The Bounded Injury Priority Method and the Learnability of Unions of Rectangles*

Zhixiang Chen† Boston University

Steven Homer‡ Boston University

Abstract

We develop a bounded version of the finite injury priority method in recursion theory. We use this to study the learnability of unions of rectangles over the domain $\{0, \ldots, n-1\}^d$ with only equivalence queries. Applying this method, we show three main results: (1) The class of unions of rectangles is polynomial time learnable for constant dimension d. (2) The class of unions of rectangles whose projections at some unknown dimension are pairwise-disjoint is polynomial time learnable. (3) The class of unions of two disjoint rectangles is polynomial time learnable with unions of two rectangles as hypotheses.

---

*The preliminary version of this paper is partially contained in [BCH].

†Computer Science Department, Boston University, Boston, MA 02215; zchen@cs.bu.edu. The author was supported by NSF grant CCR91-03055 and by a Boston University Presidential Graduate Fellowship.

‡Computer Science Department, Boston University, Boston, MA 02215; homer@cs.bu.edu. The author was supported by NSF grants CCR91-03055 and CCR94-00229.

1 Introduction

We study a central problem in the field of computational learning theory and the first application of finite injury priority arguments from recursion theory in this area. Modern computational learning theory studies the existence and efficiency of "learning algorithms" for various computational problems. Most often these problems arise in complexity theory and have interesting hard combinatorial properties. Results in this field give further insight into these fundamental problems, providing evidence as to their algorithmic properties and to their computational complexity.

We consider the on-line learning model, one of the two standard models in this field (the other being PAC learning [V], [BEHW]). On-line learning is formally defined in the next section. (See also Angluin [Aa], [Ab].) The basic idea is that an algorithm acts to learn a concept C from among a large set of possible concepts. The algorithm takes no input but works by issuing a series of hypotheses H which are attempts to extensionally specify C's identity. These hypotheses are responded to by an environment which provides counterexamples to a hypothesis if it is wrong or says "correct" if the concept has been identified. A counterexample consists of an element in the symmetric difference of the hypothesis and the concept to be learned. (In the parlance of computational learning theory this is called on-line learning with equivalence queries.)

In this paper we consider the problem of learning unions of rectangles of arbitrary dimensions. This problem has been well-studied in learning theory, as an interesting problem in its own right and because it generalizes several of the most fundamental concept classes studied in this field (see [Aa], [R]). In particular, (1) it is a generalization of learning DNF formulas, (2) it is a special case of unions of intersections of half-spaces over the domain $[0, n-1]^d$, and (3) unions of pairwise-disjoint rectangles are general cases of Boolean-decision trees over bounded integer domains.

In order to make these relationship precise we first need to define the class of rectangles we consider. Let N be the set of natural numbers. $\forall i, j \in N$, we use $[i, j]$ to denote the set $\{i, \ldots, j\}$ if $i \leq j$ or ϕ otherwise. We define the class of all discretized *axis-parallel*

rectangles (or *rectangles* for short) over the domain $[0, n-1]^d$ as follows,

$$BOX_n^d = \{\prod_{i=1}^{d}[a_i, b_i] | 0 \leq a_i \leq b_i \leq n-1, \forall i \in [1, d]\} \cup \{\phi\}.$$

We consider the following concept class of unions of rectangles over the domain $[1, n-1]^d$

$$U_k BOX_n^d = \{C_1 \cup \cdots \cup C_k | \forall i \in [1, k], C_i \in BOX_n^d\}.$$

Now given a DNF formula

$$F = x_1 x_2 \vee \overline{x}_3 \overline{x}_4 \vee x_1 x_2 \overline{x}_3 \overline{x}_4,$$

we view learning F as a special case of learning rectangles as follows. For any example $x \in [0, 1]^4$, $F(x) = 1$ if and only if x satisfies the following condition

$$(x_1 = 1 \wedge x_2 = 1) \vee (x_3 = 0 \wedge x_4 = 0) \vee (x_1 = 1 \wedge x_2 = 1 \wedge x_3 = 0 \wedge x_4 = 0).$$

i.e., x is in $C_1 \cup C_2 \cup C_3$, where

$$C_1 = [1, 1] \times [1, 1] \times [0, 1] \times [0, 1],$$

$$C_2 = [0, 1] \times [0, 1] \times [0, 0] \times [0, 0],$$

$$C_3 = [1, 1] \times [1, 1] \times [0, 0] \times [0, 0].$$

Thus, x is in a union of three rectangles $C_1, C_2, and C_3$. In general, a k-term DNF formula can be represented as a union of k rectangles.

Define a halfspace over the domain $[0, n-1]^d$ to be the set of all examples in $[0, n-1]^d$ satisfying the condition

$$a_1 x_1 + \cdots + a_d x_d \geq a_{d+1},$$

where a_i are real numbers, and $(a_1, \ldots, a_d)$ is called the weight of the halfspace. Learning half-spaces has previously been considered in computational learning theory by Littlestone and Maass and Turán ([L], [MTf]). Given any intersection of halfspaces, when any weight in any of those halfspaces contains exactly one nonzero component, then the intersection of the halfspaces is a rectangle. So learning rectangles is a special case of learning unions of intersections of half-spaces.

A Boolean decision tree is a binary tree where each non-leaf node represents a Boolean question and has a "true" and a "false" successor node, and where the leaves are labelled with a Boolean value classifying the input as being accepted or rejected. Decision trees in learning theory have previously been studied by Rivest [R]. Any Boolean decision tree with k leaf-nodes is equivalent to a k-term DNF formula such that any two of those terms are disjoint, i.e., no examples satisfy more than one term. As we observed before that a k-term DNF formula can be represented as a union of k-rectangles. Hence, any Boolean decision tree can be represented as a union of pairwise disjoint rectangles.

Relevant research on the problem of learning unions of rectangles in the pac-model (see [V]) was done by Blumer *et al* [BEHW] and, Long and Warmuth [LW]. Blumer *et al* proved that for constant dimension d, unions of non-discretized rectangles over the d dimensional Euclidean space are pac-learnable. Long and Warmuth proved that for constant k, unions of k non-discretized rectangles over arbitrary dimensional Euclidean space are pac-learnable. Recently, Jackson [J] proved that any union of polynomially many discretized rectangles over the domain $[0, n-1]^d$ such that each of those rectangles is bounded on $O(\frac{\log d}{\log \log n})$ sides is strongly pac-learnable with respect to the uniform distribution and using membership queries as well.

In the on-line learning model with only equivalence queries, problems of learning BOX_n^d by BOX_n^d and many other discretized geometric concepts have been studied in Maass and Turán [MTb, c, d, e] and, Bultman and Maass [BM]. They showed that the query complexity of learning BOX_n^d by BOX_n^d is $\Omega(d \log n)$ [MTc, d]. They also exhibited an algorithm that learns BOX_n^d by BOX_n^d using $O(2^d \log n)$ queries [MTc, d]. In contrast to Maass and Turán's $O(2^d \log n)$ upper bound, an $O(dn)$ upper bound can be derived from Valiant's work [V] on learning monomials. Finally, an $O(d^2 \log n)$ upper bound and an $\Omega(\frac{d^2 \log n}{\log d})$ lower bound were obtained respectively in Chen and Maass [CMa, b] and Auer [AU]. It is still open whether one can close the "$(\log d)$-gap" between the upper and lower bounds (see also Maass [M]). One should note that it follows from Angluin [Ab] that on-line learning with only equivalence queries implies pac-learning under any distribution. When the learner is allowed to use both equivalence and membership queries, Chen and Homer [CH] first proved that unions of k rectangles over the domain

$[0, n-1]^2$ are learnable with $O(k^3 \log n)$ queries. Later, Goldberg, Goldman and Mathias [GGM] proved that for any fixed d, unions of rectangles over the domain $[0, n-1]^d$ are polynomial time learnable with equivalence and membership queries. They also proved that for any constant k but arbitrary dimension d, unions of k rectangles are polynomial time learnable with equivalence and membership queries.

In constructing algorithms to learn unions of rectangles with only equivalence queries, obvious approaches tend to fail because one faces two *"credit assignment problems"*. The fundamental problem is in which dimension or to which rectangle can the information given by a counterexample be correctly used. On the one hand, to which rectangle in the target concept should a positive counterexample belong? On the other hand, in which dimension is the projection of a negative counterexample true for any rectangle in the target concept? A more precise and detailed explanation of these issues is given in Section 3. There is a potential relation between the *"credit assignment problem"* and the *"injury"* in an injury constructions in recursion theory. When one makes a wrong assignment, then one's goal is "injured" in the sense that one would never achieve his goal unless the assignment is undone. A solution to the credit assignment problem related to rectangle learning given by Chen and Maass [CMb] is reminiscent of the finite injury priority method. The injury constructions in this paper can be viewed as a new method to solve the credit assignment problem by applying priority methods from recursion theory in order to construct concrete algorithms.

Previous work on learning unions of rectangles with equivalence queries was able to overcome the related credit assignment problem by employing local search strategies that could tolerate certain types of one-sided errors (see [CMb]). Based on this design technique and certain more powerful local search strategies that can tolerate some two-sided errors, Chen [C] exhibited an algorithm for learning unions of two rectangles over the domain $[0, n-1]^2$ with $O(\log^2 n)$ equivalence queries and using unions of two rectangles as hypotheses.

In this paper we make the analogy with injury methods from recursion theory explicit in our proofs. This enables us to give a more precise analysis of methods present in earlier work in this area. And as well, it provides a more general and canonical construction

of learning algorithms which can be used for new cases of learning of rectangles which were not possible before. The methods developed here are far from the full strength of finite injury arguments in modern recursion theory. They make essential and strong use of the notion of requirements and of their (bounded) injury, but only little use of priorities assigned to these requirements. Nonetheless they provide a new and useful method with which to explain several complicated constructions and proofs.

Using our methods we obtain the first efficient and exact learning algorithms for three classes of unions of rectangles using equivalence queries. In Section 4 we show how to learn unions of two disjoint rectangles in polynomial time, with unions of two rectangles as hypotheses. This improves the previously known algorithm which works only for two dimensional rectangles. Improving this algorithm to apply to any pair of rectangles would imply $P = NP$ [PV]. In Section 5 we extend the methods of Section 4 to obtain an algorithm which learns unions of $k \geq 3$ rectangles whose projections on some unknown dimension are pairwise-disjoint in polynomial time and using unions of at most $(k-1)(\log n + 1)$ rectangles as hypotheses. Finally, in Section 6, we present an algorithm which learns for constant d, unions of k rectangles in $poly(k, \log n)$ time and using unions of at most $(4kd(\log n - 1) - 2k(\log n - 3))^d$ rectangles as hypotheses.

Almost at the same time and in work independent from ours, Bshouty [Bb] (see also [BGGM]), and Maass and Warmuth [MW] also proved that for fixed d, unions of rectangles over the domain $[0, n-1]^d$ are polynomial time learnable using only equivalence queries. All three approaches are different in hypothesis representations and, more importantly, in their proof techniques. Maass and Warmuth's algorithm is the strongest in the sense that its query complexity matches the lower bound. Bshouty's algorithm can also cope with certain misclassified CE's. Our approach uses unions of rectangles as hypotheses and, with introduction of priority arguments in this setting, opens the possibility that other concrete algorithms can make good use of priority arguments.

2 The Learning Model

Our learning model is the standard model for on-line learning with equivalence queries (see Angluin [Ab], Littlestone [L], Maass and Turán [MTa]). A learning algorithm for a concept class $\mathbf{C}$ over a domain $\mathbf{X}$ is viewed as a dialogue between a learner and an environment. The goal of the learner is to *learn* an unknown target concept $C_t \in \mathbf{C}$ that has been fixed by the environment. In order to obtain information about C_t the learner proposes hypotheses H from a fixed hypothesis space $\mathbf{H}$ with $\mathbf{C} \subseteq \mathbf{H} \subseteq \mathbf{2^X}$ to the environment. If $H = C_t$, the environment will answer *yes* so the learner learns it. Whenever $H \neq C_t$, the environment responds with a *counterexample* (CE) $g \in H \triangle C_t$. g is called a *positive counterexample* (PCE) if $g \in C_t - H$, and a *negative counterexample* (NCE) if $g \in H - C_t$. Each new hypothesis issued by the learner may depend on the earlier hypotheses and the received CE's. The learning complexity of the concept class $\mathbf{C}$ is in the worst case the minimum number of CE's required by the learner to learn any target concept in it. We say that a learning algorithm is polynomial if its time complexity is polynomial in the logarithm of the size of the domain and the size of the target concept.

It is known that on-line learning implies pac-learning [Ab]. It also known that there are cases in which on-line learning is strictly harder than pac-learning [B].

3 The Credit Assignment Problems

The "credit assignment problem" may be defined as "the problem of assigning credit or blame to the individual decisions that led to some overall results" (Cohen and Feigenbaum [CF]). Obviously, this problem is ubiquitous not only in Artificial Intelligence, but also in the study of adaptive neural networks, where credit or blame for the overall performance of the network has to be distributed to the individual components of the network.

In the study of learning unions of rectangles with equivalence queries, obvious approaches tend to fail because one would face two credit assignment problems: On the

one hand, in which dimension is the projection of a negative counterexample true, that is, outside of the bounds of the rectangle in that dimension ? (This occurs in the single rectangle case as well.) On the other hand, to which rectangle in the target concept does a positive counterexample belong? (This never occurs in the single rectangle case.) We illustrate those two kinds of problems in figure 1.A and 1.B, respectively.

In figure 1.A, the target concept is a rectangle $C_t = \prod_{i=1}^{2}[a_i, b_i]$. In order to learn it, we only need to learn the four parameters a_1, a_2, b_1 and b_2. To do this, we can design a global algorithm which employs four local search procedures to search for the four parameters, respectively. However, when the learner receives a NCE $x = (x_1, x_2)$ as shown in the figure, the learner does not know to which of the procedures a component of x should be assigned as a true negative counterexample. In this case, the component x_2 should not be assigned to the procedures searching for the parameters a_2 and b_2, because it is lies inside the interval $[a_2, b_2]$, unfortunately the learner does not know this.

In figure 1.B, the target concept $C_t = A \cup B$ with $A = \prod_{i=1}^{2}[a_i, b_i]$ and $B = \prod_{i=1}^{2}[e_i, f_i]$. In order to learn C_t, the learner needs to learn the parameters a_i, b_i, e_i and f_i, for $i = 1, 2$. As above, the learner will be a global algorithm that employs local search procedures to search for each of the eight parameters. In addition to the credit problem present in learning one rectangle, when the global algorithm receives a PCE $y = (y_1, y_2)$ as in figure 1.B, the learner does not know to which of the procedures a component of y should be assigned. In this case, the component y_1 should not be assigned to the procedures searching for the parameters a_1 and b_1 associated with retangle A because it lies outside the interval $[a_1, b_1]$, unfortunately the learner does not know this is the case. (This is similarly true for y_2, a_2 and b_2.)

4 The Bounded Injury Priority Method

The finite injury priority method in modern recursion theory was invented by Friedberg [F] in 1957 and independently by Muchnik [MU] in 1956, in their solutions to Post's Problem. Since then, injury arguments with priority have become the most powerful

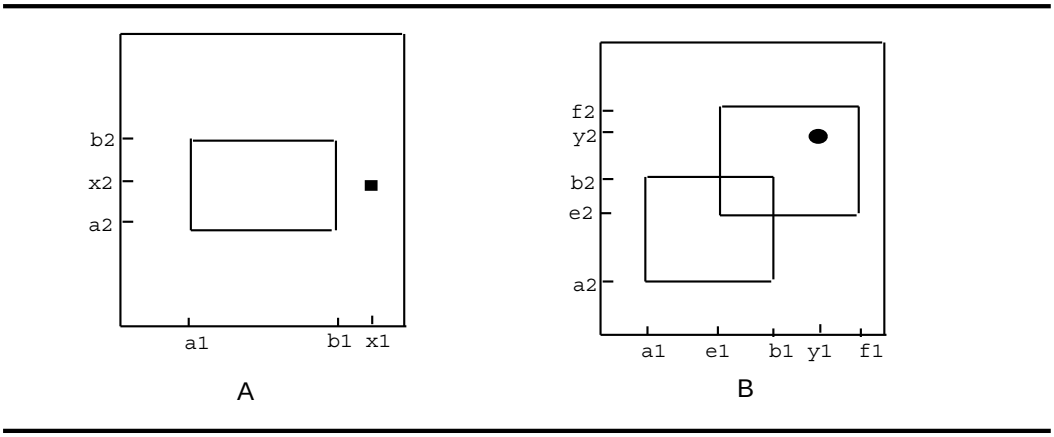

Figure 1: How to Assign a Counterexample

and dominant design technique in modern recursion theory. In a typical finite injury priority argument (see, Soare [S]), one needs to achieve a "goal" (which is usually the construction of a set with certain properties). One divides this "goal" A into a sequence of infinitely many "requirements" $\{R_i\}_{i\in N}$ such that A is achieved if all the requirements R_i are met or satisfied. Normally, one also assigns priority to the requirements. For example, if $n < m$, then the requirement R_n is assigned priority over R_m, and we say that R_n has higher priority than R_m. One then constructs a procedure which runs in stages. At each stage, one will take certain action to satisfy some requirement(s). However, actions taken at some stage s for satisfying R_m may at a later stage $t > s$ be undone when action is taken for satisfying R_n of higher priority. In this case, we say that R_m is injured at stage t. The crucial feature of all finite injury priority methods is that each requirement is injured finitely often and so comes to a limit.

Consider designing a learning algorithm for a given concept class $\mathbf{C}$ over a domain $\mathbf{X}$. For each target concept $C_t \in \mathbf{C}$, the goal of the learner is to identify C_t. One can design a sequence of requirements $\{R_i\}$ such that C_t is learned if all the requirements R_i are satisfied. However, there are substantial difficulties when one applies an injury argument with priority to design a learning algorithm A for a concept class $\mathbf{C}$ over a domain $\mathbf{X}$. First, one would like the complexity of the algorithm A to be bounded by a reasonable function $f(n, |C_t|)$ (usually f is a polynomial, or at worst exponential), where n is the size of the domain, and $|C_t|$ is the size of the target concept. Thus,

the number of requirements designed by the learner must be bounded by $f(n, |C_t|)$. Similarly, the number of injuries received by each requirements must also be bounded by $f(n, |C_t|)$. Second, in the finite injury priority method the domain is infinite. Once a requirement is injured, one can search the domain to find a new element with which to *"remedy"* the injury, i.e., to satisfy the requirement again. However, in designing the learning algorithm A, the domain $\mathbf{X}$ is finite. Hence, once a requirement is injured, it is more difficult to remedy the injury within the domain $\mathbf{X}$. Finally, in a construction using the injury method, one can decide whether a requirement is injured at each stage, because one can simply trace the entire (but finite) construction up to the current stage. However, this may not be true in designing a learning algorithm because of the desired complexity bound on the algorithm.

5 From One Interval to One Rectangle

In this section we begin with several basic results whose proofs exploit the ideas of an injury construction. The methods and techniques used illustrate how the problems mentioned here can be dealt with by injury arguments in this setting. Furthermore, they form the basis for the more complex constructions in the later sections of this paper where priority is essential.

Consider the following interval concept class

$$HEAD(n) = \{[0, k] | 0 \leq k \leq n-1\}$$

over the domain $[0, n-1]$. We now design an algorithm S to learn any target concept $C_t = [0, j] \in HEAD(n)$. Here, we consider an extended environment which may respond with an NCE outside C_t (called a true NCE) or an NCE in C_t (called a false NCE) for a given hypothesis. We do this because, when we later consider rectangles of higher dimension, such false NCE's will be of concern in our algorithms.

Algorithm S runs in stages. At each stage s, S searches a value for j, denoted by $RS(s)$, and issues a hypothesis $H_s = [0, RS(s)]$. Let x_s be the counterexample received by the

learner at stage s for the hypothesis H_s. Define

$$PS(s) = max(\{0\} \cup \{x_r | 1 \leq r \leq s \ \& \ x_r \ is \ a \ PCE\}),$$

$$NS(s) = min(\{n\} \cup \{x_r | 1 \leq r \leq s \ \& \ x_r \ is \ an \ NCE \ \& \ x_r > PS(s)\}).$$

In order to learn C_t, one only needs to satisfy the following requirement

$$R: \quad \exists s \ \ (\forall s' \geq s \ \ (RS(s') = j)).$$

One should observe that even though the learner may issue a hypothesis $H_s = [0, RS(s)] = [0, j]$ at stage s, the environment can still cheat the learner by giving a false NCE x_s for H_s to the learner. We say that R is *injured* (or R receives an *injury*) at stage s, if $PS(s) < NS(s) \leq j$. When R receives an injury at stage s, the learner has received false NCE's and will be *fooled* by them until he proves that they are false by receiving a PCE greater than their maximum.

Learning Algorithm S

Stage 1. Set $RS(1) = 0$. Ask an equivalence query for the hypothesis $H_1 = [0, RS(1)]$. If yes then stop. Otherwise, one receives a PCE x_1.

Stage $s + 1 \geq 2$. We consider three cases:

(1) If x_s is an NCE and $x_s \leq PS(s)$, then set $RS(s+1) = RS(s)$.

(2) If x_s is an NCE but $x_s > PS(s)$, then set $RS(s+1) = PS(s) + \lfloor \frac{NS(s) - PS(s)}{2} \rfloor$.

(3) If x_s is a PCE, then set $RS(s+1) = min(\{NS(s) - 1\} \cup \{RS(r) | 1 \leq r \leq s \ \& \ PS(s) \leq RS(r) < NS(s)\})$. Finally, one asks an equivalence query for the hypothesis $H_{s+1} = [0, RS(s+1)]$. If yes then stop, otherwise one receives a CE x_{s+1}.

For an NCE x_s, we say that x_s is an invalid NCE if $x_s \leq PS(s)$, otherwise we say that x_s is a valid NCE. Obviously, the extended environment cannot cheat the learner by giving an invalid NCE. In other words, Requirement R will never be injured by an invalid NCE. Algorithm S has the following properties that were originally given in [CMa]:

Property 3.1.

(i) $\forall s \geq 1$, $PS(s) + \lfloor \frac{NS(s)-PS(s)}{2} \rfloor \leq NS(s) - 1$.

(ii) $\forall s \geq 1$, $PS(s) \leq RS(s+1) \leq NS(s) - 1$.

(iii) $\forall s \geq 1$, if x_s is a (true or false) NCE, then $RS(s+1) \leq RS(s)$.

(iv) Assume that x_s and x_{s+1} are PCE's. Then, $x_{s+1} \geq NS(s)$, or $x_{s+1} > RS(r)$ for some $r \in [1, s]$ with $PS(s) \leq RS(r) < NS(s)$.

Proof.

(i) Note that $PS(s) < NS(s)$ by the definition of $NS(s)$.

(ii) This follows from (i) and the definition of $RS(s+1)$ in the construction.

(iii) If $x_s \leq PS(s)$, then $RS(s+1) = RS(s)$. Assume that $x_s > PS(s)$ and $s \geq 1$. Since $RS(s) \leq NS(s-1) - 1$ by (ii), we have $NS(s) = x_s \leq RS(s)$. Furthermore, $RS(s+1) = PS(s) + \lfloor \frac{NS(s)-PS(s)}{2} \rfloor \leq NS(s) - 1$ by (i). Thus, $RS(s+1) \leq RS(s)$.

(iv) By construction one has $x_{s+1} > RS(s+1) = min(\{NS(s) - 1\} \cup \{RS(r) | 1 \leq r \leq s \ \& \ PS(s) \leq RS(r) < NS(s)\})$. □

The following two theorems establish relationships between the query complexity of Algorithm S and the number of injuries received during the learning process of S. These two theorems are new interpretations of a result in [CMa, b].

Theorem 3.2. *Assume that R has received i injuries and, g invalid NCE's have been received during a learning process of S. Then, at most* $\log n$ *true NCE's and at most* $2(\log n + g + i) + i + 1$ *CE's occur in this learning process.*

Proof. We first estimate the number of true NCE's received in the learning process. Consider any s' and s'' such that $s' < s''$ and, $x_{s'}$ and $x_{s''}$ are true NCE's. $NS(s') = x_{s'}$, since $PS(s'-1) < x_{s'} \leq RS(s') \leq NS(s'-1) - 1$ by Property 3.1 (ii). Similarly, $NS(s'') = x_{s''}$. Because $x_{s'}$ is a true NCE, so for any PCE x_s with $s > s'$, $x_s < x_{s'}$, thus $RS(s+1) < x_{s'}$. This implies, by Property 3.1 (iii), $NS(s) < x_{s'}$, $\forall s > s'$. So, $x_{s''} = NS(s'') < x_{s'}$. Since $x_{s'}$ is an NCE, $RS(s'+1) = PS(s') + \lfloor \frac{NS(s')-PS(s')}{2} \rfloor$. We consider the following two cases.

(1) When $PS(s'') > RS(s'+1)$, we have $PS(s'') \geq PS(s') + \lfloor \frac{NS(s')-PS(s')}{2} \rfloor + 1 \geq \frac{NS(s')+PS(s')}{2}$. Hence, $NS(s'') - PS(s'') \leq NS(s') - PS(s'') \leq NS(s') - \frac{NS(s')+PS(s')}{2}$

$= \frac{NS(s')-PS(s')}{2}$.

(2) Assume that $PS(s'') \leq RS(s'+1)$. Then, $x_j \leq RS(s'+1)$, $\forall j \in [s'+1, s''-1]$ such that x_j is a PCE. Hence, $RS(j+1) \leq RS(s'+1)$ for each such j. Together with Property 3.1 (iii), this implies that $RS(j+1) \leq RS(s'+1)$, $\forall j \in [s'+1, s''-1]$. In particular, $RS(s'') \leq RS(s'+1)$, so $NS(s'') \leq RS(s'+1)$. Thus, $NS(s'') - PS(s'') \leq RS(s'+1) - PS(s'') \leq PS(s') + \lfloor \frac{NS(s')-PS(s')}{2} \rfloor - PS(s'') \leq PS(s') + \lfloor \frac{NS(s')-PS(s')}{2} \rfloor - PS(s')$ $= \frac{NS(s')-PS(s')}{2}$.

Putting the above two cases (1) and (2) together, we have $NS(s'')-PS(s'') \leq \frac{NS(s')-PS(s')}{2}$. This follows that at most $\log n$ true NCE's occur in the learning process of S.

We say that $[r', r'']$ is an injury interval if and only if, $\forall r \in [r', r'']$, Requirement R is injured at stage r, but not at stages $r'-1$ and $r''+1$.

Claim 3.3.

(i) For every true NCE x_s, s is not in any injury interval.

(ii) If x_s is a valid and false NCE, then s is in some injury interval.

Proof of Claim 3.3.

(i) Suppose by contradiction that x_s is a true NCE but s is in an injury interval $[r', r'']$. By Property 3.1 (ii), $PS(s-1) \leq RS(s) \leq NS(s-1) - 1$. Since x_s is a true NCE, $PS(s) = PS(s-1) < x_s = NS(s) \leq RS(s) \leq NS(s-1) - 1$. Because Requirement R is injured at stage s according to the definition of injury intervals, we have $NS(s) \leq j$, i.e., $x_s \leq j$, a contradiction to that x_s is a true NCE. Hence, (i) holds.

(ii) Again by Property 3.1 (ii), $PS(s-1) \leq RS(s) \leq NS(s-1) - 1$. Assume that x_s is a valid NCE. Then, $PS(s) = PS(s-1) < x_s = NS(s) \leq RS(s) \leq NS(s-1) - 1$. Since x_s is false, $PS(s) < x_s = NS(s) \leq j$, this implies that Requirement R is injured at stage s. Thus, s is in some injury interval. □

Claim 3.4. *Assume that x_s is a PCE and, s is not in any injury interval. Then, if x_{s+1} is a PCE then there must be an injury interval $[r', r'']$ such that $s = r'' + 1$. In other words, if $s \neq r'' + 1$ for any injury interval $[r', r'']$, then x_{s+1} must be an NCE.*

Proof of Claim 3.4. Suppose that both x_s and x_{s+1} are PCE's. According to the construction of Algorithm S, $RS(s+1) = NS(s) - 1$, or $RS(s+1) = RS(r)$ for some $r \in [1, s]$ with $PS(s) \leq RS(r) < NS(s)$. In the first case, $x_{s+1} \geq NS(s)$. By Property 3.1 (ii), $PS(s) = x_s \leq RS(s+1) < NS(s) \leq x_{s+1} \leq j$. This implies that Requirement R receives an injury at stage s, a contradiction to that s is not in any injury interval.

Now, we only need to consider that $x_{s+1} > RS(s+1) = RS(r)$ for some $r \in [1, s]$. Fix $m = max\{m' | RS(m') = RS(r)\ \&\ m' \in [r, s]\}$. If x_m is a PCE, then $RS(m) < x_m \leq RS(m')$, $\forall m' \geq m$. In particular, $RS(m) < x_m \leq RS(s+1) = RS(m)$, a contradiction. Hence, x_m is an NCE. Similarly, $\forall m' \in [m+1, s]$, if $x_{m'}$ is PCE then $x_{m'} \leq RS(m)$. If there is no $m' \in [m+1, s]$ such that $x_{m'}$ is an PCE and $x_m \leq x_{m'} \leq RS(m)$, then $RS(m'+1) < x_m$, $\forall m' \in [m+1, s]$. This implies that $RS(s+1) < RS(m)$, a contradiction to the assumption $RS(s+1) = RS(m)$. Hence, there is a $m' \in [m+1, s]$ such that $x_{m'}$ is an PCE and $x_m \leq x_{m'} \leq RS(m)$.

Set $t = min\{m' | m' \in [m+1, s]\ \&\ x_{m'}\ is\ an\ PCE\ \&\ x_m \leq x_{m'} \leq RS(m)\}$. According to the construction of Algorithm S, $RS(t+1) = RS(m)$. Note that $t+1 > m$. If $t < s$, i.e., $t+1 \leq s$, then $m \geq t+1$ by the definitions of m and t. Thus, $t = s$. Again, according to the definitions of t and m, $\forall m' \in [m, t-1] = [m, s-1]$, $PS(m') < x_m \leq j$. This means that Requirement R receives an injury at stage m'. Let $[r', r'']$ be the injury interval containing all those m', then $r'' = s - 1$, since s is not in any injury interval. □

Note that the learner receives i CE's within all the injury intervals. By Claim 3.3 and the previous analysis, there are at most $\log n$ true NCE and g invalid NCE outside all injury intervals. So, we only need to estimate how many PCE's are received outside all the injury intervals. Since R receives i injuries, there are at most i injury intervals. Hence, there are at most $\log n + g + i + 1$ maximal blocks of successive PCE's outside all the injury intervals in the learning process. Consider any such maximal block B that consists of $k+1$ PCE's $x_s, \ldots, x_{s+k}$. By Claim 3.4, either $k = 0$ or $k = 1$. Moreover, when $k = 1$, then the maximal block B is uniquely determined by an injury interval. Therefore, there are at most $\log n + g + 2i + 1$ PCE's occurring outside all the injury intervals.

Putting all together, we know that the learner receives at most $2(\log n + g + i) + i + 1$ CE's. □

Theorem 3.5. *Assume f false NCE's are received during a learning process for S. Then, requirement R will receive at most $3f$ injuries in the process. (Of course, the number of invalid NCE's is at most f.)*

Proof. We define injury intervals as in the proof of Theorem 3.2. We now analyze how many CE's will be received within all the injury intervals. By Claim 3.3, there are at most f (false) NCE's received within all the injury intervals. So, we now need to estimate the number of PCE's received within all the intervals.

By definition, for any injury interval $[r', r'']$, we know that one receives a false NCE at stage r'. Suppose that there are l false NCE's received within the interval. Then, there are at most l maximal blocks of successive PCE's from stage r' to stage r''. Hence, there are at most f such maximal blocks within all the injury intervals. Consider any such maximal block B that consists of $k+1$ PCE's $x_s, \ldots, x_{s+k}$. By Property 3.1 (iv), $x_{s+1} \geq NS(s)$, or $x_{s+1} > RS(r)$ for some $r \in [1, s]$. In the first case, x_{s+1} proves that $NS(s)$ is a false NCE. This happens at most once for each false NCE. In the later case, x_{s+1} refutes the earlier hypothesis $H_r = [1, RS(r)]$. The learner will never issue this hypothesis H_r again at a later stage $t > s+1$, since $RS(r) < x_{s+1} \leq RS(t)$. Furthermore, this event can only happen if the original CE x_r to H_r is a false NCE. Thus, there are at most f such hypotheses for which this event can occur. It follows from the above analysis that one receives at most $2f$ PCE's within all the injury intervals. Therefore, there are at most $3f$ CE's received within all the injury intervals, this implies that Requirement R receives at most $3f$ injuries. □

Remark 3.6. *We can design a learning algorithm S^* using a similar injury construction for the concept class $TAIL(n) = \{[j, n-1] | j \in [0, n-1]\}$ over the domain $X = [0, n-1]$ such that analogous versions of Theorem 3.2 and 3.5 hold for S^*. By a direct transformation, we have that, $\forall a, b$, Algorithm S works on the concept class $HEAD(a, b) = \{[a, j] | j \in [a, b]\}$, and Algorithm S^* works on $TAIL(a, b) = \{[j, b] | j \in [a, b]\}$.*

Remark 3.7. *For convenience, we will use S and S^* (or with subscripts in most cases) throughout this paper to stand for copies of Algorithm S and S^*, respectively. We also use $RS(s), NS(s), PS(s)$ and, $RS^*(s), NS^*(s)$, and $PS^*(s)$ (again, with subscripts in most cases) to denote the corresponding parameters as defined in the constructions of Algorithm S and S^*.*

For any $S \subseteq [0, n-1]^d$, let

$$min_i(S) = min\{x_i | x \in S \ \& \ x = (x_1, \ldots, x_d)\},$$

$$max_i(S) = max\{x_i | x \in S \ \& \ x = (x_1, \ldots, x_d)\}.$$

Define

$$rec(S) = \prod_{i=1}^{d} [min_i(S), max_i(S)].$$

It is obvious that $rec(S)$ is the minimal rectangle in BOX_n^d containing S. Given $C_t \in U_k BOX_n^d$, for any example $r \in [0, n-1]^d$ and $S \subseteq [0, n-1]^d$, we say that (r, S) is a *witness* for C_t if and only if, (1) $r \notin C_t$ and $S \subseteq C_t$, and (2) $r \in rec(S)$.

Property 3.8.

(i) $\forall S \subseteq [0, n-1]^d$, $S \subseteq rec(S)$.

(ii) $\forall A \subseteq [0, n-1]^d$, $A \in BOX_n^d$ *if and only if* $(\forall S \subseteq A) \ \ (rec(S) \subseteq A)$.

(iii) $\forall C_t \in U_k BOX_n^d$, $C_t \notin BOX_n^d$ *if and only if there is a witness* (r, S) *for it.*

Proof.

(i) $\forall z = (z_1, \ldots, z_d) \in S$, $min_i(S) \leq z_i \leq max_i(S)$, $\forall i \in [1, d]$. This implies that $z \in rec(S)$, hence $S \subseteq rec(S)$.

(ii) Suppose that $A \in BOX_n^d$. Let $A = \prod_{i=1}^{d} [a_i, b_i]$. Then, $\forall S \subseteq A$, $min_i(S) = min\{x_i | x \in S \ \& \ x = (x_1, \ldots, x_d)\} \geq a_i$ and, $max_i(S) = max\{x_i | x \in S \ \& \ x = (x_1, \ldots, x_d)\} \leq b_i$. Thus, $\forall y = (y_1, \ldots, y_d) \in rec(S)$, $a_i \leq y_i \leq b_i$, $\forall i \in [1, d]$. Hence, $y \in A$, this implies that $rec(S) \subseteq A$. On the other hand, assume that $(\forall S \subseteq A)$ $(rec(S) \subseteq A)$. In particular, $rec(A) \subseteq A$. By (i), $A \subseteq rec(A)$. Thus, $A = rec(A) \in BOX_n^d$.

(iii) Suppose that there is a witness (r, S) for C_t. If $C_t \in BOX_n^d$, then by (ii), $r \in rec(S) \subseteq C_t$, contradicting $r \notin C_t$. Now, suppose that $C_t \notin BOX_n^d$. Choose $S = C_t$.

Then, by (i), $C_t \subseteq rec(S)$. Because $C_t \notin BOX_n^d$, $rec(S) - C_t \neq \phi$. Hence, there exists at least one $r \in rec(S) - C_t$. By definition, (r, S) is a witness for C_t. $\Box$

It is known in [CMa, b] that one can learn BOX_n^d by BOX_n^d using $O(d^2 \log n)$ queries. We now prove a stronger result.

Theorem 3.9. *For any $C_t \in U_k BOX_n^d$, with $O(d^2 \log n)$ CE's one learns C_t if it is a rectangle or finds a witness for it otherwise.*

Proof. For any target concept $C_t \in U_k BOX_n^d$, we assume that it is a single rectangle $\prod_{i=1}^d [a_i, b_i]$ (of course, this may not true). Then, $\forall i \in [1, d]$, at the i-th dimension we use two procedures S_i and S_i^* to search for the parameters b_i and a_i, respectively. Our learning algorithm LR runs in stages. At stage s, the learner issues the hypothesis

$$H(s) = \prod_{i=1}^{d} [RS_i^*(s), RS_i(s)].$$

Let $x_s = (x_{s1}, \ldots, x_{sd})$ be the CE received for $H(s)$. One should note that, although x_s is a CE for the hypothesis $H(s)$, x_{si} may not necessarily be a CE for the hypothesis issued by procedure S_i or S_i^*, $i \in [1, d]$. If x_{si} is an NCE, it may be false. However, x_{si} will always be true, if it is a PCE.

Let $W(s)$ be the set of all CE's received by the end of stage s and, $P(s)$ be the set of all PCE's in $W(s)$. Because $P(s)$ is uniquely determined by $W(s)$, we will not explicitly define it in the algorithm. In order to learn C_t, one only needs to satisfy the following $2d$ requirements,

$$R(i,1): \quad \exists s \ (\forall s' \geq s \quad (RS_i(s') = b_i)), \quad i \in [1, d],$$

$$R(i,2): \quad \exists s \ (\forall s' \geq s \quad (RS_i^*(s') = a_i)), \quad i \in [1, d].$$

We say that $R(i,1)$ is injured (or receives an injury) at stage s, if either $PS_i(s) < NS_i(s) \leq b_i$, or $b_i < PS_i(s)$. In the first case, we say that $R(i,1)$ receives a negative injury. In the latter case, we say that $R(i,1)$ receives a positive injury. Similarly, we define injury, positive injury and negative injury for requirements $R(i,2)$.

Learning Algorithm LR

Stage 0. Set $H(0) = \phi$. Ask an equivalence query. If yes then stop. Otherwise one receives a PCE x_0. Let $W(0) = \{x_0\}$.

Stage $s + 1 \geq 1$. Decide whether $\exists r_s \in (W(s) - P(s))$ such that $(r_s, PS(s))$ is a witness. If yes then output it and stop. Otherwise, on the received CE x_s, $\forall i \in [1, d]$, the learner executes the following two steps: (1) Search for b_i using S_i in the domain $[x_{0i}, n-1]$, i.e., if x_{si} is a CE for the current hypothesis $RS_i(s)$ of S_i, then S_i issues a new hypothesis $RS(s+1)$, or sets $RS(s+1) = RS(s)$ otherwise. (2) Search for a_i using S_i^* in the domain $[0, x_{0i}]$, i.e., do the same as that in (1).

Ask an equivalence query for $H(s+1)$. If yes then stop, otherwise a CE x_{s+1} is received. Set $W(s+1) = W(s) \cup \{x_{s+1}\}$.

Claim 3.10. *Assume that $C_t \in BOX_n^d$. Then, $\forall i \in [1, d]$, neither $R(i, 1)$ nor $R(i, 2)$ will receive a positive injury. Furthermore, each of the procedures S_i and S_i^* receives at most $2(d-1)\log n$ invalid NCE's, and each of $R(i, 1)$ and $R(i, 2)$ receives at most $6(d-1)\log n$ negative injuries.*

Proof. Given $C_t \in BOX_n^d$, let $C_t = \prod_{i=1}^{d}[a_i, b_i]$. For each PCE $x_s = (x_{s1}, \ldots, x_{sd})$, $a_i \leq x_{si} \leq b_i, \forall i \in [1, d]$. This implies that neither $R(i, 1)$ nor $R(i, 2)$ will receive a positive injury. On the other hand, each NCE x_s provides a CE for at least one of the $2d$ procedures. Furthermore, each NCE x_s provides a true NCE for at least one procedure and a false NCE for at most $(d-1)$ procedures. By Theorem 3.2, each of the procedures receives at most $\log n$ true NCE's, thus each procedure receives at most $2(d-1)\log n$ false NCE's. Hence, each procedure receives at most $2(d-1)\log n$ invalid NCE's, and by Theorem 3.5, each of $R(i, 1)$ and $R(i, 2)$ receives at most $6(d-1)\log n$ negative injuries. □.

Claim 3.11 *Assume $C_t \in BOX_n^d$. Then, with at most $4d\log n + 44d(d-1)\log n + 2d$ CE's the learner learns C_t.*

Proof. It follows from Theorem 3.2 and Claim 3.10 that each of the $2d$ procedures receives at most $2\log n + 22(d-1)\log n + 1$ CE's. Every CE for Algorithm LR provides one CE for at least one of its $2d$ procedures. Therefore, at most $4d\log n + 44d(d-1)\log n + 2d$ CE's are required to learn C_t. □

Claim 3.12. *Assume that $C_t \notin BOX_n^d$. Then, with at most $4d\log n + 44d(d-1)\log n + 2d + 1$ CE's, the learner will find a witness for it.*

Proof. By Property 3.8 (iii), there are witnesses for C_t. Since the learner issues a hypothesis $H(s) \in BOX_n^d$ at each stage during the learning process of LR, the learner will not receive a yes from the environment. If the learner finds a witness for C_t with at most $4d\log n + 44d(d-1)\log n + 2d + 1$ CE's, then the claim is true. We now assume, by contradiction, that the learner has received $s' = 4d\log n + 44d(d-1)\log n + 2d + 1$ CE's but hasn't found any witnesses. Thus, at stage $s'+1$, $rec(P(s'))$ is consistent with all CE's in $W(s')$. Recall that $rec(PS(s')) \in BOX_n^d$. Consider the learning process of LR on the target concept $rec(P(s'))$. Since Algorithm LR is deterministic and is oblivious to the input target concept, the learning process of LR for $rec(PS(s'))$ is the same as that for C_t by the end of stage s'. There is no $s'' \leq s'$ such that at stage s'' the learner issued the hypothesis $H(s'') = rec(P(s'))$, because if so a CE $x_{s''}$ would be received and thus $x_{s''} \in W(s')$, a contradiction to the consistency of $rec(P(s'))$ with all CE's in $W(s')$. Therefore, the learner requires at least $4d\log n + 44d(d-1)\log n + 2d + 1$ CE's to learn $rec(P(s'))$, a contradiction to Claim 3.11. □

Now, Theorem 3.9 follows directly from Claim 3.11 and 3.12. □

6 Unions of Two Disjoint Rectangles

It is open whether there is a polynomial time algorithm for learning unions of two rectangles over the domain $[0, n-1]^d$ using the same type of unions as hypotheses. Pitt and Valiant's work in [PV] on the non-learnability of 2-term DNF formulas by 2-term DNF formulas under the assumption $P \neq NP$ implies that no positive solution to this problem exists if $P \neq NP$. On the other hand, a polynomial time learning algorithm using $O(\log^2 n)$ queries was given in Chen [C] over the domain $[0, n-1]^2$. It is also open whether one can learn unions of two rectangles on the plane by unions of two rectangles using $O(\log n)$ queries. This technically difficult problem was considered in Maass and Turán [MTd] as one of the most interesting open problems regarding learning discretized geometric concepts. In this section, we will show that unions of two disjoint

rectangles are polynomial time learnable using unions of two rectangles as hypotheses.

Given any two rectangles $A, B \in BOX_n^d$, let $A = \prod_{i=1}^d [a_i, b_i]$, $B = \prod_{i=1}^d [e_i, f_i]$. For any $j \in [1, d]$, we say that A and B are j-disjoint if either $b_j < e_j$ or $f_j < a_j$. Define

$$DU(j, d, n) = \{A \cup B | A, B \in BOX_n^d \ \& \ A \ and \ B \ are \ j - disjoint\},$$

$$DU(d, n) = \bigcup_{j=1}^{d} DU(j, d, n).$$

It is obvious that $DU(d, n)$ is the class of all unions of two disjoint rectangles. For any witness (r, S) for C_t, let $\alpha^r = (\alpha_1^r, \ldots, \alpha_d^r) \in S$ be an example such that $\forall x = (x_1, \ldots, x_d) \in S$, $\alpha_j^r \leq x_j$. Similarly, let $\beta^r = (\beta_1^r, \ldots, \beta_d^r) \in S$ be an example such that $\forall x = (x_1, \ldots, x_d) \in S$, $\beta_j^r \geq x_j$. Note that α^r and β^r may not be unique.

Lemma 4.1. *Given $C_t = A \cup B = \prod_{i=1}^d [a_i, b_i] \cup \prod_{i=1}^d [e_i, f_i] \in DU(j, d, n)$. Assume $b_j < e_j$. Then, for any witness (r, S) for C_t, $\alpha^r \in A$ and $\beta^r \in B$.*

Proof. Suppose that $\alpha^r \in B$. Then, $S \subseteq B$ by the definition of α^r and the assumption that $b_j < e_j$. This implies that $rec(S) \subseteq B$ by Property 3.8 (ii), thus $r \in rec(S) \subseteq B \subseteq C_t$, a contradiction to $r \notin C_t$. Hence, $\alpha^r \in A$. Similarly, $\beta^r \in B$. □

Theorem 4.2. *$\forall j \in [1, d]$, there is an algorithm for learning the concept class $DU(j, d, n)$ using $O(d^4 \log^2 n)$ queries. Moreover, the hypotheses issued by the learner are unions of two rectangles, and the time complexity of the algorithm is $poly(d, \log n)$.*

Corollary 4.3. *There is an algorithm for learning the concept class $DU(d, n)$ using $O(d^5 \log^2 n)$ queries. Moreover, the hypotheses issued by the learner are unions of two rectangles, and the time complexity of the algorithm is $poly(d, \log n)$.*

Proof of Corollary 4.3. $\forall j \in [1, d]$, run a copy of the learning algorithm for $D(j, d, n)$. Then, the above theorem follows immediately from Theorem 4.2. □

Proof of Theorem 4.2. Given the target concept $C_t = A \cup B = \prod_{i=1}^d [a_i, b_i] \cup \prod_{i=1}^d [e_i, f_i] \in DU(j, d, n)$, without loss of generality, we assume that $b_j < e_j$. $\forall i \in [1, d]$, we use four procedures $S_{A,i}$, $S_{A,i}^*$, $S_{B,i}$ and $S_{B,i}^*$ simultaneously to search for b_i, a_i, f_i, and e_i, respectively. Our learning algorithm $LU(j)$ runs in stages. At stage s, the

learner issues the hypothesis

$$H(s) = H_A(s) \cup H_B(s) = \prod_{i=1}^{d} [RS^*_{A,i}(s), RS_{A,i}(s)] \bigcup \prod_{i=1}^{d} [RS^*_{B,i}(s), RS_{B,i}(s)].$$

Let $x_s = (x_{s1}, \ldots, x_{sd})$ be the CE received for $H(s)$. Again, although x_s is a CE for the hypothesis H_s, $\forall i \in [1, d]$, x_{si} may not necessarily be a CE for the current hypothesis issued by procedures $S_{A,i}, S^*_{A,i}, S_{B,i}$, or $S^*_{B,i}$. When x_{si} is an NCE, it may be false as happened in Algorithm LR. When x_{si} is PCE, however, it may also be false, since one doesn't know that to which of A and B the CE x_s belongs. This phenomenon will also occur during the learning processes of algorithms in later sections.

Let $W_A(s)$ be the set of all *possible* CE's for A received by the end of stage s, and $P_A(s)$ be the set of all PCE's in $W_A(s)$. Because $P_A(s)$ is uniquely determined by $W_A(s)$, we will not define $P_A(s)$ explicitly in the construction of Algorithm $LU(j)$. In order to learn C_t, one only needs to satisfy the following requirements, $i \in [1, d]$,

$$R_A(i,1): \quad \exists s \ (\forall s' \geq s \ (RS_{A,i}(s') = b_i)),$$

$$R_A(i,2): \quad \exists s \ (\forall s' \geq s \ (RS^*_{A,i}(s') = a_i)),$$

$$R_B(i,1): \quad \exists s \ (\forall s' \geq s \ (RS_{B,i}(s') = f_i)),$$

$$R_B(i,2): \quad \exists s \ (\forall s' \geq s \ (RS^*_{B,i}(s') = e_i)).$$

The priority of the above requirements is given as follows,

$$R_B(i,1) = R_B(j,2) > R_A(r,1) = R_A(m,2), \quad \forall i, j, r, m \in [1, d].$$

In the same manner as we did in the construction of Algorithm LR, we define injury, positive injury and negative injury for $R_A(i,k)$ and $R_B(i,k)$, $\forall (i,k) \in [1,d] \times [1,2]$. By Theorem 3.9, we further assume without loss of generality that the learner is given a prior witness (r_0, S_0). By Lemma 4.1, we know that $\alpha^{r_0} \in A$ and $\beta^{r_0} \in B$. Fix an integer-pairing function $\langle \cdot, \cdot \rangle$.

Learning Algorithm LU(j)

Stage $\langle 0,0 \rangle$**.** Set $H_A(\langle 0,0 \rangle) = \{\alpha^{r_0}\}$ and $H_B(\langle 0,0 \rangle) = \{\beta^{r_0}\}$. Set $W_A(\langle 0,0 \rangle) = \{\alpha^{r_0}\}$. Go to stage $\langle 1,0 \rangle$.

Stage $<s, m> > 0$. Ask an equivalence query for $H(<s-1, m>)$. If yes then stop. Otherwise, one receives a CE $x_{<s-1,m>}$. Set $W_A(<s, m>) = W_A(<s-1, m>) \cup \{x_{<s-1,m>}\}$. Decide whether there is a $r_s \in (W_A(<s, m>) - P_A(<s, m>))$ such that $(r_s, P_A(<s, m>))$ is a witness. If so then execute Case (i). Otherwise, execute Case (ii).

Case (i). On the CE β^{r_s}, $\forall i \in [1, d]$, do: (1) Search for f_i using $S_{B,i}$ in the domain $[\beta_i^{r_0}, n-1]$, i.e., if $\beta_i^{r_s}$ is a CE for the current hypothesis $RS_{B,i}(<s-1, m>)$, then issue a new hypothesis $RS_{B,i}(<s, m>)$, or set $RS_{B,i}(<s, m>) = RS_{B,i}(<s-1, m>)$ otherwise. (2) Similar to (1), search for e_i using $S^*_{B,i}$ in the domain $[0, \beta_i^{r_0}]$. Reset $W_A(<s, m>) = W_A(<s, m>) - \{\beta^{r_s}\}$. Undo $S_{A,i}$ and $S^*_{A,i}$ by setting $H_A(<s, m>) = rec(P_A(<s, m>))$. Go to stage $<s+1, m>$.

Case (ii). On the CE $x_{<s-1,m>}$, $\forall i \in [1, d]$, in the same manner as (1) in Case (i), we first search for b_i using $S_{A,i}$ in the domain $[\alpha_i^{r_0}, n-1]$ and a_i using $S^*_{A,i}$ in the domain $[0, \alpha_i^{r_0}]$. We then consider two subcases: (3) If $x_{<s-1,m>}$ is a PCE, then no actions will be taken for $S_{B,i}$ and $S^*_{B,i}$. (4) If $x_{<s-1,m>}$ is an NCE, then search for f_i using $S_{B,i}$ in the domain $[\beta_i^{r_0}, n-1]$ and e_i using $S^*_{A,i}$ in the domain $[0, \beta_i^{r_0}]$. Go to stage $<s, m+1>$.

Claim 4.4. *$\forall (i, k) \in [1, d] \times [1, 2]$, $R_B(i, k)$ never receives a positive injury.*

Proof. Actions taken for $S_{B,i}$ and $S^*_{B,i}$ for PCE's only happen at Case (i) in the construction of Algorithm $LU(j)$. According to Lemma 4.1, all PCE's β^{r_s} for which actions have been taken for $S_{B.i}$ are true, i.e., in B. This implies that requirements $R_B(i, k)$ never receive any positive injuries. □

Claim 4.5. *Each of the procedures $S_{B,i}$ and $S^*_{B,i}$ receives at most $2(d-1)$ invalid NCE's, $\forall i \in [1, d]$. $\forall (i, k) \in [1, d] \times [1, 2]$, $R_B(i, k)$ receives at most $6(d-1)\log n$ negative injuries.*

Proof. Note that an NCE $x_{<s,m>}$ for $H(<s, m>)$ may not necessarily be an NCE for $H_B(<s, m>)$. As in the proof of Claim 3.10, we know that an NCE $x_{<s,m>}$ for $H_B(<s, m>)$ provides an NCE for at least one of the $2d$ procedures $S_{B,i}$ and $S^*_{B,i}$, $i \in [1, d]$. Furthermore, each NCE $x_{<s,m>}$ provides a true NCE for at least one of the

procedures and a false NCE for at most $(d-1)$ procedures. By Theorem 3.2 and Claim 4.4, each of the procedures receives at most $\log n$ true NCE's, thus each procedure receives at most $2(d-1)\log n$ false NCE's. Hence, each procedure receives at most $2(d-1)\log n$ invalid NCE's, and by Theorem 3.5, each requirement $R_B(i,k)$ receives at most $6(d-1)\log n$ negative injuries. □

Claim 4.6. *$O(d^2 \log n)$ CE's are required to satisfy all the requirements $R_B(i,k)$, $\forall (i,k) \in [1,d] \times [1,2]$.*

Proof. It follows from Claim 4.4, 4.5 and Theorem 3.2. □

Claim 4.7. *Suppose that Case (i) was executed at stage $<s,m'>$ and stage $<s+1,m''>$. Then, $m''-m' = O(d^2 \log n)$. In other words, there is a constant δ which is independent of m', m'', d and n such that $m''-m' \leq \delta d^2 \log n$.*

Proof. By the construction of Algorithm $LU(j)$, the parameter s is incremented by one only when Case (i) is executed, while the parameter m is incremented by one only when Case (ii) is executed. Thus, from stage $<s,m'>$ to stage $<s+1,m''>$ (excluding those two stages), only Case (ii) is executed. During this period of the learning process, for any CE $x_{<s,m>}$, $m \in [m'+1, m''-1]$, we use procedures $S_{A,i}$ and $S^*_{A,i}$, $\forall i \in [1,d]$, to learn A when $x_{<s,m>i}$ is a CE for $S_{A,i}$ or $S^*_{A,i}$. We also use procedures $S_{B,i}$ and $S^*_{B,i}$ to learn B when $x_{<s,m>}$ is an NCE and $x_{<s,m>i}$ is an NCE for $S_{B,i}$ or $S^*_{B,i}$. Each CE $x_{<s,m>}$ provides at least a CE for at least one of the procedures. By Theorem 3.9, and Claim 4.6, $O(d^2 \log n)$ CE's will occur during the period, this implies that $m''-m' = O(d^2 \log n)$. □

Claim 4.6 and 4.7 imply that all requirements $R_B(i,k)$, $i \in [1,d], k \in [1,2]$, will be satisfied at some stage $<s,m>$ after receiving $O(d^4 \log^2 n)$ CE's. At any stage later than $<s,m>$, the learner only uses procedures $S_{A,i}$ and $S^*_{A,i}$ to learn A, $\forall i \in [1,d]$. This requires $O(d^2 \log n)$ CE's by Theorem 3.9. Hence, the learner requires $O(d^4 \log^2 n)$ CE's to learn C_t. Each execution of one of the $4d$ procedures needs time $O(\log^2 n)$, because it only needs to do certain comparisons of two numbers and then compute a new number for next hypothesis and the sizes of all the numbers considered are of order $\log n$. Algorithm $LU(j)$ needs also to decide at each stage $<s,m>$ whether a witness

exists. This is done by searching each CE in $W_A(<s,m>) - P_A(<s,m>)$ and deciding whether it is in $rec(P_A(<s,m>))$. Thus, the time complexity of $LU(j)$ is $poly(d, \log n)$. □

7 Unions of k Disjoint Rectangles

As noted in the first section, Boolean-decision trees are special cases of unions of pairwise-disjoint rectangles over the domain $[0, n-1]^d$. Recently, in his remarkable paper, Bshouty [B] proved that Boolean-decision trees are polynomial time learnable with equivalence and membership queries. However, it is open whether unions of pairwise-disjoint rectangles are polynomial time learnable. In this section, we will show that any unions of rectangles whose projections on some unknown dimension are pairwise-disjoint are polynomial time learnable using only equivalence queries.

Now, we assume that $k \geq 3$. For any target concept $C_t = \cup_{i=1}^{k} C_i \in U_k BOX_n^d$, let $C_i = \prod_{j=1}^{d} [a_{i,j}, b_{i,j}]$. We say that that C_t is *e-dimension disjoint* if

$$\forall i_1, i_2 \in [1,d],\ i_1 \neq i_2 \ implies\ [a_{i_1,e}, b_{i_1,e}] \bigcap [a_{i_2,e}, b_{i_2,e}] = \phi.$$

Intuitively, C_t is *e-dimension disjoint* if the projections of its k rectangles on the e-th dimension are pairwise-disjoint. Define

$$DDU(e,k,d,n) = \{C_t \in U_k BOX_n^d | C_t\ is\ e-dimension\ disjoint\},$$

$$DDU(k,d,n) = \bigcup_{e=1}^{d} DDU(e,k,d,n).$$

Theorem 5.1. *$\forall e \in [1,d]$, there is an algorithm for learning $DDU(e,k,d,n)$ using $O(k^2 d^2 \log^3 n)$ queries, where the hypotheses issued by the algorithm are unions of $(k-1)(\log n + 1)$ rectangles, and its time complexity is $poly(k,d,\log n)$.*

Corollary 5.2. *There is an algorithm for learning $DDU(k,d,n)$ using $O(k^2 d^3 \log^3 n)$ queries, where the hypotheses issued by the algorithm are unions of $(k-1)(\log n + 1)$ rectangles, and its time complexity is $poly(k,d,\log n)$.*

Proof of Corollary 5.2. $\forall e \in [1,d]$, run a copy of the algorithm for learning $DDU(e,k,d,n)$. Then, the theorem follows immediately from Theorem 5.1. $\square$

Proof of Theorem 5.1. Given the target concept $C_t = \cup_{i=1}^{k} C_i \in DDU(e,k,d,n)$, let $C_i = \prod_{j=1}^{d}[a_{i,j}, b_{i,j}]$. Without loss of generality, we assume that $b_{i,e} < a_{i+1,e}$, $\forall i \in [1, k-1]$. Our learning algorithm $LDDU(e)$ runs in stages. At each stage s, we divide the e-th dimension into a set $I(s)$ of pairwise-disjoint intervals such that their union is $[0, n-1]$. Based on $I(s)$, we divide the domain $X = [0, n-1]^d$ into a set of pairwise-disjoint subdomains, i.e., $D(s) = \{E(u,v) | [u,v] \in I(s)\}$, where $E(u,v) = [0,n-1]^{e-1} \times [u,v] \times [0,n-1]^{d-e}$. When it is clear from the context, we simply use E to stand for $E(u,v)$.

For each subdomain $E \in D(s)$, we assume that $C_t \cap E$ is a rectangles (this may not true). $\forall i \in [1,d]$, within the domain E, we use procedures $S_{E,i}$ and $S^*_{E,i}$ at the i-th dimension to search for the right and left parameters (denoted by $r_{E,i}$ and $l_{E,i}, respectively$) of $C_t \cap E$. At stage s, $\forall E \in D(s)$, we issue a hypothesis

$$H_E(s) = \prod_{i=1}^{d} [RS^*_{E,i}(s), RS_{E,i}(s)].$$

The hypothesis issued for the target concept at stage s is

$$H(s) = \bigcup \{H_E(s) | E \in D(s)\}.$$

In order to learn C_t, we only need to satisfy the following requirements,

$$R(0) : \exists s \ (\forall s' \geq s \ (\forall E \in D(s') \ (C_t \cap E \in BOX^d_n))),$$

$$R(j,1) : \exists s \ (\forall s' \geq s \ (\forall E \in D(s') \ (RS_{E,j}(s') = r_{E,j}))), \quad j \in [1,d],$$

$$R(j,2) : \exists s \ (\forall s' \geq s \ (\forall E \in D(s') \ (RS^*_{E,j}(s') = l_{E,j}))), \quad j \in [1,d].$$

The priority of the requirements is

$$R(0) > R(j_1,1) = R(j_2,2), \quad \forall j_1, j_2 \in [1,d].$$

We say that $R(0)$ is injured at stage s, if $\exists E \in D(s) \quad (C_t \cap E \notin BOX^d_n)$. We also define injury, positive injury and negative injury for requirements $R(j,1)$ and $R(j,2)$, $\forall j \in [1,d]$, in the same manner as one did in the construction of Algorithm LR. Let

$W(s)$ denote the set of all the CE's received by the end of stage s, and $P(s)$ be the set of PCE's in $W(s)$. We will not define $D(s)$ and $P(s)$ explicitly in the construction of $LDDU(e)$, since they are uniquely determined by $I(s)$ and $W(s)$, respectively. Fix an integer-pairing function $<\cdot,\cdot>$.

Learning Algorithm LDDU(e)

Stage $<0,0>$. Set $I(<0,0>) = \{[0, n-1]\}$, $H(<0,0>) = \phi$ and $W(<0,0>) = \phi$. Go to stage $<1,0>$.

Stage $<s,m> > 0$. Ask an equivalence query for $H(<s-1,m>)$. If yes then stop, otherwise one receives a CE $x_{<s-1,m>}$. Let $Z_1 = W(<s-1,m>) \cup \{x_{<s-1,m>}\}$, and Z_2 be the set of all PCE's in Z_1. Decide whether $\exists E(u,v) \in D(<s-1,m>)$ and $\exists r_s \in (Z_1 - Z_2) \cap E$ such that $(r_s, Z_2 \cap E)$ is a witness for $C_t \cap E$. If yes then execute Case (i). Otherwise, execute Case (ii).

Case (i). Define $z = u + \lfloor \frac{v-u}{2} \rfloor$. Set $I(<s,m>) = (I(<s-1,m>) - \{[u,v]\}) \cup \{[u,z],[z+1,v]\}$, and $W(<s,m>) = Z_1$. Undo all procedures by setting $H_E(<s,m>) = rec(E \cap P(<s,m>))$, $\forall E \in D(<s,m>)$. Go to stage $<s+1,m>$.

Case (ii). Fix $E \in D(<s-1,m>)$ such that $x_{<s-1,m>} \in E$. Within E, search for $r_{E,j}$ using $S_{E,j}$ and $l_{E,j}$ using $S^*_{E,j}$ at the j-th dimension in the same manner as we did in Algorithm LR, $j \in [1,d]$. No actions will be taken for all the other subdomains in $D(<s-1,m>)$. Set $I(<s-1,m+1>) = I(<s-1,m>)$ and, $W(<s-1,m+1>) = Z_1$. Go to stage $<s,m+1>$.

Claim 5.3. *At stage $<s,m>$, for any $[u,v] \in I(<s,m>)$, if $[u,v]$ contains parameters at the e-th dimension from at most one rectangle in C_t, then $E(u,v) \cap C_t \in BOX^d_n$. In other words, if $E(u,v) \cap C_t \notin BOX^d_n$, then $[u,v]$ contains parameters at the e-th dimension from at least two different rectangles in C_t.*

Proof. Assume that $[u,v]$ contains parameters at the e-th dimension from at most one rectangle in C_t, say, C_1. Then, $[u,v] \subseteq [a_{1,e}, b_{1,e}]$. Hence, $[u,v] \cap [a_{i,e}, b_{i,e}] = \phi$, $\forall i \in [2,d]$, since C_t is e-dimension disjoint. This implies that $C_t \cap E([u,v]) = C_1 \cap E(u,v) \in BOX^d_n$. $\square$

Claim 5.4. *During the learning process of Algorithm LDDU(e), Case (i) can be executed at most* $(k-1)(\log n - 1)$ *times, and* $|I(<s, m>)| \leq (k-1)(\log n + 1)$, $\forall s, m \geq 1$.

Proof. We construct a tree T by tracing the generation of $I(<s, m>)$. The root of T is $[0, n-1] \in I(<0, 0>)$. At stage $<s, m>$, we expand T by adding two children to one leaf of the current T if Case (i) is executed, or do nothing otherwise. More precisely, when Case (i) is executed, $\exists [u, v] \in I(<s-1, m>)$ (i.e., $[u, v]$ is a leaf in the current T), one finds a witness in $E([u, v])$, so one adds $[u, z]$ and $[z+1, v]$ as two children to the leaf $[u, v]$.

We say a node $[u, v]$ in T is valid if it has two children and both children are leaves. Fix one leaf l in T such that the path from the root to l is the longest. Let f be the parent of l. By the construction, f must have another child g. If g is not a leaf, then the path from the root to one of its child is longer than the path from the root to l, a contradiction. Thus, g is a leaf. Hence, f is valid node. For any non-leaf node s, consider the subtree with the root s. Similarly, we can show that there is at least one valid node in this subtree. Therefore, any non-leaf node is on the path from the root to some valid node.

Because each time two children $[u, z]$ and $[z+1, v]$ are added to a node $[u, v]$, they are disjoint and of length at most half of that of their parent $[u, v]$. This implies that any two distinct nodes in T with a common ancestor distinct from themselves are disjoint, and the longest path in T is at most $\log n$. Suppose that $[u, v]$ is a valid node. Then, according to the construction of T, $E([u, v]) \cap C_t \notin BOX_n^d$. So, by Claim 5.3, $[u, v]$ contains parameters at the e-th dimension from at least two different rectangles in C_t. Since the target concept C_t has at most k rectangles, there are at most $k-1$ valid nodes in T. Hence, the number of non-leaf nodes of T is at most $(k-1)(\log n - 1)$, and the number of leaves of T is at most $2(k-1) + (k-1)(\log n - 1)$. It follows from the construction of T that, $|I(<s, m>)|$ is no more than the number of leaves of T, the number of executions of Case (i) is the number of non-leaf nodes of T. This implies that, $|I(<s, m>)| \leq 2(k-1) + (k-1)(\log n - 1) = (k-1)(\log n + 1)$ and, the number of executions of Case (i) is at most $(k-1)(\log n - 1)$. □

Claim 5.5. *Assume that Case (i) was executed at stage* $<s, m'>$ *and* $<s+1, m''>$*. Then,* $m'' - m' = O(kd^3 \log^2 n)$.

Proof. In the construction of Algorithm $LDDU(e)$, the parameter s is incremented by one only when Case (i) is executed, while the parameter m is incremented by one only when Case (ii) is executed. Thus, during the period of the learning process between stage $<s, m'>$ and $<s, m''>$ (excluding those two stages), only Case (ii) is executed. At any stage $<s, m>$, for $m \in [m', m'']$, for each domain $E \in D(<s-1, m>)$ the learner uses $2d$ procedures $S_{E,j}$ and $S^*_{E,j}$, $\forall j \in [1, d]$, to learn $C_t \cap E$. With the same analysis as we did for Algorithm LR, the learner needs $O(d^2 \log n)$ CE's to learn $C_t \cap E$ if it is in BOX^d_n, or finds a witness for it otherwise. By Claim 5.4, there are at most $(k-1)(\log n + 1)$ subdomains in $D(<s-1, m>)$. Therefore, $m'' - m' = O(kd^2 \log^2 n)$. □

Claim 5.6. *Requirement* $R(0)$ *will be satisfied after receiving* $O(k^2 d^2 \log^3 n)$ *CE's. In other words, there is a constant* δ *which is independent of* k, d *and* n *such that* $R(0)$ *will be satisfied after receiving at most* $\delta k^2 d^2 \log^3 n$ *CE's.*

Proof. By Claim 5.4 and 5.5. □

Suppose that $R(0)$ is satisfied at stage $<s, m>$ after finding the last witness in the execution of Case (i). Then, at stage $<s+1, m>$, for any $E \in D(<s, m>)$, $E \cap C_t \in BOX^d_n$. At any later stage $<s+1, m'>$ with $m' > m$, for each domain $E \in D(<s, m'>)$ the learner uses $2d$ procedures $S_{E,j}$ and $S^*_{E,j}$, $\forall j \in [1, d]$, to learn $C_t \cap E$. With the same analysis as we did for Algorithm LR, the learner needs $O(d^2 \log n)$ CE's to learn $C_t \cap E$. By Claim 5.4, there are at most $(k-1)(\log n + 1)$ subdomains in $D(<s, m'>)$, Therefore, the learner learns C_t with $O(kd^2 \log^2 n)$ CE's after stage $<s, m>$. Together with Claim 5.6, $O(k^2 d^2 \log^3 n)$ CE's are required in the learning process. Claim 5.4 also implies that each hypothesis issued is a union of no more than $(k-1)(\log n + 1)$ rectangles. By the similar analysis to Algorithm $LU(j)$, the time complexity of algorithm $LDD(e)$ is $ploy(k, d, \log n)$, □

8 Unions of k Rectangles

Goldberg, Goldman and Mathias [GGM] proved that for any fixed d, unions of rectangles over the domain $[0, n-1]^d$ are polynomial time learnable using equivalence and membership queries. In this section, we will show that their result still holds when only equivalence queries are used. As mentioned in Section 1, similar results to Theorem 6.1 have also been proved independently in Bshouty [Bb] and, Maass and Warmuth [MW]. All three approaches are different in hypothesis representations and, more importantly, in the proof techniques. Maass and Warmuth's algorithm is the strongest in the sense that its query complexity matches the lower bound. Bshouty's algorithm can also cope with certain misclassified CE's, Our approach uses unions of rectangles as hypotheses and, with introduction of priority arguments in this setting, opens the possibility that other concrete algorithms can make good use of priority arguments.

Theorem 6.1. *There are is an algorithm for learning $U_kBOX_n^d$ using $O((4kd\log n)^{d+1}d^2\log n)$ equivalence queries, where the time complexity of the algorithm is poly$(k^d, \log^d n)$, and the hypotheses issued by the learner are unions of at most $(4kd(\log n - 1) - 2k(\log n - 3))^d$ rectangles.*

Proof. Given $C_t = \cup_{i=1}^k C_i \in U_kBOX_n^d$. Let $C_i = \prod_{j=1}^d [a_{i,j}, b_{i,j}]$, $\forall i \in [1,k]$. Our learning algorithm LUR runs in stages. At each stage s, $\forall i \in [1,d]$, we divide the i-th dimension into a set $I(i,s)$ of pairwise-disjoint intervals such that their union is $[0, n-1]$. Based on $I(i,s)$, we divide the domain $[0,n-1]^d$ into a set of pairwise-disjoint subdomains $D(s) = \prod_{i=1}^d I(i,s)$. Inside each subdomain $B \in D(s)$, $\forall i \in [1,d]$, at the i-th dimension, we use $S_{B,i}$ and $S^*_{B,i}$ to search for the right and left parameters $r_{B,i}$ and $l_{B,i}$ of $C_t \cap B$, respectively. Here, we assume that $C_t \cap B$ is a rectangle (again, this may not be true). At stage s, $\forall B \in D(s)$, we issue a hypothesis

$$H_B(s) = \prod_{i=1}^{d} [RS^*_{B,i}(s), RS_{B,i}(s)],$$

and the hypothesis issued for the target concept is

$$H(s) = \bigcup \{H_B(s) | B \in D(s)\}.$$

In order to learn C_t, we only need to satisfy the following requirements,

$$R(0) : \exists s \ (\forall s' \geq s \ (\forall B \in D(s') \ (C_t \cap B \in BOX_n^d))),$$

$$R(j,1) : \exists s \ (\forall s' \geq s \ (\forall B \in D(s') \ (RS_{B,j}(s') = r_{B,j}))), \quad j \in [1,d],$$

$$R(j,2) : \exists s \ (\forall s' \geq s \ (\forall B \in D(s') \ (RS^*_{B,j}(s') = l_{B,j}))), \quad j \in [1,d].$$

The priority of the requirements is

$$R(0) > R(j_1,1) = R(j_2,2), \quad \forall j_1, j_2 \in [1,d].$$

We say that $R(0)$ is injured at stage s, if $\exists B \in D(s) \ \ (C_t \cap B \notin BOX_n^d)$. $\forall (j,k) \in [1,d] \times [1,2]$, in the same manner as we did in the construction of Algorithm LR, we define injury, positive injury and negative injury for $R(j,k)$. Let $W(s)$ be the set of all the CE's received by the end of stage s, and $P(s)$ be the set of all PCE's in $W(s)$. We will not define $D(s)$ and $P(s)$ in the construction of Algorithm LUR, since they are uniquely defined by $I(i,s)$ and $W(s)$, respectively. As in the previous section, fix an integer-pairing function $<\cdot,\cdot>$.

Learning Algorithm LUR

Stage $<0,0>$. $\forall i \in [1,d]$, set $I(i,<0,0>) = \{[0,n-1]\}$. Let $H(<0,0>) = \phi$, and $W(<0,0>) = \phi$. Go to stage $<1,0>$.

Stage $<s,m> > 0$. Ask an equivalence query for $H(<s-1,m>)$. If yes then stop, otherwise one receives a CE $x_{<s-1,m>}$. Let $Z_1 = W(<s-1,m>) \cup \{x_{<s-1,m>}\}$, and Z_2 be the set of all PCE's in Z_1. Decide whether $\exists B = \prod_{i=1}^d [e_i, f_i] \in D(<s-1,m>)$ such that, $\exists r_s \in (Z_1 - Z_2) \cap B$ such that $(r_s, Z_2 \cap B)$ is a witness for $B \cap C_t$. If yes then execute Case (i). Otherwise, execute Case (ii).

Case (i). $\forall i \in [1,d]$, let $z_i = e_i + \lfloor \frac{f_i - e_i}{2} \rfloor$. if $e_i < f_i$, then set $I(i,<s,m>) = (I(i, <s-1,m>) - \{[e_i,f_i]\}) \cup \{[e_i,z_i],[z_i+1,f_i]\}$, or set $I(i,<s,m>) = I(i,<s-1,m>)$ otherwise. Set $W(<s,m>) = Z_1$. Undo all procedures by setting $H_E(<s,m>) = rec(E \cap P(<s,m>))$, $\forall E \in D(<s,m>)$. Go to stage $<s+1,m>$.

Case (ii). Within the subdomain $B = \prod_{i=1}^{d}[e_i, f_i]$ with $x_{<s-1,m>} \in B$, in the same manner as one did in the construction of Algorithm LR, $\forall i \in [1, d]$, search for $r_{B,i}$ and $l_{B,i}$ using $S_{B,i}$ and $S^*_{B,i}$, respectively. No actions will be taken for all the other subdomains in $D(<s-1, m>)$. Set $W(<s-1, m+1>) = Z_1$, and $I(i,<s-1, m+1>) = I(i,<s-1, m>)$. Go to stage $<s, m+1>$.

Claim 6.2. *Given $B = \prod_{j=1}^{d}[e_j, f_j] \in D(<s, m>)$. If $\forall j \in [1, d]$, either $[e_j, f_j]$ contains no parameters of C_t, or $e_j = f_j$, then $C_t \cap B \in BOX_n^d$. In other words, if $C_t \cap B \notin BOX_n^d$, then $\exists j \in [1, d]$ such that $[e_j, f_j]$ contains at least one parameter of C_t and $e_j \neq f_j$.*

Proof. Assume that the condition is true. Then, $\forall i \in [1, k]$, $\forall j \in [1, d]$, either $[e_j, f_j] \subseteq [a_{i,j}, b_{i,j}]$, or $[e_j, f_j] \cap [a_{i,j}, b_{i,j}] = \phi$. Thus, for any C_i in C_t, $C_i \cap B \neq \phi$ implies $B \subseteq C_i$. Hence, $B \cap C_t = \bigcup\{B \cap C_i | i \in [1, k]\} = \bigcup\{B \cap C_i | B \cap C_i \neq \phi \ \& \ i \in [1, k]\}$ $= \bigcup\{B \cap C_i | B \subseteq C_i \ \& \ i \in [1, k]\} = \bigcup\{B | B \subseteq C_i \ \& \ i \in [1, k]\} = B$, this means that $C_t \cap B = B \in BOX_n^d$. □

Claim 6.3. *During the learning process of Algorithm LUR, Case (i) can be executed at most $2kd(\log n - 1)$ times, and $|I(i, <s, m>)| \leq 4k + 2k(\log n - 1) + 4k(d-1)(\log n - 1)$, $\forall i \in [1, d], s \geq 1, m \geq 1$.*

Proof. $\forall i \in [1, d]$, we construct a tree $T(i)$ by tracing the generation of $I(i,<s, m>)$. The root of $T(i)$ is $[0, n-1] \in I(i, <0, 0>)$. At stage $<s, m>$, we expand $T(i), \forall i \in [1, d]$, if Case (i) is executed, or do nothing otherwise. More precisely, when Case (i) is executed, $\exists B = \prod_{i=1}^{d}[e_i, f_i] \in D(<s-1, m>)$ (i.e., $[e_i, f_i]$ is a leaf in the current $T(i)$), one finds a witness in B, so one adds $[e_i, z_i]$ and $[z_i + 1, f_i]$ as two children to the leaf $[e_i, f_i]$ if $e_i < f_i$, or adds no children to $[e_i, f_i]$ otherwise. In the first case, we say that $T(i)$ receives an operation for the node $[e_i, f_i]$. Moreover, when $[e_i, f_i]$ contains at least one parameter of C_t at the i-th dimension, we say that the operation is effective.

We say a leaf in $T(i)$ is valid if it contains at least one parameter of C_t. Because each time when two children $[e_i, z_i]$ and $[z_i + 1, f_i]$ are added to a node $[e_i, f_i]$ in $T(i)$, they are disjoint and of length at most half of that of their parent $[e_i, f_i]$. This implies that

any two nodes in $T(i)$ with a common ancestor (different from themselves) are disjoint, and the longest path in $T(i)$ is at most $\log n$. Since C_t has at most $2k$ parameters at the i-th dimension, there are at most $2k$ valid leaves in $T(i)$. Assume that $T(i)$ receives an effective operation for a node w. Then, w contains at least one parameters of C_t, this implies that at least one leave of $T(i)$ with an ancestor w contains at least one parameter. Thus, an effective operation of $T(i)$ corresponds to one non-leaf node in the path from the root to some valid leaf of $T(i)$. Hence, $T(i)$ receives at most $2k(\log n - 1)$ effective operations.

Suppose that at a stage Case (i) is executed for the subdomain $B = \prod_{i=1}^{d}[e_i, f_i]$. Then, according to Algorithm LUR, $B \cap C_t \notin BOX_n^d$. So, by Claim 6.2, there is at least one j such that $[e_j, f_j]$ contains at least one parameter of C_t at the j-th dimension and $e_j < f_j$. This means that at least one of the d trees receives an effective operation, while at most $(d-1)$ trees receive a non-effective operation. By the above analysis, each $T(i)$ receives at most $2k(\log n - 1)$ effective operations and $2k(d-1)(\log n - 1)$ non-effective operations. Hence, the number of executions of Case (i) is at most $2kd(\log n - 1)$. Note that each operation produces two nodes. Thus, any tree $T(i)$ has at most $4k + 2k(\log n - 1) + 4k(d-1)(\log n - 1)$ nodes. This implies that $|I(i, <s, m>)|$ is at most $4k + 2k(\log n - 1) + 4k(d-1)(\log n - 1)$. □

Claim 6.4. *Assume that Case (i) was executed at stage* $<s, m'>$ *and* $<s+1, m''>$. *Then,* $m'' - m' = O((4kd \log n)^d d^2 \log n)$.

Proof. In the construction of Algorithm $LUR(e)$, the parameter s is incremented by one only when Case (i) is executed, while the parameter m is incremented by one only when Case (ii) is executed. Thus, during the period of the learning process between stage $<s, m'>$ and $<s, m''>$ (excluding those two stages), only Case (ii) is executed. At any stage $<s, m>$, for $m \in [m'+1, m''-1]$, for each domain $B \in D(<s, m>)$ the learner uses $2d$ procedures $S_{B,j}$ and $S^*_{B,j}$, $\forall j \in [1, d]$, to learn $C_t \cap B$. With the same analysis as we did for Algorithm LR, the learner needs $O(d^2 \log n)$ CE's to learn $C_t \cap B$ if it is in BOX_n^d, or finds a witness for it otherwise. By Claim 6.3, there are at most $4kd \log n$ intervals in $I(i, <s-1, m>)$ when $n \geq 8$. Hence, $D(<s-1, m>)$ contains at most $(4kd \log n)^d$ subdomains when $n \geq 8$. Therefore, $m'' - m' = O((4kd \log n)^d d^2 \log n)$. □

Claim 6.5. *Requirements $R(0)$ will be satisfied after receiving $O(4kd\log n)^{d+1}d^2\log n)$ CE's.*

Proof. By Claim 6.3 and 6.4. □

Suppose that $R(0)$ is satisfied at stage $<s, m>$ by finding the last witness in the execution of Case (i). Then, at stage $<s+1, m>$ for any $B \in D(<s, m>)$, $B \cap C_t \in BOX_n^d$. This implies that $R(0)$ will never be injured again. Thus, at any later stage $<s+1, m'>$ with $m' > m$, for each domain $B \in D(<s, m'>)$ the learner uses $2d$ procedures $S_{E,j}$ and $S^*_{E,j}$, $\forall j \in [1, d]$, to learn $C_t \cap B$. With the same analysis as we did for Algorithm LR, the learner needs $O(d^2 \log n)$ CE's to learn $C_t \cap E$ (because it is in BOX_n^d). By Claim 6.4, there are at most $(4kd(\log n - 1) - 2k(\log n - 3))^d$ domains in $D(<s, m'>)$. Therefore, the learner learns C_t with $O((4kd\log n)^d d^2 \log n)$ CE's after stage $<s, m>$. Together with Claim 6.5, $O(4kd\log n)^{d+1}d^2\log n)$ CE's are required in the learning process. The hypotheses issued by the learner are unions of at most $(4kd(\log n - 1) - 2k(\log n - 3))^d$ rectangles. The time complexity of Algorithm $LDD(e)$ is $ploy(k, d, \log n)$, because an execution of any procedure needs time polynomially in $\log n$ and, finding a witness at any stage $<s', m''>$ for a subdomain B needs only searching for each NCE in $(W(<s' - 1, m''>) - P(<s' - 1, m''>)) \cap B$ and testing whether it is in $rec(P(<s' - 1, m''>) \cap B)$. □

Acknowledgment. We thank Wolfgang Maass for his insights into linking the finite injury priority method to our early approach to learning unions of rectangles. We also thank Ming Li for his valuable discussions on this topic, and William Gasarch for his encouraging comments on the use of the injury priority method in this setting.

References

[Aa] D. Angluin, "Computational learning theory: Survey and selected bibliography", *Proc of the 24th Annual Symposium on Theory of Computing*, 1992, pages 351-367.

[Ab] D. Angluin, "Queries and concept learning", *Machine Learning*, 2, 1988, pages 319-342.

[AU] P. Auer, "On-line learning of rectangles in a noisy environment", *Proc of the 6th Annual Workshop on Computational Learning Theory*, 1993, pages 253-261.

[BEHW] A. Blumer, A. Ehrenfeucht, D. David, and M. Warmuth, "Learnability and the Vapnik-Chervonenkis dimension", *J. ACM*, pages 929-965, 1989.

[Ba] N. Bshouty, "Exact learning via the monotone theory", *Proc of the 35th Annual Symposium on Foundations of Computer Science*, 1993.

[Bb] N. Bshouty, "On learning discretized geometric concepts", unpublished manuscript, May, 94.

[BCH] N. Bshouty, Z. Chen, S. Homer, "On learning discretized geometric concepts", *Proc of the 35th Annual Symposium on Foundations of Computer Science*, pages 54-63, 1994.

[BGGM] N. Bshouty, P. Goldberg, S. Goldman, and D. Mathias, "Exact learning of discretized geometric concepts", Technical Report WUCS-94-19, Dept of Computer Science, Washington University at St. Louis, July, 1994.

[BM] W. Bultman, W. Maass, "Fast identification of geometric objects with membership queries", *Proc of the 4th Annual ACM Workshop on Computational Learning Theory*, pages 337-353, 1991.

[C] Z. Chen, "Learning unions of two rectangles in the plane with equivalence queries", *Proc of the 6th Annual ACM Conference on Computational Learning Theory*, pages 243-253, 1993.

[CH] Z. Chen, S. Homer, "Learning unions of rectangles with queries", Technical Report BUCS-93-10, Dept of Computer Science, Boston University, July, 93.

[CMa] Z. Chen, W. Maass, "On-line learning of rectangles", *Proc of the 5th Annual Workshop on Computational Learning Theory*, pages 16-28, 1992.

[CMb] Z. Chen, W. Maass, "On-line learning of rectangles and unions of rectangles", *Machine Learning*, 17, pages 201-223, 1994.

[CF] P. Cohen, E. Feigenbaum, "The Handbook of Artificial Intelligence", Vol. 3, William Kaufman, 1982.

[F] R. Friedberg, "Two recursively enumerable sets of incomparable degrees of unsolvability", *Proc. Natl. Acad. Sci. USA*, 43, 1957, pages 236-238. and unions of rectangles",

[GGM] P. Goldberg, S. Goldman, and D. Mathias, "Learning unions of rectangles with membership and equivalence queries", *Proc of the 7th annual ACM Conference on Computational Learning Theory*, 1994.

[J] J. Jackson, "An efficient membership-query algorithm for learning DNF with respect to the uniform distribution", *Proc of the 35th Annual Symposium on Foundations of Computer Science*, 1994.

[L] N. Littlestone, ' 'Learning quickly when irrelevant attributes abound: a new linear threshold algorithm", *Machine Learning*, 2, 1987, pages 285-318.

[BM] P. Long, M. Warmuth, "Composite geometric concepts and polynomial predictability", *Proc of the 3th Annual Workshop on Computational Learning Theory*, pages 273-287, 1991.

[M] W. Maass, "On the Complexity of Learning on Feed-forward Neural Nets", Lecture Notes, the Advanced School on Computational Learning and Cryptography of the EATCS, Vietri sul Mare, Italy, September, 1993.

[MTa] W. Maass, G. Turán, "On the complexity of learning from counterexamples", *Proc of the 30th Annual Symposium on Foundations of Computer Science*, 1989, pages 262-267.

[MTb] W. Maass, G. Turán, "On the complexity of learning from counterexamples and membership queries", *Proc of the 31th Annual Symposium on Foundations of Computer Science*, 1990, pages 203-210.

[MTc] W. Maass, G. Turán, "Algorithms and lower bounds for on-line learning of geometric concepts", Report 316 (Oct. 1991), *IIG-Report Series, Technische universitaet Graz.*

[MTd] W. Maass, G. Turán, "Algorithms and lower bounds for on-line learning of geometric concepts", *Machine Learning*, 1994, pages 251-269.

[MTe] W. Maass, G. Turán, "Lower bound methods and separation results for on-line learning models", *Machine Learning*, 1992, pages 107-145.

[MTf] W. Maass, 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, to appear.

[MW] W. Maass, M. Warmuth, "Efficient learning with virtual threshold gates", Technical Report 395 of the Institutes for Information Processing Graz, August, 1994. (To appear in *Machine Learning*.)

[MU] A. Muchnik, "On the unsolvability of the problem of reducibility in the theory of algorithms", *Dokl. Akad. Nauk SSSR*, N.S., 108, 1956, pages 194-197.

[PV] L. Pitt, L. G. Valiant, "Computational limitations on learning from examples", *J. of the ACM*, 35, 1988, 965-984.

[R] R. Rivest, "Learning decision lists", *Machine Learning*, pages 223-242, 1987.

[S] R. Soare, "Recursively Enumerable Sets and Degrees", Springer-Verlag, 1987.

[V] L. Valiant, "A theory of the learnable", *Comm. of the ACM*, 27, 1984, pages 1134-1142.