Learning Unions of Rectangles with Queries

Learning Unions of Rectangles with Queries [*]

Zhixiang Chen Steven Homer
Department of Computer Science
Boston University
Boston, MA 02215

[*]The authors were supported by NSF grant CCR91-9103055. The first author was also supported by a Boston University Presidential Graduate Fellowship. Email address: homer@cs.bu.edu, zchen@cs.bu.edu

Abstract

We investigate the efficient learnability of unions of k rectangles in the discrete plane $\{1, \ldots, n\}^2$ with equivalence and membership queries. We exhibit a learning algorithm that learns any union of k rectangles with $O(k^3 \log n)$ queries, while the time complexity of this algorithm is bounded by $O(k^5 \log n)$. We design our learning algorithm by finding "corners" and "edges" for rectangles contained in the target concept and then constructing the target concept from those "corners" and "edges". Our result provides a first approach to on-line learning of nontrivial subclasses of unions of intersections of halfspaces with equivalence and membership queries.

1 Introduction

Learning unions of rectangles is closely related to other central problems in machine learning theory. It is a generalization of learning DNF (disjunctive normal form) formulas and a special case of unions of intersections of half-spaces in $\{1, \ldots, n\}^d$, two of the major open problems in computational learning theory. In addition, unions of rectangles is a very natural concept class, and many practical problems can be easily represented as special cases of this class. Thus, any efficient learning algorithm for unions of rectangles may find applications in practice.

In the pac-model, the class of single rectangles over the domain $\{1, \ldots, n\}^d$ was shown to be polynomial time predictable by Blumer, Ehrenfeucht, Haussler and Warmuth [BEHW]. Long and Warmuth [LW] further proved that unions of a constant number of rectangles over the domain $\{1, \ldots, n\}^d$ is polynomial time predictable. However, the problem of predicting unions of nonconstant number of rectangles is open [LW].

In the on-line model with queries, the problem of learning a single rectangle with only equivalence queries over the domain $\{1, \ldots, n\}^d$ has been well-studied. The first known lower bound for this problem is $\Omega(d \log n)$ (see [L] and [MTc]). There is a learning algorithm with complexity $O(dn)$ which always issues as its next hypothesis the smallest rectangle that is consistent with all preceding counterexamples (see the algorithm for the complementary class 1-CNF in Valiant's seminal paper [V]). Another $O(2^d \log n)$ learning algorithm was exhibited in [MTa] and [MTb]. The question whether there is a learning algorithm for single rectangles whose complexity is $O(poly(d, \log n))$ was proposed by David Haussler (see also [MTb]). Chen and Maass [CMa, CMb] gave a positive solution to this open question by introducing a new design technique.

The learning algorithm in [CMa, CMb] for rectangles consists of $2d$ separate search strategies which search for the $2d$ boundaries of the target rectangle. A learning algorithm with this type of modular design tends to fail because of the well-known " credit assignment problem": which of the $2d$ local search strategies should be "blamed" when the global algorithm makes an error? This difficulty was overcome there (see also [CMc]) by employing local search strategies that are able to tolerate certain types of one-sided errors. Based on this design technique and more powerful local search strategies which can tolerate certain types of two-sided errors, Chen [C] exhibited an $O(\log^2 n)$ algorithm for learning unions of two rectangles in the discrete plane $\{1, \ldots, n\}^2$ with only equivalence queries while the hypothesis space of the learning algorithm is the same as the target concept class. Recently, Auer [AU] has designed a variation of the learning algorithm for BOX_n^d that also learns with $O(d^2 \log n)$ counterexamples, and that tolerates a fraction of at most $1/(16d^2)$ false counterexamples (in arbitrary distribution). He also proved that the number of equivalence queries required to learn BOX_n^d is bounded below by $\Omega(d^2 \log n / \log d)$.

In contrast to learning unions of k rectangles, much research has been done on learning k-term DNF formulas. The standard strategy for learning k-term DNF formulas uses a hypothesis space of l-CNF formulas and runs in time $O(d^k)$, where d is the number of input variables [V]. Another $O(d^k)$ learning algorithm which uses a representation of general DNF formulas was also obtained in [BS]. With the added ability of the learner to make membership queries, a wider collection of DNF subclasses are known to be learnable. These include monotone DNF formulas in [V] and read-twice DNF formulas in [AP] and [H]. For learning k-term DNF formulas with equivalence queries and membership queries, Angluin [Ab] designed an algorithm whose running time is $O(d^{k^2})$; another remarkable result is due to Blum and Rudich [BR], they exhibited an $O(d2^{O(k)})$ learning algorithm.

Enlightened by those positive results on learning k-term DNF formulas with queries, we begin to investigate the efficient learnability of unions of k rectangles, a general class of k-term DNF formulas. We say a concept is a "tree" if it is a union of at most k rectangles that are "piled" on some fixed "trunk" (a vertical line). We will give a formal definition of a "tree" in the next section. In section 3 we design an $O(k \log n)$ algorithm for learning a "tree" by finding its "corners" and "edges", assuming that its trunk is known to the learner. We then show how to use this design strategy to learn any union of k rectangles. We prove in section 4 that with a slightly larger hypothesis space one can learn any unions of k rectangles over the domain $\{1, \ldots, n\}^2$ using $O(k^3 \log n)$ equivalence and membership queries, while the time complexity of the learning algorithm is bounded by $O(k^5 \log n)$. Our result provides a contrast to the known upper bounds for the learning complexity of k-term DNF formulas, since all those known bounds are exponential in k.

Baum [Ba] studied the learnability of intersections of halfspaces. He has shown that intersections of halfspaces are probably almost correctly learnable in an extended version of the PAC model where the learner may also ask membership queries, provided that the distribution and the target concept are chosen in a "non-malicious manner". Baum [Bb] also studied the pac-learnabilty of a union of halfspaces. Bultman and Maass [BM] have shown that a variety of intersections of halfspaces over the domain $\{1, \ldots, n\}^2$ are efficiently learnable using only membership queries. One may note that, in essence, Bultman and Maass' learning algorithm requires exactly one equivalence query. Our result provides a first approach to on-line learning of nontrivial subclasses of unions of intersections of halfspaces with equivalence and membership queries.

2 Preliminaries

Our learning model is the standard model for on-line learning with equivalence and membership queries (see [Aa], [L] and [MTc]). A learning process for a concept class $\mathbf{C}$ over a domain X is viewed as a dialogue between a learner A and the environment. The goal of the learner A is to learn an unknown target concept $C_t \in \mathbf{C}$ that has been fixed by the environment. In order to gain information about C_t the learner proposes hypothesis H from a fixed hypothesis space $\mathbf{H}$ with $\mathbf{C} \subseteq \mathbf{H} \subseteq 2^X$. Whenever $H \neq C_t$ for the proposed hypothesis H, the environment responds with some counterexample $g \in H \triangle C_t = (C_t - H) \cup (H - C_t)$. g is called a positive counterexample (PCE) if $g \in (C_t - H)$. g is called a negative counterexample (NCE) if $g \in (H - C_t)$. The learner

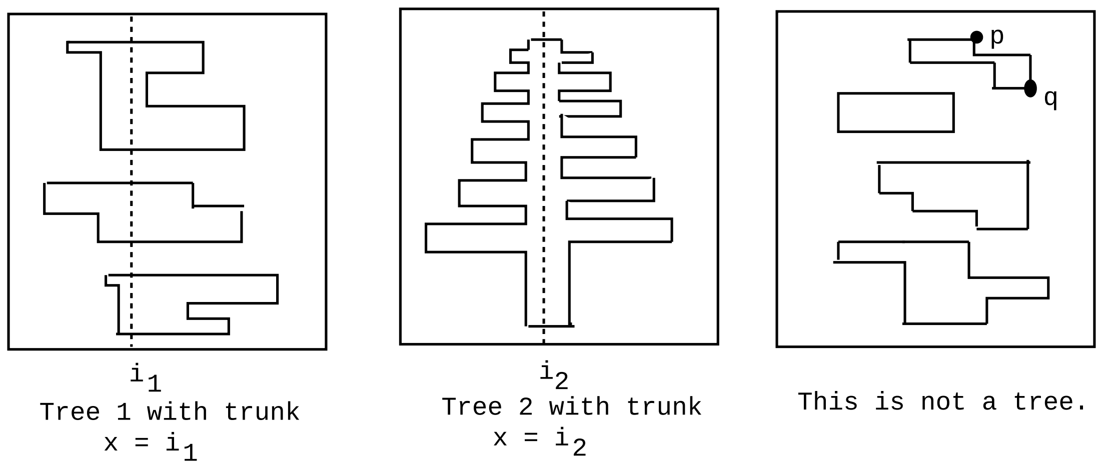


Figure 1: Tree Concept via Non-tree Concept

may also ask membership queries "$x \in C_t$?" for some $x \in X$, to which the environment responds with the reply "yes" or "no". One also calls this x a counterexample. The learning complexity LC-MEMB(A) of a learner A is the maximal number of equivalence and membership queries that A needs in some learning process of this type before it can uniquely identify the target concept C_t. The learning complexity LC-$MEMB^H(\mathbf{C})$ of the concept class $\mathbf{C}$ is the learning complexity of the best algorithm that learns $\mathbf{C}$, assuming that the algorithm issues only hypotheses in $\mathbf{H}$. One sets LC-MEMB($\mathbf{C}$)=LC-$MEMB^{\mathbf{C}}(\mathbf{C})$.

¿From now on we let $[i,j]$ denote the set $\{i, i+1, \ldots, j\}$ for any integers $i, j \in N$ with $i \leq j$. Let $BOX_n^d = \{\prod_{i=1}^d [a_i, b_i] \mid 1 \leq a_i, b_i \leq n \ for \ i = 1, \ldots, d\}$. We consider for any integer $k \geq 1$ the concept class

$$\text{U-k-}BOX_n^d = \{C_1 \cup \cdots \cup C_k \mid C_i \in BOX_n^d \ for \ i = 1, \ldots, k\}.$$

over the domain $X = [1,n]^d$. Note that for any target concept $C_t = C_1 \cup \ldots \cup C_k \in$ U-k-BOX_n^2, there may exist different sets of k rectangles $D_1, \ldots, D_k$ such that $C_t = D_1 \cup \ldots \cup D_k$. Note also that a single point and a line segment are special cases of rectangles. For any $C_t \in$ U-k-BOX_n^2, one says that C_t is a tree if there is an $i \in [1,n]$ such that, for any $z = (u,v) \in C_t$, $[i,u] \times [v,v] \subseteq C_t$ if $i \leq u$, or $[u,i] \times [v,v] \subseteq C_t$ if $u < i$. One calls the y-axis-parallel line $x = i$ the trunk of the tree C_t. For a given tree, there may exist many different trunks for it. In figure 1, the first two concepts are trees, while the third concept is not a tree, since the two points p and q can not be assigned to any trunk at the same time. Define

$$TREE_{n,k} = \{C_t \in \text{U-k-}BOX_n^2 \mid C_t \ is \ a \ tree\}$$

Let $INT_n = \{[i,j] \mid i \leq j \ and \ i,j \in [1,n]\}$. Define the concept class U-k-$INT_n = \{I_1 \cup \ldots \cup I_k \mid I_1, \ldots, I_k \in INT_n\}$ over the domain $[0, n+1]$. Note that for any $C_t \in$ U-k-BOX_n^2, for any axis-parallel segment I_t in the domain $[0, n+1]^2$, $C_t \cap I_t$ can be viewed as a concept transformed from some concept in U-k-INT_n. For convenience, we simply consider $C_t \cap I_t$ as a concept in U-k-INT_n. Consider any target concept $I_t \in$

U-k-INT_n, let $x \in I_t$ such that x is known to the learner. one defines the following binary search algorithm with membership queries to find a boundary (a, b) for I_t on the right side of x. Where one says that (a, b) is a boundary of I_t if $a \in I_t$, $b \notin I_t$, and either $b = a + 1$, or $b = a - 1$.

Algorithm BM_n**:**

```
    Set $b_1 = x, h_1 = n + 1$.
    For any $b_r, h_r, r \geq 1$, let $q_r = b_r + \lceil \frac{h_r - b_r}{2} \rceil$.
    If $h_r = b_r + 1$ then stop, otherwise ask whether $q_r$ is in $I_t$.
    If yes then set $b_{r+1} = q_r$, and $h_{r+1} = h_r$.
    Otherwise, set $b_{r+1} = b_r$, $h_{r+1} = q_r$
```

Proposition 2.1 *For any target concept $I_t \in$ U-k-INT_n, assume that $x \in I_t$ is known to the learner. Then, the algorithm BM_n finds a boundary $(b, b + 1)$ for the target concept I_t with at most $\lceil \log n \rceil + 3$ membership queries.*

Proof. In the process of the algorithm BM_n, for any $r \geq 1$, $x \leq b_r < h_r \leq n + 1$, $b_r \in I_t, h_r \notin I_t$, $h_{r+1} - b_{r+1} \leq \frac{n}{2^r} + 3$. Thus, with at most $\lceil \log n \rceil + 3$ membership queries, one can find a $b_r \in I_t$ with $h_r = b_r + 1 \notin I_t$. □

Remark. One can also define a dual version of the algorithm DBM_n which finds a boundary $(b, b - 1)$ for I_t on the left side of x with at most $\lceil \log n \rceil + 3$ queries.

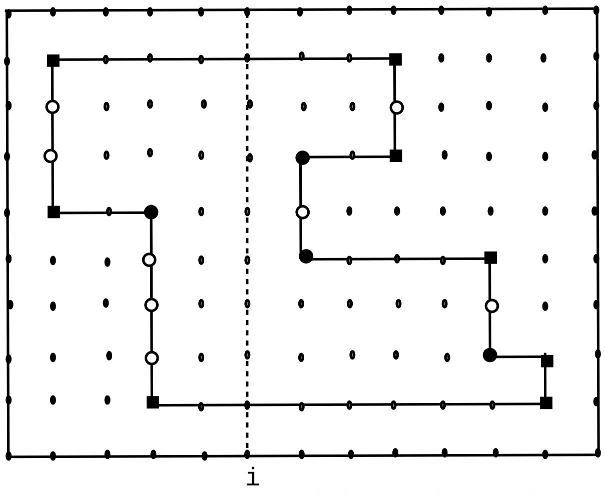
3 Learning Tree-Shape Unions of k Rectangles

We assume in this section that for each tree-shape target concept $C_t \in TREE_{n,k}$, the trunk of C_t is known to the learner when the learner begins to learn it. The idea behind this assumption is that when the learner wants to learn a target concept in U-k-BOX_n^2, he can use queries to find some trunks of the trees of the target concept. Once he knows the trunk of a tree, he then employs a copy of the algorithm for tree-shape concepts developed in this section to find the tree.

We now consider the structural properties of a tree-shape concept. We first extend our domain $[1, n]^2$ to $[0, n + 1]^2$ by assuming that all points in $[0, n + 1]^2 - [1, n]^2$ are outside any concept in U-k-BOX_n^2. For any $C_t \in TREE_{n,k}$, we say that $r =< r_1, r_2, r_3 >$ with $r_j = (x_j, y_j), j = 1, 2, 3,$ is a corner-witness for C_t, if $r_1 \in C_t$, $r_2, r_3 \notin C_t$, $y_2 = y_1, x_3 = x_1$, and one of the following conditions holds:

1. $x_2 = x_1 + 1, y_3 = y_1 + 1$;
2. $x_2 = x_1 + 1, y_3 = y_1 - 1$;
3. $x_2 = x_1 - 1, y_3 = y_1 + 1$;
4. $x_2 = x_1 - 1, y_3 = y_1 - 1$.

The intuition of a corner-witness $r =< r_1, r_2, r_3 >$ of a concept $C_t \in TREE_{n,k}$ is that r_1 is a real corner of some rectangle contained in C_t. One says that $e =< e_1, e_2 >$ with



Tree concept with trunk x = i

Every ■ point relates to a corner-witness.
Every ■ or ○ point relates to an edge-witness.
Each ● point does not relate to any witness.

Figure 2: Corner-witness via Edge-witness

$e_j = (x_j, y_j)$, $j = 1, 2$, is an edge-witness of C_t, if $e_1 \in C_t$, $e_2 \notin C_t$, $y_2 = y_1$, and one of the following conditions holds:

5. $x_2 = x_1 + 1$;
6. $x_2 = x_1 - 1$.

Note that when $e =< e_1, e_2 >$ is an edge-witness for C_t, then e_1 is on a vertical edge of some rectangle contained in C_t. For any target concept $C_t \in TREE_{n,k}$, let t_0 be the fixed trunk for C_t, $R(C_t)$ be the set of all corner-witnesses and, $E(C_t)$ be the set of all edge-witnesses. We also define an equivalence relation among edge-witnesses in $E(C_t)$. For any two edge-witnesses $e =< e_1, e_2 >$ and $d =< d_1, d_2 >$ in $E(C_t)$, $e_j = (x_j, y_j), d_j = (u_j, v_j), j = 1, 2$, we say that e is equivalent to d iff, $x_1 = u_1$, $[min\{x_1, t_0\}, max\{x_1, t_0\}] \times [min\{y_1, v_1\}, max\{y_1, v_1\}] \subseteq C_t$, and there are no corner-witnesses between the x-axis-parallel lines $y = y_1$ and $y = v_1$, i.e. for any corner-witness $< r_1, r_2, r_3 >$ with $r_j = (p_j, q_j), j = 1, 2, 3$, either $q_1 \le min\{y_1, v_1\}$, or $q_1 \ge max\{y_1, v_1\}$. Using this equivalence relation, we can divide $E(C_t)$ into disjoint equivalence classes such that any two edge-witnesses are in the same equivalence classes if they are equivalent. Let $E^*(C_t)$ denote the set of all equivalence classes of $E(C_t)$.

Lemma 3.1 $\|R(C_t)\| \le 4k$, $\|E^*(C_t)\| \le 4k - 2$.

Proof. For any corner-witness $r =< r_1, r_2, r_3 >\in R(C_t)$, by the definition, $r_1 \in C_t$, but $r_2, r_3 \notin C_t$. Fix a rectangle C_j in C_t such that $r_1 \in C_j$, then r_1 is a real corner of C_j. We also note that two different corner-witnesses correspond to two different real corners of one rectangle (or two different rectangles) in C_t. There are at most k different rectangles in C_t, each contains at most four different corners. Thus, $\|R(C_t)\| \le 4k$. For each corner-witness $r =< r_1, r_2, r_3 > \in R(C_t)$, one draws an x-axis-parallel line from the trunk of C_t through the point r_1 to the boundary of the domain. Then we know, by the definition of $E^*(C_t)$, that every equivalence class corresponds to a vertical boundary segment of some rectangles in C_t between two of those adjacent x-axis-parallel lines. There are at most k different rectangles in C_t, one can draw at most $2k$ different x-axis-parallel lines on either side of the trunk. Thus, there are at most $2(2k-1)$ different equivalence classes in $E^*(C_t)$. $\Box$

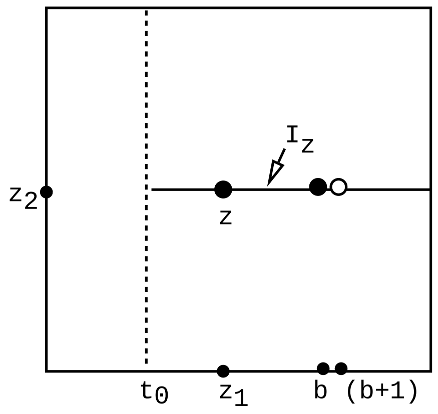
For any $C_t \in TREE_{n,k}$, define $S(C_t) = R(C_t) \cup E^*(C_t)$. In other words, $S(C_t)$ is the set of all real corners and all real vertical edges (each of those edge represents a different equivalence class) of the rectangles in C_t. An easy observation is that, given the set $S(C_t)$ of a concept C_t, one can efficiently construct C_t.

Theorem 3.2 *There is an algorithm L-TREE for learning $TREE_{n,k}$ such that, LC-MEMB(L-TREE)=$O(k \log n)$, and the hypothesis space used by the learning algorithm L-TREE is U-(4k-2)-BOX_n^2, provided that the trunk of the input concept is known to the learner. Moreover, the computation steps of the learning algorithm L-TREE are also bounded by $O(k^3 \log n)$.*

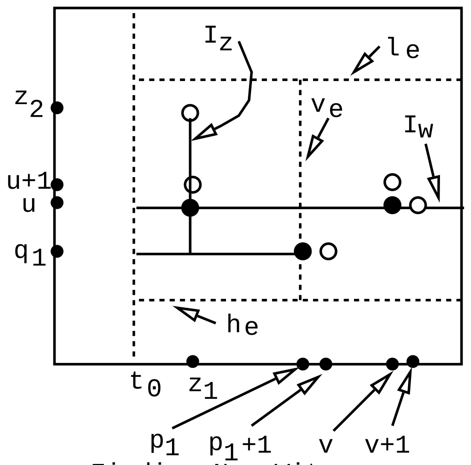
Proof. Consider any target concept $C_t \in TREE_{n,k}$. Let t_0 be the trunk of C_t which is known to the learner. The algorithm L-TREE works in stages. At each stage r, let $R_r(C_t)$ and $E_r(C_t)$ be the set of all corner-witnesses and the set of all edge-witnesses received by the learner within first r stages, respectively. Let W_r be the set of all counterexamples received by the learner within first r stages. For any corner-witness $g =< g_1, g_2, g_3 > \in R_r(C_t)$, one draws a horizontal line from the trunk $x = t_0$ through the point g_1 to the boundary of the domain. Let L_r be the set of all those horizontal lines and the additional two lines $y = 0$, i.e. the bottom boundary of the domain, and $y = n + 1$, i.e. the upper boundary of the domain. Initially, one sets $W_0 = E_0(C_t) = R_0(C_t) = \phi$, and $L_0 = \{y = 0, y = n + 1\}$.

At stage 0, the learner proposes the hypothesis $H_1 = \phi$ to the environment to ask an equivalence query. If $C_t = H_1 = \phi$ then the learner learns C_t, otherwise the learner receives a PCE $z = (z_1, z_2)$ from the environment. The algorithm L-TREE then uses a copy of the algorithm BM_n on the line segment $I_z = [0, n+1] \times [z_2, z_2]$ to find a boundary of $C_t \cap I_z$ on the right side of z if z is not on the left side of the trunk $x = t_0$, or on the left side of z if otherwise. We may assume that z is not on the left side of the trunk, the case that z is on the left side can be coped with in the same manner. By Proposition 2.1, one finds an edge-witness $e =< (b, y), (b+1, y) >$ for C_t. One sets $E_1 = \{e\}$, and sets W_1 be the set of all counterexamples received during the process of the algorithm BM_n. One keeps $R_1(C_t) = R_0(C_t)$ and $L_1 = L_0$. At any stage $r \geq 1$, the algorithm L-TREE executes the following three tasks sequentially.

Task 1: Constructing the New Hypothesis H_{r+1}.
For each edge witness $e =< e_1, e_2 > \in E_r(C_t)$, one finds the lowest horizontal line l_e above the point e_1 and the highest horizontal line h_e below e_1 among all lines in L_r. One then draws a vertical line v_e from h_e through e_1 to l_e. One finally defines the



Finding New Witness: When z is a PCE



Finding New Witness: When z is an NCE

Figure 3: Finding New Witnesses

new hypothesis H_{r+1} as the set of all those points p such that, there is an edge-witness $e \in E_r(C_t)$ such that p is in the rectangle formed by the trunk, the lines v_e, l_e and h_e.

Task 2: Asking an Equivalence Query.
One checks whether H_{r+1} is consistent with all counterexamples in W_r. If yes then one asks an equivalence query for H_{r+1} to the environment. If $C_t = H_{r+1}$ then one learns C_t, otherwise one receives a counterexamples from the environment, one then puts this counterexample into W_r.

Task 3: Finding New Witnesses.
One finds a counterexample $z = (z_1, z_2) \in W_r$. We may assume that $z_1 \geq t_0$, the case that $z_1 < t_0$ can be treated similarly. When z is a PCE, then one employs a copy of the algorithm BM_{n-t_0+1} on the line segment $I_z = [t_0, n+1] \times [z_2, z_2]$ to find a new edge-witness $e =< (b, z_2), (b+1, z_2) >$ on the right side of z.

When $z = (z_1, z_2)$ is an NCE, one then finds an edge-witness $e =< e_1, e_2 >$ such that z is in the rectangle formed by the trunk, the lines v_e, l_e and h_e (see Task 1 for the definitions of those lines). Let $e_j = (p_j, q_j), j = 1, 2$. We may also assume that $z_2 > q_1$, since the case that $z_2 < q_1$ can be treated in the same way. Because C_t is a tree and z is an NCE, $z_2 \neq q_1$. One first employs a copy of the algorithm $BM_{z_2-q_1+1}$ on the line segment $I_z = [z_1, z_1] \times [q_1, z_2]$ to find a boundary $w =< (z_1, u), (z_1, u+1) >$ for $C_t \cap I_z$. At next step, one employs another copy of the algorithm BM_{n-t_0+1} on the line segment $I_w = [t_0, n+1] \times [u, u]$ to finds an edge-witness $< (v, u), (v+1, u) >$. One also obtains a corner-witness $< (v, u), (v+1, u), (v, u+1) >$, since $(z_1, u+1) \notin C_t$ and C_t is a tree. The execution procedure of Task 3 is illustrated in figure 3.

Finally, put all the obtained edge-witnesses (corner-witnesses) into $E_r(C_t)$ $(R_r(C_t))$ to get $E_{r+1}(C_t)$ $(R_{r+1}(C_t))$. Put all the received counterexamples into W_r to get W_{r+1}.

We now analyze the complexity of the algorithm L-TREE. For each edge witness

$e =< e_1, e_2 > \in E_r(C_t)$, let l_e (h_e) be the lowest (highest) horizontal line above (below) the point e_1 among all lines in L_r. One defines an equivalence relation among edge-witnesses in $E_r(C_t)$ as follows, for any $e =< e_1, e_2 >$ and $d =< d_1, d_2 >$ in $E_r(C_t)$, $e_j = (x_j, y_j)$, $d_j = (u_j, v_j)$, $j = 1, 2$, e is equivalent to d iff, $x_1 = u_1$, $l_e = l_d, h_e = h_d$, and $[min\{t_0, x_1\}, max\{t_0, x_1\}] \times [min\{y_1, v_1\}, max\{y_1, v_1\}]$ contains no NCE's in W_r. Note that this equivalence relation is an approximation to the equivalence relation defined on edge-witnesses in $E(C_t)$. Let $E_r^*(C_t)$ denote the set of all equivalence classes derived by the equivalence relation among edge-witnesses in $E_r(C_t)$. Let $S_r(C_t) = R_r(C_t) \cup E_r^*(C_t)$.

Lemma 3.3 $\|S_{r+1}(C_t)\| \geq \|S_r(C_t)\| + 1$ *for any* $r \geq 1$, *provided that the learner doesn't learn* C_t *at stage* r.

Proof. At stage $r \geq 1$, the algorithm L-TREE first constructs the hypothesis H_{r+1} in the execution of Task 1, L-TREE then asks an equivalence query for H_r. Since the learner doesn't learn the target concept C_t at stage r by the assumption, he receives a counterexample from the environment and adds this counterexample into W_r. Thus, H_{r+1} is not consistent with counterexamples in W_r, this implies that the learner will find in the execution of Task 3 a counterexample $z = (z_1, z_2) \in W_r$ for H_r, L-TREE then finds new witnesses for C_t with z. We may assume that $z_1 \geq t_0$, the case that $z_1 < t_0$ can be proved in the same manner.

When z is a PCE, L-TREE employs a copy of the algorithm BM_{n-t_0+1} on the line segment $I_z = [t_0, n+1] \times [z_2, z_2]$ to find an edge-witness $e =< (b, z_2), (b+1, z_2) >$ on the right side of z. Let l_z (h_z) be the lowest (highest) horizontal line in L_r which is above (below) z. Since z is a PCE to H_{r+1}, according to the construction of the hypothesis H_{r+1}, there is no edge-witness $d = (d_1, d_2)$ in S_r such that, d_1 is between the lines l_z and h_z, and on the right side of the vertical line $x = z_1$. This implies that no edge-witness in $E_r(C_t)$ is equivalent to e. One sets $E_{r+1}(C_t) = E_r(C_t) \cup \{e\}$. Hence, $\|S_{r+1}(C_t)\| = \|R_{r+1}(C_t)\| + \|E_{r+1}^*(C_t)\| = \|R_r(C_t)\| + \|E_r^*(C_t)\| + 1 = \|S_r(C_t)\| + 1$.

Now assume that $z = (z_1, z_2)$ is an NCE. By the construction of H_{r+1}, there is an edge-witness $e =< e_1, e_2 > \in E_r(C_t)$ such that z is in the rectangle formed by the trunk $x = t_0$, and the lines v_e, l_e, and h_e. Note that there may exist other edge-witnesses in $E_r(C_t)$ between l_e and h_e. However, for any of those edge-witness d, $l_d = l_e, h_d = h_e$. In the execution of Task 3, the learner finds a corner-witness $g =< g_1, g_2, g_3 >$ for C_t such that g_1 is between the horizontal lines l_e and h_e, but not on either of those two lines, this implies that $g \in R_{r+1}(C_t) - R_r(C_t)$. Thus, $\|S_{r+1}(C_t)\| \geq \|S_r(C_t)\| + 1$. □

Lemma 3.4 $\|S_r(C_t)\| \leq 8k - 2$ *for any* $r \geq 1$.

Proof. By Lemma 3.1, $\|R_r(C_t)\| \leq \|R(C_t)\| \leq 4k$, since $R_r(C_t) \subseteq R(C_t)$. Note also that $E_r(C_t) \subseteq E(C_t)$. We define a mapping fun from $E_r^*(C_t)$ to $E^*(C_t)$ as follows. For any equivalence class $\delta \in E_r^*(C_t)$, let $min(\delta)$ denote the lowest edge-witness in δ. More precisely, for each edge-witness in δ we draw an x-axis-parallel line through it, then $min(\delta)$ is the edge-witness representing the lowest line. For any $\delta \in E^*(C_t)$, let $eq(\delta)$

denote the equivalence class in $E^*(C_t)$ that contains δ. Define for any $\delta \in E_r^*(C_t)$,

$$fun(\delta) = eq(min(\delta)).$$

We now show that fun is a one-to-one mapping from $E_r^*(C_t)$ to $E^*(C_t)$. In order to do so, we only need to show that, for any $\delta_1, \delta_2 \in E_r^*(C_t)$, if $\delta_1 \neq \delta_2$, then $eq(min(\delta_1)) \neq eq(min(\delta_2))$, i.e. $fun(\delta_1) \neq fun(\delta_2)$. Let $min(\delta_1) =< e_1, e_2 >$, $min(\delta_2) =< d_1, d_2 >$, $e_j = (x_j, y_j)$, $d_j = (u_j, v_j)$, $j = 1, 2$. Assume that $\delta_1 \neq \delta_2$, then, by the definition of the equivalence relation for $E_r(C_t)$, one of the following three cases is true. Case $1: \ x_1 \neq u_1$. This implies by the definition that $eq(min(\delta_1)) \neq eq(min(\delta_2))$. Case $2:$ either $l_{min(\delta_1)} \neq l_{min(\delta_2)}$, or $h_{min(\delta_1)} \neq h_{min(\delta_2)}$. We may assume without loss of generality that $l_{min(\delta_1)}$ is above $l_{min(\delta_2)}$. Thus, the PCE e_1 in the edge-witness $min(\delta_1)$ is between l_{δ_1} and l_{δ_2}, since otherwise either $l_{\delta_1} = l_{\delta_2}$ or l_{δ_1} is below l_{δ_2}, contradicting to the assumption. Hence, there is at least one corner-witness between the x-axis-parallel lines $y = y_1$ and $y = v_1$. This implies that $eq(min(\delta_1)) \neq eq(min(\delta_2))$. Case $3:$ there is an NCE in W_r which is in $[min\{t_0, x_1\}, max\{t_0, x_1\}] \times [min\{y_1, v_1\}, max\{y_1, v_1\}]$. This also implies that there is at least one corner-witness between the x-axis-parallel lines $y = y_1$ and $y = v_1$. So, $eq(min(\delta_1)) \neq eq(min(\delta_2))$. By the above analysis, we know that $\|E_r^*(C_t)\| \leq \|E^*(C_t)\| \leq 4k - 2$, the right inequality holds by Lemma 3.1. Therefore, $\|S_r(C_t)\| = \|R_r(C_t)\| + \|E_r^*(C_t)\| \leq 4k + (4k - 2) = 8k - 2$. □ 1 By Lemma 3.3 and 3.4, the algorithm L-TREE can execute at most 8k-2 stages. At each stage, the learner asks at most one equivalence query, and employs at most two copies of the algorithm BM_n. So, by Proposition 2.1, the learner asks at most $2(\lceil \log n \rceil + 3) + 1$ queries. Thus, the total queries required by the learner is at most $(8k - 2)(2(\lceil \log n \rceil + 3) + 1) = (16k - 4)\lceil \log n \rceil$ $+56k - 14$. Hence, LC-MEMB(L-TREE) $(16k - 4)\lceil \log n \rceil$ $+56k - 13 = O(k \log n)$, and $\|W_r\| \leq (16k - 4)\lceil \log n \rceil$ $+56k - 13$. Note that any two different edge-witnesses share no common points, thus $E_r(C_t) \leq \|W_r\|/2 \leq (8k - 2)\lceil \log n \rceil$ $+28k - 6$. Note also that $\|R_r(C_t)\| \leq 4k$ by Lemma 3.1. So, at stage r, the computation steps of executing Task 1 is $O(\|R_r(C_t)\| + \|E_r(C_t)\|) = O(k \log n)$. In the execution of Task 2, the learner is required to test whether each counterexample in W_r is consistent with the hypothesis H_{r+1}, while H_{r+1} can be represented by $8k - 2$ boundaries, since $\|S_r\| \leq 8k - 2$ by Lemma 3.4. So, executing Task 2 takes $\|W_r\|(8k - 2) = O(k^2 \log n)$ computation steps. In the execution of Task 3, the learner runs at most two copies of the algorithm BM_n, so the time in executing this task is $O(\log n)$. We already know that the algorithm L-TREE runs for at most $8k - 2$ stages. Therefore, the computation steps of L-TREE is bounded by $O(k^3 \log n)$. By the proof of Lemma 3.4, $E_r^*(C_t) \leq 4k - 2$. Since each equivalence class in $E_r^*(C_t)$ witnesses one rectangle in the hypothesis H_r, there are at most $4k - 2$ many different rectangles contained in H_r, it follows that the hypothesis space used by the algorithm L-TREE is U-$(4k - 2)$-BOX_n^2. This completes the proof of Theorem 3.1. □

4 An Algorithm for Learning U-k-BOX_n^2

For any $C_t \in$ U-k-BOX_n^2, we observe that there exist trees $T_1, \ldots, T_l \in TREE_n^k$ with $l \leq k$ such that $C_t = T_1 \cup \ldots \cup T_l$. Examples of this observation are given in figure 4. ¿From this observation, one may divide the task of learning C_t into at most k

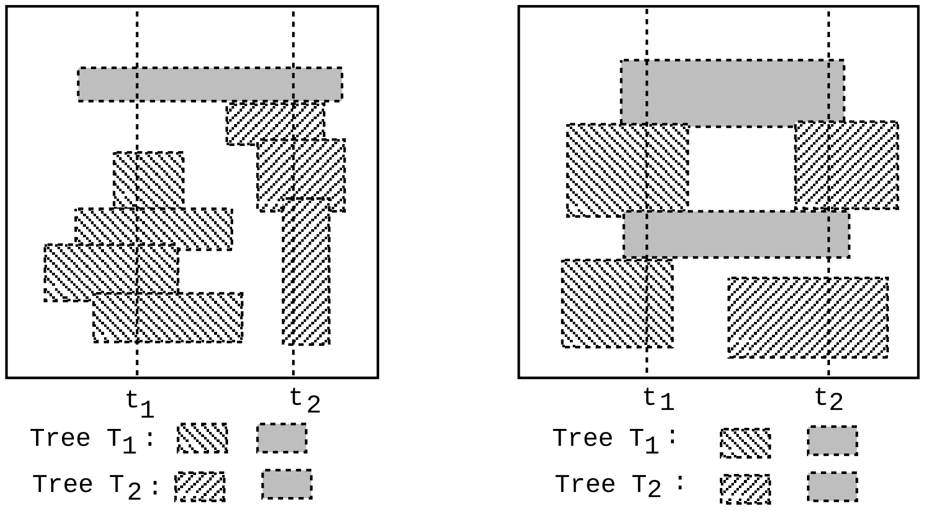


Figure 4: Dividing a Concept into Trees

subtasks, in each subtask, one learns a tree T_j, $j = 1, \ldots, l$, by employing one copy of the algorithm L-TREE. In our algorithm we may actually find C_t as a union of more than k trees. We will prove, however, that never require more than $2k + 1$ trees. However, given a corner-witness (or an edge-witness) for the target concept C_t, one does not know in general to which tree of the target concept the witness belongs. The substantial problem involved in this type of design is how one can efficiently derive a corner-witness (or an edge-witness) for a specific tree T_j from a witness for the target concept C_t. Another difficulty occurs when the learner receives a PCE z from the environment for the target concept C_t. It is trivial that z belongs to some tree T_j of the target concept. However, the learner does not know to which tree T_j z belongs. Thus, whenever the learner receives a PCE, the learner has to decide for which tree of the target concept to run a copy of the algorithm L-TREE.

Theorem 4.1 *There is an algorithm LUKB for learning U-k-BOX_n^2 such that, LC-MEMB(LUKB) = $O(k^3 \log n)$, the hypothesis space used by the algorithm LUKB is U-$(8k^2 - 2)$-BOX_n^2. Moreover, the computation steps of the algorithm LUKB is bounded by $O(k^5 \log n)$.*

Proof. Consider any target concept $C_t \in$ U-k-BOX_n^2. The algorithm LUKB works in stages. At each stage r, the algorithm chooses a set of vertical lines which divide the domain into subdomains formed by two adjacent vertical lines, and the upper boundary and the bottom boundary of the original domain. Whthin each such subdomain, one defines a tree of the target concept. One then runs a copy of the algorithm L-TREE for the tree in each subdomain. If the learner finds that the part of the target concept in some subdomain is not a tree, then the learner chooses a new vertical line dividing the subdomain into two parts, and then defines a new tree in each of the two parts. Let $V_r = \{v_1, \ldots, v_{m_r}\}$ be the set of such vertical lines fixed by the learner within first r stages to divide the original domain into subdomains. Assume that v_{i+1} is on the right side of v_i. V_r will be defined stage by stage during the execution of the algorithm

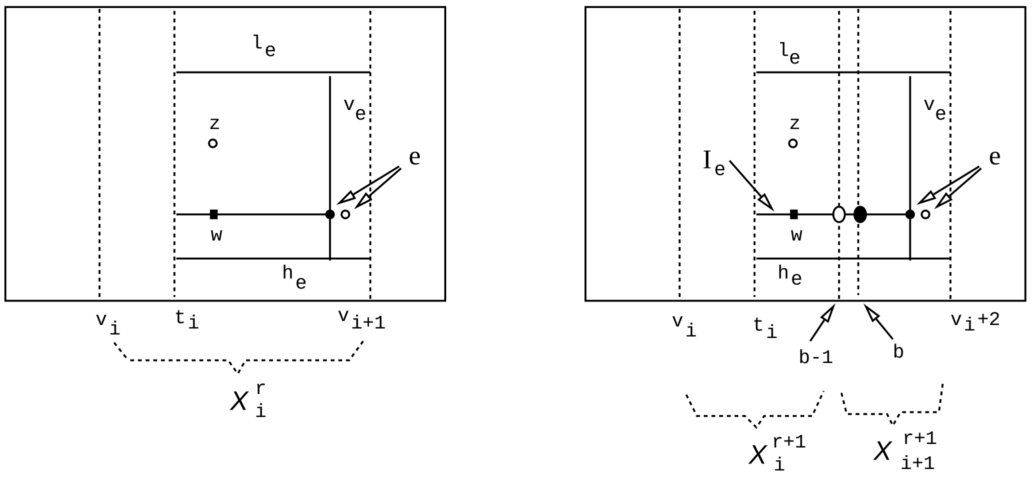
LUKB. Let X_i^r denote the subdomain formed by the vertical lines v_i, v_{i+1}, and the upper boundary and the bottom boundary of the original domain. Let T_i^r be the tree defined in the subdomain X_i^r. As in the construction of the algorithm L-TREE, let R_i^r and E_i^r be the set of all corner-witnesses and the set of all edge-witnesses received by the learner within the first r stages for the tree T_i^r, respectively. For each corner-witness $< z_1, z_2, z_3 >$ in R_i^r, one draws a horizontal line from the trunk of the tree T_i^r through the point z_1 to the boundary of the subdomain X_i^r. Let L_i^r be the set of all those horizontal lines and the additional two lines $y = 0$, i.e. the bottom boundary of the original domain, and $y = n + 1$, i.e. the upper boundary of the original domain. Let W_r be the set of all the counterexamples received by the learner within first r stages.

Initially, the learner proposes a hypothesis $H_0 = \phi$ to the environment to ask an equivalence query. If $C_t = H_0$ then the learner learns it, otherwise the learner receives a PCE $z = (z_1, z_2)$ from the environment. The learner then sets $V_0 = \{x = 0, x = n+1\}$, i.e. V_0 contains exactly the left boundary and the right boundary of the original domain. The learner defines the tree T_1^0 by giving its trunk $x = z_1$. The algorithm LUKB then uses a copy of the algorithm BM_n on the line segment $I_z = [0, n+1] \times [z_2, Z_2]$ to find an edge-witness of $C_t \cap I_z$ on the right side of z. By Proposition 2.1, one finds an edge-witness $e =< (b, y), (b+1, y) >$ for C_t. One sets $E_1^0 = \{e\}$, $R_1^0 = \phi$. Let $L_1^0 = \{y = 0, y = n+1\}$, i.e. L_1^0 contains exactly the bottom boundary and the upper boundary of the original domain. Let W_0 be the set of z and all counterexamples received during the execution of the algorithm BM_n. At any stage $r \geq 1$, the algorithm LUKB executes the following three tasks sequentially.

Task 1*: Constructing the New Hypothesis H_{r+1}.
For each two adjacent vertical lines $v_i, v_{i+1} \in V_r = \{v_1, \ldots, v_{m_r}\}$, one runs a copy of the algorithm L-TREE for the tree T_i^r in the subdomain X_i^r to construct the hypothesis H_i^{r+1}. Let $H_{r+1} = H_1^{r+1} \cup \ldots \cup H_{m_r-1}^{r+1}$ be the next hypothesis for the target concept C_t.

Task 2*: Asking an Equivalence Query.
One checks whether H_{r+1} is consistent with all counterexamples in W_r. If yes then one asks an equivalence query for H_{r+1} to the environment. If $C_t = H_{r+1}$ then one learns C_t, otherwise one receives a counterexample from the environment. One then puts this counterexample into W_r.

Task 3*: Finding New Witnesses.
One finds a counterexample $z = (z_1, z_2) \in W_r$. We may assume $z \in X_i^r$. We only consider that z is on the right side of the trunk of the tree T_i^r, the case that z is on the left side of the trunk of the tree can be coped with in the same way. If z is a PCE, then one runs the algorithm L-TREE for the tree T_i^r in the subdomain X_i^r. When $z = (z_1, z_2)$ is an NCE, one then finds an edge-witness $e =< e_1, e_2 >\in E_i^r$ such that z is in the rectangle formed by the trunk of the tree T_i^r, the vertical line v_e, and the horizontal lines $h_e, l_e \in L_i^r$. Let $e_j = (p_j, q_j), j = 1, 2$. For the definitions of the lines v_e, h_e, l_e, the reader may refer the construction of the algorithm L-TREE. The learner asks a membership query for the point $w = (z_1, q_1)$, which is the projection of the point z on the line segment from the trunk of the tree T_i^r through the edge-witness e to the boundary of the subdomain X_i^r. If the learner receives the answer that w is in the



When z is an NCE, ask whether w is in the target concept. x = t_i is the trunk of the tree in the subdomain X_i^r .

When z is an NCE and w is not in the target concept, then find a new tree with the trunk x=b, and divide the subdomain X_i^r into two parts, X_i^{r+1} and X_{i+1}^{r+1} .

Figure 5: Finding New Witnesses — a New Approach

target concept C_t, then the learner just runs the algorithm L-TREE for the tree T_i^r in the subdomain X_i^r to find new witnesses. If the answer is that w is not in the target concept, then by the definition of a tree, e_1 is not a PCE for the tree T_i^r, so e is not an edge-witness for it. In this case, the learner runs a copy of the algorithm $DBM_{p_1-z_1+1}$ on the line segment $I_e = [z_1, p_1] \times [q_1, q_1]$ to find a boundary $< (b-1, q_1), (b, q_1) >$ for $C_t \cap I_e$. Here, $(b, q_1) \in C_t \cap I_e$, $(b-1, q_1) \notin C_t \cap I_e$. The learner defines a new vertical line $x = b-1$ and lets $V_{r+1} = V_r \cup \{x = b-1\}$. The learner thus divides the subdomain X_i^r into two parts X_i^{r+1} and X_{i+1}^{r+1}. X_i^{r+1} is bounded by the lines v_i and $x = b-1$, X_i^{r+1} is bounded by the lines $x = b-1$ and v_{i+1}. Define a new tree T_i^{r+1} as the old tree T_i^r in X_i^{r+1}. Define another new tree T_{i+1}^{r+1} in X_{i+1}^{r+1} by giving its trunk $x = b$. Let R_{i+1}^{r+1} be the set of all corner-witnesses of the tree T_i^r that are on the right side of the line $x = b-1$ (not on the line). Let E_{i+1}^{r+1} be the set of all edge-witnesses of the tree T_i^r that are on the right side of the line $x = b-1$ (not on the line). Let $R_i^{r+1} = R_i^r - R_{i+1}^{r+1}$, Let $E_i^{r+1} = E_i^r - E_{i+1}^{r+1}$. All the other sets remain the same. The main part of the execution of Task 3* is illustrated in figure 5.

Remark. At any stage r, when one employs a copy of the algorithm L-TREE in the subdomain X_i^r formed by the vertical lines v_i, v_{i+1}, and the upper and bottom boundaries of the original domain, a subtle case which one has to consider is that one needs to extend the subdomain X_i^r in accordance with the construction of the algorithm L-TREE. Assume that v_i and v_{i+1} are the lines $x = a$ and $x = b$, respectively. The extended subdomain is bounded by the vertical lines $x = a-1$, $x = b+1$, and the upper and bottom boundaries of the original domain. We assume in the extended domain that all points outside X_i^r are NCE's for the tree T_i^r defined in the subdomain X_i^r. Let $x = t_i$ be the trunk of the tree T_i^r. Thus, every PCE on the lines $x = a$ and $x = b$ witnesses an edge-witness for the tree T_i^r. Suppose that $z = (z_1, z_2)$ is a PCE on one of those two

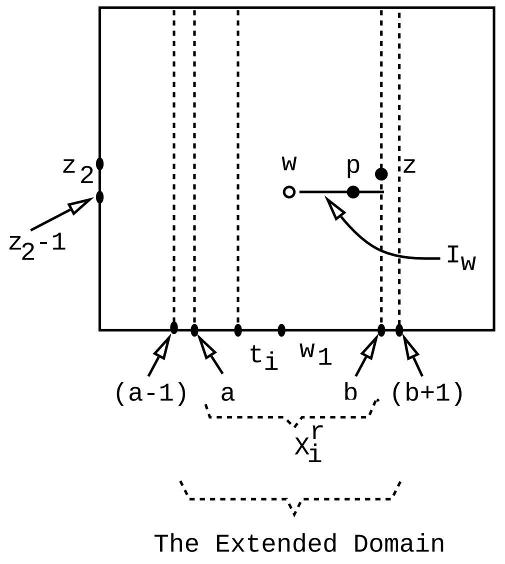


Figure 6: Extending a Subdomain

lines, say, $x = b$. If one finds an NCE $w = (w_1, w_2)$ with $t_i \leq w_1 \leq b$, and $w_2 = z_2 - 1$ or $z_2 + 1$, say, $w_2 = z_2 - 1$, then one considers in the execution of the algorithm for the tree T_i^r that z witnesses a corner-witness by assuming that all points on the line segment $I_w = [w_1 + 1, b] \times [z_2 - 1, z_2 - 1]$ are NCE's for $C_t \cap X_i^r$. If one knows that this assumption is not true by finding a PCE p on the line segment I_w, then according to the construction of Task 3* one will divide the subdomain X_i^r into two parts. Our discussions are illustrated in figure 6.

Lemma 4.1 *For any $r \geq 1$, $\|V_r\| \leq 2k + 2$.*

Proof. Since initially one puts the left boundary $x = 0$ and the right boundary $x = n+1$ of the original domain into V_0, so at any later stage $r \geq 1$, V_r contains those two vertical lines. At any stage $r \geq 1$, the learner may add at most one vertical line into V_r during the execution of Task 3*. This happens in the case that the learner finds an NCE $z = (z_1, z_2) \in W_r \cap X_i^r$ which is not consistent with the hypothesis H_{r+1} and the projection of z on the line segment, which is from the trunk of the tree T_i^r through the edge-witness $e = (e_1, e_2)$ to the boundary of the subdomain X_i^r, is an NCE, where $e_j = (p_j, q_j), j = 1, 2$. We may assume without loss of generality that z is on the right side of the trunk of the tree T_i^r. The learner then employs a copy of the algorithm $DBM_{p_1 - z_1 + 1}$ on the line segment $I_e = [z_1, p_1] \times [q_1, q_1]$ to find a boundary $< (b-1, q_1), (b, q_1) >$ for $C_t \cap I_e$. The learner defines a new vertical line $x = b - 1$ and adds it into V_r. Since $< (b-1, q_1), (b, q_1) >$ (and thus the line $x = b - 1$) witnesses a vertical boundary of some rectangle in the target concept C_t, since also there are at most k different rectangles in C_t, so the learner can add altogether at most $2k$ different vertical lines into V_r. Thus, $\|V_r\| \leq 2k + 2$. □

By Lemma 4.1, there are stages $r_{i_1}, \ldots, r_{i_j}$, $j \leq 2k$, such that at any stage $r_{i_u}, 1 \leq u \leq j$, the learner adds a vertical line into $V_{r_{i_u}}$. Between any two adjacent stages r_{i_u},

$r_{i_{u+1}}$, the learner runs at most $i_{u+1}+1$ copies of the algorithm L-TREE. By Theorem 3.2, the algorithm LUKB makes $O((i_{u+1}+1)k\log n)$ queries, does $O((i_{u+1}+1)k^3\log n)$ computation steps, and issues a hypothesis in U-$((i_{u+1}+1)(4k-2))$-BOX_n^2. Thus, the hypothesis space used by LUKB is U-$((2k+1)(4k-2))$-BOX_n^2 = U-$(8k^2-2)$-BOX_n^2, the total number of queries and the total number of computation steps required by the algorithm LUKB are bounded respectively by

$$\sum_{u=1}^{j} O((i_{u+1}+1)k\log n) = O(k^3\log n),$$

$$\sum_{u=1}^{j} O((i_{u+1}+1)k^3\log n) = O(k^5\log n).$$

□

5 Concluding Remarks

It is well known that learning unions of rectangles is closely related to the problem of learning DNF formulas and the problem of learning unions of intersections of halfspaces. In this paper, we have made a first step towards the problem of learning unions of k rectangles in the discrete plane $\{1,\ldots,n\}^2$ by designing an efficient learning algorithm that uses a slightly larger hypothesis space. The number of equivalence and membership queries required by our algorithm is bounded by $O(k^3\log n)$, while the time complexity of it is bounded by $O(k^5\log n)$. Our result provides a contrast to the known upper bounds for learning k-term DNF formulas with queries, since all those are exponential in k.

It is easy to see that, for any $k = O(n)$, the number of equivalence and membership queries required for learning unions of k rectangles in the discrete plane $\{1,\ldots,n\}^2$ is bounded below by $\Omega(k\log n)$. However, we do not know whether the upper bound $O(k^3\log n)$ of our algorithm is optimal. Another open problem is whether one can design an efficient learning algorithm that also achieves an $O(poly(k)\log n)$ upper bound but uses U-k-BOX_n^2 as the hypothesis space. Finally, since U-k-BOX_n^d is a general case of k-term DNF formulas with at most d variables, it would be very interesting to investigate whether one can design an efficient algorithm for learning U-$O(\log d)$-BOX_n^d by extending Blum and Rudich's technique developed in [BR] for learning $O(\log n)$-term DNF formulas.

Acknowledgment We would like to thank Wolfgang Maass and Ming Li for their discussions. We would also like to thank Avrim Blum for explaining the algorithm developed in [BR] to us, Anselm Blumer for his critiques on the draft of this paper, and Phil Long for his valuable comments on our work.

References

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

[Ab] D. Angluin, "Learning k-term DNF formulas using queries and counterexamples, *Technical Report YaleU/DCS/RR-559*, Yale University department of Computer Science, 1987.

[AP] H. Aizenstein, L. Pitt, "Exact learning of read-twice DNF formulas", *Proc of the 32th Annual Symposium on Foundations of Computer Science*, 1991, pages 170-179.

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

[Bb] E.B. Baum, "Polynomial time algorithms for learning neural nets, *Proc of the 3th Annual Workshop on Computational Learning Theory*, pages 258-272. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1990.

[Ba] E.B. Baum, "On learning a union of halfspaces", *Journal of Complexity*, 6(1990), pages 67-101.

[BR] A. Blum, S. Rudich, "Fast learning of k-term DNF formulas with queries", *Proc of the 24th Annual Symposium on Theory of Computing*, 1992, pages 382-389.

[BS] A. Blum, M. Singh, "Learning functions of k terms", *Proc of the 3th Annual Workshop on Computational Learning Theory*, pages 144-153. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1990.

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

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

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

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

[CMb] Z. Chen, W. Maass, "On-line learning of rectangles and unions of rectangles", to appear in *Machine Learning*.

[CMc] Z. Chen, W. Maass, " A solution of the credit assignment problem in the case of learning rectangles", *Proc of the Second International Workshop on Analogical Inductive and Inference*, pages 26-34, 1992.

[H] T. Hancock, "Learning 2μ DNF formulas and $k\mu$ decision trees", *Proc of the 4th Annual Workshop on Computational Learning Theory*, pages 199-209. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1991.

[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. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1991.

[MTa] W. Maass, G. Turán, “On the complexity of learning from counterexamples”, *Proc of the 30th Annual Symposium on Foundations of Computer Science*, 1988, pages 262-267.

[MTb] 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*; to appear in *Machine Learning*.

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

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