A Faster Algorithm for Learning Decision Trees With Module Nodes

Zhixiang Chen **Xiannong Meng**
Department of Computer Science, University of Texas – Pan American
1201 West University Drive, Edinburg, TX 78539-2999, USA.
Email: {chen, meng}@cs.panam.edu, Phone:(956)381-2320, Fax:(956)384-5099

Abstract

One of the challenging problems regarding learning decision trees is whether decision trees over the domain Z_N^n, in which each internal node is labeled by a module function $ax_i(mod\ N)$ for some $i \in \{1,\ldots,n\}$, are efficiently learnable with equivalence and membership queries [5]. Given any decision tree T, let s be the number of its leaf nodes. Let $N = p_1^{t_1} \cdots p_r^{t_r}$ be its prime decomposition, and let $\gamma(N) = (t_1 + 1) \cdots (t_r + 1)$. We show that when all the module functions in T have the same coefficient, then there is a polynomial time algorithm for learning T using at most $s(s+1)\gamma(N)$ equivalence queries and at most $s^2 n\gamma(N)$ membership queries. Our algorithm is substantially more efficient than the algorithm designed in [7] for learning such decision trees. When N is a prime, our algorithm implies the best-known algorithm in [4] for learning decision trees over a finite alphabet.

1 Introduction

There are a large body of empirical research on the induction of decision trees (see, for example, [15, 14]). A decision tree is a representation of functions as a rooted oriented tree. The internal nodes of a tree are labeled with simple tests that can be performed on the input of the function, and leaves of a tree are labeled with the possible values of the function. For each internal node there is an outgoing edge with respect to each possible outcome of the test. To evaluate the decision tree on an input instance one starts at the root and follows the path determined by taking the edge respect to the outcome of the test on the current node. The label of the leaf one finally reaches is the value of the function.

A number of systems such as Quinlan's ID3 [15] perform induction by recursively growing a decision tree to fit a sample of data from a unknown target function. However these are heuristic algorithms that may produce a very large decision tree to fit data that can in fact be explained with a small (in terms of the nodes) decision tree hypothesis. It is a long standing open question whether there exists an efficient algorithm that provably learns decision trees in the formal sense of Valiant's PAC model [16]. There have been considerable efforts on finding a solutions to the problem and many interesting results have been obtained.

In the basic PAC model, the most general result for learning decision trees is by Ehrenfeucht and Haussler [8], who gave an algorithm to learn boolean decision trees whose running time is linear in n^r. Here r is the rank of the tree, which is a parameter of its topology that measures the tree's bushiness. Algorithms that provably learn read-once decision trees (in which each variable occurs in at most one node) and read-k decision trees (in which each variable occurs in at most k nodes) on the uniform distribution were obtained in [1]. Other

provably correct decision tree learning algorithms are known in the more powerful membership query learning model, where the learner may ask an oracle to supply the correct classification for arbitrary examples during the learning process [9, 10, 11].

In the on-line learning model with equivalence and membership queries, the well-known result has been obtained by Bshouty [4]. He has shown that boolean decision trees and their extensions over any finite alphabet is polynomial time learnable with equivalence and membership queries. In [6], Bshouty's results are extended to boolean decision trees with certain types of functions as labels for leaves nodes.

However, it remains open whether one can learn a decision tree over the domain Z_N^n when the internal nodes of the tree are labeled with simple functions such as *less-than* functions $<$ (see [4]) or "module functions" $ax(mod\ N)$ (see [5]). The monotone theory developed in [4], though very powerful, could not be applied to those types of decision trees, because the number of monotone "minterms" for those trees is in general exponential in its size.

[7] studied the problem of learning decision trees in which each internal node is labeled by a module function $ax_i(mod\ N)$ for some $i \in \{1, \ldots, n\}$, over the domain Z_N^n with equivalence and membership queries [5]. Given any decision tree T, let s be the number of its leaves. It was proved [7] that when all the module functions in T have the same coefficient, then there is a polynomial time algorithm for learning T using at most $s(s+1)N$ equivalence queries and at most $s^n nN$ membership queries.

Given any decision tree T, let s be the number of its leaf nodes. Let $N = p_1^{t_1} \cdots p_r^{t_r}$ be its prime decomposition, and let $\gamma(N) = (t_1 + 1) \cdots (t_r + 1)$. In this paper we show that when all the module functions in T have the same coefficient, then there is a polynomial time algorithm for learning T using at most $s(s+1)\gamma(N)$ equivalence queries and at most $s^2 n\gamma(N)$ membership queries. Our algorithm is substantially more efficient than the algorithm designed in [7]. When N is a prime, our algorithm implies the best-known algorithm in [4] for learning decision trees over a finite alphabet.

This paper is organized as follows. In section 2, we introduce the learning model. In section 3, we define decision trees with module nodes over the domain Z_N^n. In section 4, we discuss the module DNF and CNF as well as their representations of a decision tree. In section 5, we study properties of general module DNF and CNF with the Hamming partial order over Z_N. In section 6, we design introduce the $CDNF$-algorithm for learning a module DNF via its module CNF representation obtained in [7] and its application on learning decision trees with module nodes. In section 7, we design a new algorithm to learn decision trees with module nodes which is substantially efficient than the one obtained in [7]. We point out some future research direction in section 8.

2 The Learning Model

In 1984, Valiant [16] proposed the pac-learning model and thus founded the modern computational learning theory. Later, Angluin [2] proposed the on-line learning with queries and then initiated the study of exact learning. According to Angluin [2] and Blum [3], on-line learning with equivalence queries (and with membership queries) implies pac-learning (with membership queries), and there are cases such that on-line learning with equivalence queries is strictly harder than pac-learning.

In this paper, we will focus ourselves on the on-line learning with equivalence and membership queries. For any integer $N \geq 2$, let $Z_N = \{0, \ldots, N-1\}$. The goal of a learning algorithm (or learner) for a class $\mathbf{C}$ of boolean-valued functions over the domain Z_N^n is to learn any unknown target function $f \in \mathbf{C}$ that has been fixed by a teacher. In order to obtain information about f, the learner can ask equivalence queries by proposing hypotheses h from a fixed hypothesis space $\mathbf{H}$ of functions over Z_N^n with $\mathbf{C} \subseteq \mathbf{H}$ to an equivalence oracle $EQ()$. If $h = f$, then $EQ(h) =$ "*yes*", so the learner succeeds. If $h \neq f$, then $EQ(h) = x$ for some $x \in X^n$, called a counterexample, such that $h(x) \neq f(x)$. x is called a positive counterexample if $f(x) = 1$ and a neg-

ative counterexample otherwise. The learner can also ask membership queries by presenting examples in the domain to a membership oracle $MQ()$. For any example x, $MQ(x) =$ "*yes*" if $f(x) = 1$, otherwise $MQ(x) =$ "*no*". Each new hypothesis issued by the leaner may depend on the earlier hypotheses and the examples observed so far. A learning algorithm exactly learns $\mathbf{C}$, if for any target function $f \in \mathbf{C}$, it can find a $h \in \mathbf{H}$ that is logically equivalent to f. We say that a class $\mathbf{C}$ is polynomial time learnable if there is a learning algorithm that exactly learns any target function in $\mathbf{C}$ and runs in time polynomially in the size of the domain, the size of the target function, and the size of the largest example observed during its learning process.

3 Decision Trees With Module Nodes

Given an integer $N \geq 2$, we consider decision trees with module nodes over the domain Z_N^n. Such a decision tree is a rooted N-branch tree, and each of its internal nodes is labeled by a module function $ax_i(mod\ N)$ with some $i \in \{1, \ldots, n\}$. Each internal node has N edges labeled respectively by $0, 1, \ldots, N-1$. Each of the leaf nodes is labeled by either 0 or 1. A decision T represents a boolean-valued function $f_T : Z_N^n \to \{0, 1\}$. The function f_T is evaluated recursively by starting at the root and evaluating the subtree which is connected to the root by an edge labeled by the outcome of the module function which labels the root node. When a leaf node is reached, the leaf's label is the output of the function. The size of T is defined as the number of its leaf nodes. In this paper, we consider only decision trees $T(a)$, where all the module functions labeling the internal nodes have the same coefficient $a \in Z_N$. A tree $T(2)$ with a modulus 4 is illustrated in figure 1.

4 The Module DNF and CNF of a Decision Tree

Given a decision tree $T(a)$ with module nodes, let $l_1, \ldots, l_s$ be its leaf nodes labeled by 1. Walk from the root to each l_i and let $ax_{i_1}(mod\ N), b_1, \ldots, ax_{i_s}(mod\ N), b_s$ be the module functions and edge labels encountered in

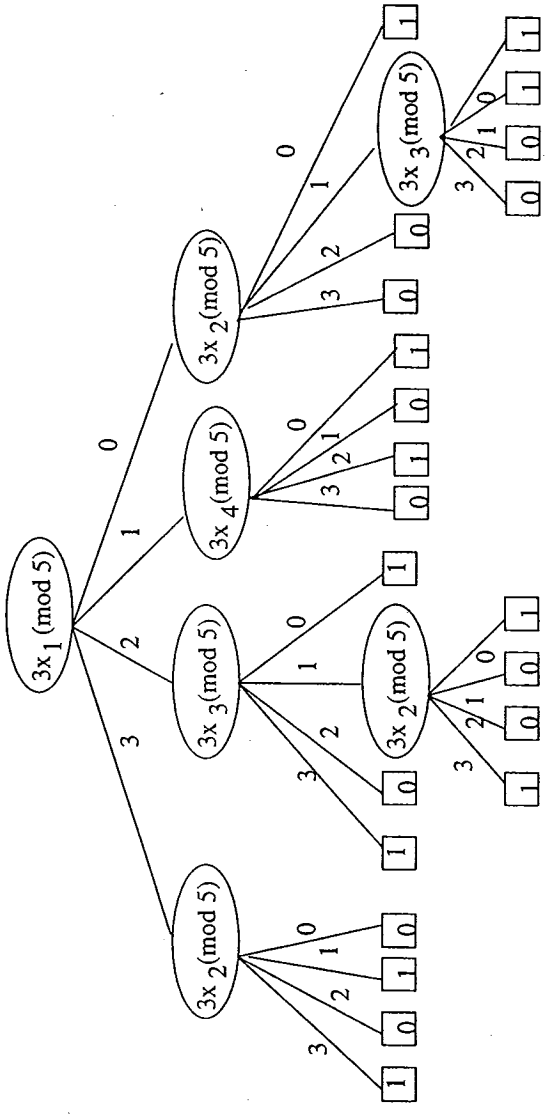

Figure 1: A decision tree $T(3)$ with a modulus 5.

this unique path to l_i. The leaf node l_i will represent the term

$$T_i = ax_{i_1} \equiv b_1 (mod\ N) \bigwedge \cdots \bigwedge ax_{i_s} \equiv b_s (mod\ N).$$

The module DNF of $T(a)$ is thus defined as

$$f^{DNF}_{T(a)} = T_1 \bigvee \cdots \bigvee T_s.$$

Similarly, let $g_1, \ldots, g_t$ be its leaf nodes labeled by 0. Walk from the root to each g_i and let $ax_{i_1}(mod\ N), b_1, \ldots, ax_{i_t}(mod\ N), b_t$ be the module functions and edge labels encountered in this unique path to g_i. The leaf g_i will represent the clause

$$C_i = ax_{i_1} \not\equiv b_1 (mod\ N) \bigvee \cdots \bigvee ax_{i_s} \not\equiv b_s (mod\ N).$$

The module CNF of $T(a)$ is thus defined as

$$f^{CNF}_{T(a)} = C_1 \bigwedge \cdots \bigwedge T_s.$$

From the above definitions we have the following two easy facts:

Fact 1. *For any example $x \in Z_N^n$, $T(a)(x) = 1$ if and only if $f^{DNF}_{T(a)}(x) = 1$.*

Fact 2. *For any example $x \in Z_N^n$, $T(a)(x) = 0$ if and only if $f^{CNF}_{T(a)}(x) = 0$.*

By Fact 1, one may think that one can learn a decision tree $T(a)$ via learning its module DNF $f^{DNF}_{T(a)}$. However, as noted in [5], the latter problem is even harder than learning boolean functions. Algorithms for learning subclasses of disjunctions of conjunctions of general counting functions have been obtained in [5].

By Fact 2, one may also think that one can learn a decision tree $T(a)$ via learning its module CNF $f^{CNF}_{T(a)}$ using Bshouty's monotone technique [4]. A very tempting approach is to apply his technique along with the Hamming partial order over Z_N to learn $f^{CNF}_{T(a)}$. Unfortunately, this process will fail because of one substantial difficulty: The number of monotone minterms for $f_{T(a)}$ is in general exponential. In this paper, we will develop a new method to overcome the above difficulty and thus we are able to use the monotone technique to design an efficient algorithm to learn a decision tree $T(a)$.

5 Hamming Partial Order, Module DNF and CNF

We first introduce the Hamming partial order [4]. The Hamming partial order $(Z_N, \leq)$ is defined as follow: For any $x, y \in Z_N$ we have

$$x \leq y \Longleftrightarrow x = y \bigvee x = 0.$$

Using the Hamming partial order we have: For any $x, y \in Z_N^n$,

$$x \leq y \Longleftrightarrow (\forall i)(x[i] = y[i] \bigvee x[i] = 0),$$

where $x[i]$ and $y[i]$ denote respectively their i-th components.

Given any $a \in Z_N$, let us consider the general forms of module DNF's and CNF's for decision trees. We say that

$$F(a) = T_1 \bigvee \cdots \bigvee T_s$$

is a module DNF, where for any $i \in \{1, \ldots, s\}$,

$$T_i = ax_{i_1} \equiv b_1 (mod\ N) \bigwedge \cdots \bigwedge ax_{i_{l_i}} \equiv b_{l_i} (mod\ N),$$

and we also say that T_i is a module term. We say that

$$C(a) = C_1 \bigwedge \cdots \bigwedge C_t,$$

is a module CNF, where for any $j \in \{1, \ldots, t\}$,

$$C_i = ax_{j_1} \not\equiv e_1 (mod\ N) \bigvee \cdots \bigvee ax_{j_{g_i}} \not\equiv e_{g_i} (mod\ N),$$

and we also say that C_j is a module clause.

For a module term T_i, for any $x \in Z_N^n$, we say that x is a minterm for T_i if

$$T_i(x) = 1 \bigwedge (\forall y \leq x)(T_i(y) = 1 \Longrightarrow x = y).$$

For a module DNF $F(a)$, for any $x \in Z_N^n$, we say that x is a minterm for $F(a)$ if x is a minterm for a term T_i of $F(a)$.

We say that a module CNF $C(a)$ is a representation of a module DNF $F(a)$ if

$$(\forall x)(C(a)(x) = 1 \Longleftrightarrow F(a)(x) = 1).$$

Given a set B of examples, we say that B is a monotone basis for $F(a)$ if for any module clause C_i in the representation $C(a)$ of $F(a)$, there is an example $x \in B$ such that $C_i(x) = 0$.

When the monotone base $B = \{u_1, \ldots, u_t\}$ for a module DNF $F(a)$ is known, one can use Bshouty's monotone technique to learn $F(a)$. The idea is as follows: We start from an empty hypothesis and use an equivalence query to get a positive counterexample for $F(a)$. For each positive counterexample v, for each $u_i \in B$, we use membership queries to get a minterm for the module DNF $F(a)(x+u_i)$. We make a new hypothesis by taking the conjunction of all H_i which is the disjunction of all minterms we have found for $F(a)(x+u_i)$. However, this process will fail because in general the number of minterms for $F(a)(x+u_i)$ is exponential.

We now work on a new way to overcome the above difficulty. For a module term

$$T = ax_{i_1} \equiv b_1 (mod\ N) \bigwedge \cdots \bigwedge ax_{i_l} \equiv b_l (mod\ N) \bigwedge ax_{j_1} \equiv 0 (mod\ N) \bigwedge \cdots \bigwedge ax_{j_f} \equiv 0 (mod\ N), \quad (1)$$

where $b_1, \ldots, b_l$ are not 0, we say that

$$\mu(T) = ax_{i_1} \equiv b_1 (mod\ N) \bigwedge \cdots \bigwedge ax_{i_l} \equiv b_l (mod\ N)$$

is the module minterm for T.

Although there may be in general exponentially many minterms for a module term T, yet its module minterm is unique. Moreover, the following lemma tells us that we can find the module minterm using one minterm.

Lemma 5.1. *Suppose that v is a minterm for a module term T. Let*

$$\mu = \bigwedge_{\forall i, v[i] \neq 0} (ax \equiv av[i] (mod\ N).$$

Then μ is the module minterm for T.

Proof. Suppose that T is a term as given in (1). Since v is a minterm for T, we have that for any $r \notin \{i_1, \ldots, i_l\}$, $v[r] = 0$, and for any $r \in \{i_1, \ldots, i_l\}$, $v[r] \neq 0$ and $av_r \equiv b_r (mod\ N)$. Hence, $\mu = \mu(T)$. □

Given any example $\alpha \in Z_N^n$, and any module term T. Consider the shift $T(x+\alpha)$ of T with α:

$$T(x+\alpha) = a(x_{i_1} + \alpha[i_1]) \equiv b_1 (mod\ N) \bigwedge \cdots \bigwedge a(x_{i_l} + \alpha[i_l]) \equiv b_l (mod\ N). \quad (2)$$

Define

$$\mu(T)_\alpha = ax_{i_{j_1}} \equiv b_{j_1} (mod\ N) \bigwedge \cdots \bigwedge ax_{i_{j_l}} \equiv b_{j_l} (mod\ N),$$

where for any $r \in \{j_1, \ldots, j_l\}$, $(b_r - a\alpha[r]) \not\equiv 0 (mod\ N)$, and for any $r \notin \{j_1, \ldots, j_l\}$, $(b_r - a\alpha[r]) \equiv 0 (mod\ N)$. We say that $\mu(T)_\alpha$ is the module α-minterm for T.

Lemma 5.2. *Given any module term T and and any $\alpha \in Z_N^n$, if v is a minterm for a module term $T(x+\alpha)$. Then*

$$\mu_\alpha = \bigwedge_{\forall i, v[i] - \alpha[i] \neq 0} ax_i \equiv av[i] (mod\ N)$$

is the module α-minterm for T.

Proof. Suppose that $T(x+\alpha)$ is a term as given in (2). Note that $a_r(x[r]) \equiv b_r (mod\ N)$ is equivalent to $a_r((x[r] - \alpha[r]) + \alpha[r]) \equiv b_r (mod\ N)$, which is equivalent to $a_r(x[r] - \alpha[r]) \equiv (b_r - \alpha[r]) (mod\ N)$. Since v is a minterm for $T(x+\alpha)$, we have that for any $r \notin \{j_1, \ldots, j_l\}$, $v[r] - \alpha[r] = 0$, and for any $r \in \{i_1, \ldots, i_l\}$, $v[r] - \alpha[r] \neq 0$ and $av_r \equiv b_r (mod\ N)$. Hence, $\mu_\alpha = \mu(T)_\alpha$. □

6 Learning a Module DNF via Its Module CNF Representation

Given any module DNF $F(a)$ and its module CNF representation $C(a)$, using the monotone technique in [4] the following algorithm was designed to learn $F(a)$ in polynomial time in the number of variables, the size of F, and the size of $C(a)$. When the learner receives a positive counterexample, it finds a *module minterm* for $F(a)$. this is in contrast with finding a minterm in the monotone technique in [4].

The $CDNF$ Algorithm for $F(a)$

(1)$t \leftarrow 0$.

(2)$EQ(1) \rightarrow v$. If the answer is "YES" then stop.

(3)$t \leftarrow t+1, H_t \leftarrow 0, S_t \leftarrow \phi, u_t \leftarrow v$.

(4)$EQ(\bigwedge_{i=1}^{t} H_i) \rightarrow v$. If the answer is "YES" then stop.

(5)$I \leftarrow \{i | H_i(v) = 0\}$.

(6)If I is empty then Goto (3).

(7)For each $i \in I$ do

(7.1) $v_i \leftarrow v$;
(7.2) Walk from v_i toward u_i while keeping $F(a)(v_i) = 1$
(7.3) $S_i \leftarrow S_i \cup \{\mu(v_i)_{u_i}\}$.

(8)$H_i = M_{u_i} S_i$ for $i = 1, \ldots, t$.
(9) Goto (4).

The following lemma was proved in [7].

Lemma 6.1. [7]. *Suppose that the coefficient a is known to the learner. Then any module DNF $F(a)$ can be learned using at most mnt membership queries and at most $(m+1)t$ equivalence queries, where m is the number of module terms in F and t is the number of module clauses in module CND representation $C(a)$ of $F(a)$.*

With the help of the above lemmas, the following main result in [7] was established.

Theorem 6.2. [7] *Any decision tree $T(a)$ with module nodes over the domain Z_N^n can be learned using at most s^2nN membership queries and at most $s(s+1)N$ equivalence queries, where s is the size of $T(a)$.*

7 A More Efficient Algorithm for Learning Decision Trees $T(a)$

Given any module decision $T(a)$, as we discussed in section 4, $T(a)$ is equivalent to its module DNF represent ion $f^{DNF}_{T(a)}$ and to its module CNF representation $f^{CNF}_{T(a)}$. Note that $f^{CNF}_{T(a)}$ is also a module CNF representation of $f^{DNF}_{T(a)}$. In order to learn $T(a)$, we only need to learn $f^{DNF}_{T(a)}$. Thus, one way to learn a decision tree $T(a)$ is as follows: When a is known, then run the $CDNF$-algorithm to learn $f^{DNF}_{T(a)}$ directly. When it is not known, then for each $a \in Z_N$, run a copy of the $CDNF$-algorithm to learn $T(a)$. This is what has been done in [7].

We now consider how to design a more efficient algorithm to learn a decision tree $T(a)$ when the coefficient a is unknown.

We first give some easy facts about the structures of submodules of Z_N.

Lemma 7.1. *For any submodule S of Z_N, there is an element $d \in Z_N$ such that $S = Z_N d$, and $d|N$ if $d \neq 0$.*

Proof. If $S = \{0\}$, then $S = Z_N 0$. Now assume that $S \neq \{0\}$. Fix $d_1 \in S$ such that $d_1 \neq 0$. If $S = Z_N d_1$ then we are done. Otherwise, fix $t_1 \in S - Z_N d_1$. Set $d_2 = gcd(d_1, t_1, N)$. Then, $Z_N d_1 + Z_N t_1 = Z_N d_2$. $t_1 \notin Z_N d_1$ implies that $d_2 \notin Z_N d_1$. If $S = Z_N d_2$, then we are done, otherwise fix $t_2 \in S - Z_N d_2$. Let $d_3 = gcd(d_2, t_2, N)$. Then, $Z_N d_2 + Z_N t_2 = Z_N d_3$, and $d_3 \notin Z_N d_2$. Repeat the above procedure. Because Z_N contains N elements, the above procedure must terminate at an element d_m with $m \leq N-1$. Hence, $S = Z_N d_m$. □

Lemma 7.2. *Given any two fixed value $a, b \in Z_N$, consider the module equation*

$$ax \equiv b (mod\ N).$$

There is a constant d such that for any particular x_0 satisfying the above equation, the set of all its solutions is

$$Z_N d + x_0 = \{y + x_0 | y \in Z_N\}.$$

Moreover, $d|N$ if $d \neq 0$.

Proof. Since the set of all solutions to the equation $ax \equiv 0 (mod\ N)$ is a submodule of Z_N, by Lemma 7.1, there is a constant d such that the set of all solution to $ax \equiv 0 (mod\ N)$ is $Z_N d$, and $d|N$ if $d \neq 0$. Given a solution x_0 to the equation $ax \equiv b (mod\ N)$, for any solution x' to the equation, we have

$$a(x' - x_0) \equiv ax' - ax_0 \equiv b - b \equiv 0 (mod\ N).$$

This implies that $x' - x_0$ is a solution to $ax \equiv 0 (mod\ N)$, i.e., $x' \in Z_N d + x_0$. On the other hand, let $x' \in Z_N d + x_0$. That is, there is $y \in Z_N d$ such that $x' = y + x_0$. We have

$$ax' \equiv a(y + x_0) \equiv ay + ax_0 \equiv 0 + b \equiv b (mod\ N).$$

This means that x' is a solution to $ax \equiv b (mod\ N)$. Hence, $Z_N d + x_0$ is the set of all solutions to the equation $ax \equiv b (mod\ N)$ □

Let $N = p_1^{t_1} \cdots p_r^{t_r}$ be the prime decomposition of N, where $t_i \geq 1$ and p_i is a prime for $i = 1, \ldots, r$. Define

$$\gamma(N) = (t_1 + 1) \cdots (t_r + 1).$$

Lemma 7.3. *There are at most $\gamma(N)$ many submodules of Z_N.*

Proof. By Lemma 7.1, each submodule of Z_N is $Z_N d$ for some $d \in Z_N$ such that $d|N$ if $d \neq 0$. There are $\gamma(N) - 1$ different such $d \neq 0$ and $Z_N 0 = \{0\}$ is also a submodule. Hence, Z_N has at most $\gamma(N)$ many submodules. □

Now, we are ready to prove our main result.

Theorem 7.4. *There is a algorithm to learn any decision $T(a)$ with module nodes over the domain Z_N^n using at most $s(s+1)\gamma(N)$ equivalence queries and at most $s^2 n\gamma(N)$ membership queries.*

Proof. In the $CDNF$-algorithm the learner needs to find a module minterm when it receives a positive counterexample. A module minterm is a conjunction of equations $ax_{i_j} \equiv b_{i_j} (mod\ N)$. When a solution x_0 to $ax_{i_j} \equiv b_{i_j} (mod\ N)$ is known, then by Lemma 7.2, the set of all its solutions is $Z_N d + x_0$ for some unknown d. Because all module functions in the decision tree $T(a)$ have the same coefficient a, the d is the same for all the module functions. Hence, we can learn $T(a)$ by running a copy the $CDNF$-algorithms for each of all such possible d's, i.e., all possible submodules of Z_N. According to Lemma 7.3, there are at most $\gamma(N)$ submodules. Thus, our theorem follows from Lemma 6.1. □

Remark 7.5. *Because $\gamma(N)$ is substantially smaller than N, our algorithm for learning a decision tree $T(a)$ with module nodes over the domain Z_N^n is much more efficient than the one obtained in [7]. In particular, when $N = p^t$ for a prime p, $\gamma(N) \leq \frac{\log N}{\log p}$. To learn a decision tree $T(a)$, our algorithm uses at most $s(s+1)\frac{\log N}{\log p}$ equivalence queries and at most $s^2 n \frac{\log N}{\log p}$ membership queries, while the algorithm in [7] uses at most $s(s+1)N$ equivalence queries and at most $s^2 nN$ membership queries.*

Remark 7.6. *When N is a prime, any equation $ax \equiv b (mod\ N)$ is equivalent to $x = a^{-1}b$ if $a \neq 0$. If $a = 0$, then $ax \equiv b (mod\ N)$ have no solutions if $b \neq 0$, otherwise its solution set is the entire Z_N. From the above analysis, when N is a prime, decision trees $T(a)$ with module nodes are the same as the decision trees over the alphabet Z_N such that every internal node of such a tree is labeled by a variable and each internal node has N outgoing edges labeled respectively by $0, \ldots, N-1$. Thus, Theorem 7.4 implies the learning algorithm obtained in [4] for learning decision trees over the alphabet Z_N.*

8 Conclusion Remarks

In this paper, we give a substantially efficient algorithms for learning any decision tree with module nodes over the domain Z_N^n, when the module coefficients at all nodes are the same. Our algorithm improves the algorithm obtained in [7]. When N is a prime, our algorithm implies the well-known algorithm for learning decision trees over any alphabet in [4].

Although the theoretical analyses of the decision tree learning algorithms obtained here and in [4, 7] are very complicated, yet the practical implementations should be, as we can see from the algorithm designs, very simple. Moreover, the complexity of those algorithms are very low. As a future research direction, we plan to apply those algorithms on solving certain practical problems.

References

[1] W. Aiello and M. Mihail, "Learning the fourier spectrum of probabilistic lists and trees", *Proc. of the ACM-SIAM Symposium on Discrete Algorithms*, 1991.

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

[3] A. Blum, "Separating PAC and mistake-bounded learning models over the boolean domain", *Proc of the 31th Annual ACM Symposium on Foundations of Computer Science*, pages 211-218, 1990.

[4] N. Bshouty, "Exact Learning via the monotone theory", *Proc of the 34th Annual ACM Symposium on Foundations of Computer Science*, pages 302-311, 1993.

[5] N. Bshouty, Z. Chen, S. Decatur, S. Homer, "On the learnability of Z_N-DNF formulas", *Proc of the 8th Annual ACM Conference on Computational Learning Theory*, pages 198-205, 1995.

[6] N. Bshouty, C. Tamon, D. Wilson, "On learning decision trees with large output domains", *Proc of the 8th Annual ACM Conference on Computational Learning Theory*, pages 190-197, 1995.

[7] Z. Chen, R. Fox, and X. Meng, "On learning decision trees with module nodes", submitted for publication.

[8] A. Ehrenfeucht and D. Haussler, "Learning decision trees from random examples", *Information and Computation*, 3, pages 231-246, 1989.

[9] T. Hancock, "Learning 2μ DNF formulas and $k\mu$ decision trees", *Proc of the 4th Annual ACM Conference on Computational Learning Theory*, pages 199-209, 1991

[10] T. Hancock, "Learning $k\mu$ decision trees on the uniform distribution", *Proc of the 6rd Annual ACM Conference on Computational Learning Theory*, pages 352-360, 1993

[11] T. Hancock and Y. Mansour, "Learning monotone $k\mu$ DNF formulas on product distribution", *Proc of the 4th Annual ACM Conference on Computational Learning Theory*, pages 179-183, 1991

[12] N. Jacobson, Basic Algebra I, W. H. Freeman and Company, New York, 1985.

[13] E. Kushilevitz and Y. Mansour, "Learning decision trees using the fourier spectrum", *Proc of the 23th Annual ACM Symposium on Theory of Computing*, pages 455-464, 1991.

[14] J. Mingers, "An empirical comparison of selection measures for decision-trees induction", *Machine Learning*, 3, pages 319-342, 1989.

[15] J. Quinlan, "Induction of decision trees", *Machine Learning*, 1, pages 81-106, 1986.

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