New Results for Random Walk Learning

Jeffrey C. Jackson[*]
Duquesne University, 600 Forbes Ave
Pittsburgh PA 15282-1754 USA
jacksonj@duq.edu

Karl Wimmer
Carnegie Mellon University, 5000 Forbes Ave
Pittsburgh, PA 15213 USA
kwimmer@andrew.cmu.edu

Abstract

In a very strong positive result for passive learning algorithms, Bshouty *et al.* showed that DNF expressions are efficiently learnable in the uniform random walk model. It is natural to ask whether the more expressive class of thresholds of parities (TOP) is similarly learnable, but the Bshouty *et al.* time bound becomes exponential in this case. We present a new approach to weak parity learning that leads to quasi-efficient random walk learnability of TOP. We also introduce a more general random walk model naturally related to the Metropolis-Hastings algorithm and show that DNF is efficiently learnable and that juntas are efficiently agnostically learnable in this model.

1 Introduction

Positive results in learning theory have often assumed that the learner has access to a membership oracle. Such a learner is *active*, actively choosing examples for which it would like information. In arguably the strongest result to date in any natural *passive* model (i.e., any model with examples chosen randomly according to some natural process), Bshouty *et al.* [BMOS05] showed that an algorithm that receives examples drawn via a uniform random walk on the Boolean cube can efficiently learn the class of DNF expressions, that is, learn an arbitrary Boolean function f in time polynomial in the minimum number of terms in any DNF expression equivalent to f (and polynomial in other parameters).

However, Bshouty *et al.* left as an open problem for the uniform random walk model the efficient learnability of polynomial-weight threshold-of-parity (poly-TOP) functions. A function $f : \{0,1\}^n \to \{-1,1\}$ is in poly-TOP if it can be represented as the sign of a weighted sum of parity functions, where the weights are integers and the sum of the magnitudes of the weights is bounded by some fixed polynomial in n. More generally, we can ask if there is an algorithm that learns any Boolean function f in time polynomial in the minimal weight of any TOP representation of f as well as in the other standard PAC learning parameters. This is an intriguing question both because TOP is a much more expressive class than is DNF and because the membership query algorithm for efficiently learning DNF, the Harmonic Sieve [Jac97], can also efficiently learn TOP.

Unfortunately, as Bshouty *et al.* point out, their approach does not seem to be capable of producing a positive TOP learning result. Actually, they give two algorithms, one using a random walk oracle and a second using a weaker oracle that can be viewed as making statistical queries to estimate noise sensitivity (which is, roughly, the probability that corrupting an input to a function changes the output). A proof in Roch [Roc07] shows that time $2^{\Omega(n)}$ is required for the latter approach, and Bshouty *et al.*'s time bound for the former approach also becomes exponential when applied to learning even a single parity function on $\Omega(n)$ bits.

In somewhat more detail, the Bshouty *et al.* weak learning algorithm is loosely based on the algorithm of Goldreich and Levin [GL89]—adopted for learning purposes by Kushilevitz and Mansour [KM93] and therefore often referred to in learning papers as the **KM** algorithm—for locating all of the Fourier coefficients of a function whose magnitude exceeds some threshold. Bshouty *et al.* replace the influence-like estimates made by **KM** with noise sensitivity estimates, and they also employ a breadth-first search rather than **KM**'s depth-first approach. Each of these changes makes the modified **KM** unsuitable for finding a weak-approximating parity on $\Omega(n)$ bits: high-order Fourier coefficients contribute only negligibly to low-order noise sensitivity estimates, which are the only estimates made by Bshouty *et al.*; employing breadth-first search for a high-degree parity using the original **KM** influence-like estimates would lead to

[*]This material is based upon work supported by the National Science Foundation under Grant No. CCF-0728939.

computing an exponential in n number of estimates before locating the parity.

We are aware of only one approach to weakly learning parity that differs fundamentally from **KM**. This approach is based on a clever algorithm due to Levin [Lev93] that randomly partitions the set of 2^n Fourier coefficients into polynomially many bins and succeeds in finding a weak approximator if one of the bins is dominated by a single coefficient, which happens with non-negligible probability. Variants of Levin's algorithm have been proposed and analyzed by others ([BJT04, Fel07]). But it is not at all clear how Levin's original algorithm or any of the variants could be adapted to use a random walk oracle rather than a membership oracle.

Thus, it seems that a fundamentally new approach to weakly learning parity functions is needed in order to efficiently learn TOP in the random walk model. We take a step in this direction, presenting an algorithm that learns TOP in time polynomial in n but exponential in $\log(s/\epsilon)$, where s is the minimal TOP-weight of the target function and ϵ is the desired accuracy of the approximation. The algorithm borrows some underlying Fourier ideas from **KM**, but differs markedly in how it employs these ideas. A key feature is that our algorithm can be viewed as an elimination algorithm: it locates a good approximating parity function by eliminating all other possibilities. The core of our algorithm can also be viewed as an agnostic learner for parity in the random walk model; it is efficient if the optimal parity is an $O(1)$-approximator to the target. In contrast, the best current algorithm for the same problem given only uniform random examples has time bound $2^{O(n/\log n)}$ [BKW03].

We give a quasi-polynomial algorithm for learning TOP in the uniform random walk model. TOP is a natural class of functions that arise in learning theory, especially when using Fourier techniques. The efficient learnability of TOP is an intriguing question both because TOP is a much more expressive class than is DNF and because the membership query algorithm for efficiently learning DNF, the Harmonic Sieve [Jac97], can also efficiently learn TOP. Hence, TOP is a good candidate for a concept class to learn after establishing the learnability of DNF. The algorithm is a seemingly counter-intuitive search method for identifying large Fourier coefficients.

Finally, we introduce a more general random walk model based on p-biased distributions. We briefly show why Fourier-based methods cannot learn TOP in this model. On a positive note, we generalize existing efficient learning results for DNF and juntas, showing that these classes are also efficiently learnable (juntas agnostically) in the product random walk model.

Also, we consider another learning model called the Noise Sensitivity model. In this model, random examples come in correlated pairs. The first comes from the standard random distribution, and the second is a noisy version of the first. It is clear that this model is at least as powerful as the standard PAC model, because we could ignore the second example in each pair. [BMOS05] also show that DNF expressions are learnable in this model; we extend this result (and the model) to product distributions. Further, we extend a membership query algorithm of [GKK08] for agnostically learning functions of few variables to product distribution Noise Sensitivity models.

2 Preliminaries

All of our results are for versions of the well-known PAC model: we are given some class of functions $\mathcal{F}$ equipped with a size measure (such as the class of all Boolean functions and size measure the TOP size); a target function $f : \{0,1\}^n \rightarrow \{-1,1\}$ is drawn from $\mathcal{F}$ adversarially; examples $\langle x, f(x)\rangle$ of this function are drawn from an oracle to be specified; given ϵ, $\delta > 0$, the goal is to with probability at least $1-\delta$ (over randomness in the oracle and learning algorithm) produce a hypothesis $h : \{0,1\}^n \rightarrow \{-1,1\}$ such that $\mathbf{Pr}[f(x) \neq h(x)] \leq \epsilon$, where the probability is with respect to some distribution D (that is related to the oracle; see below) over instances x. If this goal can be achieved for every $f \in \mathcal{F}$ and every ϵ and δ, then $\mathcal{F}$ is said to be *learnable* (from the given oracle). If ϵ is of the form $1/2-\gamma$ for some specified γ (which is typically a function of other parameters), we say that h is a γ-*weak approximation to* f, and if $\mathcal{F}$ is learnable for all ϵ so restricted then $\mathcal{F}$ is γ-*weakly learnable*. The run time dependence on δ for our algorithms, like virtually all PAC algorithms, is logarithmic. For simplicity of exposition, we will assume that δ is such a small constant, e.g. 2^{-50}, that our algorithms can rationally be relied on as if they were deterministic. The δ parameter is then subsumed in $O()$ notation and will henceforth be ignored.

Our TOP result makes use of a uniform random walk oracle. In the first example $\langle x, f(x)\rangle$ returned by this oracle, the instance x is drawn uniformly at random from $\{0,1\}^n$. Thereafter, the oracle chooses the instance to return in each example by beginning with the previous example's instance x, choosing $i \in [1..n]$ uniformly at random, and replacing bit x_i with a bit value chosen uniformly at random (i.e., chosen by a fair coin flip). This is known as *updating* x. (A more natural definition involves flipping rather than updating bits, but as [BMOS05] point out, the definitions are essentially interchangeable for uniform random walks, and the updating oracle turns out to be more convenient for our proofs.) The distribution $\mathcal{D}$ used to measure the accuracy of hypothesis h when learning from a uniform random walk oracle is the uniform distribution over $\{0,1\}^n$.

We call PAC learning in this setting the *uniform random walk model*, denoted by RW.

A class $\mathcal{F}$ is *agnostically learnable* if there is an algorithm that, given an oracle for an arbitrary Boolean function f and any $\epsilon > 0$, returns a function $h \in \mathcal{F}$ such that $\mathbf{Pr}[h \neq f] \leq \min_{g \in \mathcal{F}} \mathbf{Pr}[g \neq f] + O(\epsilon)$.

We will make extensive use of discrete Fourier analysis. For any $a \in \{0,1\}^n$, define $\chi_a : \{0,1\}^n \to \{-1,1\}$ as $\chi_a(x) = (-1)^{a \cdot x}$, where $a \cdot x$ represents the dot product of vectors a and x. $\chi_a(x)$ returns the parity (1 is even parity, -1 is odd) of the subset of components of x indexed by the 1's in a. For any function $g : \{0,1\}^n \to \mathbb{R}$ and any $a \in \{0,1\}^n$, let $\hat{g}(a) \equiv E[g(x)\chi_a(x)]$, where the expectation is over uniform choice of x from $\{0,1\}^n$. We call $\hat{g}(a)$ a *Fourier coefficient* of g. If g is Boolean (which in this paper means that g has codomain $\{-1,1\}$), then it is easily seen that $\hat{g}(a) = 1 - 2\mathbf{Pr}[\chi_a \neq g]$, where again the probability is uniform over the Boolean cube. Thus, if $|\hat{g}(a)| \geq \gamma$, then either $\mathbf{Pr}[\chi_a \neq g] \leq 1/2 - \gamma/2$ or $\mathbf{Pr}[-\chi_a \neq g] \leq 1/2 - \gamma/2$, which implies that either χ_a or $-\chi_a$ is a $(\gamma/2)$-weak approximation to g. Our primary algorithms, which focus on finding *heavy* Fourier coefficients—finding a such that $|\hat{g}(a)| \geq \gamma$ for some threshold value $\gamma > 0$ that will be clear from context—therefore can be viewed as weak learning algorithms. We use $L_2(\hat{g})$ to denote $\sum_a \hat{g}^2(a)$ and $L_\infty(g)$ to represent $\max_x |g(x)|$. By Parseval's identify, if g is Boolean, $L_2(\hat{g}) = 1$. The notation $x \oplus y$ represents the bitwise exclusive-OR of the binary vectors x and y (assumed to be of the same length). Sometimes, instead of $\hat{f}(a)$, we write $\hat{f}(A)$, where $A \subset \{1, \ldots, n\}$ is the set of coordinates at which a is 1.

We will call a string in $\{0,1,*\}^n$ a *restriction* (we use such strings to represent certain subsets, which can be viewed as restrictions of larger sets). For example, $0**1*$ represents the restriction and could be viewed as representing the set of all 5-bit strings where the first bit is 0 and the fourth is 1. The *bits of a restriction* are those symbols that are not $*$'s. Note that an n-bit string is considered to be a restriction. For $1 \leq i \leq n$ and $b \in \{0,1,*\}$, we use the notation $\alpha + (i,b)$ to represent the restriction α' that is identical to α except that its ith symbol α'_i is b. We say that a restriction a is *consistent* with a restriction α if and only if for all i such that $\alpha_i \neq *$ it is the case that $a_i = \alpha_i$. A Fourier coefficient $\hat{f}(c)$ is consistent with restriction a if c is consistent with a. A sum over $a \in \alpha$ represents a sum over the set of all bit-strings a consistent with α.

Our algorithms involve approximating various mean values using sample means. As with the PAC δ parameter, although the required sample size for such estimates is logarithmically dependent on the confidence we wish to have in the accuracy of the approximation, we will treat this dependence as a constant.

3 Finding a Heavy Parity

In this section we present and analyze our core algorithm, which given a threshold value θ and a uniform random walk oracle for a function $g : \{0,1\}^n \to \mathbb{R}$ finds the index a of a Fourier coefficient such that $|\hat{g}(a)|$ (nearly) exceeds θ, if such a coefficient exists. The algorithm's time bound is exponential in $\log(L_2(\hat{g})/\theta)$ but is otherwise polynomial. In later sections we use this algorithm to obtain other random walk results, agnostically learning parity (in polynomial time if the accuracy ϵ is constant) and learning TOP (in polynomial time for TOPs of constant size and given constant ϵ).

We will use $\mathbf{FC}(g, a, \tau)$ to denote an algorithm that uses a uniform random set of examples to estimate the Fourier coefficient $\hat{g}(a) = \mathbf{E}_{\boldsymbol{x}}[g(\boldsymbol{x})\chi_a(\boldsymbol{x})]$ within an additive error τ of the true value. By Hoeffding [Hoe63], this is easy to implement with a random walk oracle (walking $O(n \log n)$ steps from any point gives a uniform random input) in time polynomial in n and $1/\tau$.

We will use $\mathbf{SSF}(g, \alpha, \tau)$ to represent an algorithm that returns a value σ such that $|\sum_{a \in \alpha} \hat{g}^2(a) - \sigma| \leq \tau$. If α contains exactly k bits (or equivalently, k non-$*$ entries), then it follows from the analysis of Kushilevitz and Mansour [KM93] that $\sum_{a \in \alpha} \hat{g}^2(a) = \mathbf{E}_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}}[g(\boldsymbol{x}+\boldsymbol{y})g(\boldsymbol{x}+\boldsymbol{z})\chi_d(\boldsymbol{y} \oplus \boldsymbol{z})]$, where the expectation is over uniform choice of $\boldsymbol{x}$ from $\{0,1\}^{n-k}$ and $\boldsymbol{y}$ and $\boldsymbol{z}$ from $\{0,1\}^k$, where we use $\boldsymbol{x}+\boldsymbol{y}$ to mean the n-bit string formed by interleaving in the obvious way bits from $\boldsymbol{x}$ in positions where there are $*$'s in α with bits from $\boldsymbol{y}$ in non-$*$ positions, and where d is the k-bit string obtained by removing all of the $*$'s from α. Thus, **SSF** can be implemented given a membership oracle for g by randomly sampling the random variable $g(\boldsymbol{x}+\boldsymbol{y})g(\boldsymbol{x}+\boldsymbol{z})\chi_d(\boldsymbol{y} \oplus \boldsymbol{z})$ and computing the sample mean. By the Hoeffding bound [Hoe63], a sample of size polynomial in $1/\tau$ suffices, and therefore the time bound for **SSF** is polynomial in n and $1/\tau$.

It is also easy to implement **SSF** using a random walk oracle, but the time bound now depends on k, the number of bits in α. Specifically, to obtain a single sample from the random variable $g(\boldsymbol{x}+\boldsymbol{y})g(\boldsymbol{x}+\boldsymbol{z})\chi_d(\boldsymbol{y} \oplus \boldsymbol{z})$, we begin by walking $O(n \log n)$ steps to ensure independence with previous draws. We then draw from the oracle until we observe a sequence of consecutive steps that collectively update all of and only the k bits corresponding to non-$*$ characters in α (some of these bits may be updated more than once). The inputs before and after this sequence of steps give us the $\boldsymbol{x}+\boldsymbol{y}$ and $\boldsymbol{x}+\boldsymbol{z}$ values we need in order to obtain a sample from the random variable. It is not hard to see that the expected time to obtain such a sequence is $O(n^k)$, giving us an algorithm for **SSF** in RW having run time polynomial in $1/\tau$ and n^k.

Given **SSF** and **FC**, we next present an algorithm **PT** that finds a parity function weakly approximating the target, if such a function exists. See Algorithms 2 (the primary portion of the algorithm, a recursive "helper" function) and 1 (the entry point). We give some intuition for the algorithm before analyzing it formally.

Consider two calls to **SSF**, one with $\alpha = 0*^{n-1}$ and one with $\alpha = 1*^{n-1}$ (and assume for this informal description that the value returned by **SSF** is exact rather than an approximation). Since these calls return sums of squares of coefficients and since every coefficient is included in one or the other of these sums, if both of the returned values are less than θ^2 then there can be no coefficient of magnitude θ, and the algorithm returns this information. On the other hand, if exactly one of the returned values—say from the call with $\alpha = 1*^{n-1}$—returns an above-threshold value, then we know that the index of any heavy coefficient must begin with a 1. In this case, we could record this fact (this is the purpose of the a parameter of **PTH**) and next attempt to fix a value for the second bit of all heavy indices by calling **SSF** with $\alpha = *0*^{n-2}$ and $\alpha = *1*^{n-2}$. If this similarly returns a single above-threshold value, and if in fact we have similar results for all n variables, then there will remain a single candidate heavy Fourier coefficient with index a; all other possibilities will have been eliminated. We can then use **FC** to determine whether or not $\hat{f}(a)$ is heavy, and we are done.

But what if, say, the calls to **SSF** with $\alpha = *0*^{n-2}$ and $\alpha = *1*^{n-2}$ both return a value greater than θ^2? Then the second bit of a heavy Fourier coefficient index might be either 0 or 1. So, we tentatively choose one value—say, 0—for this bit and temporarily remember this bit in the a parameter of a recursive call to **PTH**. We also record this bit value in the recursive call's α parameter; as long as we are exploring the possibility that some heavy coefficient has an index with 0 as its second bit (that is, as long as the recursive call is active), this α parameter will "remember" to include 0 as the second bit in the α parameter of calls to **SSF**. For instance, when we next attempt to fix a value for the third bit, we will call **SSF** with the restrictions $\alpha = *00*^{n-3}$ and $\alpha = *01*^{n-3}$. If both of these return below-threshold values, we can conclude that no heavy coefficient has a 0 in the second bit of its index, and hence that this bit must be 1 in any heavy coefficient's index. We will then retract our tentative choice of 0 for the second bit (by returning from the recursive call to **PTH**) and replace the second bit of a with a 1. This also means that we no longer need to restrict bit 2 of α in subsequent calls to **SSF** (also retracted by the recursive return). That is, we can proceed to examine the third bit using the restrictions $\alpha = **0*^{n-3}$ and $\alpha = **1*^{n-3}$, because we know the values that the first two bits must have in any heavy index.

One possibly counter-intuitive aspect of the algorithm is that, when choosing which of two bits to tentatively select, it chooses the one corresponding to a *smaller* sum of squares of coefficients. As the formal analysis will show, this approach limits the number of non-$*$ values in the α parameter in every call to **SSF**, which in turn allows us to achieve a quasipolynomial time bound. Intuitively, this approach makes sense if we think of the algorithm as seeking to eliminate non-heavy coefficients rather than seeking to find heavy ones.

Algorithm 1 : PT

```
Input: $\theta, \epsilon > 0$
Output: thrown value, if any, is $a \in \{0,1\}^n$ such that
  $|\hat{g}(a)| \geq \theta - \epsilon/2$; normal return guarantees that there
  is no $a$ such that $|\hat{g}(a)| \geq \theta + \epsilon/2$.

  if $\theta \leq \epsilon/2$ then {any coefficient will do}
    throw $0^n$
  else
    PTH($*^n, *^n, 1, \theta, \epsilon$)
    return
  end if
```

Algorithm 2 : PTH

```
Input: $\alpha, a \in \{0,1,*\}^n$; $1 \leq i \leq n$; $\theta > 0$; $0 < \epsilon <$
  $2\theta$
Output: thrown value, if any, is $c \in \{0,1\}^n$ such that
  $|\hat{g}(c)| \geq \theta - \epsilon/2$; normal return guarantees that there
  is no $c$ consistent with $a$ such that $|\hat{g}(c)| \geq \theta + \epsilon/2$.

  while $i \leq n$ do
    $s_0 \leftarrow$ SSF($g, \alpha + (i,0), \epsilon^2/16$)
    $s_1 \leftarrow$ SSF($g, \alpha + (i,1), \epsilon^2/16$)
    if $s_0 < \theta^2$ and $s_1 < \theta^2$ then
      return
    else if $s_0 < \theta^2$ then
      $a \leftarrow a + (i,1)$
    else if $s_1 < \theta^2$ then
      $a \leftarrow a + (i,0)$
    else
      $b \leftarrow \text{argmin}_b(s_b)$
      PTH($\alpha + (i,b), a + (i,b), i+1, \theta, \epsilon$)
      $a \leftarrow a + (i, 1-b)$
    end if
    $i \leftarrow i + 1$
  end while
  if $|$FC($g, a, \epsilon/2$)$| \geq \theta$ then
    throw $a$
  else
    return
  end if
```

We prove the correctness of our algorithm in two lemmas.

Lemma 1 *Suppose c is thrown. Then $|\hat{g}(c)| \geq \theta - \epsilon/2$.*

Proof: If $\theta \leq \epsilon/2$, then 0^n is thrown immediately in **PT**, and we are done as $\theta - \epsilon/2 \leq 0$. Otherwise, a value is thrown near the end of **PTH**. Before throwing c, we must have that $|\mathbf{FC}(g, c, \epsilon/2)| \geq \theta$, which implies $|\hat{g}(c)| \geq \theta - \epsilon/2$. ■

We now need to show that a normal return implies that all the Fourier coefficients have low magnitude. We show an equivalent statement:

Lemma 2 *If there exists $c' \in \{0,1\}^n$ such that $|\hat{g}(c')| \geq \theta + \epsilon/2$, then the algorithm throws some value c.*

Proof: We assume that $\theta > \epsilon/2$, otherwise **PT** throws immediately. It suffices to consider all possibilities that make iterations of **PTH** return. Suppose **PTH** is called with a restriction a consistent with c'. When we compute s_b for $a + (i, b)$ also consistent with c', we get that s_b is at least

$$\begin{aligned}(\theta + \epsilon/2)^2 - \epsilon^2/16 &\geq \theta^2 + 2\theta\epsilon + \epsilon^2/4 - \epsilon^2/16 \\ &\geq \theta^2 + 2(\epsilon/2)\epsilon + \epsilon^2/4 - \epsilon^2/16 \\ &\geq \theta^2.\end{aligned}$$

So the algorithm won't return. Instead, it either invokes a new iteration of **PTH** with restriction $a + (i, b)$ or simply continues the current iteration with restriction $a + (i, b)$. Either way, the effect is the same: an iteration of **PTH** using a restriction a having one more bit consistent with c' is performed. If no other values are thrown, eventually we will call $\mathbf{FC}(g, c', \epsilon/2)$, which will return a value at least θ because $|\hat{g}(c')| \geq \theta + \epsilon/2$, and c' will be thrown, completing the proof. ■

What remains is to analyze the run time. First, notice that each recursive call receives an argument α that contains exactly as many bits k as the level of the recursion. Furthermore, every recursive call has a different α argument, and in fact if $S(\alpha)$ represents the set of locations of bits in restriction α, and if α_1 and α_2 are the values of the α arguments in any two distinct calls to **PTH**, then $S(\alpha_1) \neq S(\alpha_2)$. Therefore, if the maximum level of recursion is k, then there are $O(n^k)$ many recursive calls. Furthermore, each of these calls runs (excluding the time for any recursive calls it makes) in time polynomial in n^k and $1/\epsilon$, where the bulk of the time is spent implementing **SSF** with a random walk oracle. Therefore, the run time is similarly bounded, and our task is reduced to finding a bound on k.

As a first approximation to bounding k, note that if each call to **SSF** returned exactly the requested sum of squares, then the maximum depth of the recursion would clearly be $O(\log(L_2(\hat{g})/\theta))$, since the recursion would always be performed on the smaller of two halves of a larger sum of squares of coefficients and recursion would never be performed on a sum of squares less than θ^2.

When using the actual **SSF**, the first thing to note is that we might recurse on a restriction such that the true sum of squares for that restriction (call it t_b) could be as small as $\theta^2 - \epsilon^2/16$ (which, since $\theta \geq \epsilon/2$, is at least $3/4$ of θ^2). Furthermore, it could be that the restriction recursed on actually corresponds to the larger of the two sums of squares; in the worst case, $t_b = t_{1-b} + \epsilon^2/8$. Combining these observations with some straightforward analysis gives us that for $\theta \geq \epsilon/2$, the maximum value of $t_b/(t_b + t_{1-b})$ is $3/4$. Thus, even when employing an imperfect **SSF**, the algorithm recurses on a sum of squares that is reduced by a constant factor and recurses down to a value nearly the same as that in the perfect **SSF** case. So we still have that the maximum depth k of the recursion is $O(\log(L_2(\hat{g})/\theta))$.

We have now shown the following:

Lemma 3 *Given any $g : \{0,1\}^n \to \mathbb{R}$ and any positive θ and ϵ, **PT**(θ, ϵ) runs in time polynomial in $1/\epsilon$ and $n^{\log(L_2(\hat{g})/\theta)}$ and produces the output claimed in Algorithm 1.*

4 TOP Learning

The Harmonic Sieve [Jac97] learns—from a membership oracle and with accuracy measured respect to the uniform distribution—Threshold of Parity (TOP) functions in time polynomial in their size s as well as in n and $1/\epsilon$ (recall that the TOP-size of a function f is the minimum weight representation of that function as a threshold of an integer-weighted sum of parity functions). The algorithm's proof of correctness is based in part on the following fact [Jac97]:

Fact 4 *For every f of TOP-size s and every distribution D over $\{0,1\}^n$, there exists a parity χ_a such that*

$$|E_D[f\chi_a]| \geq \frac{1}{2s+1}$$

Defining $g \equiv 2^n f D$, it follows that for any TOP of size s, there exists a such that $|\hat{g}(a)| \geq 1/(2s+1)$. The original Sieve uses its membership queries in two ways: 1) to obtain uniform random examples for purposes of estimating hypothesis accuracy; 2) to implement **KM** in order to locate heavy a's for D's (and hence g's) defined by a certain boosting algorithm. The original Sieve boosting algorithm (and some other boosting algorithms which could be used instead and give asymptotically better bounds; see, e.g., [KS99]) has the property that, when learning with respect to uniform, the D's it defines all have the property that $2^n \max_x D(x)$ is polynomial in $1/\epsilon$. It follows that any g defined using such a D has $L_2(\hat{g})$ that is also polynomial in $1/\epsilon$ [Jac97].

It is a simple matter, then, to replace the membership-query **KM** algorithm in the Sieve with the random-walk **PT** algorithm. Since $1/\theta$ is $O(s)$ (we can assume s is known, since a simple binary search technique can be used otherwise), in the context of TOP learning **PT** will run in time polynomial in $n^{\log s/\epsilon}$. And as noted earlier, the uniform random examples required by the Sieve can be obtained using a uniform random walk oracle with $O(n \log n)$ run-time cost per example. We therefore obtain the following:

Theorem 5 *TOP is learnable in the uniform random walk model in time* $n^{O(\log(s/\epsilon))}$.

When employing certain boosting algorithms, the Harmonic Sieve produces a TOP as its hypothesis. Thus, TOP is actually *properly* learnable in RW in the stated time.

5 Agnostic Parity Learning

It is straightforward to employ **PT** to agnostically learn parity in RW. First, some analysis. Let $o = \operatorname{argmax}_a |\hat{f}(a)|$; that is, o represents the index of the optimal approximating parity (or its negation, but we will assume without loss of generality that $\hat{f}(o) > 0$). By the definition of agnostic learning, we want an $a \in \{0,1\}^n$ such that $\mathbf{Pr}[\chi_a \neq f] \leq \mathbf{Pr}[\chi_o \neq f] + \epsilon$ (we are also assuming, again without loss of generality, that $\hat{f}(a) > 0$). Since for any a, $\mathbf{Pr}[\chi_a \neq f] = (1 - \hat{f}(a))/2$, we can achieve this goal if we find an a such that $|\hat{f}(o)| - |\hat{f}(a)| \leq 2\epsilon$.

Now we describe the agnostic algorithm, which is given a single parameter $\epsilon > 0$. First, we run **PT** with $\theta = 1$ (by Parseval, this is the value of $L_2(\hat{f})$ when f is Boolean). If it throws a vector a, then we know that $|\hat{f}(a)| \geq 1 - \epsilon/2$ and, since the maximum possible value of $|\hat{f}(o)|$ is 1, that χ_a (or its negation, if $\hat{f}(a)$ is negative) is a suitable hypothesis for purposes of agnostic learning. If instead **PT** returns rather than throwing a value, we will iteratively call **PT**, each time dividing θ by 2. This process is guaranteed to terminate, since **PT** will throw 0^n if θ becomes smaller than $\epsilon/2$. If the iteration terminates because **PT** is called with $\theta \leq \epsilon/2$, then we know that there was no extra heavy coefficient for the next larger θ, which could be at most ϵ. This means that $|\hat{f}(o)| \leq 3\epsilon/2$, which in turn means that χ_{0^n} or its negation is a suitable agnostic hypothesis.

If **PT** throws a value using a value of θ that is neither 1 nor less than $\epsilon/2$, we employ binary search, using the successful θ as the initial lower bound of the search and twice this value as the initial upper bound. The search continues until these bounds are within ϵ of one another. Call the lower and upper bounds θ_l and θ_u; we know $\theta_u - theta_l \leq \epsilon$. From Lemma 1 we know that the thrown value c is such that $|\hat{f}(c)| \geq \theta_l - \epsilon/2$. From Lemma 2 we know that $|\hat{f}(o)| \leq \theta_u + \epsilon/2 \leq \theta_l + 3\epsilon/2$. Since $\theta_l > \epsilon/2$ in this case, we have that $|\hat{f}(c) - \hat{f}(o)| \leq 2\epsilon$ as desired.

Because the θ multiplicative step size begins at $1/2$, is reduced by a factor of 2 at each iteration, and ends when it reaches $O(\epsilon)$, this procedure calls **PT** $O(\log 1/\epsilon)$ times. Furthermore, since all calls to **PTH** have $\theta \geq \epsilon/2$, every call to **PT** runs in time polynomial in $n^{\log 1/\epsilon}$. We therefore have the following:

Theorem 6 *Parity is agnostically learnable in the uniform random walk model in time* $n^{O(\log 1/\epsilon)}$.

6 Product Random Walk Model

We next turn to results for a generalization of the uniform random walk model to certain non-uniform walks. In this section, we give definitions and some preliminary observations. Subsequent sections generalize existing learning results to this model.

6.1 Properties of p-biased Distributions

A p-biased distribution over $\{0,1\}^n$ is a distribution where $\mathbf{Pr}[x_i = 0] = p$ for each index i. Define $q = 1-p$. Further, all bits are independent of each other, so a p-biased distribution $\mathcal{D}$ assigns probability weight

$$(\Pi_{i:x_i=0}p)\,(\Pi_{i:x_i=1}q)$$

to a string x.

Given a p-biased distribution $\mathcal{D}$, we define the inner product of two functions f and g to be $\mathbf{E}_{\boldsymbol{x}\sim\mathcal{D}}[f(\boldsymbol{x})g(\boldsymbol{x})]$.

Given a string x and an index i, define

$$z_i(x) = \begin{cases} \sqrt{q/p} & \text{if } x_i = 0, \\ -\sqrt{p/q} & \text{if } x_i = 1 \end{cases}$$

and for a set $S \subseteq [n]$, define $\phi_S = \Pi_{i \in S} z_i$. It is shown in [Bah61] that the 2^n functions ϕ_S form an orthonormal basis on real valued functions on $\{0,1\}^n$ with respect to the inner product defined in this section. We define the p-biased Fourier coefficient of $f : \{0,1\}^n \to \mathbb{R}$ as

$$\widetilde{f}(S) = \mathop{\mathbf{E}}_{\boldsymbol{x}\sim\mathcal{D}}[f(S)\phi_S].$$

Notice that when $p = 1/2$, we recover the uniform distribution. Indeed, many theorems from Fourier analysis with respect to the uniform distribution are true in p-biased distributions, such as Parseval's identity:

$$\mathop{\mathbf{E}}_{\boldsymbol{x}\sim\mathcal{D}}[f(\boldsymbol{x})^2] = \sum_S \widetilde{f}(S)^2.$$

With the standard Fourier coefficients, if $\hat{f}(a)$ is heavy then the corresponding parity χ_a (or its negation) is a weak approximator to f. But ϕ_S is not a Boolean function, so it cannot be directly used as a hypothesis if $\widetilde{f}(S)$ is heavy. However, [Jac97] shows how to produce a weak approximating Boolean function from a heavy $\widetilde{f}(S)$ as long as $|S|$ is logarithmic, which will be the case for our results.

6.2 Product Random Walk Oracle

Given the definition of p-biased distributions, it is natural to define a product random walk oracle as follows. The initial $\boldsymbol{x}$ will be chosen at random according to the p-biased distribution. After this, given that the previous call to the oracle returned $\langle \boldsymbol{x}, f(\boldsymbol{x}) \rangle$, the next call will uniformly at random choose a coordinate i and update bit i of $\boldsymbol{x}$ with a new random p-biased bit. The resulting $\boldsymbol{x}'$ is returned along with its label. We call PAC learning from such an oracle and measuring the accuracy of the hypothesis against the associated p-biased distribution the *Product Random Walk* model, or pRW for short.

As an alternative definition in a more general framework, we could define our pRW oracle using the Metropolis-Hastings algorithm [Has70], a well-studied Markov chain Monte Carlo method for generating samples from a distribution. Then, we could alternately define our random walk oracle for a product distribution D as the oracle that generates samples as the Metropolis-Hastings does in generating new states converging towards a random draw from D. In other words, as the Metropolis-Hastings algorithm generates $\boldsymbol{x}^1, \boldsymbol{x}^2, \boldsymbol{x}^3, \ldots$, our random walk oracle returns $\langle \boldsymbol{x}^1, f(\boldsymbol{x}^1) \rangle, \langle \boldsymbol{x}^2, f(\boldsymbol{x}^2) \rangle, \langle \boldsymbol{x}^3, f(\boldsymbol{x}^3) \rangle$. We will see that for very simple parameter choices in the Metropolis-Hastings algorithm, the two definitions are equivalent. We take this as evidence that our definition of the pRW oracle is indeed natural.

We will denote the general version of the Metropolis-Hastings algorithm **MH**. Let μ be a probability distribution we want to generate samples from. The **MH** algorithm, given a state x^t and a probability distribution $Q(\cdot; x^t)$ depending on x^t, proceeds as shown in Algorithm 3.

Algorithm 3 : MH

```
Input: $x^t$, a probability distribution $Q(\cdot; x^t)$
Output: $x^{t+1}$
  Draw $x' \sim Q(x'; x^t)$.
  if $\mu(x') \geq \mu(x^t)$ then
    return $x'$.
  else
    With probability $\mu(x')/\mu(x^t)$, return $x'$.
    return $x^t$
  end if
```

In our case $\mu(x)$ is a p-biased distribution, which we will denote $\mu_p(x)$. Let $(x^t)^{(i)}$ denote x^t with the ith bit flipped. We assume without loss of generality that $p \leq \frac{1}{2}$. We will define the distribution $Q(\cdot; x^t)$ is as follows: x^t itself has probability mass p, and for all $1 \leq i \leq n$, $(x^t)^{(i)}$ has probability mass q/n. In the **MH** algorithm described above, $x^{t+1} \neq x^t$ only if $x' = (x^t)^{(i)}$ for some i and $x^{t+1} = x'$.

We claim that **MH** as shown gives samples with the same distribution as samples from the pRW oracle. For any fixed i, $x' = (x^t)^{(i)}$ with probability q/n. Let us assume $x' = (x^t)^{(i)}$. If $x_i^t = 1$, then $x^{t+1} = x'$, as $\mu_p((x_t)^{(i)}) \geq \mu_p(x^t)$ when $p \leq \frac{1}{2}$. If $x_i^t = 0$, then $x^{t+1} = x'$ with probability $\mu_p((x^t)^{(i)})/\mu_p(x^t) = p/q$. Since drawing from $Q(\cdot; x^t)$ and accepting a proposal are independent events, we note that for all $1 \leq i \leq n$,

$$\mathbf{Pr}[x^{t+1} = (x^t)^{(i)}] = \begin{cases} q/n & \text{if } x_i^t = 1 \\ p/n & \text{if } x_i^t = 0 \end{cases}$$

and all of the other remaining probability mass for the value of x^{t+1} lies on the event $x^{t+1} = x^t$.

Our pRW oracle proceeds by uniformly picking a coordinate and updating it with a p-biased bit. The probability any coordinate is chosen is $1/n$. Upon updating a bit, if that bit is 0 it is flipped with probability q, and if it is 1 it is flipped with probability p. Since the choice of flipping and choosing a coordinate are independent, we see that **MH** and the pRW oracle sample from the same distribution.

6.3 The Noise Sensitivity Model

We will actually prove our product random walk results in a weaker learning model. The *product ρ-noise sensitivity model* is defined similarly to the ρ-noise sensitivity model introduced by [BMOS05]. Specifically, each call to the oracle for this model will return a 5-tuple $\langle \boldsymbol{x}, f(\boldsymbol{x}), \boldsymbol{y}, f(\boldsymbol{y}), S \rangle$, where $\boldsymbol{x}$ is generated at random from a fixed p-biased distribution and $\boldsymbol{y}$ is constructed from $\boldsymbol{x}$ as follows: for each of the n bits of $\boldsymbol{x}$, independently and with probability $1-\rho$ update the bit by choosing a new p-biased value for it (if a bit of $\boldsymbol{x}$ is not updated, it is unchanged in $\boldsymbol{y}$). S is the set of coordinates of the bits updated. The accuracy of the hypothesis produced will be measured using the underlying p-biased distribution. We refer to this model as pρNS for short. The following lemma is an immediate generalization of Proposition 10 of [BMOS05].

Lemma 7 *For any $0 < \rho < 1$, any algorithm in the pρNS model can be simulated in the pRW model at the cost of a $O(n \log n)$ factor in run time.*

7 Positive and Negative Results in pRW

Having introduced the pRW model, we would like to transfer our uniform random walk result for agnostically learning parity to the pRW model. Unfortunately, this is not possible using Fourier methods, due to the "smearing" of the Fourier spectrum of parity under p-biased distributions that we show next. Here, we think of parity as a function from $\{-1,1\}^n$ into $\{-1,1\}$, where p is the probability that any bit is 1, and q is the probability that any bit is -1.

Claim 8 *Let $\chi_{[n]}$ be the parity function on n bits. Then the p-biased Fourier coefficient $\widetilde{f}(S)$ of $\chi_{[n]}$ is*

$$(p-q)^{n-|S|}(2\sqrt{pq})^{|S|}.$$

Proof: Assume we are working under the p-biased distribution; all expectations here are with respect to this distribution. Then

$$\begin{aligned}\widetilde{f}(S) &= \mathbf{E}[\chi_{[n]}\phi_S] \\ &= \mathbf{E}[\Pi_{i\in[n]}\chi_i\Pi_{i\in S}\phi_i] \\ &= \mathbf{E}[\Pi_{i\in S}\chi_i\phi_i\Pi_{i\notin S}\chi_i] \\ &= \Pi_{i\in S}\mathbf{E}[\chi_i\phi_i]\Pi_{i\notin S}\mathbf{E}[\chi_i].\end{aligned}$$

It is straightforward to check that $\mathbf{E}[\chi_i] = p-q$ and $\mathbf{E}[\chi_i\phi_i] = -1(-\sqrt{p/q})q + 1(\sqrt{q/p})p = 2\sqrt{pq}$, proving the claim. ■

When p is bounded away from $\frac{1}{2}$ by a constant, both $p-q$ and $2\sqrt{pq}$ are bounded away from 1 by a constant, so every $\widetilde{f}(S)$ is exponentially small. Thus, our agnostic parity algorithm, which relies on finding heavy Fourier coefficients, cannot succeed in the product random walk model (for most p).

Despite this negative beginning, we are able to obtain two positive results for pRW. We begin by observing that many of the algorithms used in learning under the uniform distribution using Fourier analysis techniques work by estimating certain (possibly weighted) sums of squares of Fourier coefficients. Often, the algorithm efficiently estimates these sums by estimating expectations, and the correctness of these expectations depends only on the orthogonality of the χ_S functions. When only orthogonality is used, it is straightforward to extend the algorithm to product distributions. However, the structural theorems used to prove correctness may not follow, as we have just seen in the case of parity. In addition, the complexity of the algorithm may increase when extending to product distributions.

We will show two positive learning results in pRW (via results in $p\rho NS$):

Theorem 9 *DNF formulas can be efficiently learned in the pRW model, where the efficiency depends on the parameters of the product distribution.*

Theorem 10 *Juntas can be efficiently agnostically learned in the pRW model, where the efficiency depends on the parameters of the product distribution.*

The proofs are given in the next two sections.

8 Product Learning of DNF

In [BMOS05], the authors proceed by estimating certain weighted sums of squares of Fourier coefficients. These weighted sums correspond to noise sensitivity estimates. In fact, most of these proofs (e.g., Lemma 6, Theorem 7, and Theorem 11 of their paper) use only the orthogonality of the χ_S basis functions to show that the expectations that their algorithm estimates yield the appropriate weighted sum of squares of Fourier coefficients. These proofs can be generalized to $p\rho NS$ almost immediately by replacing all occurrences of $\hat{f}$ with $\widetilde{f}$. The generalization of their Claim 12 is not as immediate and is therefore given here.

We start with some definitions from [BMOS05]. Given a $p\rho NS$ oracle and a set $I \subseteq [n]$, let $D_\alpha^{(I)}$ be the distribution on pairs $(\boldsymbol{x},\boldsymbol{y})$ as follows: $\boldsymbol{x}$ is a random string from a p-biased distribution, and $\boldsymbol{y}$ is formed from $\boldsymbol{x}$ by updating each bit in I with probability 1 and updating each bit not in I with probability $1-\alpha$. Using a $p\rho NS$ oracle, we can simulate this distribution. We simply keep drawing $\{\boldsymbol{x}, f(\boldsymbol{x}), \boldsymbol{y}, f(\boldsymbol{y}), S\}$ until we get a 5-tuple with $I \subseteq S$. With high probability, we need at most $\mathrm{poly}((1-\alpha)^{|I|})$ examples until this happens. Then $(\boldsymbol{x},\boldsymbol{y})$ is our desired draw from $D_\alpha^{(I)}$.

Define $\mathcal{T}'(I) = \mathbf{E}_{(\boldsymbol{x},\boldsymbol{y})\sim D_\alpha^{(I)}}[f(\boldsymbol{x})f(\boldsymbol{y})]$. It is easy to see that with a $p\rho NS$ oracle we can estimate $\mathcal{T}'(I)$. Now we prove the analog of Claim 12 in the product setting. Our proof will demonstrate that we only use orthonormality of the ϕ_S functions to achieve this result.

Claim 11 $\mathcal{T}'(I) = \sum_{S\bigcap I=\emptyset}\alpha^{|S|}\widetilde{f}(S)^2$

Proof: All expectations in this proof are over $(\boldsymbol{x},\boldsymbol{y}) \sim D_\alpha^{(I)}$.

$$\begin{aligned}&\mathbf{E}[f(\boldsymbol{x})f(\boldsymbol{y})] \\ =\ &\mathbf{E}[(\sum_S \widetilde{f}(S)\phi_S(\boldsymbol{x}))(\sum_T \widetilde{f}(T)\phi_T(\boldsymbol{y}))] \\ =\ &\sum_S\sum_T \widetilde{f}(S)\widetilde{f}(T)\mathbf{E}[\phi_S(\boldsymbol{x})\phi_T(\boldsymbol{y})] \\ =\ &\sum_S\sum_T \widetilde{f}(S)\widetilde{f}(T)\mathbf{E}[\Pi_{i\in S}z_i(\boldsymbol{x})\Pi_{j\in T}z_j(\boldsymbol{y})].\end{aligned}$$

Because we are working over product distributions, $z_i(\boldsymbol{x})$ and $z_j(\boldsymbol{x})$ are independent when $i \neq j$, and $\boldsymbol{x}$ can be replaced with $\boldsymbol{y}$ in either or both cases. Notice that if $S\setminus T$

or $T \setminus S$ is nonempty, the expectation is 0. We will assume without loss of generality that $T \setminus S$ is nonempty and $j \in T \setminus S$. Then $z_j(\boldsymbol{y})$ is independent of every other term in the expectation, and $\mathbf{E}[z_j(\boldsymbol{y})]$ is 0. So the only nonzero terms in the sum occur when $S = T$, and the sum becomes

$$\begin{aligned} &\sum_S \widetilde{f}(S)^2 \mathbf{E}[\Pi_{i \in S} z_i(\boldsymbol{x}) z_i(\boldsymbol{y})] \\ = &\sum_S \widetilde{f}(S)^2 \Pi_{i \in S} \mathbf{E}[z_i(\boldsymbol{x}) z_i(\boldsymbol{y})] \end{aligned}$$

using independence again. Note that if $i \in I$, then $z_i(\boldsymbol{x})$ and $z_i(\boldsymbol{y})$ are independent and thus $\mathbf{E}[z_i(\boldsymbol{x}) z_i(\boldsymbol{y})] = \mathbf{E}[z_i(\boldsymbol{x})]\mathbf{E}[z_i(\boldsymbol{y})] = 0$. For $i \notin I$, we see that $\boldsymbol{y}_i$ is updated with probability $1 - \alpha$. The probability distribution is on $z_i(\boldsymbol{x}) z_i(\boldsymbol{y})$ is as follows:

$$z_i(\boldsymbol{x}) z_i(\boldsymbol{y}) = \begin{cases} \frac{q}{p} & \text{with probability } p(\alpha + (1-\alpha)p) \\ \frac{p}{q} & \text{with probability } q(\alpha + (1-\alpha)q) \\ -1 & \text{with probability } 2pq(1-\alpha) \end{cases}$$

The first case corresponds to both bits being 1, the second for both bits being 0, and the third for when the bits are different. The expectation is $q(\alpha + (1-\alpha)p) + p(\alpha + (1-\alpha)q) - 2pq(1-\alpha) = (p+q)\alpha + (1-\alpha)pq + (1-\alpha)pq - 2pq(1-\alpha) = \alpha$. So the sum reduces to

$$\sum_S \widetilde{f}(S)^2 \Pi_{i \in S} \mathbf{E}[z_i(\boldsymbol{x}) z_i(\boldsymbol{y})]] = \sum_{S : S \bigcap I = \emptyset} \alpha^{|S|} \widetilde{f}(S)^2,$$

as claimed. ■

In addition to these generalizations, we need (in using the Bounded Sieve; see [BMOS05] for details on the use of this algorithm) to employ the p-biased version of Jackson's DNF fact [Jac97]. The end result is the following:

Theorem 12 *The class of s-term DNF formulas over n variables can be learned with error ϵ and confidence $1 - \delta$ in the $p\rho NS$ and pRW models in time* $\text{poly}(n, s^{\log(1/p)}, \epsilon^{-\log(1/p)}, 1/\delta)$.

9 Agnostically Learning Juntas

We also claim that we can agnostically learn juntas using random walks. Our proof is very similar to the proof in [GKK08]. In fact, our algorithm is virtually the same. However, rather than working in the model of uniform distribution plus membership queries, we extend this algorithm to product distributions as well as restricting our oracle access to a $p\rho NS$ oracle.

The algorithm from [GKK08] makes use of its membership queries by using KM to identify all Fourier coefficients $\hat{f}(S)$ of heavy magnitude with $|S| \leq k$. Since the size of S is bounded for their purposes, it suffices to use the Bounded Sieve, just as in [BMOS05]. In showing above that DNF is efficiently learnable in the $p\rho NS$ model, we have effectively also shown that the Bounded Sieve works even in the $p\rho NS$ model. The Fourier methods used in their algorithm again only use orthogonality and are not specific to the uniform distribution. Therefore, after translating expectations to the correct product distribution, we can copy their algorithm.

Suppose we wish to agnostically learn a k-junta. We will start by running the Bounded Sieve in $p\rho NS$, stopping at level k and setting the threshold $\theta = \epsilon 2^{-k/2}$. In this fashion, we find all heavy Fourier coefficients S of f with $|S| \leq k$. Let $\mathcal{S}$ be the set of heavy Fourier coefficients found, and set $g(x) = \sum_{S \in \mathcal{S}} \widetilde{g}(S)\phi_S(x)$. Following [GKK08], let R be the set defined where $i \in R$ if and only if $\sum_{i \in S} \widetilde{g}(S)^2 \leq \epsilon^2/k$. Finally, for every set $K \subseteq R$ of size k, estimate the error of $\text{sgn}(g'_K)$ on f, where $g'_K = \sum_{S \subseteq K} \widetilde{g}(S)\phi_S(x)$. The function $\text{sgn}(g'_K)$ of least error over choices of $K \subseteq R$ is our hypothesis.

We will now prove the correctness of this algorithm, as well as a brief runtime analysis. We will prove a sequence of lemmas very similar to those in [GKK08]. First, an analog of their Lemma 13:

Lemma 13 *Given $K \subset [n]$, and $f : \{-1,1\}^n \to \{-1,1\}$, let $f_K(x) = \sum_{S \subseteq K} \widetilde{f}(S)\phi_S(x)$. The K-junta that minimizes* $\text{err}_f(\cdot)$ *is given by* $h_K(x) = \text{sgn}(f_K(x))$. *Also,* $\text{err}_f(h_K) = \frac{1}{2}(1 - \|f_K(X)\|_1)$.

Proof: The proof follows similarly to Lemma 13 of [GKK08]. The major difference is that in the $\{\phi_S\}$ basis, x_i is not a unbiased bit. However, the ϕ_S functions are orthonormal, which is the property used to derive (in the ϕ basis) that $\mathbf{E}[\phi_S(x)|x_K = u] = \mathbf{1}[S \subseteq K]\phi_S(u)$. The rest of the argument follows directly by changing $\hat{f}$ to $\widetilde{f}$. ■

Their Lemma 14 is also easily generalized:

Lemma 14 *Let $g_K : \{-1,1\} \to \mathbb{R}$ be such that $\|f_K(x) - g_K(x)\|_1 < \epsilon$, and let* $h'_K = \text{sgn}(g_K)$. *Then* $\text{err}_f(h'_K) < \text{err}_f(h_K) + 2\epsilon$.

Proof: This lemma follows from straightforward probability and from part of Lemma 13; the only modification is to "average" in a product distribution sense over the choice of u rather than a uniform distribution sense. ■

We note that we need an extra running time factor of $\text{poly}(p^{-k})$, since the nonzero Fourier coefficients could be exponentially small in p. If we think of p as constant, then p^{-k} is subsumed by k^k anyway. So finally, their Theorem 15 generalizes using orthogonality as well:

Theorem 15 *The described algorithm agnostically learns k-juntas to error* $\text{Opt} + 5\epsilon$ *in the $p\rho NS$ model in time* $\text{poly}(n, k^k, \epsilon^{-k}, p^{-k})$.

10 Further Work

Although we have made progress on learning TOP in the uniform random walk model, it would of course be preferable to have a polynomial-time algorithm. In the product model, it is obvious that n-bit parity is learnable by simply observing which bits are relevant during a walk. Can TOP be learned (quasi)-efficiently in this model?

11 Acknowledgments

The authors thank Ryan O'Donnell for suggesting the Metropolis-Hastings algorithm and Rocco Servedio for a helpful discussion.

References

[Bah61] R. R. Bahadur. A representation of the joint distribution of responses to n dichotomous items. In Herbert Solomon, editor, *Studies in Item Analysis and Prediction*, pages 158–168. Stanford University Press, Stanford, California, 1961.

[BJT04] Nader H. Bshouty, Jeffrey C. Jackson, and Christino Tamon. More efficient PAC-learning of DNF with membership queries under the uniform distribution. *J. Comput. Syst. Sci.*, 68(1):205–234, 2004.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, 2003.

[BMOS05] Nader H. Bshouty, Elchanan Mossel, Ryan O'Donnell, and Rocco A. Servedio. Learning dnf from random walks. *J. Comput. Syst. Sci.*, 71(3):250–265, 2005.

[Fel07] Vitaly Feldman. Attribute-efficient and non-adaptive learning of parities and dnf expressions. *J. Mach. Learn. Res.*, 8:1431–1460, 2007.

[GKK08] Parikshit Gopalan, Adam Tauman Kalai, and Adam R. Klivans. Agnostically learning decision trees. In *STOC '08: Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 527–536, New York, NY, USA, 2008. ACM.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 25–32, 1989.

[Has70] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, April 1970.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *American Statistical Association Journal*, 58:13–30, 1963.

[Jac97] Jeffrey Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. *Journal of Computer and System Sciences*, 55(3):414–440, 12 1997.

[KM93] Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the Fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, December 1993. Earlier version appeared in *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 455–464, 1991.

[KS99] Adam Klivans and Rocco A. Servedio. Boosting and hard-core sets. In *Proc. 40th Annu. IEEE Symposium on Foundations of Computer Science*, pages 624–633, 1999.

[Lev93] Leonid A. Levin. Randomness and non-determinism. *Journal of Symbolic Logic*, 58(3):1102–1103, 1993.

[Roc07] Sébastien Roch. On learning thresholds of parities and unions of rectangles in random walk models. *Random Struct. Algorithms*, 31(4):406–417, 2007.