On Learning Branching Programs and Small Depth Circuits

Francesco Bergadano
Universitá di Torino

Nader H. Bshouty
University of Calgary

Christino Tamon*
Clarkson University

Stefano Varricchio
Universitá di L'Aquila

Abstract

This paper studies the learnability of branching programs and small depth circuits with modular and threshold gates in both the exact and PAC learning models with and without membership queries. Some of the results extend earlier works in [GG95, ERR95, BTW95]. The main results are as follows. For branching programs we show the following.

1. Any monotone width two branching program (defined by Borodin *et al.* [BDFP86]) is PAC learnable with membership queries under the uniform distribution. This extends Jackson's result [J94] for learning DNF formulae.
2. Any width two branching program with a bounded number of sinks is exactly learnable using equivalence queries only. This extends the result of PAC learning width two branching programs with two sinks [BTW95]. This cannot be extended to an unbounded number of sinks unless k-term DNF is learnable from equivalence queries, for non-constant k.
3. Any width three and width four even permutation branching program (defined by Barrington [B89]) are exactly learnable with equivalence and membership queries. These results cannot be extended to width five unless NC^1 is learnable with membership queries [B89, AK95].
4. Any mod_p of polynomially many Obdds (ordered binary decision diagrams) and any constant Boolean combination of mod_p of Obdds are exactly learnable using equivalence and membership queries, assuming that p is a fixed prime. This extends a result of Gavaldà and Guijarro [GG95].

For small depth circuits with modular and threshold gates we prove the following.

1. Any depth two circuit with a mod_p gate at the top, for a fixed prime p, and arbitrary modular gates at the bottom level is exactly learnable using equivalence and membership queries.
2. Any depth two circuit with a mod_p gates at the top, for a fixed prime p, and arbitrary threshold gates at the bottom level is exactly learnable using equivalence and membership queries. We note that by a result of Krause and Pudlàk [KP94] learning a threshold of modular gates will imply the learnability of DNF formulae.
3. Any $f(g_1, g_2, \ldots, g_k)$, where k is constant, f is any Boolean function and $g_1, g_2, \ldots, g_k$ are one of the above two classes, is exactly learnable using equivalence and membership queries.

Keywords: Branching programs. Small depth modular and threshold circuits. Computational learning. Ordered binary decision diagrams.

*Work done while the author was a student at the Dept. Computer Science, University of Calgary.

1 Introduction

Branching program is a well-studied computational model in complexity theory. The interest was in proving space and time-space tradeoff lower bounds in a non-uniform model of computation. One of the earliest famous conjectures is that majority cannot be computed by a bounded width branching program of polynomial size [BDFP86]. Barrington [B89] disproved this with the surprising result that the computational power of width five permutation branching programs or S_5-PBPs is equivalent to NC^1. Recently, under the name of *binary decision diagrams*, branching programs have found numerous applications in computer-aided circuit design and verification (see [GM93] and the references therein).

The problem of learning branching programs has been studied in earlier works [RW93, ERR95, GG95, BTW95]. We will review these results in the following and outline the contributions of this paper.

The learnability of bounded width branching programs was studied initially by Ergün, Ravi Kumar, and Rubinfeld [ERR95]. In that paper they proved that restricted width two read-once branching programs with two sinks is PAC learnable under any distribution. They also showed that learning width three branching programs is as hard as learning DNF formulae. In [BTW95] strict width two branching programs $\mathcal{SW}_2$ (as defined by Borodin *et al.* [BDFP86]) is shown to be *properly* PAC learnable under any distribution. This is an improvement over [ERR95] since the latter is a more general class. It was also observed that learning monotone width two branching programs (as defined by Borodin *et al.* [BDFP86]) is as hard as learning DNF formulae.

In this paper we improve the above results in two ways. First we show that any width two branching programs with a bounded number of sinks is exactly learnable using equivalence queries only. This improves upon the result of [BTW95] showing the learnability of width two branching programs with two sinks. Second, we show that any monotone width two branching program is PAC learnable with membership queries under the uniform distribution. This extends Jackson's known result [J94] on learning DNF formulae since DNF formulae form a proper subclass of monotone width two branching programs.

For branching programs with width more than two, Barrington [B89] proved that width three permutation branching programs are equivalent to depth two circuits with a mod_3 gate at the top and parity gates at the bottom level. There is also a characterization of width four even permutation branching programs as depth three circuits with $\wedge$, mod_2, and mod_3 gates. We exploit these alternative characterizations to obtain exact learning algorithm using equivalence and membership queries for these two classes of permutation branching programs. On the other hand, by the work of Angluin and Kharitonov [AK95], we know that learning width five permutation branching programs with membership queries is hard under cryptographic assumptions.

The learnability of branching programs with read restrictions was studied in [RW93, GG95]. They proved that μ-branching programs (each variable appears at most once in the entire program) and ordered binary decision diagrams or Obdds (each variable appears at most once along any path) are exactly learnable using equivalence and membership queries. We extend the latter result in several ways. Let p be a fixed prime. We prove that a constant Boolean combination of a mod_p of polynomially many Obdds is exactly learnable from equivalence and membership queries.

Further in the paper, we study the problem of learning depth two circuits that consist of threshold and modular gates. We show that any depth two circuit with a mod_p gate at the top, for a fixed prime p, and arbitrary modular gates at the bottom level (possibly with different modularities) is exactly learnable using equivalence and membership queries. Also we show that any depth two circuit with a mod_p gate at the top, for a fixed prime p, and arbitrary threshold

gates at the bottom level is exactly learnable using equivalence and membership queries. We note that by a result of Krause and Pudlák [KP94], DNF formulae are contained in the class of depth two circuits with a threshold gate at the top and parity gates at the bottom level. Hence learning a threshold of modular gates will imply the learnability of DNF formulae.

Finally we prove that any Boolean combination of a constant number of concepts taken from the above classes is exactly learnable using equivalence and membership queries.

The rest of the paper is organized as follows. In section 2, we provide some necessary notation and definitions for the concept classes considered in the paper, such as decision trees and lists, branching programs, and multiplicity automata. We also define the learning models that we consider. In section 3 we prove the result on monotone width two branching program after first reviewing the basic theory of Fourier transform for Boolean functions. In section 4 we turn to width two branching programs with bounded number of sinks and prove that they are learnable from equivalence queries only. The last two sections, sections 5 and 6, we will use a single technique, exact learning of multiplicity automata, to prove that certain classes of permutation branching programs and ordered binary decision diagrams as well as small depth circuits with modular and threshold gates are exactly learnable from equivalence and membership queries.

2 Preliminaries

In this section we define some necessary notation and definition for various models of branching programs and the learning models considered in this paper.

We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$ and $[a, b]$ to denote $\{a, a+1, \ldots, b\}$. The Iversonian $I[statement]$ notation means 1 if the *statement* is true and 0 otherwise. For $a \in \{0,1\}^n$, let a_i denote the i-th bit of a. The vector $e_i \in \{0,1\}^n$ denotes the vector with all entries equal to zeros except for the i-th bit which is 1. The all zero vector is denoted 0_n. The Hamming weight of a, i.e. the number of ones in a, is denoted by $|a|$. The bitwise exclusive-or operation is denoted $\oplus$; sometimes we also use the $+$ sign. The inner product operation of $a \in \{0,1\}^n$ is denoted $a^T x$ or $a \cdot x$.

The ring of integers is denoted by Z. The finite field of q elements is denoted by $GF(q)$. The expectation operation is denoted $\mathbf{E}[]$; unless otherwise stated, the expectation is assumed to be over a uniform distribution (denoted U).

2.1 Decision trees and parity classes

Let A and B be two concept classes over $\{0,1\}^n$. An (A, B)-decision tree or (A, B)-DT is a rooted binary tree whose internal nodes are labeled with functions from A and whose leaves are labeled with functions from B. Each internal node has precisely two outgoing edges, one labeled with 0 and the other labeled with 1.

An (A, B)-decision tree computes a Boolean function from $\{0,1\}^n$ to $\{0,1\}$ in the following natural way. Given an assignment $a \in \{0,1\}^n$, the computation starts at the root node, evaluating each function that labels the internal node according to its label, and taking the consistent edge out to the next internal node. The computation stops at a leaf node and outputs the value of the function that labels the leaf node. An (A, B)-decision list or (A, B)-DL is a degenerate (A, B)-decision tree. In notation, we will write

$$[(f_1, g_1), (f_2, g_2), \ldots, (f_m, g_m)]$$

where $f_1, f_2, \ldots, f_m \in A$ and $g_1, g_2, \ldots, g_m \in B$, to represent a (A, B)-DL. We implicitly assume that the last function f_m is the constant one (always true) function.

The class *const* consists of the constant functions always false and always true. We will shorthand $(A, const)$ to A, for example $(A, const)$-DT is abbreviated A-DT.

We need to define the notion of *rank* of a decision tree.

Definition 1 (rank of a decision tree)
Let T be a decision tree. Then the rank of a node in T is defined inductively as follows. For a non-leaf node v, let v_L and v_R be the left and right child, respectively, of v.

$$rank(v) = \begin{cases} 0 & \text{if } v \text{ is a leaf} \\ 1 + rank(v_L) & \text{if } rank(v_L) = rank(v_R) \\ \max\{rank(v_L), rank(v_R)\} & \text{if } rank(v_L) \neq rank(v_R) \end{cases}$$

The rank of the tree T is the rank of its root node.

Another concept class that we consider is the class of parity functions $\oplus_k = \{a^T x + b | a \in \{0,1\}^n, b \in \{0,1\}, |a| \leq k\}$, where the parity may depend on at most k variables. Note that $\oplus_1$ is the set of literals and $\oplus_n$ is the set of all parities. We also consider the class of parities of k-monomials (monomials of size at most k) which we denote $\oplus \cap_k$.

2.2 Branching programs

A branching program M over $X_n = \{x_1, \ldots, x_n\}$ is an acyclic directed graph whose nodes labeled with variables from X_n and whose edges are labeled with the constants $\{0, 1\}$. It has a unique source (a node with no incoming edges) and at least two sinks (a node with no outgoing edges). The sinks are labeled with 0 (rejecting) and 1 (accepting), and both labels must be present. An assignment $a \in \{0,1\}^n$ to the variables induces a selection on the edges of M; it keeps alive all edges that are consistent with the assignment a. Then the branching program is said to accept a if there is a directed path from the source to an accepting sink.

The *size* of a branching program is the number of nodes in the branching program. A branching program is called *leveled* if there is ordered partition $\Pi = (L_1, L_2, \ldots)$ of the nodes of the branching program such that all of the edges connect nodes of one level to the next one in the partition. The *width* of a leveled branching program is the maximum number of nodes in any level in the ordered partition.

Borodin *et al.* [BDFP86] defined several variations of the width two branching programs. A width two branching program is called *strict* if it has exactly one accepting sink and one rejecting sink. A width two branching program is called *monotone* if it has exactly one rejecting sink. It was pointed out in [BDFP86] that any DNF can be converted into a width two monotone branching program. We will need the following properties of strict width two branching programs in terms of decision lists as shown in [BTW95].

Fact 1 *The class $\mathcal{SW}_2$ of strict width two branching programs is equivalent to $(\oplus_2, \oplus_n)$-DL. Moreover, any decision list in $(\oplus_2, \oplus_n)$-DL has length at most n^2.*

A width w *permutation branching program* is a leveled branching program of width w whose edges labeled with one form a permutation on $[w]$ and the same rule applies for edges labeled with zero. Also we require that the nodes in each level of the partition is labeled with a unique variable from X_n. Thus, sometimes we will say a G-permutation branching program or G-PBP, for some

permutation group G, to denote a specific permutation used by the branching program. The notion of acceptance in a permutation branching program is slightly different. For this we need to fix some subset S of G. Given an input assignment a, the entire branching program will compute a product of permutations from G. If this product is a permutation from S then the branching program accepts otherwise it rejects.

Next we define ordered binary decision diagrams or Obdds. Let π be an ordering or bijection of $\{1, 2, \ldots, n\}$. For an assignment $x \in \{0,1\}^n$, let $\pi(x)$ be the string $x_{\pi(1)} \ldots x_{\pi(n)}$. For an ordering π and a branching program M, we say that M is π-ordered bdd or π-Obdd if the labels of the nodes along any path in M are consistent with the ordering π. An ordered branching program or decision diagram is one that is π-ordered, for some π. The notion of accepting is the same for ordinary branching programs.

2.3 Small depth circuits with modular and threshold gates

In this section we introduce some notation for describing small depth Boolean circuits with modular and threshold gates. Our notation follows loosely the ones used by Krause and Pudlák [KP94].

A mod_p gate over n Boolean inputs $x_1, \ldots, x_n$ returns the value $x_1 + x_2 + \ldots + x_n (mod_p)$. A threshold gate or function over n Boolean inputs with integer weights $a = a_1, a_2, \ldots, a_n \in Z$ and a threshold of $b \in Z$, denoted by $f_{a,b}$, returns the value

$$f_{a,b}(x) = \begin{cases} 1 & \text{if } a_1x_1 + a_2x_2 + \ldots + a_nx_n \geq b \\ 0 & \text{otherwise} \end{cases}$$

In another notation, $f_{a,b}(x) = [\sum_{i=1}^{n} a_i x_i \geq b]$. The class of Boolean function computable by a threshold gate with integer weights is denoted by $\widehat{LT}_1$. For a threshold function $f_{a,b}$, we define $w(f_{a,b})$ to be $|b| + \sum_{i=1}^{n} |a_i|$. The *representation size* of a threshold function f is $w(f)$.

The class mod_Z is the class of mod_q gates, for all integers $q \in Z$. For two classes of gates A and B, we denote A-B circuits to be the class of functions computable by a depth two Boolean circuit with a gate from A at the top and gates from B at the bottom level. For example, a mod_p-mod_Z circuit is a depth two Boolean circuit that has a mod_p gate at the top and arbitrary mod_q gates, $q \in Z$, at the bottom level. Note that we allow mod_q gates with possibly different q's at the bottom level (not just for a single value q).

2.4 Learning models

In this paper we will consider two standard learning models, namely the Probably Approximately Correct (PAC) model and the exact learning model with equivalence and membership queries.

First we define the Probably Approximately Correct (PAC) learning model [V84]. Let C, H be two classes of Boolean functions over n variables, let D be a probability distribution over $\{0,1\}^n$, and let $f \in C$ be a target function chosen from C. The learning algorithm has access to an example oracle $EX(f, D)$ which generates random labeled examples $(a, f(a))$, where $a \in \{0,1\}^n$ is drawn according the distribution D.

We say that C is *PAC learnable* using H if there exists an algorithm A so that: for any concept $f \in C$, for any distribution D over $\{0,1\}^n$, for any $0 < \epsilon, \delta < 1$, if A is given access to $EX(f, D)$ and inputs ϵ, δ, then with probability at least $1 - \delta$, A outputs a hypothesis $h \in H$ satisfying $D(f \triangle h) \leq \epsilon$. The last probability is taken over the internal randomization of A along with the randomization in the calls to $EX(f, D)$. We also require that the learning algorithm A runs in time polynomial in $n, \frac{1}{\epsilon}, \frac{1}{\delta}$, and the size of the target function f. We say that C is weakly PAC learnable

under distribution D if there is a fixed polynomial p and a learning algorithm that succeeds for an error $\epsilon = \frac{1}{2} - \frac{1}{p(n,s(f))}$.

Next we review the exact learning model using equivalence and membership queries [Ang88, L88]. As above we have two concept classes C, H over $\{0,1\}^n$ and a target concept $f \in C$. In the exact learning model the learner asks certain oracles certain types of questions or queries about the target function f. The goal of the exact learning algorithm is to halt after time polynomial in n and the *size of the representation* for f in the class, and output a representation $h \in H$ that is logically equivalent to f. The following types of queries are allowed. In an *equivalence query*, the learning algorithm supplies any function $h \in H$ as input to $EQ_f()$ and the reply of the oracle is either *yes*, signifying that $h \equiv f$, or a *counterexample*, which is an assignment b such that $h(b) \neq f(b)$. In a *membership query*, the learning algorithm supplies an assignment b to $MQ_f()$ and the reply of the oracle is $f(b)$.

Note that, initially, the learner has no knowledge about f other than its membership to the target class. Learning must succeed against any valid choice of counterexamples by the teacher.

In both of these learning models, we assume that the hypothesis h produced by the learner is taken from the representation class H. The only requirement on H that we impose is that it be polynomial-time computable (or evaluatable), i.e., given $x \in \{0,1\}^n$, $h(x)$ is computable in polynomial time.

2.5 Multiplicity automata

In this section we describe relevant definitions from the theory of multiplicity automata and a recent result on the learnability of multiplicity automata. We will also prove a lemma that describes a non-trivial closure operation on this class of automata.

Let K be a field. A nondeterministic automaton M with multiplicity is a five-tuple $M(A, Q, E, I, F)$ where A is a finite alphabet, Q is the finite set of states, $I, F : Q \to K$ are two mappings associated with the initial and final states, respectively, and

$$E : Q \times A \times Q \to K$$

is a map that associates a multiplicity to each edge of M. We will sometimes call M a K-automaton for brevity.

Let $x = (x_1, \ldots, x_n) \in A^*$. A path for x is a sequence

$$p = (p_1, x_1, p_2), (p_2, x_2, p_3), \ldots, (p_n, x_n, p_{n+1}).$$

Let $Path_M(x)$ denote the set of all paths for x. The *behavior* of M is a mapping $S_M : A^* \to K$ defined as follows: for each $x = (x_1, \ldots, x_n) \in A^*$

$$S_M(x) = \sum_{p \in Path_M(x)} I(p_1) \left(\prod_{i=1}^{n} E(p_i, x_i, p_{i+1}) \right) F(p_{n+1}).$$

For a Boolean function over $f : A^* \to \{0,1\}$, we say that a multiplicity automata M computes f if for all $x \in A^*$ we have $S_M(x) = f(x)$. Alternatively one may think of f as a characteristic function of a language over A^*.

In the following we will describe several operations on multiplicity automata, namely the Hadamard product, union, and scalar multiplication. In effect we will argue that multiplicity automata are closed under these operations.

Definition 2 (Closure operations)
Let K be a field. Let $M_1(A, Q_1, E_1, I_1, F_1)$ and $M_2(A, Q_2, E_2, I_2, F_2)$ be two K-automata.

1. *The* Hadamard product *of M_1 and M_2, denoted by $M_1 \odot M_2$, is a K-automaton $M(A, Q, E, I, F)$ where $Q = Q_1 \times Q_2$, and I, F, E are defined as $I(q_1, q_2) = I_1(q_1)I_2(q_2)$, $F(q_1, q_2) = F_1(q_1)F_2(q_2)$, and $E((q,p), a, (q', p')) = E_1(q, a, q')E_2(p, a, p')$. Note that M has $|Q_1||Q_2|$ states. Moreover M satisfies*
$$S_M(x) = S_{M_1}(x)S_{M_2}(x).$$

2. *Assume that Q_1 and Q_2 are two disjoint sets of states. The* union *of M_1 and M_2, denoted simply by $M_1 \cup M_2$, is a K-automaton where $M(A, Q, E, I, F)$ where $Q = Q_1 \cup Q_2$, and I, F, E are defined as $I(q) = I_1(q)[q \in Q_1] + I_2(q)[q \in Q_2]$, $F(q) = F_1(q)[q \in Q_1] + F_2(q)[q \in Q_2]$, $E(q, a, p) = E_1(q, a, p)[q, p \in Q_1] + E_2(q, a, p)[q, p \in Q_2]$. Note that M has $|Q_1| + |Q_2|$ states. Moreover M satisfies*
$$S_M(x) = S_{M_1}(x) + S_{M_2}(x).$$

3. *For any $\lambda \in K$, the automaton λM_1 is defined to be K-automaton $M(A, Q, E, I, F)$ where $Q = Q_1$, $I = \lambda I_1$, $F = F_1$, and $E = E_1$. Note that $|Q| = |Q_1|$ and that M satisfies*
$$S_M(x) = \lambda S_{M_1}(x).$$

Next we prove a result that adds another closure operation, namely constant Boolean combinations of multiplicity automata.

Lemma 2.1 *Let p be a fixed prime. Let $g_1, g_2, \ldots, g_k$ be Boolean functions that can be computed by $GF(p)$-automata of size at most s. Then for any Boolean function f on k inputs, $f(g_1, g_2, \ldots, g_k)$ can be computed by a $GF(p)$-automaton with at most $2^k s^k$ states.*

Proof The function $f(g_1, g_2, \ldots, g_k)$ can be written as
$$\sum_{\alpha \in Z^k} \lambda_\alpha g_1^{\alpha_1} \times \ldots \times g_k^{\alpha_k}.$$
Since $g_1, g_2, \ldots, g_k$ take values $\{0, 1\}$, we may assume that $\alpha_1, \ldots, \alpha_k \in \{0, 1\}$. Therefore we can write
$$f = \sum_{\beta \in \{0,1\}^k} \lambda_\beta g_1^{\beta_1} \times \ldots \times g_k^{\beta_k}.$$
By the properties of Hadamard product $g_1^{\beta_1} \times \ldots \times g_k^{\beta_k}$ has a multiplicity $GF(p)$-automaton of size at most s^k. The multiplication with λ_β admits another $GF(p)$-automaton of size s^k. Then summing 2^k of such $GF(p)$-automata gives an automaton with size at most $2^k s^k$. ∎

A *multiplicity oracle* $MUL_M()$ for a K-automaton M is an oracle that receives as input a string $x \in A^*$ and returns $S_M(x)$. The following result was recently established in [BV94].

Theorem 2.1 *[BV94] The class of K-automata, for a field K, is exactly learnable from equivalence and multiplicity queries.*

3 Monotone width two branching programs

In this section we prove that monotone width two branching programs are PAC learnable with membership queries under the uniform distribution. To prove the claim we need to introduce some notation and fact from harmonic or Fourier analysis of Boolean functions. In fact our analysis will use some of the standard methods from this area.

3.1 Fourier transform of Boolean functions

We will review the basic setting for the Fourier transform of Boolean functions [M94]. In this setting Boolean functions will be thought of having range $\{-1,+1\}$. To avoid confusion we will denote "normal" Boolean functions, i.e., ones with the range $\{0,1\}$, with lower-case letters, such as $f:\{0,1\}^n \to \{0,1\}$, and their *corresponding* $\{-1,+1\}$-range *counterpart* with upper-case letters, e.g., $F:\{0,1\}^n \to \{-1,+1\}$. It is easy to see that the following relations hold between f and F:

$$F = 2f - 1, \qquad f = \frac{1+F}{2}.$$

Let $F:\{0,1\}^n \to \{-1,+1\}$ be a Boolean function. The Fourier transform of Boolean functions over the uniform distribution is defined as follows. First we define the inner product over the 2^n-dimensional vector space of all real-valued functions over $\{0,1\}^n$.

$$\langle F, G\rangle = 2^{-n}\sum_x F(x)G(x) = \mathbf{E}[FG].$$

Next we define for each $a \in \{0,1\}^n$ the basis function χ_a as follows:

$$\chi_a(x) = (-1)^{\sum_{i=1}^n a_i x_i}.$$

These functions are *orthonormal*, i.e., $\langle \chi_a, \chi_b\rangle = [a \neq b]$, and they are *decomposable*, i.e., $\chi_{ab}(xy) = \chi_a(x)\chi_b(y)$, where xy is the concatenation of strings x and y (possibly of different lengths).

Given the orthonormality of these χ_a's we get the Fourier representation of any Boolean function F as

$$F(x) = \sum_a \hat{F}(a)\chi_a(x),$$

where $\hat{F}(a) = \langle F, \chi_a\rangle = \mathbf{E}[F\chi_a]$. Also because of orthonormality we have Parseval's identity: $\sum_a \hat{F}^2(a) = 1$. Finally note that $\chi_{0_n}(x)$ is the constant function 1.

3.2 Jackson's DNF learning algorithm

In a beautiful paper Jackson [J94] proved the following theorem.

Theorem 3.1 *[J94] The class of DNF formulae is PAC learnable from membership queries under the uniform distribution.*

We outline the arguments used by Jackson and illustrate how we modify it to prove the learnability of monotone width two branching programs. The first key fact about DNF formulae is that they correlate well with *some* parity function χ_A, $A \in \{0,1\}^n$, under *any* distribution. In notation, if f is a DNF formulae of size s (f has s terms) and D is an arbitrary distribution, then there is some A such that

$$|\mathbf{E}_D[F\chi_A]| \geq \frac{1}{2s+1}.$$

We remind the reader that F is the $\{-1,+1\}$-version of f. Using the above inequality, since $\mathbf{E}_D[F\chi_A] = \Pr_D[F = \chi_A] - \Pr_D[F \neq \chi_A]$ we derive the following. Assume without loss of generality that $\mathbf{E}_D[F\chi_A]$ is positive (the other case is symmetrically similar).

$$\mathbf{E}_D[F\chi_A] = 1 - 2\Pr_D[F \neq \chi_A] \geq \frac{1}{2s+1} \implies \Pr_D[F \neq \chi_A] \leq \frac{1}{2} - \frac{1}{2(2s+1)}.$$

This is good news since it means that the parity function χ_A is a potential hypothesis for *weak* learning f. The problem is that we do not know what A is in relation to F (other than it correlates well with F).

The second key fact is that there is an efficient algorithm due to Kushilevitz and Mansour [KM93] to find all parities that correlate well with certain Boolean functions assuming that the underlying distribution is uniform. So weakly learning DNF under the uniform distribution is possible by combining these two facts [BFJKMR94].

The third ingredient is a *boosting* algorithm, developed by Freund [F90], that can turn any weak learning algorithm into a strong learning algorithm. This does not solve the DNF learning problem immediately since the boosting algorithm assumes that the weak learning algorithm works under *arbitrary* distributions (not just the uniform distribution). This is because the boosting method works by running the weak learning algorithm on a carefully chosen set of modified distributions.

Jackson then supplied the missing pieces: he proved that the boosting algorithm of Freund combined with a modified version of Kushilevitz and Mansour's algorithm will still work since the distribution is not being perturbed too much (he quantified precisely this intuition in [J94]). Also by the first fact, DNF formulae are still guaranteed to correlate well with some parity when the distribution is changed. In fact the only property that is ever needed about DNF formulae to get the learning result is the first fact. Jackson called this DNF learning algorithm the *harmonic sieve* algorithm.

To prove our PAC learning result we will show in the next section that the first fact holds for monotone width two branching programs. Using this we can then claim the following theorem.

Theorem 3.2 *The class $\mathcal{MW}_2$ of monotone width two branching programs is PAC learnable with membership queries under the uniform distribution.*

3.3 A correlation lemma

In this section we prove the following lemma that states any monotone width two branching program correlates well with some parity function under *any* distribution. The fact that the lemma is true for any distribution is critical for the boosting stage in Jackson's harmonic sieve algorithm.

Lemma 3.1 *For any $F \in \mathcal{MW}_2$ with s accepting sinks and for any distribution D there is a parity χ_C that satisfies*

$$|\mathbf{E}_D[F\chi_C]| \geq \frac{1}{2sn^2+1}.$$

Proof Let $f \in \mathcal{MW}_2$ be computed by a monotone width two branching program with s sinks. Note that each accepting sink defines a subportion of the branching program that is a strict width two branching program. Let $g_1, \ldots, g_s$ be the functions computed by the s subportions associated with the s strict width two branching programs. Note that

$$f = g_1 \vee g_2 \vee \ldots \vee g_s.$$

Hence there is a subportion g_i so that

$$\Pr_D[g_i = 1] \geq \frac{1}{s} \Pr_D[f = 1].$$

We fix our attention to this subportion g_i and call it g.

Using Fact 1, $g \in \mathcal{SW}_2$ is equivalent to a decision list in $(\oplus_2, \oplus_n)$-DL. If g is defined as

$$G = [(\chi_{a_1}, \chi_{b_1}), (\chi_{a_2}, \chi_{b_2}), \ldots, (\chi_{a_m}, \chi_{b_m})]$$

(recall that G is the $\{-1, +1\}$-version of g) then it can be rewritten as

$$g = \sum_{i=1}^{m} \frac{1 + \chi_{a_i}}{2} \frac{1 + \chi_{b_i}}{2} \prod_{j=1}^{i-1} \frac{1 - \chi_{a_j}}{2}.$$

Let

$$h_i = \frac{1 + \chi_{a_i}}{2} \frac{1 + \chi_{b_i}}{2} \prod_{j=1}^{i-1} \frac{1 - \chi_{a_j}}{2}.$$

Therefore

$$\begin{aligned} |\mathbf{E}_D[Fg]| &= \left| \mathbf{E}_D \left[F \left(\sum_{i=1}^{m} \frac{1 + \chi_{a_i}}{2} \frac{1 + \chi_{b_i}}{2} \prod_{j=1}^{i-1} \frac{1 - \chi_{a_j}}{2} \right) \right] \right| \\ &= \left| \mathbf{E}_D \left[F \sum_{i=1}^{m} h_i \right] \right| = \left| \sum_{i=1}^{m} \mathbf{E}_D[Fh_i] \right| \\ &\leq \sum_{i=1}^{m} |\mathbf{E}_D[Fh_i]| \leq m \, |\mathbf{E}_D[Fh_{i_0}]| \end{aligned}$$

where $i_0 \in [m]$ is such that $|\mathbf{E}_D[Fh_{i_0}]|$ is maximum. We now rewrite

$$\prod_{j=1}^{i_0-1} \frac{1 - \chi_{a_j}}{2} = \mathbf{E}_{\alpha \in \{0,1\}^{i_0-1}}[(-1)^{|\alpha|} \chi_{A_\alpha}],$$

where $A_\alpha = \sum_{t=1}^{i_0} \alpha_t a_t$. Thus we have

$$\begin{aligned} h_{i_0} &= \frac{1 + \chi_{a_{i_0}}}{2} \frac{1 + \chi_{b_{i_0}}}{2} \prod_{j=1}^{i_0-1} \frac{1 - \chi_{a_j}}{2} \\ &= \frac{1 + \chi_{a_{i_0}}}{2} \frac{1 + \chi_{b_{i_0}}}{2} \mathbf{E}_{\alpha \in \{0,1\}^{i_0-1}}[(-1)^{|\alpha|} \chi_{A_\alpha}] \\ &= \mathbf{E}_S[(-1)^{|\alpha|} \chi_B] \end{aligned}$$

where the probability space S is over α uniformly chosen from $\{0,1\}^{i_0-1}$ and β, γ uniformly chosen from $\{0,1\}$ and B is defined as $B = A_\alpha + \beta a_{i_0} + \gamma b_{i_0}$. Combining this into an earlier expression we get

$$|\mathbf{E}_D[Fh_{i_0}]| = \left| \mathbf{E}_D[F \mathbf{E}_S[(-1)^{|\alpha|} \chi_B]] \right| = \left| \mathbf{E}_S[(-1)^{|\alpha|} \mathbf{E}_D[F\chi_B]] \right| \leq \mathbf{E}_S[|\mathbf{E}_D[F\chi_B]|].$$

Given this we may claim that there is a choice $\alpha_0, \beta_0, \gamma_0$ with $C = A_{\alpha_0} \oplus \beta_0 a_{i_0} \oplus \gamma_0 b_{i_0}$ so that

$$|\mathbf{E}_D[F\chi_C]| \geq |\mathbf{E}_D[Fh_{i_0}]| \geq \frac{|\mathbf{E}_D[Fg]|}{m}.$$

But notice that since g implies f we have the following relation (we remind the reader that F has range $\{-1, +1\}$ and g has range $\{0, 1\}$)

$$\mathbf{E}_D[Fg] = \mathbf{E}_D[g] = \Pr_D[g = 1] \geq \frac{1}{s} \Pr_D[f = 1] = \frac{\mathbf{E}_D[F] + 1}{2s}.$$

Hence, we have

$$|\mathbf{E}_D[F\chi_C]| \geq \frac{\mathbf{E}_D[F] + 1}{2sm}.$$

So either $|\mathbf{E}_D[F\chi_C]| \geq 1/(2sm + 1)$ or $\mathbf{E}_D[F] = \mathbf{E}_D[F\chi_{0_n}] \leq -1/(2sm + 1)$. Noting that by Fact 1 $m \leq n^2$, we obtain the desired claim. ∎

4 Width two branching programs with bounded number of sinks

In this section we prove width two branching programs with a constant number of sinks is exactly learnable from equivalence queries only. In notation we use k-sink $\mathcal{W}_2$ to denote the class of width two branching programs with at most k sinks. We prove the following theorem.

Theorem 4.1 *k-sink $\mathcal{W}_2$ is exactly learnable using equivalence queries.*

In the next lemma we prove that k-sink width two branching program is equivalent to rank-k decision trees with parity nodes.

Lemma 4.1 *The class of k-sink, $k \geq 2$, width two branching programs is a subclass of the class rank-k $\oplus_n$-DT decision trees with parity nodes.*

Proof We prove by induction on k. For $k = 2$, the claim states that strict width two branching program or $\mathcal{SW}_2$ is a subclass of rank-2 $\oplus_n$-DT. But this is true by Fact 1, since $\mathcal{SW}_2$ is equivalent to $(\oplus_2, \oplus_n)$-DL, and an element $f \in (\oplus_2, \oplus_n)$-$DL$ can be turned into an element of rank-2 $\oplus_n$-DT (by trivially adding two new nodes for each leaf of f).

Assume that the claim is true for all width two branching programs with at most $k - 1$ sinks, $k \geq 3$. Consider a width two branching program B with k sinks. B can be decomposed into k strict width two branching programs. Let L_1 be the first strict width two branching program and let $b \in \{0, 1\}$ be the label of its sink. By Fact 1 L_1 can be converted into a decision list of type $(\oplus_2, \oplus_n)$-DL. By induction the remaining portion of B can be written as a rank-$(k - 1)$ decision tree T with parity nodes. We attach T to each leaf node of L_1 as follows. For each leaf node l of L_1, we create an outgoing edge labeled with -1 (or 0) going into T and an outgoing edge labeled with $+1$ going into the constant function b. Note that the resulting decision tree is of rank k. To see this, notice that the new rank of the leaves of L_1 is now k (the rank of T) and hence the new rank of the internal nodes of L_1 is also k. This completes the inductive argument and hence the lemma. ∎

Next, using an idea due to Blum [B92], we transform further the target width two branching program into a decision list representation.

Lemma 4.2 *The class of rank-k decision trees with parity nodes is a subclass of decision lists whose nodes are parity of monomials, where each monomial is of size at most k. In notation, rank-k $\oplus_n$-DT is a subclass of $\oplus \cap_k$ -DL.*

Proof The proof follows is an adaptation of Blum's argument [B92]. Let T be a rank-k decision trees with parity nodes. Because of its rank, T has a leaf node that is of depth at most k (Lemma 1 in [B92]); call this leaf node x. Let p be the parent of x and let T_p be the other subtree of p.

Create a node n_x in the decision list that is labeled with a conjunction of at most k parity questions (induced by the root to leaf path ending in x). This conjunction of parities can be converted into a parity of conjunctions where each conjunction is of size at most k.

The next crucial step is that we can remove from T, the nodes p and x, and reattach the parent of p directly to T_p. The resulting tree is still a rank-k' decision tree with parity nodes, where $k' \leq k$. If $k' = k$ then we may repeat the same process until the rank reduces to $k - 1$. At that point we appeal to an inductive hypothesis and complete the lemma. ∎

Finally we show in the following lemma that $\oplus \cap_k$ -DL, and hence k-sink width two branching programs, are exactly learnable from equivalence queries. The idea is to use the closure algorithm for learning nested differences of intersection-closed concept classes due to Helmbold, Sloan, and Warmuth [HSW90].

Lemma 4.3 *The class $\oplus \cap_k$ -DL of decision lists whose nodes are parity of monomials of size at most k is exactly learnable using equivalence queries.*

Proof By the transformation technique of Littlestone [L88], it suffices to prove the claim for the concept class of decision list with parity nodes, i.e., $\oplus_n$-DL. That is we can create new variables for each k-subset of the variables and learn the target concept as a new function over at most $n + n^k$ variables.

To exactly learn $\oplus_n$-DL we will appeal to the exact learning algorithm for nested differences of intersection-closed concept classes due to Helmbold, Sloan, and Warmuth [HSW90]. For this we will argue that we can express any element of $\oplus_n$-DL as a nested difference of vector spaces over $GF(2)^n$.

Assume that the target concept $f \in \oplus_n$-DL is given by

$$f = [(\chi_{a_1}, b_1), (\chi_{a_2}, b_2), \ldots, (\chi_{a_k}, b_k)],$$

where $a_1, a_2, \ldots, a_k \in \{0,1\}^n$ and $b_1, b_2, \ldots, b_k \in \{0,1\}$. We *compress* consecutive leaves that output the same value. This is fine since consecutive parity tests can be turn into a membership test for a linear subspace L that halts at the leaf if the test failed and proceeds to the next node if the test is passed. When the compression process is finished, we will end up with a decision list whose internal nodes are labeled with membership tests for linear subspaces. So assume that we have

$$f = [(L_1, c_1), (L_2, c_2), \ldots, (L_t, c_t)],$$

where $t \leq k$, the L_i's stand for linear subspaces, and $c_1, \ldots, c_t \in \{0,1\}$ are alternating in values. Again we remind the reader that the value c_1 will be output if the example does not belong to the linear subspace L_1, the value c_2 will be output if the example passed test L_1 but not test L_2, and so on. Assume without loss of generality that $c_1 = 0$ (the case when $c_1 = 1$ can be treated as easily). Then we have the following form

$$f = L_1 - (L_2 - (L_3 - (\ldots))).$$

Note crucially that the class of linear subspaces are closed under intersection and hence we may safely apply the closure algorithm given in [HSW90]. ∎

5 Small depth circuits and bounded width permutation branching programs

In this section we will show that some small depth circuits with modular and threshold gates and certain types of bounded width permutation branching programs are exactly learnable using equivalence and membership queries. First we consider depth two circuits with modular gates.

Theorem 5.1 *For any fixed prime p, the class of mod_p-mod_Z circuits is exactly learnable using equivalence and membership queries.*

Proof It suffices to exhibit a multiplicity automaton for the target mod_p-mod_Z circuit. Let $K = GF(p)$. We construct for each mod_q gate, $q \in Z$, a K-automaton that accepts it (a simple cycle-like automata will suffice here). Next we combine these automata into a single K-automaton by taking the union of all the automata for the mod_q gates. By Fermat's Little theorem, the Hadamard product of M with itself $p-1$ times computes the target mod_p-mod_Z circuit. ∎

In the following we will exploit circuit characterizations of permutation branching programs to prove the learnability of S_3 and A_4 permutation branching programs, where S_3 is the symmetric group on [3] and A_4 is the alternating group on [4]. The following fact about S_3-PBPs was proved by Barrington [B86].

Fact 2 *The class S_3-PBP is equivalent to mod_3-mod_2 circuit.*

Theorem 5.2 *S_3-PBPs are exactly learnable using equivalence and membership queries.*

Proof Follows from Theorem 5.1. ∎

It was implicitly shown in [BST90] that A_4-PBP is equivalent to (mod_2, mod_2)-mod_3 circuit, i.e., a depth "two" circuit consisting of mod_3 gates at the bottom level coming into two mod_2 gates at the second level. The output of the two mod_2 gates can then be combined using *any* Boolean operation.

We will prove a more general result and as a corollary we will show that A_4 permutation branching programs are exactly learnable using equivalence and membership queries.

Theorem 5.3 *Let $g_1, g_2, \ldots, g_k$ be Boolean functions that can be computed by a multiplicity $GF(p)$-automata of size at most s. Then for any Boolean function f on k inputs, $f(g_1, g_2, \ldots, g_k)$ is exactly learnable using equivalence and membership queries in time $s^{O(k)}$.*

Proof Follows from Lemma 2.1 and Theorem 2.1. ∎

Corollary 5.1 *A_4 PBPs are exactly learnable using equivalence and membership queries.*

In the following we will show that Boolean functions computable by threshold gates with integer weights can be represented as a multiplicity automata.

Lemma 5.1 *The class $\widehat{LT}_1$ admits a representation as a $GF(p)$-automata, for any prime p.*

Proof Suppose that $f(x) = [a^T x \geq b]$ where $a \in Z^n$ and $b \in Z$. Let $A = |a_1| + |a_2| + \ldots + |a_n| + 1$. We construct the automata M with state set $Q = \{q_{i,j} : i \in [-A, A], j \in [n+1]\}$. The edge set contain only the following edges (assigned a multiplicity of 1):

$$(q_{i,j}, 0, q_{i,j+1}), (q_{i,j}, 1, q_{i+a_j,j+1}) \in E$$

for all $i \in [A]$ and $j \in [n]$. Set $I = \{q_{0,0}\}$ and $F = \{q_{i,n+1} | i \geq b\}$. ∎

Using the above lemma we can claim (as before) that a mod_p and a constant Boolean combination of threshold functions are exactly learnable.

Corollary 5.2 *The class of mod_p-$\widehat{LT}_1$ is exactly learnable using equivalence and membership queries.*

Corollary 5.3 *For any Boolean function f on k inputs, and for $g_1, \ldots, g_k$ taken from the class mod_p-$\widehat{LT}_1$, the Boolean function $f(g_1, g_2, \ldots, g_k)$ is exactly learnable using equivalence and membership queries.*

We remark that proving the learnability of either $\widehat{LT}_1$-mod_p or $\widehat{LT}_1$-$\widehat{LT}_1$ functions will prove the learnability of DNF formulae [KP94].

6 Ordered binary decision diagrams

In this section we provide exact learning algorithms for certain Boolean combinations of ordered binary decision diagrams (Obdds). In particular we consider any mod_p and any constant Boolean combinations of Obdds.

In all of our learning results we always assume that the learning algorithm is given the ordering π. As pointed out by Gavaldà and Guijarro [GG95] the problem of learning using the best ordering (that minimizes the size of the branching program) is hard. So from now on we will assume that the ordering is simply the identity ordering $x_1 < x_2 < \ldots < x_n$.

Now we prove that a mod_p of polynomially many Obdds are learnable. First we state a result that is implicit in the work of Gavaldà and Guijarro [GG95].

Fact 3 *[GG95] For every Obbd there is an equivalent $GF(p)$-automaton, for any prime p, that accepts it. Moreover the transformation can be carried out in polynomial-time in the size of the Obdd.*

Theorem 6.1 *Let p be a fixed prime. A mod_p of a polynomially many Obdds is exactly learnable using equivalence and membership queries.*

Proof First we convert each Obdd into a $GF(p)$-automaton using Fact 3. Finally we combine all these into a single $GF(p)$-automaton by taking the union and by taking the Hadamard product $p-1$ times to *booleanize* the output. ∎

Theorem 6.2 *For any Boolean function f on k inputs, the function $f(g_1, g_2, \ldots, g_k)$, where $g_1, \ldots, g_k$ are mod_p of Obdds, is exactly learnable using equivalence and membership queries.*

Proof First we convert each mod_p of Obdds into a $GF(p)$-automaton using the previous theorem, Theorem 6.1. To learn the target function, we apply Lemma 2.1 and Theorem 2.1. ∎

Acknowledgments

We thank David A. Mix Barrington for his generosity in sending us his thesis and for his help on permutation branching programs.

References

[Ang88] Dana Angluin. Queries and Concept Learning. *Machine Learning*, 2:319–342, 1988.

[AK95] Dana Angluin and Michael Kharitonov. When Won't Membership Queries Help? In *Journal of Computer and System Sciences*, 50:336–355, 1995.

[B86] David A. Barrington. Bounded-Width Branching Programs. PhD thesis, Massachusetts Institute of Technology, 1986.

[B89] David A. Barrington. Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC^1. In *Journal of Computer and System Sciences*, 38:150–164, 1989.

[B92] Avrim Blum. Rank-r Decision Trees are a Subclass of r-Decision Lists. In *Information Processing Letters*, 42:183–185, 1992.

[BCV] Francesco Bergadano, D. Catalano, and Stefano Varricchio. Learning Sat-k-DNF Formulas from Membership Queries. Manuscript.

[BDFP86] Allan Borodin, Danny Dolev, Faith Fich, and Wolfgang Paul. Bounds for Width Two Branching Programs. In *SIAM Journal on Computing*, 15(2):549–560, 1986.

[BFJKMR94] Avrim Blum, Merrick Furst, Jeffrey Jackson, Michael Kearns, Yishay Mansour, and Steven Rudich. Weakly Learning DNF and Characterizing Statistical Query Learning using Fourier Analysis. In *Proceedings of the Twenty Sixth Annual ACM Symposium on Theory of Computing*, pages 253–262, 1994.

[BST90] David A. Mix Barrington, Howard Straubing, and Denis Thérien. Non-uniform Automata over Groups. In *Information and Computation*, 89:109–132, 1990.

[BTW95] Nader H. Bshouty, Christino Tamon, and David K. Wilson. On Learning Width Two Branching Programs. Manuscript, 1995.

[BV94] Francesco Bergadano and Stefano Varricchio. Learning Behaviors of Automata from Multiplicity and Equivalence Queries. In *Proceedings of the Second Italian Conference on Algorithms and Complexity (CIAC 94)*, Lecture Notes in Computer Science No. 778, M. Bonuccelli, P. Crescenzi, R. Petreschi (Eds.), Springer-Verlag, 1994. To appear in SIAM Journal on Computing.

[ERR95] Funda Ergün, S. Ravi Kumar, and Ronitt Rubinfeld. On Learning Bounded-Width Branching Programs. In *Proceedings of the 8th Annual ACM Conference on Computational Learning Theory*, 361–368, 1995.

[F90] Yoav Freund. Boosting a Weak Learning Algorithm by Majority. In *Proceedings of the 3rd Annual Workshop on Computational Learning Theory*, 202–216, 1990.

[GG95] Ricard Gavaldà and David Guijarro. Learning Ordered Binary Decision Diagrams. In *6th International Workshop on Algorithmic Learning Theory*, Lecture Notes in Artificial Intelligence No. 997, Jantke, Shinohara, Zeugmann (Eds.), Springer-Verlag, 1995.

[GM93] Jordan Gergov and Christoph Meinel. MOD-2-OBDD's – A Generalization of OBDD's and EXOR-Sum-Of-Products. In *IFIP WG 10.5 Workshop on the Applications of Reed-Muller Expansion in Circuit Design*, Hamburg, 1993.

[HSW90] David Helmbold, Robert Sloan, and Manfred Warmuth. Learning Nested Differences of Intersection-Closed Concept Classes. In *Machine Learning*, 5:165–196, 1990.

[J94] Jeffrey C. Jackson. DNF is Efficiently Learnable under the Uniform distribution with Membership Queries. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, 42–53, 1994.

[KM93] Eyal Kushilevitz and Yishay Mansour. Learning Decision Trees using the Fourier Spectrum. In *SIAM Journal on Computing*, **22**:6, pages 1331–1348, 1993.

[KP94] Matthias Krause and Pavel Pudlák. On the Computational Power of Depth 2 Circuits with Threshold and Modulo gates. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, 48–57, 1994.

[L88] Nick Littlestone. Learning Quickly When Irrelevant Attributes Abound: A New Linear-Threshold Algorithm. *Machine Learning*, **2**, 285–318, 1988.

[M94] Yishay Mansour. Learning Boolean Functions via the Fourier Transform. Tutorial Notes for the *Workshop on Computational Learning Theory*, 1994.

[RW93] Vijay Raghavan and Dawn Wilkins. Learning Branching Programs with Queries. In *Proceedings of the 6th Annual Workshop on Computational Learning Theory*, 27–36, 1993.

[V84] Leslie G. Valiant. A Theory of the Learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.