Testing by implicit learning: a brief survey

Rocco A. Servedio*
Columbia University
rocco@cs.columbia.edu

February 9, 2010

Abstract

We give a high-level survey of the "testing by implicit learning" paradigm, and explain some of the property testing results for various Boolean function classes that have been obtained using this approach.

1 Introduction

This brief survey is about an approach by which proper learning algorithms from computational learning theory can sometimes be leveraged to obtain query-efficient property testing algorithms. It does not contain full proofs but gives a high-level overview of the main ideas along with complete statements of the key results. Readers who are interested in more details are referred to the full proofs in [DLM$^+$07, DLM$^+$08, GOS$^+$09].

After presenting some relevant background from earlier work in property testing, in Section 2 we first give an explanation of the basic approach, called "testing by implicit learning," and describe a general condition under which a class $\mathcal{C}$ of functions can be efficiently tested using the basic approach. (Roughly speaking, the condition is that functions in the class must have some sort of concise representation, and must be well-approximated by juntas in the class.) The results in Section 2 are from [DLM$^+$07].

We then present two extensions of the basic "testing by implicit learning" approach. First, in Section 3 we describe how in one particular case (testing the class of s-sparse polynomials over $\mathbb{F}_2$) it is possible to augment the basic approach to obtain a testing algorithm which is *computationally* efficient as well as query-efficient. These results were first presented in [DLM$^+$08]. For the second extension, we explain (at a high level) how the "testing by implicit learning" approach can be carried out in the Fourier domain; very roughly speaking, in this extension the 2^n parity functions over $x_1, \dots, x_n$ play the role that the n Boolean variables $x_1, \dots, x_n$ play in the original approach. This extension, and the testing results obtained using it, were first established in [GOS$^+$09] and are presented here in Section 4. We close in Section 5 with suggested directions for future work.

Notation and Conventions. In [DLM$^+$07] the basic "testing by implicit learning" approach is presented in a rather general setting in which the functions being tested are mappings from Ω^n to X, where Ω and X may be arbitrary finite sets. For simplicity and ease of exposition, in this note we will only consider testing Boolean functions which map $\{0,1\}^n \to \{0,1\}$ (this is the framework for the extensions of both Sections 3 and 4). Thus for us a "property of Boolean functions" is a class $\mathcal{C}$ of Boolean functions over $\{0,1\}^n$ as described above.

*Supported in part by NSF award CCF-0915929, by NSF award CCF-0347282, and by DARPA grant HR0011-08-1-0069.

We work in the usual property testing model for Boolean functions. To recap, we view $\{0,1\}^n$ as endowed with the uniform distribution. Two functions $f_1, f_2 : \{0,1\}^n \rightarrow \{0,1\}$ are said to be *ϵ-close* if $\Pr[f_1(x) \neq f_2(x)] \leq \epsilon$, and are *$\epsilon$-far* if $\Pr[f_1(x) \neq f_2(x)] > \epsilon$. A *testing algorithm for class $\mathcal{C}$* is an algorithm A which takes black-box oracle access to an unknown and arbitrary function $f : \{0,1\}^n \rightarrow \{0,1\}$. If f belongs to $\mathcal{C}$ then A must output "yes" with probability at least $2/3$ (over its internal coin tosses), and if f is ϵ-far from every $g \in \mathcal{C}$ then A must output "no" with probability at least $2/3$ (thus we consider testers with two-sided error).

A function $f : \{0,1\}^n \rightarrow \{0,1\}$ is said to be a *J-junta* if there exists a set $\mathcal{J} \subseteq [n]$ of size at most J such that $f(x) = f(y)$ for every two assignments x, y that agree on all the coordinates in $\mathcal{J}$ (in other words, f is a J-junta if it depends on at most J out of the n input variables).

1.1 Some previous work on testing classes of Boolean functions.

The "testing by implicit learning" approach is useful for classes of Boolean functions that have some sort of "concise representation." There has been considerable previous research on testing functions for properties corresponding to different notions of having a concise representation. An important precursor of the "testing by implicit learning" work is the paper of Parnas *et al.* [PRS02], which gave algorithms for testing whether Boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}$ have certain very simple representations as Boolean formulae. They gave an $O(1/\epsilon)$-query algorithm for testing whether f is a single Boolean literal or a Boolean conjunction, and an $\tilde{O}(s^2/\epsilon)$-query algorithm for testing whether f is an s-term monotone DNF. Parnas *et al.* posed as an open question whether a similar testing result can be obtained for the broader class of general (non-monotone) s-term DNF formulas; as we will see, the "testing by implicit learning" method gives an affirmative answer to this question.

Another closely related work is that of Fischer *et al.* [FKR$^+$04], who gave an algorithm to test whether a Boolean function $f : \Omega^n \rightarrow \{0,1\}$ is a J-junta (i.e. depends only on at most J of its n arguments) with query complexity polynomial in J and $1/\epsilon$. As described below, the "testing by implicit learning" approach makes crucial use of techniques from [FKR$^+$04], in combination with ideas from computational learning theory.

1.2 Relevant earlier work relating property testing and learning.

The basic idea of using a learning algorithm to do property testing goes back to Goldreich *et al.* [GGR98]. They observed that any proper learning algorithm for a class $\mathcal{C}$ can immediately be used as a testing algorithm for $\mathcal{C}$. (Recall that a proper learning algorithm for $\mathcal{C}$ is one which outputs a hypothesis h that itself belongs to $\mathcal{C}$.) The idea behind this observation is that if the function f being tested belongs to $\mathcal{C}$ then a proper learning algorithm will succeed in constructing a hypothesis that is close to f, while if f is ϵ-far from every $g \in \mathcal{C}$ then any hypothesis $h \in \mathcal{C}$ that the learning algorithm outputs must necessarily be far from f.

This observation shows that any class $\mathcal{C}$ can be tested to accuracy ϵ using essentially the same number of queries that are required to properly learn the class to accuracy $\Theta(\epsilon)$. However, it is well known that proper learning algorithms for virtually every interesting class of n-variable Boolean functions (such as all the classes listed in Table 1, including such simple classes as Boolean literals) must make at least $\Omega(\log n)$ queries. In many cases the "testing by implicit learning" approach can be used to test a class $\mathcal{C}$ with fewer queries than are required for learning – in particular, with query complexity independent of n (again see Table 1).

Class of functions	Number of Queries	Reference
Boolean literals (dictators), conjunctions	$O(1/\epsilon)$	[PRS02]
s-term monotone DNFs	$\tilde{O}(s^2/\epsilon)$	[PRS02]
J-juntas	$\tilde{O}(J^2/\epsilon)$, $\Omega(J)$ (adaptive) $\tilde{O}(J/\epsilon)$	[FKR+04], [CG04] [Bla09]
decision lists	$\tilde{O}(1/\epsilon^2)$	[DLM+07]
size-s decision trees, size-s branching programs, s-term DNFs, size-s Boolean formulas	$\tilde{O}(s^4/\epsilon^2)$, $\Omega(\log s/\log\log s)$ (adaptive)	[DLM+07]
s-sparse polynomials over $\mathbb{F}_2$	$\tilde{O}(s^4/\epsilon^2)$, $\tilde{\Omega}(\sqrt{s})$	[DLM+07]
size-s Boolean circuits	$\tilde{O}(s^6/\epsilon^2)$	[DLM+07]
functions with Fourier degree $\leq d$	$\tilde{O}(2^{6d}/\epsilon^2)$, $\tilde{\Omega}(\sqrt{d})$	[DLM+07]
induced subclasses of functions with k-dimensional Fourier spectra	$2^{O(k)} \cdot \text{poly}(1/\epsilon)$, $\Omega(2^{k/2})$ (adaptive)	[GOS+09]

Table 1: Selected results on testing various classes of Boolean functions over $\{0,1\}^n$. The "testing by implicit learning" method is how the upper bounds in lines marked by "[DLM+07]" and "[GOS+09]" are achieved. The upper bounds for those results are for adaptive algorithms (though it is shown in [DLM+07] that very similar bounds can be achieved by non-adaptive algorithms), and the lower bounds are for non-adaptive algorithms unless otherwise indicated by (adaptive).

2 The basic "testing by implicit learning" approach

2.1 Overview of the approach.

An observation which is at the heart of the approach is that many interesting classes $\mathcal{C}$ of functions are "well-approximated" by juntas in the following sense: every function in $\mathcal{C}$ is close to some function in $\mathcal{C}_J$, where $\mathcal{C}_J \subseteq \mathcal{C}$ and every function in $\mathcal{C}_J$ is a J-junta. For example, every s-term DNF over $\{0,1\}^n$ is τ-close to an s-term DNF that depends on only $s \log(s/\tau)$ variables, since each term with more than $\log(s/\tau)$ variables can be removed from the DNF at the cost of at most τ/s error.

Roughly speaking, the "testing by implicit learning" approach to testing whether f belongs to $\mathcal{C}$ works by attempting to learn the "structure" of the junta in $\mathcal{C}_J$ that f is close to *without actually identifying the relevant variables on which the junta depends*. If the algorithm finds such a junta function, it accepts, and if it does not, it rejects. The approach is described as "implicit learning" (as opposed to the explicit proper learning of Goldreich *et al.* [GGR98]), since it learns the structure of the junta to which f is close without explicitly identifying its relevant variables. Indeed, avoiding identifying the relevant variables is what makes it possible to have query complexity independent of n.

The basic algorithm finds the structure of the junta f' in $\mathcal{C}_J$ that f is close to by using the techniques of [FKR+04]. As in [FKR+04], it begins by randomly partitioning the variables of f into subsets and identifying which subsets contain an influential variable (the random partitioning ensures that with high probability, each subset contains at most one such variable if f is indeed in $\mathcal{C}$). Next, the algorithm creates a sample of random labeled examples $(x^1, y^1), (x^2, y^2), ..., (x^m, y^m)$, where each x^i is a string of length J (not length n; this is crucial to the query complexity of the algorithm) whose bits correspond to the influential variables of f, and where y^i corresponds with high probability to the value of junta f' on x^i. We note that the number m of examples created is the number of examples required to learn the class $\mathcal{C}_J$ of J-variable functions using "Occam's Razor" [BEHW87]; it is independent of n. Finally, the algorithm exhaustively checks whether any function in $\mathcal{C}_J$ – over J input variables – is consistent with this labeled sample. This step takes at least $|\mathcal{C}_J|$ time steps, which is exponential in s for the classes in Table 1; but since $|\mathcal{C}_J|$ is independent of n the overall approach has query complexity that is independent of n. (The overall time complexity is linear as a function of n; note that such a runtime dependence on n is inevitable since it takes n time steps simply to prepare a length-n query string to the black-box function.)

In the rest of this section we give some more details on the algorithm and its performance.

2.2 Subclass approximators.

Let $\mathcal{C}$ denote a class of functions from $\{0,1\}^n$ to $\{0,1\}$. We will be interested in classes of functions that can be closely approximated by juntas in the class. We have the following:

Definition 1. *For $\tau > 0$, we say that a subclass $\mathcal{C}(\tau) \subseteq \mathcal{C}$ is a $(\tau, J(\tau))$-*approximator *for $\mathcal{C}$ if*

- *$\mathcal{C}(\tau)$ is closed under permutation of variables, i.e. if $f(x_1, \dots, x_n) \in \mathcal{C}(\tau)$ then $f(x_{\sigma_1}, \dots, x_{\sigma_n})$ is also in $\mathcal{C}(\tau)$ for every permutation σ of $[n]$; and*
- *for every function $f \in \mathcal{C}$, there is a function $f' \in \mathcal{C}(\tau)$ such that f' is τ-close to f and f' is a $J(\tau)$-junta.*

Typically for us $\mathcal{C}$ will be a class of functions with size bound s in some particular representation, and $J(\tau)$ will depend on s and τ. (A good running example to keep in mind is that $\mathcal{C}$ is the class of all functions that have s-term DNF representations. In this case we may take $\mathcal{C}(\tau)$ to be the class of all s-term $\log(s/\tau)$-DNFs, and we have $J(\tau) = s\log(s/\tau)$.) The approach succeeds on function classes $\mathcal{C}$ for which $J(\tau)$ is a slowly growing function of $1/\tau$ such as $\log(1/\tau)$.

We write $\mathcal{C}(\tau)_k$ to denote the subclass of $\mathcal{C}(\tau)$ consisting of those functions that depend only on variables in $\{x_1, \dots, x_k\}$. We may (and will) view functions in $\mathcal{C}(\tau)_k$ as taking k arguments rather than n.

2.3 More detailed explanation of the basic algorithm.

Given $\epsilon > 0$ and black-box access to f, the algorithm performs three main steps:

1. **Identify critical subsets.** In Step 1, the algorithm first randomly partitions the variables $x_1, \dots, x_n$ into r disjoint subsets $I_1, \dots, I_r$. It then attempts to identify a set of $j \le J(\tau^\star)$ of these r subsets, which we refer to as *critical* subsets because they each contain a "highly relevant" variable. (For now the value $\tau^\star$ should be thought of as a small quantity; we discuss how this value is selected below.) This step is essentially the same as the 2-sided test for J-juntas from Section 4.2 of Fischer *et al.* [FKR$^+$04]. The analysis shows that if f is close to a $J(\tau^\star)$-junta then this step will succeed w.h.p., and if f is far from every $J(\tau^\star)$-junta then this step will fail w.h.p.

2. **Construct a sample.** Let $I_{i_1}, \dots, I_{i_j}$ be the critical subsets identified in the previous step. In Step 2 the algorithm constructs a set S of m labeled examples $\{(x^1, y^1), \dots, (x^m, y^m)\}$, where each x^i is independent and uniformly distributed over $\{0,1\}^{J(\tau^\star)}$. The analysis shows that if f belongs to $\mathcal{C}$, then with high probability there is a fixed $f'' \in \mathcal{C}(\tau^\star)_{J(\tau^\star)}$ such that each y^i is equal to $f''(x^i)$. On the other hand, if f is far from $\mathcal{C}$, then the analysis shows that w.h.p. no such $f'' \in \mathcal{C}(\tau^\star)_{J(\tau^\star)}$ exists.

   Each labeled example is constructed by again borrowing a technique outlined in [FKR$^+$04]. The algorithm starts with a uniformly random $z \in \{0,1\}^n$. It then attempts to determine how the j highly relevant coordinates of z are set. Although the algorithm does not know which of the coordinates of z are highly relevant, it does know that, assuming the previous step was successful, there should be one highly relevant coordinate in each of the critical subsets. It uses the independence test of [FKR$^+$04] repeatedly to determine the setting of the highly relevant coordinate in each critical subset.

For example, suppose that I_1 is a critical subset. To determine the setting of the highly relevant coordinate of z in critical subset I_1, the algorithm subdivides I_1 into two sets: the subset $\Omega_0 \subseteq I_1$ of indices where z is set to 0, and the subset $\Omega_1 = I_1 \backslash \Omega_0$ of indices where z is set to 1. It then uses the independence test separately on both Ω_0 and Ω_1 to find out which one contains the highly relevant variable. This tells it whether the highly relevant coordinate of z in subset I_1 is set to 0 or 1. The algorithm repeats this process for each critical subset in order to find the settings of the j highly relevant coordinates of z; these form the string x. (The other $J(\tau^\star) - j$ coordinates of x are set to random values; intuitively, this is okay since they are essentially irrelevant.) The algorithm ultimately uses $(x, f(z))$ as the labeled example.

3. **Check consistency.** Finally, in Step 3 the algorithm searches through $\mathcal{C}(\tau^\star)_{J(\tau^\star)}$ looking for a function f'' over $\{0,1\}^{J(\tau^\star)}$ that is consistent with all m examples in S. (Note that this step takes $\Omega(|\mathcal{C}(\tau^\star)_{J(\tau^\star)}|)$ time but uses no queries.) If it finds such a function it accepts f, otherwise it rejects.

2.4 Sketch of the analysis.

We now give an intuitive explanation of the analysis of the test.

Completeness. Suppose f is in $\mathcal{C}$. Then there is some $f' \in \mathcal{C}(\tau^\star)$ that is $\tau^\star$-close to f. Intuitively, $\tau^\star$-close is so close that for the entire execution of the testing algorithm, the black-box function f might as well actually be f' (the algorithm only performs $\ll 1/\tau^\star$ many queries in total, each on a uniform random string, so w.h.p. the view of the algorithm will be the same whether the target is f or f'). Thus, for the rest of this intuitive explanation of completeness, we pretend that the black-box function is f'.

Recall that the function f' is a $J(\tau^\star)$-junta. Since f' is a junta, in Step 1 the test will be able to identify a collection of $j \leq J(\tau^\star)$ "critical subsets" with high probability. Intuitively, these subsets have the property that:

- each "highly relevant" variable occurs in one of the critical subsets, and each critical subset contains at most one highly relevant variable (in fact at most one relevant variable for f');
- the variables outside the critical subsets are so "irrelevant" that w.h.p. in all the queries the algorithm makes, it doesn't matter how those variables are set (randomly flipping the values of these variables would not change the value of f' w.h.p.).

Given critical subsets from Step 1 that satisfy the above properties, in Step 2 the test constructs a sample of labeled examples $S = \{(x^1, y^1), \ldots, (x^m, y^m)\}$ where each x^i is independent and uniform over $\{0,1\}^{J(\tau^\star)}$. In [DLM+07] it is shown that w.h.p. there is a $J(\tau^\star)$-junta $f'' \in C(\tau^\star)_{J(\tau^\star)}$ with the following properties:

- there is a permutation $\sigma : [n] \rightarrow [n]$ for which $f''(x_{\sigma(1)}, \ldots, x_{\sigma(J(\tau))})$ is close to $f'(x_1, \ldots, x_n)$;
- the sample S is labeled according to f''.

Finally, in Step 3 the test does a brute-force search over all of $\mathcal{C}(\tau^\star)_{J(\tau^\star)}$ to see if there is a function consistent with S. Since f'' is such a function, the search will succeed and the test outputs "yes" with high probability overall.

Soundness. Suppose now that f is ϵ-far from $\mathcal{C}$.

One possibility is that f is ϵ-far from every $J(\tau^\star)$-junta; if this is the case then w.h.p. the test will output "no" in Step 1.

The other possibility is that f is ϵ-close to a $J(\tau^\star)$-junta f' (or is itself such a junta). Suppose that this is the case and that the testing algorithm reaches Step 2. In Step 2, the algorithm tries to construct a set of labeled examples that is consistent with f'. The algorithm may fail to construct a sample at all; if this happens then it outputs "no." If the algorithm succeeds in constructing a sample S, then w.h.p. this sample is indeed consistent with f'; but in this case, w.h.p. in Step 3 the algorithm will not find any function $g \in \mathcal{C}(\tau^\star)_{J(\tau^\star)}$ that is consistent with all the examples. (If there were such a function g, then standard arguments in learning theory show that w.h.p. any such function $g \in \mathcal{C}(\tau^\star)_{J(\tau^\star)}$ that is consistent with S is in fact close to f'. Since f' is in turn close to f, this would mean that g is close to f. But g belongs to $\mathcal{C}(\tau^\star)_{J(\tau^\star)}$ and hence to $\mathcal{C}$, so this violates the assumption that f is ϵ-far from $\mathcal{C}$.)

2.5 The main theorem and its consequences.

The main theorem about the "testing by implicit learning" algorithm sketched above is stated below. A full proof can be found in [DLM$^+$07].

Theorem 2. *There is an algorithm $\mathcal{A}$ with the following properties:*

Let $\mathcal{C}$ be a class of functions from $\{0,1\}^n$ to $\{0,1\}$. Suppose that for every $\tau > 0$, $\mathcal{C}(\tau) \subseteq \mathcal{C}$ is a $(\tau, J(\tau))$-approximator for $\mathcal{C}$. Suppose moreover that for every $\epsilon > 0$, there is a τ satisfying

$$\tau \leq \kappa \cdot \frac{\epsilon^2}{J(\tau)^2 \cdot \ln^2(J(\tau)) \cdot \ln\ln(J(\tau)) \cdot \ln^2(|\mathcal{C}(\tau)_{J(\tau)}|) \cdot \ln(\frac{1}{\epsilon}\ln|\mathcal{C}(\tau)_{J(\tau)}|)}, \tag{1}$$

where $\kappa > 0$ is a fixed absolute constant. Let $\tau^\star$ be the largest value τ satisfying (1) above. Then algorithm $\mathcal{A}$ makes:

$$\tilde{O}\left(\frac{1}{\epsilon^2} J(\tau^\star)^2 \ln^2(|\mathcal{C}(\tau^\star)_{J(\tau^\star)}|)\right)$$

many black-box queries to f, and satisfies the following:

- *If $f \in \mathcal{C}$ then $\mathcal{A}$ outputs "yes" with probability at least $2/3$;*
- *If f is ϵ-far from $\mathcal{C}$ then $\mathcal{A}$ outputs "no" with probability at least $2/3$.*

Here are some observations to help interpret the bound (1). Note that if $J(\tau)$ grows too rapidly as a function of $1/\tau$, e.g. $J(\tau) = \Omega(1/\sqrt{\tau})$, then there will be no $\tau > 0$ satisfying inequality (1). On the other hand, if $J(\tau)$ grows slowly as a function of $1/\tau$, e.g. $\log(1/\tau)$, then it is may be possible to satisfy (1).

In many cases, including all of the applications stated below, $J(\tau)$ will grow as $O(\log(1/\tau))$, and $\ln|C(\tau)_{J(\tau)}|$ will always be at most $\mathrm{poly}(J(\tau))$, so (1) will always be satisfiable. The most typical case is that $J(\tau) \leq \mathrm{poly}(s)\log(1/\tau)$ (where s is a size parameter for the class of functions in question) and $\ln|C(\tau)_{J(\tau)}| \leq \mathrm{poly}(s) \cdot \mathrm{poly}\log(1/\tau)$, which yields $\tau^\star = \tilde{O}(\epsilon^2)/\mathrm{poly}(s)$ and an overall query bound of $\mathrm{poly}(s)/\tilde{O}(\epsilon^2)$.

Theorem 2 can be used to establish that many well-studied classes of Boolean functions have query-efficient testing algorithms. The following results, which are proved in [DLM$^+$07], are also summarized in Table 1.

Theorem 3. *For any s and any $\epsilon > 0$, Algorithm $\mathcal{A}$ yields a testing algorithm for*

(i) decision lists using $\tilde{O}(1/\epsilon^2)$ queries;

(ii) size-s decision trees using $\tilde{O}(s^4/\epsilon^2)$ queries;

(iii) size-s branching programs using $\tilde{O}(s^4/\epsilon^2)$ queries;

(iv) s-term DNF using $\tilde{O}(s^4/\epsilon^2)$ queries;

(v) size-s Boolean formulas (with unbounded-fanin AND, OR and NOT gates) using $\tilde{O}(s^4/\epsilon^2)$ queries;

(vi) size-s Boolean circuits (with unbounded-fanin AND, OR and NOT gates) using $\tilde{O}(s^6/\epsilon^2)$ queries;

(vii) functions with Fourier degree at most d using $\tilde{O}(2^{6d}/\epsilon^2)$ queries;

(viii) s-sparse $\mathbb{F}_2$ polynomials using $\tilde{O}(s^4/\epsilon^2)$ queries.

3 Efficiently testing sparse $\mathbb{F}_2$ polynomials

The main drawback of the basic "testing by implicit learning" approach described in Section 2 is its time complexity. The original [DLM$^+$07] algorithm has running time $2^{\omega(s)}$ as a function of s and $\omega(\text{poly}(1/\epsilon))$ as a function of ϵ for each of the "size-s" function classes (ii) through (viii) in Theorem 3.[1]

[DLM$^+$07] asked whether any of these classes can be tested with both time complexity and query complexity poly$(s, 1/\epsilon)$. This question was answered in [DLM$^+$08], where it was shown that the class of *s-sparse $\mathbb{F}_2$ polynomials* can be so tested. Recall that a $\mathbb{F}_2$ polynomial $p : \{0,1\}^n \rightarrow \{0,1\}$ is a multilinear polynomial with coefficients from $\mathbb{F}_2$, i.e. all nonzero coefficients are 1. Such a polynomial may be viewed as a parity of monotone conjunctions (monomials). It is *s-sparse* if it contains at most s monomials (including the constant-1 monomial if it is present). The main result of [DLM$^+$08] is a time-efficient and query-efficient tester for $\mathbb{F}_2$ polynomials:

Theorem 4. *There is a poly$(s, 1/\epsilon)$-query algorithm with the following performance guarantee: given parameters s, ϵ and black-box access to any $f : \{0,1\}^n \rightarrow \{-1,1\}$, it runs in time $\text{poly}(s, 1/\epsilon)$ and tests whether f is an s-sparse $\mathbb{F}_2$ polynomial versus ϵ-far from every s-sparse polynomial.*

At a high level, the algorithm of [DLM$^+$08] augments the basic "testing by implicit learning" approach by using a sophisticated proper learning algorithm due to Schapire and Sellie [SS96] in place of the naive brute-force search which is used as the learning step in the basic approach. However, significant complications arise in the attempt to "implicitly" run the [SS96] algorithm that do not arise with the brute-force search of [DLM$^+$07]. In the rest of this subsection we briefly describe those complications and how they are addressed in [DLM$^+$08].

We first note that if f is an s-sparse $\mathbb{F}_2$ polynomial, an easy argument shows that there is a function f' - obtained by discarding from f all monomials of degree more than $\log(s/\tau)$ - that is τ-close to f and depends on at most $r = s\log(s/\tau)$ variables. As described in the previous section, the basic "testing by implicit learning" approach uses ideas of [FKR$^+$04] for testing juntas to construct a sample of uniform random examples over $\{0,1\}^r$ which with high probability are all labeled according to f'. At this point, the [DLM$^+$07] algorithm uses a naive brute-force search to check all s-sparse $\mathbb{F}_2$ polynomials over r (as opposed to n) variables, to see if any one of them is consistent with the labeled sample. This leads to a running time of roughly $2^{\omega(s)} \cdot (1/\epsilon)^{\log\log(1/\epsilon)}$.

The proper learning algorithm of Schapire and Sellie [SS96] runs in time polynomial in r and s to exactly learn any unknown s-sparse $\mathbb{F}_2$ polynomial over r variables; thus a natural idea is to use it instead of the computationally inefficient brute-force search of [DLM$^+$07]. However, the [SS96] learning algorithm requires access to a *membership query* oracle, i.e. a black-box oracle for the function being learned. Thus,

[1]As discussed in the previous section, an $\Omega(n)$ running time is necessary since the testing algorithm must prepare n-bit strings for the black-box oracle for f. The algorithms of this section and the next one both have running times that are linear in n for this reason; henceforth we discuss the running times of our testers only as a function of the other parameters.

in order to run the Schapire/Sellie algorithm in the "testing by implicit learning" framework, it is necessary to simulate membership queries to an approximating function f' which is close to f but depends on only r variables. This is significantly more challenging than generating uniform random examples labeled according to f', which is all that is required in the original [DLM+07] approach.

To see why membership queries to f' are more difficult to simulate than uniform random examples, recall that f and the f' described above (obtained from f by discarding high-degree monomials) are τ-close. Intuitively this is extremely close, disagreeing only on a $1/m$ fraction of inputs for an m that is much larger than the number of random examples required for learning f' via brute-force search (this number is "small" – independent of n – because f' depends on only r variables). Thus in the [DLM+07] approach it suffices to use f, the function to which we actually have black-box access, rather than f' to label the random examples used for learning f'; since f and f' are so close, and the examples are uniformly random, with high probability all the labels will also be correct for f'. However, in the membership query scenario of [DLM+08], things are no longer that simple. For any given f' which is close to f, one can no longer assume that the learning algorithm's queries to f' are uniformly distributed and hence unlikely to hit the error region – indeed, it is possible that the learning algorithm's membership queries to f' are clustered on the few inputs where f and f' disagree.

Thus, in order to successfully simulate membership queries, the algorithm must consistently answer queries according to a particular approximator f', even though it only has oracle access to f. This must be done implicitly in a query-efficient way, since explicitly identifying even a single variable relevant to f' requires at least $\Omega(\log n)$ queries. [DLM+08] does this by showing that for any s-sparse polynomial f, an approximating f' can be obtained as a restriction of f by setting certain carefully chosen subsets of variables to zero. Roughly speaking, this restriction is obtained by randomly partitioning all of the input variables into r subsets and zeroing out all subsets whose variables have small "collective influence."

3.1 The [DLM+08] algorithm and its analysis.

In the rest of this section we give a high-level description of the actual [DLM+08] testing algorithm, called **Test-Sparse-Poly**, and its analysis.

Test-Sparse-Poly is based on the idea that if f is a sparse polynomial then it only has a small number of "high-influence" variables, and it is close to another sparse polynomial f' (obtained from f by fixing some input variables to zero) that depends only on those high-influence variables. Roughly speaking, the algorithm works by first isolating the high-influence variables into distinct subsets, and then attempting to implicitly learn f' using the [SS96] algorithm.

We now describe the testing algorithm in tandem with a sketch of why the test is complete, i.e. why it accepts s-sparse polynomials (later we will give a sketch of the soundness argument). The first thing **Test-Sparse-Poly** does (Step 1) is to randomly partition the variables into $r = \mathrm{poly}(s/\tau)$ subsets. If f is an s-sparse polynomial, then it indeed has few high-influence variables, so with high probability at most one such variable will be present in each subset.

Next (Step 2) the algorithm attempts to distinguish subsets that contain a high-influence variable from subsets that do not; this is done by using the independence test of [FKR+04] as described in Section 2.

Once the subsets that contain high-influence variables have been identified, next (Step 3) the algorithm defines a function f' which "zeroes out" all of the variables in all low-influence subsets. Note that if the original function f is an s-sparse polynomial, then f' will be one too. Step 4 of **Test-Sparse-Poly** checks that f is close to f'; it is shown in [DLM+08] that this is indeed the case with high probability if f is an s-sparse polynomial.

The final step (Step 5) of **Test-Sparse-Poly** is to implicitly run the [SS96] algorithm to learn a sparse polynomial, which we call f'', which is isomorphic to f' but is defined only over the high-influence variables of f (recall that there is at most one from each high-variation subset). The overall **Test-Sparse-Poly** algorithm accepts f if and only if the learning algorithm successfully returns a final hypothesis (i.e. does not halt and output "fail"). It is shown in [DLM$^+$08] that for f an s-sparse polynomial, with high probability the subsets that are not restricted in Step 3 have a certain "nice structure:" essentially, they each have one variable with very high influence and all other variables with very low influence. [DLM$^+$08] show that this makes it possible to simulate the membership queries that the [SS96] algorithm requires, so it is possible to implicitly run the [SS96] learning algorithm. Thus, for f an s-sparse polynomial the [SS96] algorithm can run successfully, and the test will accept.

Sketch of soundness. We close this section with a brief sketch of the soundness argument, that if **Test-Sparse-Poly** accepts f with high probability then f must be close to some s-sparse polynomial.

If f passes Step 2 with high probability, then [DLM$^+$08] shows that **Test-Sparse-Poly** must have obtained a partition of variables into subsets that contain a high-influence variable, and subsets that have low "collective influence." If f passes Step 4, then it must moreover be the case that f is close to the function f' obtained by zeroing out the low-influence subsets.

In the last step, **Test-Sparse-Poly** attempts to run the [SS96] algorithm to learn f'' using the high-influence subsets; in the course of doing this, it attempts to simulate membership queries to f''. Since f could be an arbitrary function, we do not know whether each high-influence subset has at most one variable relevant to f'. However, [DLM$^+$08] show that, if the routine to simulate membership queries with high probability never returns "fail," then f' must be close to a junta g whose relevant variables are the individual "highest-influence" variables in each of the high-influence subsets. Now, given that the [SS96] algorithm halts successfully, it must be the case that it constructs a final hypothesis h that is itself an s-sparse polynomial and that agrees with a large sample of many random examples. From this it is possible to argue that h must be close to g (using standard arguments from learning theory), hence close to f', and hence close to f. So indeed if **Test-Sparse-Poly** accepts f with high probability then f must be close to some s-sparse polynomial.

4 Testing induced subclasses of functions with k-dimensional Fourier spectra

In [DLM$^+$07] and [DLM$^+$08] the learning is "implicit" in the sense that a learning algorithm is executed to learn some function depending on r of the n input variables, without the identity of those r variables ever being explicitly determined by the algorithm. [GOS$^+$09] extended this methodology to learn some function depending on k of the 2^n *parity functions* $\{\chi_\alpha\}$ over input variables without ever explicitly identifying those parity functions, and obtained testing results using this extended notion of "implicit learning."

We establish some terminology and recall some background in order to state the results of [GOS$^+$09]. We view the domain $\{0,1\}^n$ as $\mathbb{F}_2^n$, so a real-valued function over the Boolean cube is a mapping $\mathbb{F}_2^n \to \mathbb{R}$. Every such function f has a unique representation as

$$f(x) = \sum_{\alpha \in \mathbb{F}_2^n} \hat{f}(\alpha)\chi_\alpha(x) \qquad \text{where} \qquad \chi_\alpha(x) \stackrel{\text{def}}{=} (-1)^{\langle \alpha, x\rangle} = (-1)^{\sum_{i=1}^n \alpha_i x_i}.$$

The coefficients $\hat{f}(\alpha)$ are the *Fourier coefficients* of f, and the functions $\chi_\alpha(\cdot)$ are sometimes referred to as *linear functions* or *characters*; they are simply parity functions over all possible subsets of the n input variables. In addition to treating input strings x as lying in $\mathbb{F}_2^n$, we also index the characters by vectors $\alpha \in \mathbb{F}_2^n$; this is to emphasize the fact that we are concerned with the linear-algebraic structure. We write $\mathrm{Spec}(f)$ for the Fourier spectrum of f, i.e. the set $\{\alpha \in \mathbb{F}_2^n : \hat{f}(\alpha) \neq 0\}$.

A Boolean function $f : \mathbb{F}_2^n \to \{0,1\}$ is said to be *k-dimensional* if $\mathrm{Spec}(f)$ lies in a k-dimensional subspace of $\mathbb{F}_2^n$. An equivalent definition is that f is k-dimensional if it is a function of k characters

$\chi_{\alpha_1}, \ldots, \chi_{\alpha_k}$, i.e. f is a junta over k parity functions (this is easily seen by picking $\{\alpha_i\}$ to be a basis for $\mathrm{Spec}(f)$). Thus the class of all k-dimensional Boolean functions consists of all Boolean functions of the form $g(\chi_{\alpha_1}, \ldots, \chi_{\alpha_k})$ where g is any k-junta and $\chi_{\alpha_1}, \ldots, \chi_{\alpha_k}$ are any parity functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2$

Let $\mathcal{C}$ be a class of n-variable Boolean functions. We say that $\mathcal{C}$ is an *induced subclass of k-dimensional functions* if there is some collection $\mathcal{C}'$ of k-variable Boolean functions such that $\mathcal{C}$ is the class of all functions $f = g(\chi_{\alpha_1}, \ldots, \chi_{\alpha_k})$ where g is any function in $\mathcal{C}'$ and $\chi_{\alpha_1}, \ldots, \chi_{\alpha_k}$ are any parity functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2$ as before. For example, let $\mathcal{C}$ be the class of all k-sparse polynomial threshold functions over $\{-1,1\}^n$; i.e., each function in $\mathcal{C}$ is the sign of a *real* polynomial with at most k nonzero terms. This is an induced subclass of k-dimensional functions, corresponding to the collection $\mathcal{C}' = \{$all linear threshold functions over k Boolean variables$\}$.

[GOS$^+$09] shows that any induced subclass of k-dimensional functions has a query-efficient testing algorithm:

Theorem 5. *Let $\mathcal{C}$ be any induced subclass of k-dimensional functions. There is a nonadaptive* $\mathrm{poly}(2^k, 1/\epsilon)$-*query algorithm for ϵ-testing $\mathcal{C}$.*

The algorithm combines the "testing by implicit learning" approach with a technique from [GOS$^+$09] (building on [FGKP06]) of pairwise independently hashing the Fourier coefficients of f. Very roughly speaking, this pairwise independent hashing is used to "isolate" each nonzero Fourier coefficient of a low-dimensional function f, similar to how a random partition of the variables of a junta into disjoint subsets is used in [FKR$^+$04, DLM$^+$07] to "isolate" each of the relevant variables of a junta. With the Fourier coefficients isolated in this way, given an n-bit example $(z, f(z))$ it is possible to determine the value $\chi_\alpha(z)$ that each character function corresponding to a nonzero Fourier coefficient $\hat{f}(\alpha)$ takes on z, without explicitly identifying the string α. This makes it possible to build a "data set" for implicitly learning the function $f = g(\chi_{\alpha_1}, \ldots, \chi_{\alpha_k})$; each example in the data set consists of a vector of values for all of the parity functions corresponding to nonzero Fourier coefficients, and the corresponding label is the value $f = g(\chi_{\alpha_1}, \ldots, \chi_{\alpha_k})$. This is actually done in an "exhaustive" way, using $2^{\Theta(k)}$ examples, so the "data set" really is more akin to a truth table – it contains an entry for each of the 2^k possible vectors of values that $(\chi_{\alpha_1}, \ldots, \chi_{\alpha_k})$ can take. (It is shown in [GOS$^+$09] that if f is far from any k-dimensional function, then with high probability the construction of the "data set" will reveal this.) Thus one obtains a complete truth table for the k-variable function g, and from this it is trivial (without making any additional queries) to determine whether g belongs to $\mathcal{C}'$.

5 Open problems and directions for future work

There are natural goals for future work related to each of the three main results described above.

As described in Section 2, the basic "testing by implicit learning" approach gives $\mathrm{poly}(s/\epsilon)$-query upper bounds for testing many natural classes such as size-s decision trees, s-term DNF, size-s Boolean formulas, and more. What can be said about lower bounds for these classes? By adapting arguments of Chockler and Gutfreund [CG04], Diakonikolas et al. [DLM$^+$07] establish $\tilde{\Omega}(\log s)$ lower bounds (for adaptive algorithms), but it is quite possible that a $\mathrm{poly}(s)$ lower bound in fact holds. (We note that [DLM$^+$07] does establish a $\tilde{\Omega}(\sqrt{s})$ lower bound for nonadaptive algorithms that test the class of s-sparse $\mathbb{F}_2$ polynomials.)

An obvious goal related to [DLM$^+$08] is to obtain $\mathrm{poly}(s, 1/\epsilon)$-time testing algorithms for other classes beyond just s-sparse $\mathbb{F}_2$ polynomials. Polynomial-time proper learning algorithms are not known for classes such as s-term DNF or size-s decision trees (even if membership queries are allowed), and thus it seems that new ideas may be needed for these classes. A (potentially more modest) goal is to extend the [DLM$^+$08] results to testing s-sparse polynomials over other finite fields; see the conclusion of [DLM$^+$08] for more discussion of this.

Finally, an interesting direction related to the use of "testing by implicit learning" over the Fourier domain, as in [GOS$^+$09], is whether sharper query complexity bounds can be obtained for specific classes of interest. [GOS$^+$09] give a $2^{\Omega(k)}$ lower bound for testing the entire class of all k-dimensional functions, and thus the $2^{O(k)} \cdot \mathrm{poly}(1/\epsilon)$-query upper bound for induced subclasses of k-dimensional functions cannot be improved much in the worst case. But what about specific classes of k-dimensional functions, such as the class of all k-sparse polynomial threshold functions over $\{-1,1\}^n$? It would be interesting to determine whether this class can be tested using only $\mathrm{poly}(k, 1/\epsilon)$ queries.

References

[BEHW87] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, 1987.

[Bla09] Eric Blais. Testing juntas nearly optimally. In *Proc. 41st Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–158, 2009.

[CG04] H. Chockler and D. Gutfreund. A lower bound for testing juntas. *Information Processing Letters*, 90(6):301–305, 2004.

[DLM$^+$07] I. Diakonikolas, H. Lee, K. Matulef, K. Onak, R. Rubinfeld, R. Servedio, and A. Wan. Testing for concise representations. In *Proc. 48th Ann. Symposium on Computer Science (FOCS)*, pages 549–558, 2007.

[DLM$^+$08] I. Diakonikolas, H. Lee, K. Matulef, R. Servedio, and A. Wan. Efficiently testing sparse GF(2) polynomials. In *Proc. 16th International Colloquium on Algorithms, Languages and Programming (ICALP)*, pages 502–514, 2008.

[FGKP06] V. Feldman, P. Gopalan, S. Khot, and A. Ponnuswami. New results for learning noisy parities and halfspaces. In *Proc. 47th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 563–576, 2006.

[FKR$^+$04] E. Fischer, G. Kindler, D. Ron, S. Safra, and A. Samorodnitsky. Testing juntas. *Journal of Computer & System Sciences*, 68:753–787, 2004.

[GGR98] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45:653–750, 1998.

[GOS$^+$09] P. Gopalan, R. O'Donnell, R. Servedio, A. Shpilka, and K. Wimmer. Testing Fourier dimensionality and sparsity. In *Proc. 36th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 500–512, 2009.

[PRS02] M. Parnas, D. Ron, and A. Samorodnitsky. Testing basic boolean formulae. *SIAM J. Disc. Math.*, 16:20–46, 2002.

[SS96] R. Schapire and L. Sellie. Learning sparse multivariate polynomials over a field with queries and counterexamples. *J. Comput. & Syst. Sci.*, 52(2):201–213, 1996.