Finally, it is easily seen that this algorithm extends to learning any discretized geometric concept generated by hyperplanes with slopes from $S = \{e_i\}$ (the standard basis) while receiving at most $C_S(g)/2$ counterexamples and using time and membership queries polynomial in $C_S(g)$ and $\log n$ for d any constant.

5.3. Using a hypothesis class of unions of boxes. We now describe how a consistent hypothesis can be converted to the union of $O(md \log m)$ boxes from BOX_n^d. Since all equivalence queries are made with consistent hypotheses, such a conversion enables our algorithm to learn the union of m boxes from BOX_n^d using as a hypothesis class the union of $O(md \log m)$ boxes from BOX_n^d. Note that it is NP-hard to find a minimum covering of a concept from $\bigcup_{\leq m} \text{BOX}_n^d$ by individual boxes [19].

Recall that a consistent 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 [18] to perform the conversion. The set X of objects to cover will simply contain all positive regions in h. Thus $|X| \leq (4m+1)^d$. 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 $|\mathcal{F}| \leq (2m)^{2d}$ since there are at most $2m$ values in each dimension that can form the two sides of the box. Furthermore, $\mathcal{F}$ contains a subset of size m that covers all items in X. Finally, we can apply the greedy set covering heuristic to find a set of at most $m(\ln |X| + 1) = m(d \ln(4m+1) + 1) = O(md \log m)$ boxes that cover all positive regions. The time to perform the conversion is $O((2m)^{2d} md \log m)$. Thus, since at most $md + 1$ equivalence queries are made, the total time spent in converting the internal hypotheses into hypotheses that are unions of boxes is at most $O((md)^2 \log m \cdot (2m)^{2d})$.

6. Learning unions of boxes using only equivalence queries. We now describe a simple method to remove the use of membership queries in LEARN-WITH-MQS. First observe that the use of membership queries in this algorithm can easily 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 known point 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 the region that can be used for the binary search. (Of course, this modification dramatically increases the number of equivalence queries used.)

Now to remove the use of membership queries in a binary search between y_+ and y_- we simply take the midpoint between y_+ and y_- (i.e., the first point on which a membership query would be made) and insert a hyperplane going through that point for each of the d dimensions. There are $2dm$ hyperplanes defining the target. We require at most $2\lceil \log n \rceil$ counterexamples (that cause $\mathcal{H}$ to be modified) to find each hyperplane. Thus, the total number of counterexamples required (that cause $\mathcal{H}$ to be modified) is at most $4dm\lceil \log n \rceil$. For each such counterexample (i.e., one that causes $\mathcal{H}$ to be modified) we insert d hyperplanes in the hypothesis. Thus the number of hyperplanes in the hypothesis is at most $4d^2m\lceil \log n \rceil$. By (5.1) there are at most $(8d^2m\lceil \log n \rceil + 1)^d$ regions in the final hypothesis. There is at most one equivalence query made for each of the hyperplanes found and at most one equivalence query made for each region. Thus we obtain the following result.

THEOREM 2. *Given any $g \in \bigcup_{\leq m} \text{BOX}_n^d$, there is an algorithm that achieves exact identification of g using $O((8d^2m \log n)^d)$ equivalence queries (and* no *membership queries). The time complexity is $O((8d^2m \log n)^d)$.*

It is easily seen that for d constant this algorithm exactly learns $\bigcup_{\leq m} \text{BOX}_n^d$ *using only equivalence queries* with both time and the number of equivalence queries polynomial in m and $\log n$.

7. Extending S to arbitrary known slopes.

We now present a modification of the equivalence query algorithm described in section 6 that handles the situation in which S can be an arbitrary set of known slopes versus just being the standard basis. We let s denote the number of distinct slopes in S (i.e., $s = |S|$).

Let $S = \{a_1, \ldots, a_s\} \subset \mathcal{Z}^d$ be a set of slopes. Thus $a_i = (a_{i1}, a_{i2}, \ldots, a_{id})$ (for $1 \leq i \leq s$) defines a slope for a d-dimensional hyperplane, that is, a hyperplane of the form $\sum_{j=1}^d a_{ij}x_j = b$ (or equivalently $a_i x^T = b$) for any real constant b. Let $B = \{B_1, \ldots, B_s\}$, where $B_i \subset \mathcal{R}$ defines a set of $|B_i|$ possible values for the constant term "b" for those hyperplanes with slope given by a_i. Thus together S and B define a set of hyperplanes

$$\mathcal{H} = \{a_i x^T = b \mid x = (x_1, \ldots, x_d),\ i = 1, \ldots, s,\ b \in B_i\}.$$

The hypothesis $h = h(G_{\mathcal{H}}, A)$ is that which classifies each $x \in \mathcal{N}_n^d$ according to the unique classification of all known points (if any) in the region of $G_{\mathcal{H}}$ containing x. If there are no known points in the region containing x, then negative is used as the default.

As in our basic equivalence query algorithm, this algorithm begins with the entire region classified as negative. After the first two equivalence queries are made (the first with $h = \emptyset$ and the second with $h = \mathcal{N}_n^d$), the algorithm will have a positive counterexample y and a negative counterexample u. Thus the straight line between y and u must intersect one of the hyperplanes that define g. Let $v = (y + u)/2$ be the midpoint of the line between y and u. Without the ability to make membership queries it is not possible to find a $+/-$ pair. Furthermore, even if we could find a $+/-$ pair, we would not be able to determine the slope of the hyperplane that created that $+/-$ pair. As in the previous section, we address this problem by adding to our set of hyperplanes $\mathcal{H}$ a hyperplane passing through the midpoint v for *each* slope in S. We repeatedly use this process until an equivalence query made with $h(G_{\mathcal{H}}, A)$ is correct.

For ease of exposition, in this section we will not discuss the details of how to represent $\mathcal{H}$ so that $h(G_{\mathcal{H}}, A)$ can be efficiently evaluated (in terms of all parameters).

```
LEARN-GENERAL-SLOPES($S = \{a_1, \ldots, a_s\}$)
 $\mathcal{H} \leftarrow \emptyset$
 Let $r$ be the single region of $G_{\mathcal{H}}$, let $A[r] = 0$
 While Equiv($h(G_{\mathcal{H}}, A)$) $\neq$ "yes"
     Let $y$ be the counterexample where $y$ is in region $r$ of $h(G_{\mathcal{H}}, A)$
     $A[r] = 1 - h(G_{\mathcal{H}}, A)(y)$
     If there is a known point $u$ in $r$ classified opposite $y$
         $v = (y + u)/2$
         For $i = 1$ to $s$
             $\mathcal{H} \leftarrow \mathcal{H} \bigcup (a_i x^T = b = (a_i v^T))$
         Update $A$ to incorporate the new regions created
 Return $h(G_{\mathcal{H}}, A)$
```

FIG. 4. *Algorithm for exactly learning a discretized geometric concept defined by slopes in S using only equivalence queries.*

However, the technique of section 5 of using s balanced search trees, one for each element of S, generalizes in an obvious manner.

Our algorithm, at a high level, is shown in Figure 4. In this algorithm, LEARN-GENERAL-SLOPES, S is the set of the possible slopes of the hyperplanes. We initialize $\mathcal{H}$ to be the empty set and the classification of the single region to be 0. (And thus the initial hypothesis, $h(G_{\mathcal{H}}, A)$, is simply the always false hypothesis.) We ask the equivalence query $h(G_{\mathcal{H}}, A)$ and use the counterexample y to update A. If this counterexample is the first counterexample in its region then we just update A. Otherwise, if this counterexample is in some region for which we have already seen a point u, then y and u have different classifications in g and the line that passes through y and u must intersect a defining hyperplane of g. We then define $v = (y + u)/2$ and add all possible hyperplanes that pass through v to the set $\mathcal{H}$. We then repeat this process until $h(G_{\mathcal{H}}, A)$ is logically equivalent to g.

7.1. Analysis. We now prove the correctness of our algorithm and analyze its complexity. We use the following lemma to bound the maximum number of regions that will be contained in $G_{\mathcal{H}}$.

LEMMA 2. *Any t d-dimensional hyperplanes in a $(d+1)$-dimensional space divide the space into at most $t^{d+1} + 1$ regions.*

This result is well known. See Edelsbrunner [20] for a proof. We are now ready to analyze our algorithm LEARN-GENERAL-SLOPES.

THEOREM 3. *Let S be a set of slopes. Then* LEARN-GENERAL-SLOPES *exactly learns any target concept g from the class of discretized geometric concepts generated from hyperplanes with slopes from S using time and equivalence queries polynomial in $\|S\|, C_S(g)$, and $\log n$ for any constant d.*

Proof. The correctness follows trivially since the algorithm returns only a hypothesis $h(G_{\mathcal{H}}, A)$ for which Equiv($h(G_{\mathcal{H}}, A)$) returns "yes," hence the algorithm is correct upon returning a hypothesis.

We now analyze the query and time complexity. Let $m = C_S(g)$. Recall that $\|S\|$ denotes the sum of the logarithms of the absolute values of the integers in S. Thus, by the definition of $\|S\|$, any $a_i \in S$ can be represented using at most $\|S\|$ bits. Thus there are at most $2^{\|S\|}$ possible values for a slope. Also, any example $x \in \mathcal{N}_n^d = \{1, \ldots, n\}^d$ and thus has n^d possible values. Let $H_1, \ldots, H_m$ be a minimum-size set of hyperplanes

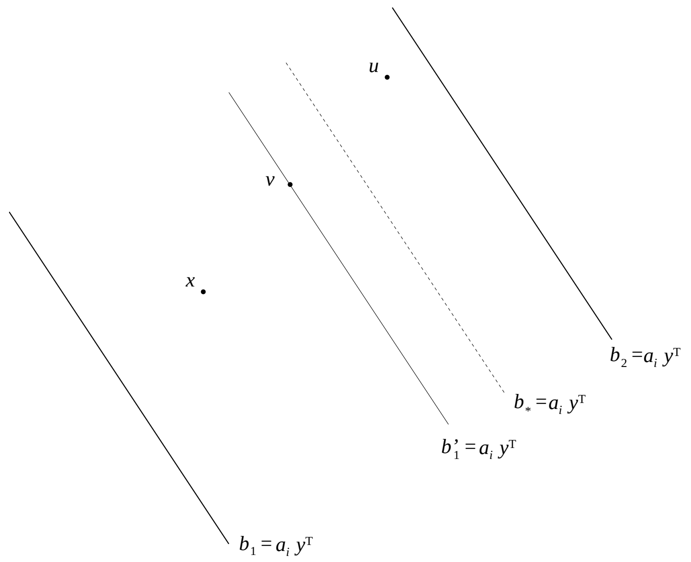
u
v
x
$b_2 = a_i\, y^{\mathrm{T}}$
$b_* = a_i\, y^{\mathrm{T}}$
$b_1' = a_i\, y^{\mathrm{T}}$
$b_1 = a_i\, y^{\mathrm{T}}$

FIG. 5. *Here y is a counterexample to the current hypothesis and $b_1 = a_i x^T$ and $b_2 = a_i x^T$ are the nearest hyperplanes. The hyperplane $b_1' = a_i x^T$ is added to $\mathcal{H}$ by* LEARN-GENERAL-SLOPES.

with slopes from S that generate the boundary of the geometric concept g. Since each hyperplane defining g is of the form $b = a_i x^T$, we have that the maximum number of values that any b can have is

$$\|b_i\| \le \|a_i x^T\| \le 2^{\|S\|} \cdot n^d = \gamma.$$

At any time during execution of our algorithm, for each of $H_1, \ldots, H_m$, there are two closest hyperplanes in $\mathcal{H}$ (one for each side) with the given slope. (At the beginning of execution, imagine two hyperplanes with each slope outside the domain and anchored at opposing corners of the domain.) We now show that for every counterexample that causes $\mathcal{H}$ to be modified there exists some H_i ($i = 1, \ldots, m$) for which the distance between it and one of its closest hyperplanes is reduced by at least a factor of 2.

Suppose that y is a counterexample to the current hypothesis $h(G_{\mathcal{H}}, A)$ and that the region r of $G_{\mathcal{H}}$ that contains y already contains the point u (otherwise $\mathcal{H}$ is not modified). By the definition of $h(G_{\mathcal{H}}, A)$ it follows that $g(u) \neq g(y)$. Therefore the line segment between y and u must intersect some hyperplane—say, $H_* \equiv (a_i x^T = b_*)$—of g for $a_i \in S$ and $b_* \in \mathcal{Z}$. Let $b_1 = a_i x^T$ and $b_2 = a_i x^T$ be the two hyperplanes with slope a_i nearest H_* in $\mathcal{H}$. Let $v = (y + u)/2$ be the midpoint between y and u. Without loss of generality we assume that the hyperplane $b_1' = a_i x^T$ that passes through v with slope a_i is between $b_1 = a_i x^T$ and $b_* = a_i x^T$ as illustrated in Figure 5. We denote the hyperplane that passes through y (respectively, u) with slope a_i by $b_y = a_i x^T$ (respectively, $b_u = a_i x^T$). Thus we have that $b_1 \le b_y \le b_1' \le b_* \le b_u \le b_2$.

Let $\Delta = b_* - b_1$. (Thus Δ is proportional to the distance between H_* and $b_1 = a_i x^T$.) Observe that LEARN-GENERAL-SLOPES will add the hyperplane $b_1' = a_i x^T$ to $\mathcal{H}$, and this will now replace $b_1 = a_i x^T$ as a closest hyperplane of slope a_i to H_*. Let $\Delta' = b_* - b_1'$. (Thus Δ' is proportional to the new distance between H_* and its closest hyperplane in $\mathcal{H}$ in that direction.) We now show that $\Delta' \le \Delta/2$. Observe that

$$\Delta' = b_* - b_1'$$

$$= b_* - \left(\frac{(b_u + b_y)}{2}\right)$$
$$\leq b_* - \left(\frac{(b_* + b_1)}{2}\right)$$
$$= \frac{b_* - b_1}{2} = \frac{\Delta}{2}$$

Thus the distance between H_* and one of its two nearest hyperplanes is reduced by a factor of two. Finally, when the distance between H_* and both of its two nearest hyperplanes is less than 1, the algorithm has determined the discretized hyperplane H_* and no other hyperplanes will be added for H_*.

Since the distance between each of $H_1, \ldots, H_m$ with both of its closest hyperplanes in $\mathcal{H}$ is at most γ it follows that the number of counterexamples needed to find one hyperplane is

$$2\lceil \log(\gamma) \rceil = 2\lceil \log(2^{\|S\|} \cdot n^d) \rceil = O(\|S\| + d \log n).$$

The number of hyperplanes is $m = C_S(g)$, and at each iteration we add $s = |S| \leq \|S\|$ hyperplanes to the hypothesis. Therefore the number of hyperplanes generated by our algorithm is $O(ms(\|S\| + d \log n))$. Thus by Lemma 2 the number of regions of the hypothesis is

$$O(ms(\|S\| + d \log n))^d.$$

The number of iterations that do not add any hyperplanes is bounded by the number of regions; thus the number of equivalence queries made by Learn-General-Slopes is

$$O(ms(\|S\| + d \log n))^d,$$

and clearly the time complexity is also polynomial in m, $\|S\|$, and $\log n$ for any constant d as desired. ∎

Finally, we note that since any slope in the discretized space $\mathcal{N}_n^d$ can be defined by two points from the domain, there are at most n^{2d} possible values for any one of the s slopes and thus $\|S\| \leq 2ds \log n$.

8. Handling lies in the counterexamples. In this section we consider the case in which the learner is provided with an l-liar teacher. Recall that an l-liar teacher is a teacher that can provide incorrect counterexamples as answers to at most l equivalence queries. This noise is not persistent. That is, if the teacher provides the same incorrect counterexample twice, then it counts as two lies (even if both were in response to the same query). Note that our learner in this model is not given access to a membership oracle. Thus, to learn the true classification of the points about which the environment has lied, it is necessary for the learner to isolate these instances. That is, the learner must create a region that consists of the single point that was the counterexample. This will force the teacher to give, eventually (after its l lies are exhausted), a correct classification for this instance if it is not already correctly classified by the hypothesis.

A degenerate region (a region consisting of fewer than d dimensions) is created whenever our algorithm adds to the hypothesis a hyperplane, passing through a given point, for each slope in S. We want to isolate the counterexample on which the lie occurred in a region of dimension 0. Notice, however, that given some set of slopes, it is possible that the hyperplanes defined by the slopes, passing through the counterexample, do not create a 0-dimensional region.

Therefore, to ensure that the 0-dimensional region is created, we include in S the slopes of the elementary vectors $\{e_i\}$. Then the faulty points will be bounded by hyperplanes and at the end the oracle must give their real values. Thus, the main difference between our algorithm in this model and in the noise free model is the addition of hyperplanes through the counterexamples. Through the midpoints of $+/-$ pairs we still add only $|S|$ hyperplanes. It is only through the counterexamples that we add $|S| + d$ hyperplanes. This changes $C_S(g)$ to $C_S(g) + dl$, the number of slopes to $|S| + d$, and the complexity to

$$O((C_S(g) + dl)(|S| + d)(\|S\| + d + d\log n))^d.$$

9. A return to learning unions of boxes. Observe that by extending the hyperplane defined by a $+/-$ pair across the entire domain, our initial algorithm Learn-With-MQs may unnecessarily split a consistent 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 splitting only the region in which the counterexample is contained. We show that this algorithm runs in polynomial time for *either* m or d constant.

We begin by examining how one might measure the complexity of a concept from $\bigcup_{\le m} \text{BOX}_n^d$. Observe that the minimum number m of boxes 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 in Figure 6. While both targets are composed of six 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 dimension of the target concept. We now introduce such a new complexity measure, σ, to better capture the complexity of the target concept. We define a *segment* of the target concept g as a *maximal portion* of one of the sides of g that lies either entirely inside or entirely outside of each of the other halfspaces defining the target. (A halfspace defines the target if it labels some $+/-$ pair consistently with the labeling given by the target. See the definition of $+/-$ pair in Definition 1.) Note that two adjacent segments will intersect in a region of dimensionality less than the segments. We observe that there exist other possible measures of complexity besides σ that are potentially worthy of study. One example is the number of sides of a target polyhedron, where a side is a maximal connected set of coplanar boundary points of the polyhedron such that the interior of the polyhedron is on the same side of the set of boundary points. Sides (as opposed to segments) have the drawback of being of varying complexity (at least for polyhedra of more than two dimensions), as well as being apparently harder to work with. That is, we can have $0, 1, \ldots, d-1$ dimensional sides.

For example the target concept shown in Figure 1 has 14 sides. The same target concept has 22 segments. (The heavy black line defining the target concept is split into 20 clearly visible segments by the thin lines that correspond to the halfspaces that define the target concept. The final 2 segments come from the fact that the points $(20, 52)$ and $(49, 52)$ are segments due to the halfspaces $y_2 \ge 52$ and $y_2 \le 52$.) The polygon in Figure 6(a) has 60 segments, since each of the long edges of the 6 boxes contributes 4 segments. By contrast the polygon in Figure 6(b) (defined by the same number of boxes) has 24 segments, since none of the sides of the individual boxes straddles any of the hyperplanes defining the boxes.

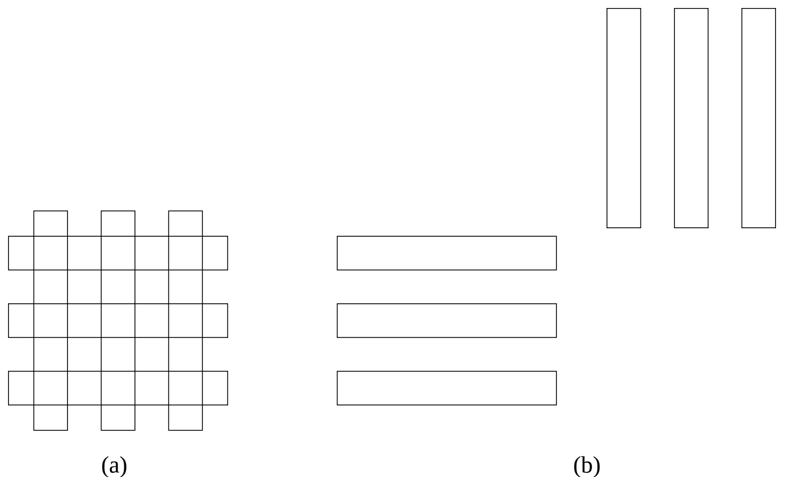

FIG. 6. *These concepts illustrate why "number of boxes" is an inadequate measure of complexity.* (a) *is more complex than* (b) *because of the large number of intersections of the sides of the boxes. For* (a) $\sigma = 60$. *For* (b) $\sigma = 24$.

For a concept $g \in \bigcup_{\leq m} \text{BOX}_n^d$ but not in $\bigcup_{\leq m-1} \text{BOX}_n^d$, we let σ denote the number of segments in the target concept corresponding to g. (In other words, suppose that m is the minimum number of boxes whose union is g.) First, observe that $m \leq \sigma$ since none of the boxes are contained within the union of the other $m-1$ boxes (which follows from the minimality of m). Furthermore, observe that $\sigma \leq (2m)^d$ since there are $2m$ halfspaces per dimension (one corresponding to each of the $2m$ sides per dimension) that can intersect to form at most $(2m)^d$ connected regions each of which would be part of at most one segment. Finally, it then follows that for any $g \in \bigcup_{\leq m} \text{BOX}_n^d$, the number of segments in $g \leq (2m)^d$ since the minimum number of boxes whose union is g is at most m.

The hypothesis class we use in this algorithm is a decision tree over the halfspaces defining the target concept. Namely, each hypothesis T is a rooted binary tree where each internal node is labeled with a halfspace and in which 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 the right subtree otherwise. When a leaf is reached its label is output. Observe that each node of T corresponds to a subregion 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 subregions, with the left child being the subregion given by $\overline{H} \cap r$ and the right child being the subregion given by $H \cap r$. The leaves correspond to a 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. In Observation 1 we show that the height of the final decision tree will be at most $2md$. Thus the hypothesis can be evaluated in time polynomial in both m and d.

We now describe our algorithm. It has complexity polynomial in σ and $\log n$ as well as being an efficient algorithm for either m or d constant. We initialize T to be a single 0 leaf node. (Again we implicitly use the instance space $\{0,1,\ldots,n,n+1\}^d$.) When a counterexample is received, we first search T to find the leaf v containing it. Let r be the subregion corresponding to v. Then as in LEARN-WITH-MQS we use a binary search to find a $+/-$ pair contained in r that defines a halfspace H. We

```
ALT-LEARN-WITH-MQS
 Initialize $T$ to be the single 0-leaf
 While Equiv($T$) $\neq$ "yes"
     Let $y$ be the counterexample
     Search in $T$ to find the leaf $v$ corresponding to the region containing $y$
     $T \leftarrow$SPLIT-REGION$(T, v, y)$
 Return $T$

SPLIT-REGION$(T, v, y)$
$\triangleright T$ is the decision tree that defines the current hypothesis
$\triangleright y$ is a counterexample in a region $r$ defined by $T$
$\triangleright v$ is the leaf of $T$ that corresponds to $r$
 Perform binary search between $y$ and a corner of region $r$
 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 $r'$
   Call these corners upper and lower
   If both corners have classification $b \in \{0, 1\}$ let $v'$ be a $b$-leaf of $T$
     Let the predicted label of $v'$ become $b$
   Else
     Let $v'$ be a leaf of $T$ with $v'$ having the same classification as lower
     SPLIT-REGION$(T, v', upper)$
```

FIG. 7. *Alternate algorithm for learning unions of d-dimensional axis-parallel boxes that runs in polynomial time for* either *m or d constant. We note that the corner points of regions in the hypothesis, used throughout the algorithm, are easy to compute given the hypothesis.*

replace v with an internal node labeled with H, having left child leaf v_L corresponding to the region given by $\overline{H} \cap r$ and right child leaf 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 consistent region. If the region r' associated with leaf v' is consistent (a region is consistent if its opposing corners have the same classification), then the classification field is filled; otherwise we use a binary search to obtain a halfspace H' for r' and replace v' with 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 7. One possible final hypothesis that could be constructed by this algorithm, for the target concept shown in Figure 1, is shown in Figure 8. Figure 9 shows the decomposition of $\mathcal{N}_n^d$ that corresponds to the decision tree shown in Figure 8.

9.1. Analysis. We first show that the height of the final decision tree comprising our hypothesis is at most $2md$.

OBSERVATION 1. *The height of the final decision tree constructed by* ALT-LEARN-WITH-MQS (*shown in Figure* 7) *is at most* $2md$ *since each of the at most* $2md$ *halfspaces defining the target polyhedra can appear at most once on any path from*

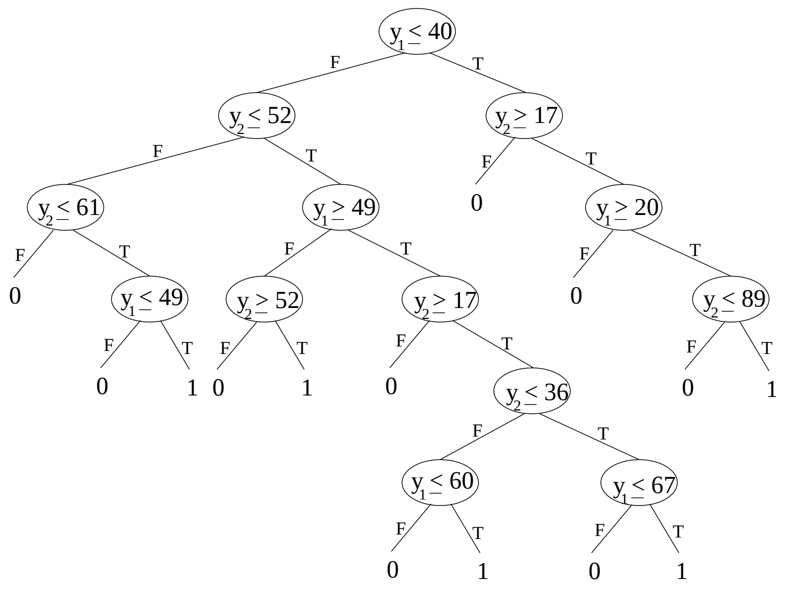

FIG. 8. *This shows the final decision tree that could be constructed by* ALT-LEARN-WITH-MQS *for the target geometric concept shown in Figure* 1.

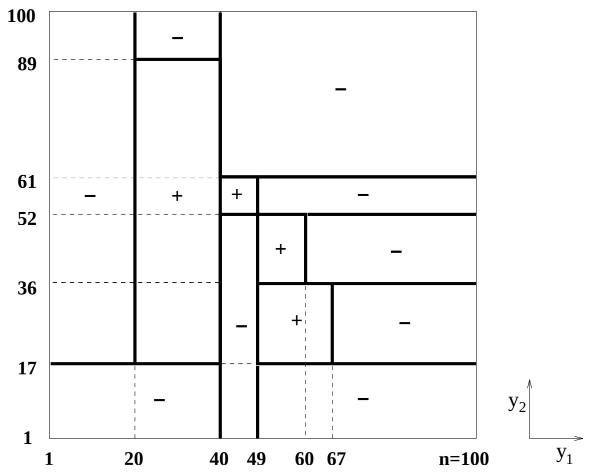

FIG. 9. *This shows the decomposition of* $\mathcal{N}_n^d$ *corresponding to the decision tree shown in Figure* 8.

the root to any leaf.

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 σ and $\log n$ (and thus polynomial in m 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 m any constant.

THEOREM 4. *Given any target concept* $g \in \bigcup_{\leq m} \text{BOX}_n^d$*, the algorithm* ALT-LEARN-WITH-MQS *achieves exact identification of* g *making at most* $(\sigma/2+1)$ *equivalence queries, and using* $O(\sigma(\sigma + \log n + d))$ *time and* $O(\sigma(\log n + d))$ *membership queries.*

Proof. Observe that each segment of g causes 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 as a result of 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 two membership queries made to query the upper and lower corners of each leaf, and $\lceil \log n \rceil + d$ membership queries used in the binary searches for the σ halfspaces, it follows that the number of membership queries made is at most $2\sigma + \sigma(\lceil \log n \rceil + d) = O(\sigma(\log n + d))$.

Observe that the depth of T is at most σ since any of the halfspaces defined by the σ segments in the target concept will appear at most once on any path from the root to a leaf. Thus the time to locate the region to split is $O(\sigma)$ and it immediately follows that the time complexity is $O(\sigma(\sigma + \log n + d))$. $\square$

COROLLARY 5. *The algorithm* ALT-LEARN-WITH-MQS *achieves exact identification for any* $g \in \bigcup_{\leq m} \text{BOX}_n^d$ *using time and queries polynomial in* m *and* $\log n$ *for* d *constant.*

Proof. This follows immediately from Theorem 4 and the observation that $\sigma \leq (2m)^d$. $\square$

Finally, observe that just as we described in section 5.3, our final hypothesis can be converted to the union of $O(md \log m)$ boxes from BOX_n^d. Recall that the time to perform the conversion is $O((2m)^{2d} md \log m)$ and thus, this will be efficient only if d is constant. Also, using the technique of section 6 we can refine this algorithm to use only equivalence queries.

We now use a different method of analysis to show that ALT-LEARN-WITH-MQS uses time and queries polynomial in d and $\log n$ for m 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 $g \in \bigcup_{\leq m} \text{BOX}_n^d$ be the target concept and T be the final decision tree output by our algorithm. For each internal node of T there is an associated region of $\mathcal{N}_n^d$. For any node r in T, let h_r denote the height of the subtree of T rooted at r (we define the height of a leaf to be 0), and let m_r be the minimum number of boxes needed to cover the part of g contained in the region of $\mathcal{N}_n^d$ associated with r. Thus, for the region r corresponding to the root of T we have that $m_r \leq m$ and $h_r \leq 2md$ since, by Observation 1, the height of T is at most $2md$.

Let $L(m, h)$ denote the maximum number of leaves in a decision tree rooted at a node r with $m_r = m$ and $h_r = h$. Then we have that $L(m, h) = L(m, h-1) + L(m-1, h-1)$, where, for all $m \geq 0$, $L(m, 0) = 1$, and for all $h \geq 1$, $L(0, h) = 1$. 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 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 m boxes, that box will not appear in one of the recursive calls. (In the worst case all other boxes will be split).

Consider the "largest possible" decision tree, according to the upper bound $L(m, h)$. (Note that a tree constructed from a problem instance will generally have fewer leaves.) Notice that in such a "largest" decision tree, there are no leaves in the decision tree on levels 0 (i.e., the root level) to $m - 1$. On each level from m to h there are leaves caused by the base case $m = 0$ of the recurrence. The total number of these leaves is given by

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

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

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

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

$$\begin{aligned}\sum_{j=m}^{h} \binom{j-1}{m-1} + \sum_{j=0}^{m-1} \binom{h}{j} &= \sum_{j=m-1}^{h-1} \binom{j}{m-1} + \sum_{j=0}^{m-1} \binom{h}{j} \\ &= \sum_{j=0}^{h-1} \binom{j}{m-1} + \sum_{j=0}^{m-1} \binom{h}{j} \\ &= \binom{h}{m} + \sum_{j=0}^{m-1} \binom{h}{j} \\ &= \sum_{j=0}^{m} \binom{h}{j}. \end{aligned} \tag{9.1}$$

Recall that $h \leq 2md$. In the following lemma we show that $(2md)^m$ is an upper bound on the summation in (9.1).

LEMMA 3. *The number of leaves in any hypothesis decision tree constructed by* ALT-LEARN-WITH-MQS *is bounded above by $(2md)^m$ for $m > 1$ and is $2d + 1$ for $m = 1$.*

Proof. Let $n = md$ and $k = m$ for $m > 1$. 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 $\sum_{j=0}^{k} \binom{2n}{j} \leq (2n)^k$ for $n \geq k > 1$, and thus the result follows for $m > 1$. Finally, for $m = 1$ the number of leaves in the hypothesis is $2d + 1$. $\square$

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

THEOREM 6. *Given any target concept $g \in \bigcup_{\leq m} \text{BOX}_n^d$, the algorithm* ALT-LEARN-WITH-MQS *achieves exact identification of g, making at most $(2md)^m$ equivalence queries, making $O((2md)^m \cdot (\log n + d))$ membership queries, and using $O((2md)^{m+1} \cdot (\log n + d))$ time.*

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

Equation (9.1) shows that $L(m,h) = \sum_{j=0}^{m} \binom{h}{j}$, and Lemma 3 proves a bound for the number of leaves of T of at most $\sum_{j=0}^{m} \binom{2md}{j} \leq (2md)^m$. Thus it immediately follows that at most $(2md)^m$ equivalence queries are made. Since at most $\lceil \log n \rceil + d$ membership queries are used by the binary search procedure when splitting a node, it follows that the number of membership queries made by Alt-Learn-With-MQs is $O((2md)^m \cdot (\log n + d))$. Finally, since it takes $O(md)$ time to find the node corresponding to the region containing the counterexample and at most $O(\log n + d)$ time for each binary search, it follows that the time complexity of Alt-Learn-With-MQs is $O((2md)^{m+1} \cdot (\log n + d))$. □

Thus Alt-Learn-With-MQs achieves exact identification for any $g \in \bigcup_{\leq m} \mathrm{BOX}_n^d$ using time and queries polynomial in d and $\log n$ for m constant. We note that for $m \geq 6$ we can remove a factor of 2^m in the complexity by using the tighter upperbound that $\sum_{j=0}^{m} \binom{2md}{m} \leq (md)^m$.

10. Concluding remarks.

We have given an efficient algorithm that uses membership and equivalence queries to exactly identify any concept from $\bigcup_{\leq m} \mathrm{BOX}_n^d$ for d constant. Furthermore, this algorithm makes at most $md + 1$ equivalence queries, all of which can be formulated as the union of $O(md \log m)$ axis-parallel boxes.

We have also shown how to extend our basic algorithm to learn efficiently, using *only* equivalence queries, any discretized geometric concept generated from any number of halfspaces with any number of known (to the learner) slopes in a constant dimensional space. In particular, our algorithm exactly learns (from equivalence queries *only*) unions of discretized axis-parallel boxes in constant dimensional space in polynomial time. Further, this algorithm can be modified to handle a polynomial number of lies in the counterexamples provided by the environment.

Finally, we have introduced a new complexity measure, σ, that better captures the complexity of the union of m boxes than simply the number of boxes and the dimension. We presented an algorithm that uses time and queries polynomial in σ and $\log n$. In fact, the time and queries (both membership and equivalence) used by this single algorithm are polynomial for *either* m 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 $\bigcup_{\leq m} \mathrm{BOX}_n^d$ (or the more general class of a discretized geometric concept) in time polynomial in d and $\log n$ when the number of boxes (or defining hyperplanes) is constant.

While we have provided an algorithm to efficiently learn geometric concepts defined by hyperplanes that are *not* axis parallel, to achieve this goal it was necessary that the learner be given a priori knowledge as to the slopes of the hyperplanes. An interesting direction is to explore the learnability (even for fixed dimensions) of geometric concepts defined by hyperplanes whose exact slopes are not known to the learner.

Finally, it would be interesting to see if $\bigcup_{\leq m} \mathrm{BOX}_n^d$ can be efficiently learned in time polynomial in m and $\log n$ for $d = O(\log m)$ or in time polynomial in d and $\log n$ for $m = O(\log d)$ (i.e., a generalization of the Blum and Rudich [9] result that $O(\log n)$-term DNF formulas are exactly learnable). Of course, since $\bigcup_{\leq m} \mathrm{BOX}_n^d$ generalizes the class of DNF formulas, it seems very unlikely that one could develop an algorithm for the unrestricted case of $\bigcup_{\leq m} \mathrm{BOX}_n^d$ that is polynomial in m, $\log n$, and d. It may be possible, however, to obtain a truly polynomial algorithm for some subclass of $\bigcup_{\leq m} \mathrm{BOX}_n^d$.

Acknowledgments. We thank Peter Auer and the COLT committee for their comments. We thank Wolfgang Maass for suggesting we consider the material discussed in section 5.3. We also thank the anonymous referees whose comments improved this manuscript.

REFERENCES

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

[2] D. Angluin and M. Krikis, *Learning with malicious membership queries and exceptions*, in Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory, 1994, pp. 57–66.

[3] D. Angluin and P. Laird, *Learning from noisy examples*, Machine Learning, 2 (1988), pp. 343–370.

[4] E. Arkin, P. Belleville, J. Mitchell, D. Mount, K. Romanik, S. Salzberg, and D. Souvaine, *Testing simple polygons*, in Proceedings of the 5th Canadian Conference on Computational Geometry, 1993, pp. 387–392.

[5] E. Arkin, H. Meijer, J. Mitchell, D. Rappaport, and S. Skiena, *Decision trees for geometric models*, in Proceedings of the 9th Annual Symposium on Computational Geometry, 1993, pp. 369–378.

[6] P. Auer, *On-line learning of rectangles in noisy environments*, in Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory, 1993, pp. 253–261.

[7] P. Belleville and T.C. Shermer, *Probing polygons mimimally is hard*, Comput. Geom., 2 (1993), pp. 255–265.

[8] A. Blum, *Separating distribution-free and mistake-bounded learning models over the Boolean domain*, SIAM J. Comput., 23 (1994), pp. 990–1000.

[9] A. Blum and S. Rudich, *Fast learning of k-term DNF formulas with queries*, J. Comput. System Sci., 51 (1995), pp. 367–373.

[10] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, *Learnability and the Vapnik-Chervonenkis dimension*, J. Assoc. Comput. Mach., 36 (1989), pp. 929–965.

[11] N. Bshouty, Personal communication, University of Calgary, 1994.

[12] N. Bshouty, Z. Chen, and S. Homer, *On learning discretized geometric concepts*, in 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 54–63.

[13] N. H. Bshouty, S. A. Goldman, T. R. Hancock, and S. Matar, *Asking questions to minimize errors*, J. Comput. System Sci., 52 (1996), pp. 268–286.

[14] W. J. Bultman and W. Maass, *Fast identification of geometric objects with membership queries*, in Proc. 4th Annual Workshop on Computational Learning Theory, San Mateo, CA, Morgan Kaufmann, 1991, pp. 337–353.

[15] Z. Chen, *Learning unions of two rectangles in the plane with equivalence queries*, in Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory, ACM Press, 1993, pp. 243–252.

[16] Z. Chen and S. Homer, *The Bounded Injury Priority Method and the Learnability of Inions of Rectangles*, unpublished manuscript, 1994.

[17] Z. Chen and W. Maass, *On-line learning of rectangles*, Machine Learning, 17 (1994), pp. 23–50.

[18] V. Chvatal, *A greedy heuristic for the set covering problem*, Math. Oper. Res., 4 (1979), pp. 233–235.

[19] J. C. Culberson and R. A. Reckhow, *Covering polygons is hard*, in 29th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 601–611.

[20] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, New York, 1987.

[21] M. Frazier, S. Goldman, N. Mishra, and L. Pitt, *Learning from a consistently ignorant teacher*, J. Comput. System Sci., 52 (1996), pp. 472–492.

[22] P. W. Goldberg, S. A. Goldman, and H. D. Mathias, *Learning unions of boxes with membership and equivalence queries*, in Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory, 1994, pp. 198–207.

[23] Y.-B. Jia and M. Erdmann, *The complexity of sensing by point sampling*, in Proceedings of the First Workshop of the Algorithmic Foundations of Robotics, 1994.

[24] Y.-B. Jia and M. Erdmann, *Geometric sensing of known planar shapes*, International Journal of Robotics Research, 15 (1996), pp. 290–299.

[25] S. Homer and Z. Chen, *Fast Learning Unions of Rectangles with Queries*, unpublished

manuscript, 1993.

[26] M. Kearns and M. Li, *Learning in the presence of malicious errors*, in Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, 1988.

[27] P. D. Laird, *Learning from Good and Bad Data*, Kluwer International Series in Engineering and Computer Science, Kluwer Academic Publishers, Boston, 1988.

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

[29] P. M. Long and M. K. Warmuth, *Composite geometric concepts and polynomial predictability*, in Proceedings of the Third Annual Workshop on Computational Learning Theory, Morgan Kaufmann, 1990, pp. 273–287.

[30] W. Maass and G. Turán, *On the complexity of learning from counterexamples*, in 30th Annual Symposium on Foundations of Computer Science, 1989, pp. 262–267.

[31] W. Maass and G. Turán, *Lower bound methods and separation results for on-line learning models*, Machine Learning, 9 (1992), pp. 107–145.

[32] W. Maass and G. Turán, *Algorithms and lower bounds for on-line learning of geometrical concepts*, Machine Learning, 14 (1994), pp. 251–269.

[33] W. Maass and G. Turán, *How fast can a threshold gate learn?*, in Computational Learning Theory and Natural Learning Systems: Constraints and Prospects, G. Drastal, S. J. Hanson, and R. Rivest, eds., MIT Press, Cambridge, MA, 1994, pp. 318–414.

[34] W. Maass and M. Warmuth, *Efficient learning with virtual threshold gates*, in Proceedings of the 12th International Conference on Machine Learning, Morgan Kaufmann, 1995, pp. 378–386.

[35] K. Romanik and C. Smith, *Testing Geometric Objects*, Technical report UMIACS-TR-90-69, University of Maryland College Park, Department of Computer Science, 1990.

[36] S. Skiena, *Problems in geometric probing*, Algorithmica, 4 (1989), pp. 599–605.

[37] R. H. Sloan, *Four types of noise in data for PAC learning*, Information Processing Letters, 54 (1992), pp. 157–162.

[38] L. Valiant, *A theory of the learnable*, Communications of the ACM, 27 (1984), pp. 1134–1142.