Foundations and Trends® in
Machine Learning
Vol. 1, No. 3 (2008) 297–394

DOI: 10.1561/2200000004

now
the essence of knowledge

Property Testing: A Learning Theory Perspective

Dana Ron

Department of Electrical Engineering — Systems, Tel-Aviv University, Ramat-Aviv, Tel-Aviv 69978, Israel, danar@eng.tau.ac.il

Abstract

Property testing deals with tasks where the goal is to distinguish between the case that an object (e.g., function or graph) has a pre-specified property (e.g., the function is linear or the graph is bipartite) and the case that it differs significantly from any such object. The task should be performed by observing only a very small part of the object, in particular by querying the object, and the algorithm is allowed a small failure probability.

One view of property testing is as a relaxation of learning the object (obtaining an approximate representation of the object). Thus property testing algorithms can serve as a preliminary step to learning. That is, they can be applied in order to select, very efficiently, what hypothesis class to use for learning. This survey takes the learning-theory point of view and focuses on results for testing properties of functions that are of interest to the learning theory community. In particular, we cover results for testing algebraic properties of functions such as linearity, testing properties defined by concise representations, such as having a small DNF representation, and more.

1

Introduction

Property testing [82, 128] is the study of the following class of problems.

> *Given the ability to perform (local) queries concerning a particular object the problem is to determine whether the object has a predetermined (global) property or differs significantly from any object that has the property. In the latter case we say it is* far *from (having) the property. The algorithm is allowed a small probability of failure, and typically it inspects only a small part of the whole object.*

For example, the object may be a graph and the property is that it is bipartite, or the object may be a function and the property is that it is linear. It is usually assumed that the property testing algorithm is given query access to the object. When the object is a function f the queries are of the form: "what is $f(x)$?" while if the object is a graph then queries may be: "is there an edge between vertices u and v" or: "what vertex is the i^{th} neighbor of v?". In order to determine what it means to be far from the property, we need a distance measure between objects. In the case of functions it is usually the weight according to the uniform

distribution of the symmetric difference between the functions, while in the case of graphs it is usually the number of edge modifications divided by some upper bound on the number of edges. When dealing with other objects (e.g., the object may be a set of points and the property may be that the set of points can be clustered in a certain way) one has to define both the types of queries allowed and the distance measure.

1.1 Property Testing as Relaxed Decision

Property testing problems are often viewed as a relaxation of decision problems. Namely, instead of requiring that the algorithm decide whether the object has the property or does not have the property, the algorithm is required to decide whether the object has the property or is far from having the property. Given this view there are several scenarios in which property testing may be useful.

- If the object is very large, then it is infeasible to examine all of it and we must design algorithms that examine only a small part of the object and make an approximate decision based on what they view.
- Another scenario is when the object is not too large to fully examine, but the exact decision problem is $\mathcal{NP}$-hard. In such a case some form of approximate decision is necessary if one seeks an efficient algorithm and property testing suggest one such form. We note that in some cases the approximation essentially coincides with *standard* notions of approximation problems (e.g., Max-Cut [82]) while in others it is quite different (e.g., k-Colorability [82]).
- It may be the case that the object is not very large and there is an efficient (polynomial-time) algorithm for solving the problem exactly. However, we may be interested in a *very* efficient (sublinear-time) algorithm, and are willing to tolerate the approximation/error it introduces.
- Finally, there are cases in which typical no-instances of the problem (that is, objects that do not have the property) are actually relatively far from having the property. In such cases we may first run the testing algorithm. If it rejects the object

then we reject it and otherwise we run the exact decision procedure. Thus, we save time on the typical no-instances. This is in particular useful if the testing algorithm has one-sided error so that it never rejects yes-instances (that have the property).

In all the aforementioned scenarios we are interested in testing algorithms that are much more efficient than the corresponding decision algorithms, and in particular have complexity that is sublinear in the size of the object.

1.2 Property Testing and Learning (Estimation)

Here when we say *learning* we mean outputting a good estimate of the target object.[1] Thus, another view of property testing is as a relaxation of learning (with queries and under the uniform distribution).[2] Namely, instead of asking that the algorithm output a good estimate of the function (object), which is assumed to belong to a particular class of functions $\mathcal{F}$, we only require that the algorithm decide whether the function belongs to $\mathcal{F}$ or is far from any function in $\mathcal{F}$. Given this view, a natural motivation for property testing is to serve as a preliminary step before learning (and in particular, agnostic learning (e.g., [107]) where no assumption is made about the target function but the hypothesis should belong to a particular class of functions): we can first run the testing algorithm to decide whether to use a particular class of functions as our hypothesis class.

Here too we are interested in testing algorithms that are more efficient than the corresponding learning algorithms. As observed in [82], property testing is no harder than *proper* learning (where the learning algorithm is required to output a hypothesis from the same class of functions as the target function). Namely, if we have a proper learning

[1] One may argue that property testing is also a certain form of learning as we learn information about the object (i.e., whether it has a certain property or is far from having the property). However, we have chosen to adopt the notion of learning usually used in the computational learning theory community.

[2] Testing under non-uniform distributions (e.g., [1, 92]) and testing with random examples (e.g., [105]) have been considered (and are discussed in this survey), but most of the work in property testing deals with testing under the uniform distributions and with queries.

algorithm for a class of functions F then we can use it as a subroutine to test the property: "does the function belong to F" (see Section 2.2 for a formal statement and proof).

Choosing between the two viewpoints. The choice of which of the aforementioned views to take is typically determined by the type of objects and properties in question. Much of property testing deals with combinatorial objects and in particular graphs. For such objects it is usually more natural to view property testing as a relaxation of exact decision. Indeed, there are many combinatorial properties for which there are testing algorithms that are much more efficient than the corresponding (exact) decision algorithms. On the other hand, when the objects are functions, then it is usually natural to look at property testing from a learning theory perspective. In some cases, both viewpoints are appropriate. This survey focuses on the latter perspective.

1.3 Property Testing and Hypothesis Testing

The notion of property testing is related to that of *hypothesis testing* (see e.g., [108, Chap. 8]) and indeed the distinction between estimation and testing is well known in the mathematical statistics literature. In this context, having the tested property (belonging to the corresponding class of objects) is called the *null* hypothesis, while being ϵ-far from the property (where ϵ is the distance parameter that the algorithm is given as input) is the *alternative* hypothesis. There are two major mathematical approaches to the study of testing in statistics (see, e.g., [136] and [113]). In the first, the alternative is taken to approach the null hypothesis at a certain rate as a function of the number of data points; when the correct rate is chosen the error probabilities stabilize at values strictly greater than zero and strictly less than one. In the second approach, the alternative is held fixed as the number of data points grows; in this case error probabilities go to zero and large deviation methods are used to assess the rate at which error probabilities go to zero. Aspects of both of these approaches can be found in the property testing literature.

While in many cases the particular problems studied in the property testing literature are somewhat different from those typically studied

in the mathematical statistics literature, the work on testing properties of distributions (which is discussed shortly in Section 6.3) deals with problems that are similar (or even the same) as those studied in mathematical statistics.

We also note that there are several works with a mathematical statistics flavor that are related to property testing and appeared in the computational learning literature (e.g., [33, 112, 137]).

1.4 Topics and Organization

We start with some preliminaries, which include basic definitions and notations. The preliminaries also include a precise statement and proof of the simple but important observation that testing is no harder than learning.

In Section 3, we consider the first type of properties that were studied in the context of property testing: algebraic properties. These include testing whether a function is (multi-)linear and more generally whether it is a polynomial of bounded degree. This work has implications to coding theory, and some of the results played an important role in the design of Probabilistically Check Proof (PCP) systems.

In Section 4, we turn to the study of function class that have a concise (propositional logic) representation such as singletons, monomials, and small DNF formula. This section includes a general result that applies to many classes of functions, where the underlying idea is that testing is performed by *implicit* learning.

The results in Sections 3 and 4 are in the *standard* model of testing. That is, the underlying distribution is uniform and the algorithm may perform queries to the function. In Section 5, we discuss distribution-free testing, and testing from random examples alone.

Finally, in Section 6, we give a more brief survey of other results in property testing. These include testing monotonicity, testing of clustering, testing properties of distributions, and more.

2

Preliminaries

2.1 Definitions and Notations

For any positive integer k, let $[k] = \{1, \ldots, k\}$. For a string $x = x_1, \ldots, x_n \in \{0,1\}^n$, we use $|x|$ to denote the number of indices i such that $x_i = 1$. We use "$\cdot$" to denote multiplication (e.g., $a \cdot b$) whenever we believe it aids readability.

For two functions $f, g : X \to R$ over a finite domain X and a distribution D over X, we let

$$\mathrm{dist}_D(f,g) \stackrel{\mathrm{def}}{=} \Pr_{x \in X}[f(x) \neq g(x)] \tag{2.1}$$

denote the distance between the functions according to the underlying distribution D. Since we mostly deal with the uniform distribution U, we use the shorthand $\mathrm{dist}(f,g)$ for $\mathrm{dist}_U(f,g)$.

When we use the term "with high probability," we mean with probability at least $1 - \delta$ for a small constant δ. When the claim is for higher success probability (e.g., $1 - \mathrm{poly}(1/n)$, where n is the input size), then this is stated explicitly. When considering the probability of a certain event we usually denote explicitly over what the probability is taken (e.g., $\Pr_{x \in X}[f(x) \neq g(x)]$), unless it is clear from the context (in which case we may write $\Pr[f(x) \neq g(x)]$).

Let $\mathcal{P}$ be a *property* of functions (from domain X to range R). That is, $\mathcal{P}$ defines a subset of functions, and so we shall use the notation $g \in \mathcal{P}$ to mean that function g has the property $\mathcal{P}$. For a function $f : X \to R$ and a distribution D over X, we let

$$\text{dist}_D(f, \mathcal{P}) = \min_{g \in \mathcal{P}} \{\text{dist}_D(f, g)\}, \tag{2.2}$$

(where there may be more than one function g that attains the minimum on the right-hand side). If $\text{dist}_D(f, \mathcal{P}) = \epsilon$, then we shall say that f is *at distance ϵ from (having) $\mathcal{P}$* (or *has distance ϵ to $\mathcal{P}$*).

What we shall refer to as *standard* testing, assumes D is the uniform distribution over the domain, and allows queries, as is defined precisely next.

Definition 2.1 (Standard Testing). A (standard) testing algorithm for property $\mathcal{P}$ (of functions from domain X to range R) is given a distance parameter ϵ and query access to an unknown function $f : X \to R$.

- If $f \in \mathcal{P}$ then the algorithm should accept with probability at least $2/3$;
- If $\text{dist}(f, \mathcal{P}) > \epsilon$ then the algorithm should reject with probability at least $2/3$.

We shall be interested in bounding both the query complexity and the running time of the testing algorithm. In some cases our focus will be on the query complexity, putting aside the question of time complexity. This can be seen as analogous to the study of sample complexity bounds in learning theory. We observe that the choice of a success probability of $2/3$ is arbitrary and can clearly be improved to $1 - \delta$, for any $\delta > 0$ at a multiplicative cost of $\log(1/\delta)$ in the complexity of the algorithm. We say that a testing algorithm has *one-sided error* if it accepts every $f \in \mathcal{P}$ with probability 1. Otherwise, it has *two-sided error*.

Variants of standard testing. In the spirit of PAC Learning (see Section 2.2 for a formal definition), we consider two variations/

generalizations of Standard Testing. In the *Distribution-free* testing model with queries there is an unknown underlying distribution D over X. The testing algorithm is given access to examples $x \in X$ distributed according to D in addition to its query access to f. As in the standard model, if $f \in \mathcal{P}$ then the algorithm should accept with probability[1] at least 2/3. The difference is that the algorithm should reject (with probability at least 2/3) if $\text{dist}_D(f, \mathcal{P}) > \epsilon$. In the model of *testing from random examples*, the algorithm is not given query access to the function f but is only given random examples labeled by f. The random examples are either distributed according to the uniform distribution, or, more generally, according to some unknown distribution D.

2.2 A Basic Observation on the Relation Between Learning and Testing

Our starting point is that, roughly speaking, testing is easier than learning. The result for testing is in the model that corresponds to the learning result (i.e., with or without queries, and distribution-free vs. distribution specific). In particular, if the learning model allows queries and the underlying distribution is uniform, then the corresponding testing model is the standard one from Definition 2.1. We first recall what a *Probably Approximately Correct* (*PAC*) [134] learning algorithm is, and in particular what is a *proper* learning algorithm.

In what follows a *representation class* is a set of functions together with a language for describing the functions in the set. In the context of efficient learning it is assumed that there is an efficient procedure that, given a string in the language representing a function f, outputs a circuit for computing f. Here, we also assume that membership in the representation language is efficiently decidable. Usually the language is not specified explicitly since it is clear that it is straightforward to construct such a language and we only refer to the function class (e.g., monomials (conjunctions)).

[1] An alternative definition would require that the algorithm accept (with high probability) if $\text{dist}_D(f, \mathcal{P}) = 0$. We adopt the requirement that $f \in \mathcal{P}$ since the known results are under this definition.

Definition 2.2 (PAC Learning (including variants)). A learning algorithm L for a representation class $\mathcal{F}$ using a representation (hypothesis) class $\mathcal{H}$ is given parameters $0 < \epsilon, \delta \leq 1$ and access to points distributed according to a fixed distribution D and labeled by an unknown function $f \in \mathcal{F}$. The algorithm should output a hypothesis $h \in \mathcal{H}$ such that with probability at least $1 - \delta$, $\text{dist}_D(h, f) \leq \epsilon$, where the probability is taken over the choice of the sample points and possibly the internal coin flips of L.

If D is unknown then the algorithm is distribution-free, while if D is known then it is distribution specific. A special case of interest is when D is the uniform distribution. The algorithm may also be allowed to perform queries of the form: "what is $f(x)$" for any x of its choice, in which case we refer to the learning model as learning with queries. Finally, if $\mathcal{H} = \mathcal{F}$ then the algorithm is a proper learning algorithm.

Note that the learning algorithm works under the *promise* that the examples it gets are indeed labeled by a fixed function[2] $f \in \mathcal{F}$. If the examples are not labeled by a function $f \in \mathcal{F}$ then nothing can be assumed about the output of the algorithm (and it may even halt without an output or not halt).

Proposition 2.1. If a function class $\mathcal{F}$ has a proper learning algorithm L, then $\mathcal{F}$ has a property testing algorithm T with sample complexity

$$m_T(n, \epsilon) = m_L(n, \epsilon/2) + O(1/\epsilon),$$

where $m_L(\cdot, \cdot)$ is the sample complexity of L as a function of the input size n and the error parameter ϵ when executed with the confidence parameter δ set to $1/6$. If L is also allowed queries then the query complexity of T is the same as that of L.[3] The running time $t_T(n, \epsilon)$ of the testing algorithm satisfies $t_T(n, \epsilon) = t_L(n, \epsilon/2) + O(1/\epsilon)t_E(n)$,

[2] This is as opposed to what is known as *agnostic* learning [107] (or learning in the unrealizable case) where no such assumption is made. In such a case the distance of the output of the algorithm from f should (with high probability) be at most ϵ larger than the minimum distance that any function in $\mathcal{H}$ has from f.

[3] When working with the standard testing model, that is, under the uniform distributions and with queries, one usually does not distinguish between sample points, which are viewed

where $t_L(\cdot,\cdot)$ is the running time of the learning algorithm, and $t_E(\cdot)$ is the maximum over all $g \in \mathcal{F}$ of the time it takes to evaluate g on a given input.

We note that a certain claim in the reverse direction (from a form of weak testing to weak learning) is presented in Proposition 5.3.

Proof. In order to test if $f \in \mathcal{F}$ or is ϵ-far from any function in $\mathcal{F}$, we first run the learning algorithm L with confidence parameter $\delta = 1/6$, and accuracy parameter $\epsilon/2$, using random examples labeled by f (and possibly queries if L is allowed to perform queries). If L does not output a hypothesis or outputs a hypothesis that is not in $\mathcal{F}$, or L passes the upper bound on its query complexity or running time, then we reject f. Otherwise (L outputs a hypothesis $h \in \mathcal{F}$), we approximate the distance between h and f by uniformly and independently drawing an additional (labeled) sample $S = \{(x^i, f(x^i)\}_{i=1}^m$ of $m = \Theta(1/\epsilon)$ points and consider the empirical error of h on S, that is, $\hat{\epsilon}_S(h) \stackrel{\text{def}}{=} \frac{1}{m}|\{1 \leq i \leq m : h(x^i) \neq f(x^i)\}|$. If $\hat{\epsilon}_S(h) \leq 3\epsilon/4$ then we accept, otherwise we reject.

In case $f \in \mathcal{F}$, with probability at least 5/6, L's output, h, is $(\epsilon/2)$-close to f. Conditioned on this event, we next show that by a multiplicative Chernoff bound [53] (see Appendix A), with probability at least 5/6, $\hat{\epsilon}_S(h) \leq 3\epsilon/4$. This implies that if $f \in \mathcal{F}$ then with probability at least 2/3 f is accepted. To verify the bound on $\hat{\epsilon}_S(h)$, let $\chi_1, \ldots, \chi_m$ be 0/1 random variables, where $\chi_i = 1$ if and only if $h(x^i) \neq f(x^i)$ (recall that x^i denotes the i^{th} point in the additional sample S). By the definition of χ_i, $\Pr_S[\chi_i = 1] = \text{dist}(f,h) \leq \epsilon/2$ and $\hat{\epsilon}_S(h) = \frac{1}{m}\sum_{i=1}^m \chi_i$. Since the probability that $\hat{\epsilon}_S(h) > 3\epsilon/4$ increases with $\text{dist}(f,h)$, we may assume without loss of generality that $\text{dist}(f,h) = \epsilon/2$ (rather than $\text{dist}(f,h) \leq \epsilon/2$). By Theorem A.1 (setting $p = \epsilon/2$ and $\gamma = 1/2$),

$$\Pr_S[\hat{\epsilon}_S(h) > 3\epsilon/4] = \Pr_S\left[\frac{1}{m}\sum_{i=1}^m \chi_i > (1+1/2)\text{dist}(f,h)\right] \tag{2.3}$$

$$< \exp(-(1/2)^2(\epsilon/2)m/3) \leq 1/6, \tag{2.4}$$

where the last inequality holds for $m \geq 72/\epsilon$.

---

as uniformly and independently selected queries, and queries that are selected in a different manner. However, since Proposition 2.1 is stated for more general models, the distinction is made.

In case f is ϵ-far from $\mathcal{F}$, the hypothesis $h \in \mathcal{F}$ that is output by L is at least ϵ-far from f. If we define $\chi_1, \ldots, \chi_m$ as in the foregoing discussion, then by Theorem A.1,

$$\Pr_S[\hat{\epsilon}_S(h) \leq 3\epsilon/4] \leq \Pr_S\left[\frac{1}{m}\sum_{i=1}^{m}\chi_i < (1-1/4)\mathrm{dist}(f,h)\right] \tag{2.5}$$

$$< \exp(-(1/4)^2\epsilon m/2) \leq 1/3, \tag{2.6}$$

where the last inequality holds for $m \geq 64/\epsilon$. Therefore, with probability at least $2/3$ over the additional sample S, f is rejected. □

Observe that it was crucial that the learning algorithm L be a proper learning algorithm. If L is not a proper learning algorithm then it is possible that f is ϵ-far from $\mathcal{F}$ (so that the promise that $f \in \mathcal{F}$ is violated) but still L outputs a hypothesis $h \in \mathcal{H}$ that is ϵ-close to f.

Thus, our focus is on designing testing algorithms that are strictly more efficient than the corresponding learning algorithms. We note that, as opposed to the situation with respect to learning, having a testing algorithm for a class of functions $\mathcal{F}$ does *not* imply that we have a testing algorithm for $\mathcal{F}' \subset \mathcal{F}$.

3

Algebraic Properties

In this section, we survey results on testing some algebraic families (properties) of functions: linear functions and more generally, low-degree polynomials. These families were first studied in the context of Program Testing, and played an important role in the design of *Probabilistically Checkable Proof* (*PCP*) systems. As we shall see, there is a similar underlying structure in all proofs. The main results mentioned in this section are summarized in Table 3.1.

We note that the results described in this section also have an interpretation from the point of view of coding theory. Namely, each of the properties (function classes) correspond to a code (or family of codes): The Hadamard code, Reed–Solomon codes, Reed Muller codes, and Generalized Reed Muller codes. If we view functions as words (e.g., for the domain $\{0,1\}^n$, the word is of length 2^n), then the test distinguishes between codewords and words that are ϵ-far from every codeword. This is referred to as *local testing of codes* (see, e.g., [80]).

3.1 Linearity

For the sake of simplicity we consider functions from $\{0,1\}^n$ to $\{0,1\}$. The result extends to functions $f : G \to H$, where G and H are groups.

Table 3.1 Results for testing polynomial function classes over a finite field F. Unless stated otherwise, all function classes consist of multivariate functions (that is, functions of the form $f : F^n \to F$). We note that by building on [21] it is possible to obtain a linear dependence on d in the case of degree-d polynomials and sufficiently large fields [131].

Class of functions	Number of queries	References
linear functions	$O(1/\epsilon)$	[42]
univariate deg-d Polynomials	$O(d + 1/\epsilon)$	[128]
deg-d polynomials, $\lvert F\rvert \geq d + 2$	$O(\text{poly}(d)/\epsilon)$	[128]
deg-d polynomials, $\lvert F\rvert = 2$	$O\left(\frac{1}{\epsilon} + d2^{2d}\right)$	[12]
	$\Omega\left(\frac{1}{\epsilon} + 2^d\right)$	
deg-d polynomials, $\lvert F\rvert = p^s$	$O(1/\epsilon + \ell\lvert F\rvert^{2\ell+1})$	[99, 103]
	$\Omega(1/\epsilon + \lvert F\rvert^{\ell-1})$	
	$(\ell = \lceil \frac{d+1}{\lvert F\rvert - \lvert F\rvert/p} \rceil)$	
s-sparse polynomials	$\tilde{O}((s\lvert F\rvert)^4/\epsilon^2)$	[61]
	$\tilde{\Omega}(\sqrt{s})$ $(\lvert F\rvert = O(1))$	

Thus addition is modulo 2, and for $x, y \in \{0,1\}^n$, $x + y$ is the bitwise sum (XOR) of the two strings, that is, it is the string $z \in \{0,1\}^n$ such that $z_i = x_i + y_i$.

Definition 3.1 (Linearity). We say that $f : \{0,1\}^n \to \{0,1\}$ is a linear function if there exist coefficients $b_1, \ldots, b_n \in \{0,1\}$ such that for $x = x_1, \ldots, x_n \in \{0,1\}^n$, $f(x) = \sum_{i=1}^n b_i x_i$. In other words, there exists a subset $S \subseteq \{1, \ldots, n\}$ such that $f(x) = \sum_{i \in S} x_i$.

Linearity testing is essentially the first property testing problem studied, though the term "Property Testing" was not yet explicitly defined at the time. Linearity testing was first studied by Blum, Luby and Rubinfeld [42] in the context of *Program Testing*. Namely, they were interested in designing algorithms (program-testers) that, given access to a program that is supposed to compute a particular function f, distinguish between the case that the program computes f correctly on all inputs, and the case that it errs on at least a certain fraction ϵ of the domain elements. The program-tester should be *much simpler* than the program itself, and is typically based on calls to the program and some basic operations on the resulting outputs.

In the case of testing whether a program computes a particular linear function, the program-tester first distinguishes between the case

that the program computes *some* linear function and the case that the function it computes is far from any linear function. That is, it first performs property testing of linearity. The starting point of the BLR linearity test is the following characterization of linear functions, which is not hard to verify (and some would actually use it as a definition of linear functions).

Fact 3.1. A function $f : \{0,1\}^n \to \{0,1\}$ is linear if and only if $f(x) + f(y) = f(x + y)$ for every $x, y \in \{0,1\}^n$.

The BLR test is described next.

Algorithm 3.1 (Linearity Test).

(1) Repeat the following $\Theta(1/\epsilon)$ times.
   (a) Uniformly and independently select $x, y \in \{0,1\}^n$.
   (b) If $f(x) + f(y) \neq f(x + y)$ then output reject (and exit).

(2) If no iteration caused rejection then output accept.

Before we prove the correctness of the algorithm, we remark on its complexity: the algorithm performs only $O(1/\epsilon)$ queries. In particular, its query complexity is *independent* of n. This is in contrast to the query complexity of any learning algorithm for the class of linear (*parity*) functions, which is $\Omega(n)$. This is true simply because every two linear functions have distance $1/2$ between them (under the uniform distribution), and a linear function is not uniquely determined by fewer than n labeled points. We note that the difference in the running time between testing and learning is less dramatic (linear in n versus quadratic in n), since the testing algorithm reads all n bits of each sampled string.

Theorem 3.1. Algorithm 3.1 is a one-sided error testing algorithm for linearity. Its query complexity is $O(1/\epsilon)$.

Let $\mathcal{L}$ denote the class of linear functions over $\{0,1\}^n$. By Fact 3.1, Algorithm 3.1 accepts every function $f \in \mathcal{L}$ with probability 1. We turn to proving that if $\mathrm{dist}(f,\mathcal{L}) > \epsilon$ then the algorithm rejects with probability at least 2/3. Let $\epsilon_{\mathcal{L}}(f)$ denote the distance of f to being linear. Namely, $\epsilon_{\mathcal{L}}(f) \stackrel{\text{def}}{=} \mathrm{dist}(f,\mathcal{L})$. We would like to prove that for every given $\epsilon > 0$, if $\epsilon > \epsilon_{\mathcal{L}}(f)$ then the probability that the test rejects is at least 2/3. This will follow from showing that if the constraint $f(x) + f(y) = f(x+y)$ is violated for relatively few pairs (x,y), then f is close to some linear function. In other words (using the terminology of [42, 128]), the characterization provided by Fact 3.1 is *robust*. To this end we define:

$$\eta(f) \stackrel{\text{def}}{=} \mathrm{Pr}_{x,y}[f(x) + f(y) \neq f(x+y)] \,, \tag{3.1}$$

where in Equation (3.1) and elsewhere in this subsection, the probability is taken over a uniform choice of points in $\{0,1\}^n$. That is, $\eta(f)$ is the probability that a single iteration of the algorithm "finds evidence" that f is not a linear function. We shall show that $\eta(f) \geq \epsilon_{\mathcal{L}}(f)/c$ for some constant $c \geq 1$ (this can actually be shown for $c = 1$ but the proof uses Discrete Fourier analysis [32] while the proof we show builds on first principles). It directly follows that if $\epsilon_{\mathcal{L}}(f) > \epsilon$ and the number of iterations is at least $2c/\epsilon$, then the probability that the test rejects is at least

$$1 - (1 - \eta(f))^{2c/\epsilon} > 1 - e^{-2c\eta(f)/\epsilon} \geq 1 - e^{-2} > 2/3 \,, \tag{3.2}$$

thus establishing Theorem 3.1.

Somewhat unintuitively, showing that $\eta(f) \geq \epsilon_{\mathcal{L}}(f)/c$ is easier if $\epsilon_{\mathcal{L}}(f)$ is not too large. Specifically, it is not hard to prove the following claim.

Claim 3.2. For every function f, $\eta(f) \geq 3\epsilon_{\mathcal{L}}(f)(1 - 2\epsilon_{\mathcal{L}}(f))$. In particular, if $\epsilon_{\mathcal{L}}(f) \leq \frac{1}{4}$ then $\eta(f) \geq \frac{3}{2}\epsilon(f)$ (and more generally, if $\eta(f) = \frac{1}{2} - \gamma$ for $\gamma > 0$, then $\eta(f) \geq 6\gamma\epsilon_{\mathcal{L}}(f)$, which gives a weak bound as $\eta(f)$ approaches 1/2).

It remains to prove that even when $\epsilon_{\mathcal{L}}(f)$ is not bounded away (from above) from 1/2 then still $\eta(f) \geq \epsilon_{\mathcal{L}}(f)/c$ for a constant c. To this end we define the following *majority* function: for each fixed choice of

$x \in \{0,1\}^n$, $g^f(x) = 0$ if $\Pr_y[f(x+y) - f(y) = 0] \geq 1/2$, and $g^f(x) = 1$ otherwise. Let

$$V_y^f(x) \stackrel{\text{def}}{=} f(x+y) - f(y) = f(y) + f(x+y) \tag{3.3}$$

be the *Vote* that y *casts* on the value of x. By the definition of $g^f(x)$ it is the majority vote taken over all y. Note that if f is linear then $V_y^f(x) = f(x)$ for every $y \in \{0,1\}^n$.

We shall prove two lemmas, stated next.

Lemma 3.3. $\text{dist}(f, g^f) \leq 2\eta(f)$.

Lemma 3.4. If $\eta(f) \leq \frac{1}{6}$ then g^f is a linear function.

By combining Lemmas 3.3 and 3.4 we get that $\eta(f) \geq \frac{1}{6}\epsilon_{\mathcal{L}}(f)$. To see why this is true, observe first that if $\eta(f) > \frac{1}{6}$, then the inequality clearly holds because $\epsilon_{\mathcal{L}}(f) \leq 1$. (In fact, since it can be shown that $\epsilon_{\mathcal{L}}(f) \leq 1/2$ for every f, we actually have that $\eta(f) \geq \frac{1}{3}\epsilon_{\mathcal{L}}(f)$.) Otherwise ($\eta(f) \leq \frac{1}{6}$), since g^f is linear and $\text{dist}(f, g^f) \leq 2\eta(f)$, we have that $\epsilon_{\mathcal{L}}(f) \leq \text{dist}(f, g^f) \leq 2\eta(f)$, so that $\eta(f) \geq \epsilon_{\mathcal{L}}(f)/2$, and we are done.

Proof of Lemma 3.3: Let $\Delta(f, g^f) = \{x : g^f(x) \neq f(x)\}$ be the set of points on which f and g^f differ. By the definition of $g^f(x)$, it is the majority value of $V_y^f(x)$ taken over all y. Hence, for every fixed choice of $x \in \Delta(f, g^f)$, we have that $\Pr_y[V_y^f(x) \neq f(x)] \geq 1/2$. Therefore,

$$\begin{aligned}
&\Pr_{x,y}[f(x) \neq V_y^f(x)] \\
&\quad \geq \Pr_x[x \in \Delta(f, g^f)] \cdot \Pr_y[f(x) \neq V_y^f(x) \,|\, x \in \Delta(f, g^f)] \\
&\quad \geq \frac{1}{2}\Pr_x[g^f(x) \neq f(x)] \,.
\end{aligned} \tag{3.4}$$

Since $\Pr_{x,y}[f(x) \neq V_y^f(x)] = \eta(f)$, it must hold that $\Pr_x[g^f(x) \neq f(x)] \leq 2\eta(f)$. □

Proof of Lemma 3.4: In order to prove this lemma, we first prove the next claim.

Claim 3.5. For every $x \in \{0,1\}^n$ it holds that $\Pr_y[g^f(x) = V_y^f(x)] \geq 1 - 2\eta(f)$.

Note that by the definition of g^f as the "majority-vote function," $\Pr_y[g^f(x) = V_y^f(x)] \geq \frac{1}{2}$. Claim 3.5 says that the majority is actually "stronger" (for small $\eta(f)$).

Proof. Fixing x, let $p_0(x) = \Pr_y[V_y^f(x) = 0]$, and let $p_1(x) = \Pr_y[V_y^f(x) = 1]$. We are interested in lower bounding $p_{g^f(x)}(x)$, where, by the definition of g^f, $p_{g^f(x)}(x) = \max\{p_0(x), p_1(x)\}$. Now,

$$p_{g^f(x)}(x) = p_{g^f(x)}(x) \cdot (p_0(x) + p_1(x)) \geq (p_0(x))^2 + (p_1(x))^2 \,. \quad (3.5)$$

Since $(p_0(x))^2 + (p_1(x))^2 = \Pr_{y,z}[V_y^f(x) = V_z^f(x)]$, in order to lower bound $p_{g^f(x)}(x)$, it suffices to lower bound $\Pr_{y,z}[V_y^f(x) = V_z^f(x)]$, which is what we do next.

In what follows we shall use the fact that the range of f is $\{0,1\}$.

$$\begin{aligned}
&\Pr_{y,z}[V_y^f(x) = V_z^f(x)] \\
&\quad = \Pr_{y,z}[V_y^f(x) + V_z^f(x) = 0] \\
&\quad = \Pr_{y,z}[f(y) + f(x+y) + f(z) + f(x+z) = 0] \\
&\quad = \Pr_{y,z}[f(y) + f(x+z) + f(y+x+z) \\
&\qquad + f(z) + f(x+y) + f(z+x+y) = 0] \\
&\quad \geq \Pr_{y,z}[f(y) + f(x+z) + f(y+x+z) = 0 \\
&\qquad \wedge f(z) + f(x+y) + f(z+x+y) = 0] \\
&\quad = 1 - \Pr_{y,z}[f(y) + f(x+z) + f(y+x+z) = 1 \\
&\qquad \vee f(z) + f(x+y) + f(z+x+y) = 1] \\
&\quad \geq 1 - \big(\Pr_{y,z}[f(y) + f(x+z) + f(y+x+z) = 1] \\
&\qquad + \Pr_{y,z}[f(z) + f(x+y) + f(z+x+y) = 1]\big) \\
&\quad = 1 - 2\eta(f).
\end{aligned}$$

□

In order to complete the proof of Lemma 3.4, we show that for any two given points $a, b \in \{0,1\}^n$, $g^f(a) + g^f(b) = g^f(a+b)$. We prove this by the probabilistic method. Specifically, we show that there exists a point y for which the following three equalities hold simultaneously:

(1) $g^f(a) = f(a+y) - f(y)$ $(= V_y^f(a))$.
(2) $g^f(b) = f(b + (a+y)) - f(a+y)$ $(= V_{a+y}^f(b))$.
(3) $g^f(a+b) = f(a+b+y) - f(y)$ $(= V_y^f(a+b))$.

But in such a case,

$$g^f(a) + g^f(b) = f(b + a + y) - f(y) = g^f(a + b) \,, \tag{3.6}$$

and we are done. To see why there exists such a point y, consider selecting y uniformly at random. For each of the three equalities, by Claim 3.5, the probability that the equality does not hold is at most $2\eta(f)$. By the union bound, the probability (over a uniform selection of y) that any one of the three does not hold is at most $6\eta(f)$. Since $\eta(f) < 1/6$, this is bounded away from 1, and so the probability that there *exists* a point y for which all three equalities hold simultaneously is greater than 0, implying that there exists at least one such pair. □

3.1.1 Self-Correction

One of the nice features of the analysis of the linearity tester (which we shall use in two different contexts, in Subsection 4.1.1 and in Section 5.1) is that it implies that f can be *self-corrected* (assuming it is sufficiently close to being linear). That is, for any x of our choice, if we want to know the value, on x, of the closest linear function (or, in the coding theory view, we want to know the correct bit in the position corresponding to x in the closest code-word), then we do the following. We select, uniformly at random, $y_1, \ldots, y_t$ and take the majority vote of $V^f_{y_1}(x), \ldots, V^f_{y_t}(x)$ (where the choice of t determines the probability that the majority is correct).

3.1.2 On the Relation Between $\eta(f)$ and $\epsilon_{\mathcal{L}}(f)$

By combining Claim 3.2 with Lemmas 3.3 and 3.4 we get that $\eta(f) \geq \max\{3\epsilon_{\mathcal{L}}(f)(1 - 2\epsilon_{\mathcal{L}}(f)), \frac{1}{6}\epsilon_{\mathcal{L}}(f)\}$. The analysis of [42] actually showed that the constant $\frac{1}{6}$ can be replaced with $2/9$.[1] An interesting question is what is the true behavior of $\eta(f)$ as a function of $\epsilon_{\mathcal{L}}(f)$? Bellare et al. [32] studied this question and showed several lower bounds, some of which are tight. One of the interesting phenomenon they observe is that the lower bound $3\epsilon_{\mathcal{L}}(f)(1 - 2\epsilon_{\mathcal{L}}(f))$ is tight as long as $\epsilon_{\mathcal{L}}(f) \leq 5/16$, implying that there is a nonmonotonic behavior (since

[1] We note that this holds for every $f : G \to H$, where G and H are any two groups, and not only for Boolean functions over the Boolean hypercube.

$3x(1-2x)$ increases until $x = 1/4$ but then *decreases* between $x = 1/4$ and $x = 5/16$). Recent progress was made on understanding the relation between $\eta(f)$ and $\epsilon_{\mathcal{L}}(f)$ in [102].

3.2 Low-Degree Polynomials

Let F be a finite field, and consider first the univariate case, that is, testing whether a function $f : F \to F$ is of the form $f(x) = \sum_{i=0}^{d} C_i^f x^i$ for a given degree bound d (where the coefficients C_i^f belong to F). In this case, the testing algorithm [128] works by simply trying to interpolate the function f on $\Theta(1/\epsilon)$ collections of $d+2$ uniformly selected points, and checking whether the resulting functions are all polynomial of degree at most d. Thus the algorithm essentially works by trying to learn the function f.[2]

We now turn to multivariate polynomials.

Definition 3.2 (Multivariate Polynomials). Let F be a finite field. A function $f : F^n \to F$ is a (multivariate) polynomial of degree (at most) d, if there exist coefficients C_α^f in F for every $\alpha \in \{0, \ldots, d\}^n$ satisfying $\sum_{i=1}^{n} \alpha_i \leq d$, such that $f(x) = \sum_{\alpha \in \{0,\ldots,d\}^n} C_\alpha^f \prod_{i=1}^{n} x_i^{\alpha_i}$.

Let $\mathrm{POLY}_{n,d}$ denote the class of all functions $f : F^n \to F$ that are polynomials of total degree at most d (where the degree in each variable is at most $|F| - 1$).

Low-degree testing has been studied extensively in the context of PCP systems [22, 23, 24, 66, 77, 78, 128]. The aforementioned results all apply to testing polynomials over fields that are larger than the degree-bound, d (where in some cases the bound is on the total degree, and in some case it is on the degree in each variable). In particular, if $|F| \geq d + 2$, then the dependence on d is known to be polynomial [77, 128].[3]

[2] In fact, a slightly more efficient version of the algorithm would select $d+1$ arbitrary points, find (by interpolating), the unique polynomial g^f of degree d that agrees with f on these points, and then check that g^f agrees with f on an additional sample of $\Theta(1/\epsilon)$ uniformly selected points.

[3] To be precise, the requirement that $|F| \geq d + 2$ is sufficient for prime fields, and otherwise needs to be slightly modified.

By combining [128] with [21] it is possible to get complexity that is only linear in d for sufficiently large fields [131].

Alon et al. [12] studied low-degree testing for the case $|F| = 2$ and $d \geq 2$. Namely, they considered the case in which the degree-bound may be (much) larger than the field size, for $F = GF(2)$. They showed that the number of queries both necessary and sufficient in this case is *exponential* in d. Hence, we encounter a very large gap in terms of the dependence on d between the query complexity when $|F| > d$ and the query complexity when $|F| = 2$. In [99, 103] this gap is bridged. We give more precise details in Subsection 3.2.2, but for now we state an approximate version of the result for the case that F is a prime field: there is a testing algorithm whose complexity grows roughly like $|F|^{2(d+1)/(|F|-1)}$. Thus, for any degree d, as the field size $|F|$ increases, the exponent $2(d+1)/(|F|-1)$ decreases (from $O(d)$ when $|F| = 2$ to $O(1)$ when $|F| = d + 3$).

Results for testing sparse polynomials (where there is a dependence on the number of terms rather than the degree) were obtained by Diakonikolas et al. [61] using techniques that are discussed in Section 4.3. They show that polynomials with at most s terms can be tested using $\tilde{O}((s|F|)^4/\epsilon^2)$ queries and that $\tilde{\Omega}(\sqrt{s})$ queries are necessary for $|F| = O(1)$. They also show that $\tilde{O}(s^4|F|^3/\epsilon^2)$ queries are sufficient for size-s algebraic circuits and computation trees over F.

The problem of learning polynomials in various learning models (including using only membership queries, i.e., interpolation), has been considered in many papers (e.g., [31, 35, 38, 41, 49, 51, 55, 76, 89, 115, 127, 129, 138, 139]). In particular, when allowed membership queries and equivalence queries (which implies learnability in the PAC model with membership queries), Beimel et al. [31] show that learning n-variate polynomials with s terms over $GF(p)$ can be done with query complexity and running time $\text{poly}(n, s, p)$. When the field is large (possibly even infinite), then the dependence on the field size can be replaced with a polynomial dependence on the degree d. The running time was recently improved in the work of Bisht et al. [38].

In the next subsection, we describe a low-degree test for large fields and in the following subsection we turn to the general case.

3.2.1 The Case of Large Fields

In this subsection, we describe an algorithm of Rubinfeld and Sudan [128], where as in [128] we consider the case that F is the prime field $GF(p)$. The idea of the algorithm is to select a random *line* in F^n, defined by two (random) points $x, h \in F^n$, and to verify that the restriction of f to this line is a (univariate) polynomial of degree at most d. To be precise, the algorithm does not query all points on the line, but rather $d+2$ evenly spaced points. The algorithm is based on the following characterization of degree-d polynomials, which was proved in [77] (improving on what is shown in [128] where the requirement was that $|F| \geq 2d+1$).

Theorem 3.6. Let d be an integer and let F be a prime field such that $|F| \geq d+2$. A function $f : F^n \to F$ belongs to $\mathrm{POLY}_{n,d}$ if and only if for all $x, h \in F^n$, $\sum_{i=0}^{d+1} \tau_i f(x + i \cdot h) = 0$, where $\tau_i = (-1)^{i+1}\binom{d+1}{i}$.

We note that if F is not a prime field but rather $|F| = p^s$ for some prime p and integer $s > 1$, then the requirement on the size of the field is that $|F| \geq (d+1)(p/(p-1))$.

Algorithm 3.2 (Low-Degree Test (for $|F| \geq d+2$)).

(1) Repeat the following $\Theta(1/\epsilon + d^2)$ times.
   (a) Uniformly and independently select $x, y \in F^n$.
   (b) If $\sum_{i=0}^{d+1} \tau_i f(x + i \cdot y) \neq 0$ (where the coefficients τ_i are as defined in Theorem 3.6), then output reject (and exit).

(2) If no iteration caused rejection then output accept.

The query complexity and running time of the algorithm are $O(d/\epsilon + d^3)$. As noted previously, by combining [128] with [21] it is possible to get complexity that is only linear in d for sufficiently large fields [131].

Theorem 3.7. Assuming $|F| \geq d+2$, Algorithm 3.2 is a one-sided error testing algorithm for $\mathrm{POLY}_{n,d}$. Its query complexity is $O(d/\epsilon + d^3)$.

By Theorem 3.6, if $f \in \mathrm{POLY}_{n,d}$ (and $|F| \geq d+2$), then the test accepts with probability 1. It remains to prove that if f is ϵ-far from any degree-d polynomial, then the test rejects with probability at least 2/3. Here too this follows from proving that the characterization in Theorem 3.6 is robust.

For any $x, y \in F^n$, let

$$V_y^f(x) \stackrel{\text{def}}{=} \sum_{i=1}^{d+1} \tau_i f(x + i \cdot y) \,. \tag{3.7}$$

Recalling that $\tau_i = (-1)^{i+1}\binom{d+1}{i}$ so that $\tau_0 = -1$, the condition $\sum_{i=0}^{d+1} \tau_i f(x + i \cdot y) = 0$ is equivalent to $-f(x) + V_y^f(x) = 0$, that is $f(x) = V_y^f(x)$. Thus, analogously to the analysis of the linearity test, $V_y^f(x)$ is the value that $f(x)$ "should have" so that the condition $\sum_{i=0}^{d+1} \tau_i f(x + i \cdot y) = 0$ holds. In all that follows, probabilities are taken over points in F^n. Let

$$\eta(f) \stackrel{\text{def}}{=} \Pr_{x,y}[f(x) \neq V_y^f(x)] \tag{3.8}$$

denote the probability that the test rejects in a single iteration of the algorithm. We now define the function $g^f : F^n \to F$ as follows:

$$g^f(x) \stackrel{\text{def}}{=} \mathrm{plurality}_{y \in F^n}\{V_y^f(x)\}, \tag{3.9}$$

where the plurality value is simply the value that appears most often in the set $\{V_y^f(x)\}_{y \in F^n}$ (so that it generalizes the notion of the majority). For g^f as defined in Equation (3.9) (and $\eta(f)$ as defined in Equation (3.8)) we have:

Lemma 3.8. For any function f and for η and g^f as defined in Equations (3.8) and (3.9), respectively:

(1) $\mathrm{dist}(f, g^f) \leq 2\eta(f)$.
(2) If $\eta(f) \leq \frac{1}{2(d+2)^2}$, then g^f is a degree-d polynomial.

We prove the lemma momentarily, but first show why it follows that the algorithm rejects any function f that is ϵ-far from $\mathrm{POLY}_{n,d}$ with probability at least 2/3 (thus establishing Theorem 3.7). Let f be a function that is ϵ-far from $\mathrm{POLY}_{n,d}$. If $\eta(f) > \frac{1}{2(d+2)^2}$, then, for a sufficiently large constant in the $\Theta(\cdot)$ notation for the number of iterations of the algorithm, the algorithm rejects with high probability. Thus, assume that $\eta(f) \leq \frac{1}{2(d+2)^2}$. By the second item in Lemma 3.8, it follows that g^f is a degree-d polynomial. Since f is ϵ-far from any degree-d polynomial, in particular $\mathrm{dist}(f, g^f) > \epsilon$. By the first item of Lemma 3.8, we have that $\eta(f) > \epsilon/2$. Once again, for a sufficiently large constant in the $\Theta(\cdot)$ notation for the number of iterations of the algorithm, it rejects with high probability.

The proof of the first item in Lemma 3.8 is essentially the same as the proof of Lemma 3.3. In order to prove the second item we first prove the following claim. By the definition of g^f as the plurality function over $V_y^f(x)$, for every x we have that $g^f(x)$ agrees with $V_y^f(x)$ for at least $1/|F|$ of the choices of y. The claim shows (similarly to Claim 3.5) that there is actually a much larger agreement, assuming $\eta(f)$ is sufficiently small.

Claim 3.9. For every $x \in F^n$ we have that $\Pr_y[g^f(x) = V_y^f(x)] \geq 1 - 2(d+1)\eta(f)$.

Proof. By slightly extending the argument used in the proof of Claim 3.5, we can get that for every fixed choice of x, $\Pr_y[g^f(x) = V_y^f(x)] \geq \Pr_{y_1,y_2}[V_{y_1}^f(x) = V_{y_2}^f(x)]$, and hence we turn to lower bounding $\Pr_{y_1,y_2}[V_{y_1}^f(x) = V_{y_2}^f(x)]$.

Observe that if we select, uniformly at random, y_1 and y_2 in F^n, then for any choice of $1 \leq i, j \leq d+1$ we have that $x + i \cdot y_1$ and $x + j \cdot y_2$ are uniformly distributed in F^n. Therefore, by the definition of $\eta(f)$ we have that for each fixed choice of $1 \leq i, j \leq d+1$,

$$\Pr_{y_1,y_2}\left[f(x + i \cdot y_1) = \sum_{j=1}^{d+1} \tau_j f((x + i \cdot y_1) + j \cdot y_2)\right] \geq 1 - \eta(f) \tag{3.10}$$

and

$$\Pr_{y_1,y_2}\left[f(x+j\cdot y_2)=\sum_{i=1}^{d+1}\tau_i f((x+j\cdot y_2)+i\cdot y_1)\right]\geq 1-\eta(f)\,. \tag{3.11}$$

Therefore, with probability at least $1-2(d+1)\eta(f)$ over the choice of y_1 and y_2, we have that

$$\sum_{i=1}^{d+1}\tau_i f(x+i\cdot y_1)=\sum_{i=1}^{d+1}\tau_i\cdot\sum_{j=1}^{d+1}\tau_j f\big((x+i\cdot y_1)+j\cdot y_2\big) \tag{3.12}$$

$$=\sum_{j=1}^{d+1}\tau_j\cdot\sum_{i=1}^{d+1}\tau_i f\big((x+j\cdot y_2)+i\cdot y_1\big) \tag{3.13}$$

$$=\sum_{j=1}^{d+1}\tau_j f(x+j\cdot y_2). \tag{3.14}$$

In Equation (3.12) we applied Equation (3.10), in Equation (3.13) we simply reordered the summands, and to obtain Equation (3.14) we applied Equation (3.11). Thus, $\Pr_{y_1,y_2}[V^f_{y_1}(x)=V^f_{y_2}(x)]\geq 1-2(d+1)\eta(f)$, as desired. □

Proof of Lemma 3.8: As noted previously, the proof of the first item in Lemma 3.8 is essentially the same as the proof of Lemma 3.3. In order to prove the second item we show that if $\eta(f)\leq\frac{1}{2(d+2)^2}$, then $\sum_{j=0}^{d+1}\tau_j g^f(x+j\cdot y)=0$ for all $x,y\in F^n$. The second item of the lemma then follows from Theorem 3.6.

Consider any fixed choice of x and y. Observe that if we select, uniformly at random, two points, r_1 and r_2 in F^n, then for any choice of $0\leq j\leq d+1$, the point $r_1+j\cdot r_2$ is uniformly distributed in F^n. Therefore, by Claim 3.9, for any choice of $0\leq j\leq d+1$,

$$\Pr_{r_1,r_2}\left[g^f(x+j\cdot y)=V^f_{r_1+j\cdot r_2}(x+j\cdot y)\right]\geq 1-2(d+1)\eta(f). \tag{3.15}$$

That is,

$$\Pr_{r_1,r_2}\left[g^f(x+j\cdot y)=\sum_{i=1}^{d+1}\tau_i f((x+j\cdot y)+i\cdot(r_1+j\cdot r_2))\right]$$
$$\geq 1-2(d+1)\eta(f). \tag{3.16}$$

For our fixed choice of $x, y \in F^n$ and a uniformly selected $r_1, r_2 \in F^n$, we have that $x + i \cdot r_1$ and $y + i \cdot r_2$ are uniformly distributed in F^n for any $1 \leq i \leq d+1$. Therefore, by the definition of $\eta(f)$ we get that for every $1 \leq i \leq d+1$,

$$\Pr_{r_1,r_2}\left[\sum_{j=0}^{d+1} \tau_j f((x + i \cdot r_1) + j \cdot (y + i \cdot r_2)) = 0\right] \geq 1 - \eta(f). \tag{3.17}$$

Since $\eta(f) \leq \frac{1}{2(d+2)^2}$, there exists a choice of r_1 and r_2 such that for every $0 \leq j \leq d+1$,

$$\tau_j g^f(x + j \cdot y) = \tau_j \sum_{i=1}^{d+1} \tau_i f\big((x + j \cdot y) + i \cdot (r_1 + j \cdot r_2)\big) \tag{3.18}$$

$$= \sum_{i=1}^{d+1} \tau_i \tau_j f\big((x + i \cdot r_1) + j \cdot (y + i \cdot r_2)\big) \tag{3.19}$$

and for every $1 \leq i \leq d+1$,

$$\sum_{j=0}^{d+1} \tau_j f\big((x + i \cdot r_1) + j \cdot (y + i \cdot r_2)\big) = 0. \tag{3.20}$$

But this implies that

$$\sum_{j=0}^{d+1} \tau_j g^f(x + j \cdot y) = \sum_{i=0}^{d+1} \tau_i \sum_{j=0}^{d+1} \tau_j f\big((x + i \cdot r_1) + j \cdot (y + i \cdot r_2)\big) = 0, \tag{3.21}$$

as claimed. □

3.2.2 The General Case

Here we follow [103], who generalize the algorithm and the analysis of [128], described in Subsection 3.2.1, as well as that of [12] (for the special case of $|F| = 2$). A similar result, using different techniques, was obtained by Jutla et al. [99], where they focused on prime fields (the result in [103] holds also for nonprime fields).

A main building block of the analysis of the general case is the following characterization of degree-d multivariate polynomials over finite fields.

Theorem 3.10. Let $F = GF(q)$, where $q = p^s$ and p is prime. Let d be an integer, and let $f : F^n \to F$. The function f is a polynomial of degree at most d if and only if its restriction to every affine subspace of dimension $\ell = \left\lceil \frac{d+1}{q-q/p} \right\rceil$ is a polynomial of degree at most d.

Theorem 3.10 generalizes the characterization result of Friedl and Sudan [77] (a variant of which is stated in Theorem 3.6), which refers to the case $q - q/p \geq d + 1$. That is, the size of the field F is sufficiently larger than the degree d, and the affine subspaces considered are of dimension $\ell = 1$. As stated in Theorem 3.6, when $\ell = 1$ it suffices to verify that a certain condition holds for $d + 2$ points on the line (one-dimensional affine subspace).

We also note that this value, ℓ, of the dimension of the considered subspaces, is tight. Namely, there exist polynomials of degree greater than d whose restrictions to affine subspaces of dimension less than ℓ are all degree-d polynomials.

The testing algorithm we describe next utilizes the characterization in Theorem 3.10 (which is shown to be robust). Specifically, the algorithm selects random affine subspaces (of dimension ℓ as defined in Theorem 3.10), and checks that the restriction of the function f to each of the selected subspaces is indeed a polynomial of degree at most d. Such a check is implemented by verifying that various linear combinations of the values of f on the subspace sum to 0.

Before giving more details, we introduce some notations.

Definition 3.3. For $m \geq 1$ and any choice of a point $x \in F^n$ and m linearly independent points $y_1, \ldots, y_m \in F^n$, let $S(x, y_1, \ldots, y_m)$ denote the affine subspace of dimension m that contains all points of the form $x + \sum_{i=1}^{m} a_i y_i$, where $a_1, \ldots, a_m \in F$.

Definition 3.4. For a function $f : F^n \to F$, a point $x \in F^n$, and m linearly independent points $y_1, \ldots, y_m \in F^n$, we denote by $f_{|(x,y_1,\ldots,y_m)}$ the restriction of f to the affine subspace $S(x, y_1, \ldots, y_m)$. Namely,

$f_{|(x,y_1,\ldots,y_m)} : F^m \to F$ is defined as follows: for every $a \in F^m$, $f_{|(x,y_1,\ldots,y_m)}(a) = f(x + \sum_{i=1}^m a_i y_i)$.

With a slight abuse of notation (and for the sake of succinctness), we shall sometimes use the notation $f_{|S}$ instead of $f_{|(x,y_1,\ldots,y_m)}$, where $S = S(x, y_1, \ldots, y_m)$ is the subspace spanned by the points. In case the set of points spanning the subspace S is not explicitly stated, then $f_{|S}$ is determined by some canonical choice of a basis.[4]

Algorithm 3.3. **(Low-Degree Test (for[5] $|F| = O(d)$))**

(1) Let $\ell = \ell(q,d) = \left\lceil \frac{d+1}{q-q/p} \right\rceil$ and repeat the following $t = \Theta\left(\ell q^{\ell+1} + \frac{1}{\epsilon q^\ell}\right)$ times:
   (a) Uniformly at random select ℓ linearly independent points $y_1, \ldots, y_\ell \in F^n$, and a point $x \in F^n$.
   (b) If $f_{|(x,y_1,\ldots,y_\ell)} \notin \mathrm{POLY}_{\ell,d}$ then output reject (and exit).

(2) If no step caused rejection then output accept.

Checking whether $f_{|S} \notin \mathrm{POLY}_{\ell,d}$ (where $S = S(x, y_1, \ldots, y_\ell)$) can be done by querying f on all points in the subspace S and verifying that the points obey certain linear constraints. Specifically, we consider the standard and unique representation of the function $f_{|S}$ as a polynomial of degree at most $q-1$ in each variable. That is, for each $\alpha \in \{0, \ldots, q-1\}^\ell$, there is a coefficient $C_\alpha^{f_{|S}}$ such that $f_{|S}(a) = \sum_{\alpha \in \{0,\ldots,q-1\}^\ell} C_\alpha^{f_{|S}} \prod_{i=1}^\ell a_i^{\alpha_i}$ for every $a \in F^\ell$. The test checks whether $C_\alpha^{f_{|S}} = 0$ for every α satisfying $\sum_{i=1}^\ell \alpha_i > d$. Each coefficient $C^{f_{|S}}$ is a certain linear combination of the points in S. For example, the coefficient corresponding to the highest degree monomial (that is,

[4] Our interest lies in the *degree* of these functions (represented as polynomials). Since for any given subspace this degree is invariant with respect to the choice of the basis, the particular choice of the basis is only a matter of convenience.

[5] The test can be applied to any field size, but when $|F|$ is much larger than d, it is possible to use Algorithm 3.2, which has lower complexity.

$C^{f_{|S}}_{\langle q-1,\ldots,q-1\rangle}$), equals $(-1)^\ell \sum_{a\in F^\ell} f_{|S}(a)$. Hence the total number of queries performed by the algorithm is $O(tq^\ell) = O(\ell q^{2\ell+1} + \frac{1}{\epsilon})$ and the running time is at most q^ℓ times larger.

Theorem 3.11. Algorithm 3.3 is a one-sided error testing algorithm for $\mathrm{POLY}_{n,d}$. Its query complexity is $O(\ell q^{2\ell+1} + \frac{1}{\epsilon})$, where $\ell = \left\lceil \frac{d+1}{q-q/p} \right\rceil$.

If $f \in \mathrm{POLY}_{n,d}$ then clearly $f_{|S} \in \mathrm{POLY}_{\ell,d}$ for every affine subspace S of dimension ℓ and hence Algorithm 3.3 accepts with probability 1. We therefore turn to the case that $\mathrm{dist}(f, \mathrm{POLY}_{n,d}) > \epsilon$. Since the full analysis is fairly lengthy, we describe only the high level structure of the proof and the underlying ideas. As in the proofs of Theorems 3.1 and 3.7, we define a function g^f (a "corrected" version of f) and show that: (1) the distance between g^f and f is upper bounded by a function of the probability that the test rejects in a single iteration; (2) if this probability is not too large then g^f belongs to $\mathrm{POLY}_{n,d}$.

Let

$$\eta(f) \stackrel{\mathrm{def}}{=} \Pr_{x,y_1,\ldots,y_\ell}\left[f_{|(x,y_1,\ldots,y_\ell)} \notin \mathrm{POLY}_{\ell,d}\right], \tag{3.22}$$

where the probability is taken over x and $y_1,\ldots,y_\ell$ such that $y_1,\ldots,y_\ell$ are linearly independent points in F^n. By the definition of Algorithm 3.3, $\eta(f)$ is the probability that a single step of the algorithm causes f to be rejected. That is, it is the probability that the restriction of f to a random affine subspace $S = S(x,y_1,\ldots,y_\ell)$ of dimension ℓ is not a polynomial of degree at most d. As noted in the foregoing discussion, this is equivalent to the requirement that in the standard representation of $f_{|S}$ as a polynomial, all coefficients corresponding to monomials with total degree greater than d are 0.

In particular, by our choice of ℓ, this should be true of the coefficient of the highest degree monomial, in which all variables have the highest degree, $q-1$ (since $\ell(q-1) \geq d+1$). As noted previously, this coefficient equals $(-1)^\ell$ times the sum of the values of f taken over all points in the subspace. We denote by $V^f(x;y_1,\ldots,y_\ell)$ the value that

$f(x)$ "should have" so that this coefficient equal 0. That is,

$$V^f(x; y_1, \ldots, y_\ell) \stackrel{\text{def}}{=} -\sum_{v \in F^\ell \setminus \{\vec{0}\}} f\left(x + \sum_{i=1}^{\ell} v_i y_i\right). \tag{3.23}$$

We refer to $V^f(x; y_1, \ldots, y_\ell)$ as the *vote* of $(y_1, \ldots, y_\ell)$ on the value assigned to x. Note that for $\eta(f)$ as defined in Equation (3.22), $\eta(f) \geq \Pr_{x, y_1, \ldots, y_\ell}[V^f(x; y_1, \ldots, y_\ell) \neq f(x)]$ (where the probability is taken over $y_1, \ldots, y_\ell$ that are linearly independent). This is true since the test checks that *all* coefficients $C^f_\alpha(x, y_1, \ldots, y_\ell)$ for which $\sum_{i=1}^{\ell} \alpha_i > d$ are 0.

We are now ready to define the (self) corrected version of f, denoted by g^f.

$$g^f(x) \stackrel{\text{def}}{=} \text{plurality}_{y_1, \ldots, y_\ell \in F^n} \left\{ V^f(x; y_1, \ldots, y_\ell) \right\}. \tag{3.24}$$

Recall that the plurality value is the value that appears most often in the set $\{V^f(x; y_1, \ldots, y_\ell)\}$. We note that in the definition of g^f the plurality is taken over all (not necessarily linearly independent) ℓ-tuples $y_1, \ldots, y_\ell \in F^n$.

Lemma 3.12. For any function f and for $\eta(f)$ and g^f as defined in Equations (3.22) and (3.24), respectively:

(1) $\text{dist}(f, g^f) \leq 2\eta(f)$.
(2) If $\eta(f) < \frac{1}{2(\ell+1)q^{\ell+1}}$ then $g^f \in \text{POLY}_{n,d}$.

Similarly to the proof of Theorem 3.7, we can show that Lemma 3.12 implies that $\eta(f) \geq \min\left\{\frac{1}{2(\ell+1)q^{\ell+1}}, \epsilon/2\right\}$. This in turn implies the correctness of a slightly less efficient version of Algorithm 3.3, which performs $t = \Theta(\ell q^{\ell+1} + 1/\epsilon)$ iterations (rather than $t = \Theta\left(\ell q^{\ell+1} + \frac{1}{\epsilon q^\ell}\right)$ iterations). To prove that the number of iteration performed by Algorithm 3.3 suffices (from which Theorem 3.11 follows), there is a need for another technical lemma, which we omit.

The proof of the first item of Lemma 3.12 is essentially the same as the proof of Lemma 3.3. The proof of the second item is based on showing that for every x, the value of $g^f(x)$, which is the plurality vote

of $V^f(x; y_1, \ldots, y_\ell)$, taken over all $y_1, \ldots, y_\ell$, equals the vote of a large fraction of the ℓ-tuples $y_1, \ldots, y_\ell$ (assuming $\eta(f)$ is sufficiently small). Namely,

---

Claim 3.13. For any fixed $x \in F^n$, let

$$\gamma(x) \stackrel{\text{def}}{=} \Pr_{y_1,\ldots,y_\ell}\left[V^f(x; y_1, \ldots, y_\ell) = g(x)\right]. \tag{3.25}$$

Then $\gamma(x) \geq 1 - 2q\ell\eta(f)$.

Proof Sketch. In order to prove Claim 3.13 it will actually be more convenient to work with another measure of "correctness" (or "consistency") of a point x. Specifically, for any fixed $x \in F^n$, let

$$\delta(x) = \Pr_{y_1,\ldots,y_\ell,z_1,\ldots,z_\ell}\left[V^f(x; y_1, \ldots, y_\ell) = V^f(x; z_1, \ldots, z_\ell)\right]. \tag{3.26}$$

Similarly to what was shown in the proof of Claim 3.5, $\gamma(x) \geq \delta(x)$. Hence it suffices to obtain a lower bound on $\delta(x)$. In order to show that $\delta(x)$ is large (and hence $\gamma(x)$ is large), it will be useful to consider the following *auxiliary graph*. The definition of this graph was inspired by the way Shpilka and Wigderson used Cayley graphs in their work [130]. Each vertex in this graph is labeled by a subset (multiset) of ℓ points, $\{y_1, \ldots, y_\ell\}$, $y_i \in F^n$. The neighbors of $\{y_1, \ldots, y_\ell\}$ are of the form $\{y_2, \ldots, y_{\ell+1}\}$. Each vertex corresponds to ℓ points that can "vote" on the value of $f(x)$, for any given x and hence we refer to it as the *voting graph*. For a fixed point $x \in F^n$, we say that an edge between $\{y_1, \ldots, y_\ell\}$ and $\{y_2, \ldots, y_{\ell+1}\}$ is *good with respect to* x if $V^f(x; y_1, \ldots, y_\ell) = V^f(x; y_2, \ldots, y_{\ell+1})$.

For any (random) choice of $y_1, \ldots, y_\ell$ and $z_1, \ldots, z_\ell$, and for each $0 \leq i \leq \ell$, let $v_i = \{y_1, \ldots, y_i, z_{i+1}, \ldots, z_\ell\}$, where we view v_i as a vertex in the voting graph. In particular, $v_\ell = \{y_1, \ldots, y_\ell\}$ and $v_0 = \{z_1, \ldots, z_\ell\}$. Since $y_1, \ldots, y_\ell, z_1, \ldots, z_\ell$ are selected uniformly at random, each v_i is a random variable. Consider the path $v_\ell, \ldots, v_0$ between v_ℓ and v_0.

In what follows we shall use the shorthand $V^f(x; v_i)$ for the vote $V^f(x; y_1, \ldots, y_i, z_{i+1}, \ldots, z_\ell)$. Recall that an edge (v_i, v_{i-1}) is good if $V^f(x; v_i) = V^f(x; v_{i-1})$. We claim that the probability (taken over the choice of $y_1, \ldots, y_\ell, z_1, \ldots, z_\ell$) that an edge (v_i, v_{i-1}) on the path

is not good is at most $2q\eta(f)$. By taking a union bound it follows that the probability that all the edges on the path are good is at least $1 - 2q\ell\eta(f)$. That is, with probability at least $1 - 2q\ell\eta(f)$, $V^f(x; y_1, \ldots, y_\ell) = V^f(x; y_1, \ldots, y_{\ell-1}, z_\ell) = \cdots = V^f(x; z_1, \ldots, z_\ell)$, and the lemma follows. Hence, it remains to show that an edge on the path is not good with probability at most $2q\eta(f)$.

Consider any edge (v_i, v_{i-1}). We say that $V^f(x; v_i)$ is an *independent vote* for x if $y_1, \ldots, y_i, z_{i+1}, \ldots, z_\ell$ are linearly independent points, otherwise we say that $V^f(x; v_i)$ is a *dependent vote* for x. For a dependent vote $V^f(x; v_i)$, it can be shown that $V^f(x; v_i) = f(x)$. Therefore, if both votes $V^f(x; v_i)$ and $V^f(x; v_{i-1})$ for x are dependent then $V^f(x; v_i) = V^f(x; v_{i-1})$ and the edge is good. If one of the votes is a dependent vote and the other is an independent vote then the probability that the edge is not good is the probability that an independent vote for y differs from $f(y)$, which is $\eta(f)$.

Finally, if $V^f(x; v_i)$ and $V^f(x; v_{i-1})$ are both independent votes then it can be shown that $V^f(x; v_i) - V^f(x; v_{i-1})$ is the sum, over $2(q-1)$ affine subspaces, $S_1, \ldots, S_{2q}$ (defined by x and $y_1, \ldots, y_i, z_i, \ldots, z_\ell$) of the largest degree coefficient of $f_{|S_j}$. For each subspace the probability that this coefficient is nonzero is at most $\eta(f)$, and so, by taking a union bound, the probability that $V^f(x; v_i) = V^f(x; v_{i-1})$ is at least $1 - 2q\eta(f)$, as claimed. □

Showing how Lemma 3.12 follows Claim 3.13 is somewhat technical, and hence we only provide the high-level idea. Consider any fixed set of points $x, y_1, \ldots, y_\ell \in F^n$ such that $y_1, \ldots, y_\ell$ are linearly independent. The goal is to show that $g^f_{|(x, y_1, \ldots, y_\ell)} \in \mathrm{POLY}_{\ell,d}$. The second item in Lemma 3.12 then follows by applying Theorem 3.10. By using a probabilistic argument it can be shown that there exists a choice of a subset of elements, denoted $\{z_{i,j}\}$, for which the following conditions hold. First, the value of g^f on every point w in the subspace $S(x, y_1, \ldots, y_\ell)$ equals the vote on w of a set, T_w, of ℓ points that are linear combinations of the $z_{i,j}$'s. Next, for each of these sets of points T_w, the restriction of f to the affine subspace defined by w and T_w, is a polynomial of degree at most d. That is, all high degree coefficients in each of these restrictions are 0. The next step is to show that each high degree coefficient in the restriction of g^f to the subspace $S(x, y_1, \ldots, y_\ell)$ is a linear combination

of the high degree coefficients in the abovementioned restrictions of f, and is hence 0.

A lower bound. By extending an argument given in [12], the following lower bound is proved in [103].

Theorem 3.14. Every algorithm for testing $\mathrm{POLY}_{n,d}$ with distance parameter ϵ must perform $\Omega\big(\max\{\frac{1}{\epsilon}, q^{\ell-1}\}\big)$ queries when q is prime, and $\Omega\big(\max\{\frac{1}{\epsilon}, q^{\lceil \ell/2 \rceil - 1}\}\big)$ queries otherwise.

Thus, a dependence on $q^{\Omega(\ell)}$ is unavoidable.

4

Basic (Boolean) Function Classes

In this section, we describe and analyze several algorithms for testing various families of Boolean functions over $\{0,1\}^n$. For all families, the query complexity of the algorithms is independent of n. This stands in contrast to the fact that for these families the number of queries required for learning (under the uniform distribution) depends on n. In some cases the results extend to more general domains and/or ranges, and this is noted when the result is discussed. The main results mentioned in this section are summarized in Table 4.1.

4.1 Singletons, Monomials, and Monotone DNF

In this subsection, we describe the results of [125]. We give full details for the simplest case of testing the class of singleton functions, and only the high-level ideas for the more complex problems of testing monomials and monotone DNF. The query complexity of the algorithms for testing singletons and for testing monomials is $O(1/\epsilon)$, and the query complexity of the algorithm for testing monotone DNF is $\tilde{O}(s^2/\epsilon)$, where s is an upper bound on the number of terms in the DNF formula. In Section 4.3, we shall describe a general result that in

Table 4.1 Results for testing basic Boolean functions over $\{0,1\}^n$ (or, in the case of linear threshold functions, over $\{+1,-1\}^n$).

Class of functions	Number of queries	Reference
singletons and monomials	$O(1/\epsilon)$	[125]
s-term monotone DNF	$\tilde{O}(s^2/\epsilon)$	[125]
k-Juntas	$\tilde{O}(k^2/\epsilon)$	[71]
	$\Omega(k)$	[54]
decision lists	$\tilde{O}(1/\epsilon^2)$	[61]
size-s decision trees,	$\tilde{O}(s^4/\epsilon^2)$	[61]
size-s branching programs,	$\Omega(\log s/\log\log s)$	
s-term DNF and		
size-s Boolean formulae		
s-sparse polynomials over $GF(2)$	$\tilde{O}(s^4/\epsilon^2)$	[61]
	$\tilde{\Omega}(\sqrt{s})$	
size-s Boolean circuits	$\tilde{O}(s^6/\epsilon^2)$	[61]
functions with Fourier degree $\leq d$	$\tilde{O}(2^{6d}/\epsilon^2)$	[61]
	$\tilde{\Omega}(\sqrt{d})$	
linear threshold functions	$\mathrm{poly}(1/\epsilon)$	[117]

particular implies that (general) DNF with s terms are testable using $\tilde{O}(s^4/\epsilon^2)$ queries.

Definition 4.1 (Singletons, Monomials, and DNF Functions). A function $f : \{0,1\}^n \to \{0,1\}$ is a singleton function if there exists an $i \in [n]$ such that $f(x) = x_i$ for every $x \in \{0,1\}^n$ or $f(x) = \bar{x}_i$ for every $x \in \{0,1\}^n$.

We say that f is a monotone k-monomial for $1 \leq k \leq n$ if there exist k indices $i_1, \ldots, i_k \in [n]$ such that $f(x) = x_{i_1} \wedge \cdots \wedge x_{i_k}$ for every $x \in \{0,1\}^n$. If we allow some of the x_{i_j}'s above to be replaced with $\bar{x}_{i_j}$, then f is a k-monomial. The function f is a monomial if it is a k-monomial for some $1 \leq k \leq n$.

A function f is an s-term DNF function if it is a disjunction of at most s monomials. If all monomials are monotone, then it is a monotone DNF function.

Singletons, and more generally monomials, can be easily learned under the uniform distribution. The learning algorithm uniformly selects a sample of size $\Theta(\log n/\epsilon)$ and queries the function f on all

sample strings. It then searches for a monomial that is consistent with f on the sample. Finding a consistent monomial if such exists can be done in time linear in the sample size and in n. A simple probabilistic argument, which is a slight variant of Occam's Razor [43],[1] can be used to show that a sample of size $\Theta(\log n/\epsilon)$ is sufficient to ensure that with high probability any monomial that is consistent with the sample is an ϵ-good approximation of f.

There is a large variety of results on learning DNF functions, and in particular monotone DNF, in several different models. We restrict our attention to the model most relevant to the result described here, namely when membership queries are allowed and the underlying distribution is uniform. The best known algorithm results from combining the works of Bshouty et al. [50] and Klivans and Servedio [109], and builds on Jackson's celebrated Harmonic Sieve algorithm [98]. This algorithm has query complexity $\tilde{O}\big(r \cdot \big(\frac{\log^2 n}{\epsilon} + \frac{s^2}{\epsilon^2}\big)\big)$, where r is the number of variables appearing in the DNF formula, and s is the number of terms. However, this algorithm does not output a DNF formula as its hypothesis. On the other hand, Angluin [20] describes a proper learning algorithm for monotone DNF formulae that uses membership queries and works under arbitrary distributions. The query complexity of her algorithm is $\tilde{O}(s \cdot n + s/\epsilon)$. Using the same preprocessing technique as suggested in [50], if the underlying distribution is uniform, then the query complexity can be reduced to $\tilde{O}\big(\frac{r \cdot \log^2 n}{\epsilon} + s \cdot \big(r + \frac{1}{\epsilon}\big)\big)$. Recall that the query complexity of the testing algorithm has similar dependence on s and $1/\epsilon$ but does not depend on n.

4.1.1 Singletons

We start by describing an algorithm for testing singletons. The testing algorithm for k-monomials generalizes this algorithm. More precisely, we describe an algorithm for testing whether a function f is a *monotone* singleton. In order to test whether f is a singleton we can check

[1] Applying the theorem known as Occam's Razor would give a stronger result in the sense that the underlying distribution may be arbitrary (that is, not necessarily uniform). This however comes at a price of a linear, as opposed to logarithmic, dependence of the sample/query complexity on n.

whether either f or $\bar{f}$ pass the monotone singleton test. For the sake of succinctness, in what follows we refer to monotone singletons simply as singletons. As we shall see, an interesting feature of the singletons test is that it applies the linearity test (that was described and analyzed in Section 3.1) as a subroutine.

When the identity of the function f is clear from the context, we may use the following notation: $F_1 \stackrel{\text{def}}{=} \{x : f(x) = 1\}$. For $x, y \in \{0,1\}^n$ we shall use $x \wedge y$ to denote the bitwise "AND" of the two strings. That is, $z = x \wedge y$ satisfies $z_i = x_i \wedge y_i$ for every $1 \leq i \leq n$.

The following characterization of monotone k-monomials motivates the tests we describe.

Lemma 4.1. Let $f : \{0,1\}^n \to \{0,1\}$. The function f is a monotone k-monomial if and only if the following two conditions hold:

(1) $\Pr[f(x) = 1] = \frac{1}{2^k}$;
(2) $f(x \wedge y) = f(x) \wedge f(y)$ for all $x, y \in \{0,1\}^n$.

In what follows we shall say that a pair of points $x, y \in \{0,1\}^n$ are *violating with respect to* f if $f(x \wedge y) \neq f(x) \wedge f(y)$.

Proof. If f is a k-monomial then clearly the conditions hold. We turn to prove the other direction. We first observe that the two conditions imply that $f(x) = 0$ for all $|x| < k$, where $|x|$ denotes the number of ones in x. In order to verify this, assume in contradiction that there exists some x such that $|x| < k$ but $f(x) = 1$. Now consider any y such that $y_i = 1$ whenever $x_i = 1$. Then $x \wedge y = x$, and therefore $f(x \wedge y) = 1$. But by the second condition, since $f(x) = 1$, it must also hold that $f(y) = 1$. However, since $|x| < k$, the number of such points y is strictly greater than 2^{n-k}, contradicting the first condition.

Next let $y = \bigwedge_{x \in F_1} x$. Using the second condition in the claim we get:

$$f(y) = f\left(\bigwedge_{x \in F_1} x\right) = \bigwedge_{x \in F_1} f(x) = 1. \tag{4.1}$$

However, we have just shown that $f(x) = 0$ for all $|x| < k$, and thus $|y| \geq k$. Hence, there exist k indices $i_1, \ldots, i_k$ such that $y_{i_j} = 1$ for all $1 \leq j \leq k$. But $y_{i_j} = \bigwedge_{x \in F_1} x_{i_j}$. Hence, $x_{i_1} = \cdots = x_{i_k} = 1$ for every $x \in F_1$. The first condition now implies that $f(x) = x_{i_1} \wedge \cdots \wedge x_{i_k}$ for every $x \in \{0,1\}^n$. □

Given Lemma 4.1, a natural candidate for a testing algorithm for singletons would take a sample of uniformly selected pairs (x, y), and for each pair verify that it is not violating with respect to f. In addition, the test would check that $\Pr[f(x) = 0]$ is roughly $1/2$ (or else any monotone k-monomial would pass the test). As shown in [125], the correctness of this testing algorithm can be proved as long as the distance between f and the closest singleton is bounded away from $1/2$. It is an open question whether this testing algorithm is correct in general.

We next describe a modified version of this algorithm, which consists of two stages. In the first stage, the algorithm tests whether f belongs to (is close to) a more general class of functions (that contains all singleton functions). In the second stage, it applies a slight variant of the original test (as described in the previous paragraph). Specifically, the more general class of functions is the class $\mathcal{L}$ of linear Boolean functions over $\{0,1\}^n$, which was discussed in Section 3.1. Clearly, every singleton function $f(x) = x_i$ is a linear function. Hence, if f is a singleton function, then it passes the first stage of the test (the linearity test) with probability 1. On the other hand, if it is far from any linear function, then it will be rejected already by the linearity test. As we shall see, if f is far from every singleton function, *but* it is close to some linear function that is not a singleton function (so that it may pass the linearity test), then we can prove that it will be rejected in the second stage of the algorithm with high probability.

In order to motivate the modification we introduce in the original singleton test, we state the following lemma and discuss its implications.

Lemma 4.2. Let $S \subseteq [n]$, and let $g_s(x) = \sum_{i \in S} x_i$ (where the sum is taken modulo 2). If $|S|$ is even then

$$\Pr_{x,y}[g_s(x \wedge y) = g_s(x) \wedge g_s(y)] = \frac{1}{2} + \frac{1}{2^{|S|+1}}$$

and if $|S|$ is odd then

$$\Pr_{x,y}[g_s(x \wedge y) = g_s(x) \wedge g_s(y)] = \frac{1}{2} + \frac{1}{2^{|S|}}.$$

Proof. Let $s = |S|$, and let x, y be two strings such that (i) x has $0 \leq i \leq s$ ones in S, that is, $|\{\ell \in S : x_\ell = 1\}| = i$; (ii) $x \wedge y$ has $0 \leq k \leq i$ ones in S; and (iii) y has a total of $j + k$ ones in S, where $0 \leq j \leq s - i$.

If $g_s(x \wedge y) = g_s(x) \wedge g_s(y)$, then either (1) i is even and k is even, or (2) i is odd and j is even. Let $Z_1 \subset \{0,1\}^n \times \{0,1\}^n$ be the subset of pairs x, y that obey the first constraint, and let $Z_2 \subset \{0,1\}^n \times \{0,1\}^n$ be the subset of pairs x, y that obey the second constraint. Since the two subsets are disjoint,

$$\Pr_{x,y}[g_s(x \wedge y) = g_s(x) \wedge g_s(y)] = 2^{-2n}(|Z_1| + |Z_2|)\,. \tag{4.2}$$

It remains to compute the sizes of the two sets. Since the coordinates of x and y outside S do not determine whether the pair x, y belongs to one of these sets, we have

$$|Z_1| = 2^{n-s} \cdot 2^{n-s} \cdot \left(\sum_{i=0, i \text{ even}}^{s} \binom{s}{i} \sum_{k=0, k \text{ even}}^{i} \binom{i}{k} \sum_{j=0}^{s-i} \binom{s-i}{j} \right) \tag{4.3}$$

and

$$|Z_2| = 2^{n-s} \cdot 2^{n-s} \cdot \left(\sum_{i=0, i \text{ odd}}^{s} \binom{s}{i} \sum_{k=0}^{i} \binom{i}{k} \sum_{j=0, j \text{ even}}^{s-i} \binom{s-i}{j} \right) \tag{4.4}$$

The right-hand side of Equation (4.3) equals

$$2^{2n-2s} \cdot (2^{2s-2} + 2^{s-1}) = 2^{2n-2} + 2^{2n-s-1} = 2^{2n} \cdot (2^{-2} + 2^{-(s+1)}). \tag{4.5}$$

The right-hand side of Equation (4.4) equals $2^{2n} \cdot (2^{-2} + 2^{-(s+1)})$ if s is odd and 2^{2n-2} if s is even. The lemma follows by combining Equations (4.3) and (4.4) with Equation (4.2). □

Hence, if f is a linear function that is not a singleton and is not the all-0 function, that is $f = g_s$ for $|S| \geq 2$, then the probability that a uniformly selected pair x, y is violating with respect to f is at least $1/8$. In this case, a sample of 16 such pairs will contain a violating pair with probability at least $1 - (1 - 1/8)^{16} \geq 1 - e^{-2} > 2/3$.

However, what if f passes the linearity test but is only close to being a linear function? Let g denote the linear function that is closest to f and let δ be the distance between them. (Note that g is unique, given that f is sufficiently close to a linear function). What we would like to do is to check whether g is a singleton, by selecting a sample of pairs x, y and checking whether it contains a violating pair with respect to g. Observe that, since the distance between functions is measured with respect to the uniform distribution, for a uniformly selected pair (x, y), with probability at least $(1-\delta)^2$, both $f(x) = g(x)$ and $f(y) = g(y)$. However, we cannot make a similar claim about $f(x \wedge y)$ and $g(x \wedge y)$, since $x \wedge y$ is *not* uniformly distributed. Thus, it is not clear that we can replace the violation test for g with a violation test for f. In addition we need to verify that g is not the all-0 function.

The solution is to use a *self-corrector* for linear functions [42] as described in Subsection 3.1.1. Namely, given query access to a function $f : \{0,1\}^n \to \{0,1\}$, which is strictly closer than $1/4$ to some linear function g, and an input $x \in \{0,1\}^n$, the procedure Self-Correct(f, x) returns the value of $g(x)$, with probability at least $9/10$. The query complexity of the procedure is constant.

We are now ready to describe the testing algorithm for singletons.

Algorithm 4.1 (Test for Singleton Functions).

(1) Apply the linearity test (Algorithm 3.1) to f with distance parameter $\min(1/5, \epsilon)$. If the test rejects then output reject (and exit).

(2) If Self-Correct$(f, \vec{1}) = 0$ (where $\vec{1}$ is the all-1 vector), then output reject (and exit).

(3) Uniformly and independently select $m = 64$ pairs of points x, y.

- For each such pair, let $b_x =$ Self-Correct(f, x), $b_y =$ Self-Correct(f, y) and $b_{x \wedge y} =$ Self-Correct$(f, x \wedge y)$.
- Check that $b_{x \wedge y} = b_x \wedge b_y$.

(4) If one of the checks fails then output reject. Otherwise output accept.

Theorem 4.3. Algorithm 4.1 is a one-sided error testing algorithm for monotone singletons. The query complexity of the algorithm is $O(1/\epsilon)$.

Proof. Since the linearity testing algorithm has a one-sided error, if f is a singleton function then it always passes the linearity test. In this case the self corrector always returns the value of f on every given input point. In particular, Self-Correct$(f, \vec{1}) = f(\vec{1}) = 1$, since every monotone singleton has value 1 on the all-1 vector. Similarly, no violating pair can be found in Step 3. Hence, Algorithm 4.1 always accepts a singleton.

Assume, without loss of generality, that $\epsilon \leq 1/5$. Consider the case in which f is ϵ-far from any singleton. If it is also ϵ-far from any linear function, then it will be rejected with probability at least 9/10 in the first step of the algorithm. Otherwise, there exists a unique linear function g such that f is ϵ-close to g. If g is the all-0 function, then f is rejected with probability at least 9/10 (in Step 2).

Otherwise, g is a linear function of at least 2 variables. By Lemma 4.2, the probability that a uniformly selected pair x, y is a violating pair with respect to g is at least 1/8. Given such a pair, the probability that the self-corrector returns the value of g on all the three calls (that is, $b_x = g(x)$, $b_y = g(y)$, and $b_{x \wedge y} = g(x \wedge y)$), is at least $(1 - 1/10)^3 > 7/10$. The probability that Algorithm 4.1 obtains a violating pair with respect to g *and* all calls to the self-corrector return the correct value, is greater than 1/16. Therefore, a sample of 64 pairs will ensure that a violation $b_{x \wedge y} \neq b_x \wedge b_y$ will be found with probability at least 9/10. The total probability that f is accepted, despite being ϵ-far from any singleton, is hence at most $3 \cdot (1/10) < 1/3$.

The query complexity of the algorithm is dominated by the query complexity of the linearity tester, which is $O(1/\epsilon)$. The second stage has constant query complexity. □

4.1.2 Monomials

The testing algorithm for monomials has a high level structure that is similar to the algorithm for singletons, and its query complexity

is $O(1/\epsilon)$ as well. Here we consider testing monotone k-monomials, where k is given as a parameter to the algorithm. It is possible to remove the monotonicity assumption as well as the assumption that the algorithm receives a parameter k, without increasing the complexity of the algorithm.

Upon receiving the parameters k and ϵ, the algorithm first checks what is the relation between ϵ and 2^{-k}. If $\epsilon > 4 \cdot 2^{-k}$ then the algorithm decides whether to accept or reject solely based on estimating $\Pr[f(x) = 1]$ using a sample of size $\Theta(1/\epsilon)$. Let α be this estimate. If $\alpha \leq 3\epsilon/8$, then the algorithm accepts and if $\alpha > 3\epsilon/8$, then the algorithm rejects.

To verify that the algorithm makes a correct decision with high constant probability, consider first the case that f is a k-monomial. In such a case $\Pr[f(x) = 1] = 2^{-k} < \epsilon/4$. By a multiplicative Chernoff bound (see Appendix A), the probability that $\alpha > 3\epsilon/8$ (which causes the algorithm to reject), is a small constant. On the other hand, every function f that satisfies $\Pr[f(x) = 1] \leq \epsilon/2$ is ϵ-close to every k monomial. This is true since for any k-monomial g, $\Pr[f(x) \neq g(x)] \leq \Pr[f(x) = 1] + \Pr[g(x) = 1] < \epsilon$. Therefore, if f is ϵ-far from every k monomial, then $\Pr[f(x) = 1] > \epsilon/2$. In such a case, by a multiplicative Chernoff bound, the probability that $\alpha \leq 3\epsilon/8$ (which causes the algorithm to accept), is a small constant. Thus the case that ϵ is large relative to 2^{-k} is straightforward, and we turn to the case that ϵ is of the same order, or smaller, than 2^{-k}.

Similarly to the simple case considered in the foregoing discussion, in the first step of the algorithm a *size test* is performed. That is, the algorithm verifies that $\Pr[f(x) = 1]$ is close to 2^{-k}, as it should be if f is a k-monomial. This step requires a sample of size $\Theta(2^k) = O(1/\epsilon)$. Assuming f passes the size test, the algorithm performs an *affinity test*. This step plays a similar role to that of the linearity test in the singleton testing algorithm (Algorithm 4.1). That is, on the one hand, every monotone k-monomial passes this test (with probability 1). On the other hand, if a function passes this test but is far from any monotone k-monomial, then the third step (which is described momentarily, and plays a similar role to Step 3 in Algorithm 4.1), will reject the function with high probability. We next describe both steps.

Recall that $F_1 = \{x : f(x) = 1\}$. The affinity test checks whether F_1 is (close to being) an affine subspace. We next recall the definition of an affine subspace.

Definition 4.2 (Affine Subspaces). A subset $H \subseteq \{0,1\}^n$ is an *affine subspace* of $\{0,1\}^n$ if and only if there exist an $x \in \{0,1\}^n$ and a linear subspace V of $\{0,1\}^n$ such that $H = V + x$. That is,

$$H = \{y \mid y = v + x, \quad \text{for some } v \in V\}.$$

The following is a well-known alternative characterization of affine subspaces, which is a basis for the affinity test.

Fact 4.1. H is an affine subspace if and only if for every $y_1, y_2, y_3 \in H$ we have $y_1 + y_2 + y_3 \in H$.

Note that the above fact also implies that for every $y_1, y_2 \in H$ and $y_3 \notin H$ we have $y_1 + y_2 + y_3 \notin H$. The affinity test selects, uniformly and independently, $m = \Theta(1/\epsilon)$ points $a_1, \ldots, a_m \in \{0,1\}^n$ and $t = \Theta(1)$ pairs of points $(x_1, y_1), \ldots, (x_t, y_t) \in F_1 \times F_1$. If for some $1 \leq i \leq m$, $1 \leq j \leq t$, the equality $f(a_i + x_j + y_j) = f(a_i)$ does not hold, then the test rejects. The analysis of the affinity test shows that passing this step with sufficiently high probability ensures that f is close to some function g for which $g(x) + g(y) + g(z) = g(x + y + z)$ for all $x, y, z \in G_1 = \{x | g(x) = 1\}$. That is, G_1 is an affine subspace.

In the last step, which is referred to as the Closure-under-intersection test, the algorithm selects, uniformly and independently, a constant number of points $x \in F_1$ and $\Theta(2^k)$ points $y \in \{0,1\}^n$. If for some pair x, y selected, Self-Correct$(f, x \wedge y) \neq$ Self-Correct(f, y), then the test rejects. Here Self-Correct is a procedure that given any input z and oracle access to f, returns with high probability the value $g(z)$, where g is as described in the previous paragraph.

In both the affinity test and the closure-under-intersection test, we need to select strings in F_1 uniformly. This is simply done by sampling from $\{0,1\}^n$ and using only x's for which $f(x) = 1$. Since in both tests

the number of strings selected from F_1 is a constant, the total number of queries required is $O(2^k) = O(1/\epsilon)$. (If the algorithm does not obtain sufficiently many strings from F_1, then it can give an arbitrary output, since this event occurs with small constant probability.)

The correctness of the algorithm follows from the next two lemmas (whose proofs can be found in [125]).

Lemma 4.4. Let f be a function for which $|\Pr[f(x) = 1] - 2^{-k}| < 2^{-k-3}$. If the probability that the affinity test accepts f is greater than $1/10$, then there exists a function $g : \{0,1\}^n \to \{0,1\}$ for which the following holds:

(1) $\text{dist}(f, g) \leq \epsilon/2^5$.
(2) $G_1 \stackrel{\text{def}}{=} \{a : g(a) = 1\}$ is an affine subspace of dimension $n - k$.
(3) There exists a procedure Self-Correct that given any input $a \in \{0,1\}^n$ and oracle access to f, asks a constant number of queries and returns the value $g(a)$ with probability at least $1 - 1/40$.

Furthermore, if F_1 is an affine subspace then the affinity tests always accepts, $g = f$, and $\text{Self-Correct}(f, a) = f(a)$ with probability 1 for every $a \in \{0,1\}^n$.

Lemma 4.5. Let $f : \{0,1\}^n \to \{0,1\}$ be a function for which $|\Pr[f(x) = 1] - 2^{-k}| < 2^{-k-3}$. Suppose that there exists a function $g : \{0,1\}^n \to \{0,1\}$ such that:

(1) $\text{dist}(f, g) \leq 2^{-k-3}$.
(2) $G_1 \stackrel{\text{def}}{=} \{x : g(x) = 1\}$ is an affine subspace of dimension $n - k$.
(3) There exists a procedure Self-Correct that given any input $a \in \{0,1\}^n$ and oracle access to f returns the value $g(a)$ with probability at least $1 - 1/40$.

If g is not a monotone k-monomial, then the probability that the Closure-Under-Intersection Test rejects is at least $9/10$.

4.1.3 Monotone DNF

In the previous subsection we described an algorithm for testing whether a function is a (single) monomial. A natural generalization is to test whether a function is of the form $f = T_1 \vee T_2 \vee \cdots \vee T_s$, where each term T_i is a monomial. Here we consider a restricted case where all terms are monotone monomials. For the sake of simplicity, from this point on when we say "term" or "monomial" we mean that it is monotone.

The basic idea underlying the algorithm is to test whether the set $F_1 \stackrel{\text{def}}{=} \{x : f(x) = 1\}$ can be "approximately covered" by at most s terms (monomials). To this end, the algorithm finds strings $x_i \in \{0,1\}^n$ and uses them to define functions f_i that are tested for being monomials. If the original function f is in fact an s-term DNF, then, with high probability, each such function f_i corresponds to one of the terms of f. In what follows we give a little more of the flavor of the algorithm. For full details see [125].

Let f be a monotone s-term DNF, and let its terms be $T_1, \ldots, T_s$. Then, for any $x \in \{0,1\}^n$, we let $S(x) \subseteq \{1, \ldots, s\}$ denote the subset of indices of the terms satisfied by x. That is: $S(x) \stackrel{\text{def}}{=} \{i : T_i(x) = 1\}$. In particular, if $f(x) = 0$ then $S(x) = \emptyset$. This notion extends to a set $R \subseteq F_1$, where $S(R) \stackrel{\text{def}}{=} \bigcup_{x \in R} S(x)$. We observe that if f is a monotone s-term DNF, then $S(x \wedge y) = S(x) \cap S(y)$ for every $x, y \in \{0,1\}^n$.

Definition 4.3 (Single-Term Representatives). Let f be a monotone s-term DNF. We say that $x \in F_1$ is a single-term representative for f if $|S(x)| = 1$. That is, x satisfies only a single term in f.

Definition 4.4 (Neighbors). Let $x \in F_1$. The set of neighbors of x, denoted by $N(x)$, is defined as follows:

$$N(x) \stackrel{\text{def}}{=} \{y \mid f(y) = 1 \text{ and } f(x \wedge y) = 1\}.$$

The notion of neighbors extends to a set $R \subseteq F_1$, where $N(R) \stackrel{\text{def}}{=} \bigcup_{x \in R} N(x)$.

Note that the above definition of neighbors is very different from the standard notion (that is, strings at Hamming distance 1), and in particular depends on the function f.

Consider the case in which x is a single-term representative of f, and $S(x) = \{i\}$. Then, for every neighbor $y \in N(x)$, we must have $i \in S(y)$ (or else $S(x \wedge y)$ would be empty, implying that $f(x \wedge y) = 0$). Notice that the converse statement holds as well, that is, $i \in S(y)$ implies that x and y are neighbors. Therefore, the set of neighbors of x is exactly the set of all strings satisfying the term T_i. The goal of the algorithm is to find at most s such single-term representatives $x \in \{0,1\}^n$, and for each such x to test that its set of neighbors $N(x)$ satisfies some common term. It can be shown that if f is in fact a monotone s-term DNF, then all these tests pass with high probability. On the other hand, if all the tests pass with high probability, then f is close to some monotone s-term DNF.

4.2 Juntas

In this subsection, we describe the main result of Fischer et al. [71]. Rather than considering families of functions characterized by their logical structure as done in [125], the paper [71] considers families of functions characterized by the number of variables they depend on.

Definition 4.5 (Juntas). A function $f : \{0,1\}^n \to \{0,1\}$ is a k-junta for an integer $k \leq n$ if f is a function of at most k variables. Namely, there exists a set $J \subseteq [n]$, where $|J| \leq k$ such that $f(x) = f(y)$ for every $x, y \in \{0,1\}^n$ that satisfy $x_i = y_i$ for each $i \in J$. We say in such a case that J dominates the function f.

The main result of [71] is stated next.

Theorem 4.6. For every fixed k, the property of being a k-junta is testable using $\text{poly}(k)/\epsilon$ queries.

Fischer et al. [71] establish Theorem 4.6 by describing and analyzing several algorithms. The algorithms vary in the polynomial dependence on k (ranging between $\tilde{O}(k^4)$ to $\tilde{O}(k^2)$), and in two properties: whether

the algorithm is nonadaptive or adaptive (that is, queries may depend on answers to previous queries), and whether it has one-sided error or two-sided error. They also prove a lower bound of $\tilde{\Omega}(\sqrt{k})$, which was later improved to $\Omega(k)$ by Chockler and Gutreund [54], thus establishing that a polynomial dependence on k is necessary. While we focus here on the domain $\{0,1\}^n$ and on the case that the underlying distribution is uniform, Theorem 4.6 holds for other domains and when the underlying distribution is a product distribution.

The class of k-juntas can be viewed as generalizing the class of k-monomials, where the function over the k variables is unrestricted. Fischer et al. [71] also consider testing whether a function is identical to a fixed function h up to a permutation of its variables. For any given function h over k variables, the testing algorithm performs a number of queries that is polynomial in $1/\epsilon$ and in the number of variables of h.

Before describing one of the algorithms for testing k-juntas, we briefly discuss the relation to learning. Knowing that a function depends on only a small number of variables can be especially useful in the context of learning. For various function classes there exist algorithms that are attribute efficient (*cf.* [40, 114, 133]). That is, they have a polynomial dependence on the number of relevant variables of the function being learned and only a logarithmic dependence on the total number of variables. Learning k-juntas under the uniform distribution (but without queries) was studied by Mossel, O'Donnel, and Servedio [119]. They give an algorithm for this learning problem that runs in time $(n^k)^{\omega/(\omega+1)}$, where $\omega < 2.376$ is the matrix multiplication exponent.

Perhaps, the most closely related learning-theory work is given by Guijarro et al. [90]. In this work, Guijarro et al. [90], consider the following problem. For a fixed but unknown distribution D over $\{0,1\}^n$ and an unknown Boolean function f over $\{0,1\}^n$ that is known to be a k-junta, the algorithm is given access to examples drawn according to D and query access to f. The goal of the algorithm is to find a subset J of size at most k that dominates a function f' that is ϵ-close to f. They describe an algorithm for this problem whose query complexity is $O(k(\log(k+1)/\epsilon + \log n))$. Their algorithm can be used to test for

the property of be a k-junta. While the query complexity of the [90] algorithm has a better dependence on k, it depends on n.

4.2.1 Preliminaries

In order to describe and analyze the testing algorithm, we first introduce some definitions and notations. The domain of the functions we consider is always $\{0,1\}^n$ and it will be convenient to assume that the range of the function is $\{1,-1\} = \{(-1)^0,(-1)^1\}$ (rather than $\{0,1\}$).

Partial Assignments. For a subset $S \subseteq [n]$ we denote by $\mathcal{A}(S)$ the set of *partial assignments* to the variables x_i, where $i \in S$. Each $w \in \mathcal{A}(S)$ can be viewed as a string in $\{0,1,*\}^n$, where for every $i \in S$, $w_i \in \{0,1\}$, and for every $i \notin S$, $w_i = *$. In particular, $\mathcal{A}([n]) = \{0,1\}^n$. For two *disjoint* subsets $S, S' \subset [n]$, and for partial assignments $w \in \mathcal{A}(S)$ and $w' \in \mathcal{A}(S')$, we let $w \sqcup w'$ denote the partial assignment $z \in \mathcal{A}(S \cup S')$ defined by: $z_i = w_i$, for every $i \in S$, $z_i = w'_i$ for every $i \in S'$, and $z_i = w_i = w'_i = *$ for every $i \in [n] \setminus \{S \cup S'\}$. In particular, we shall consider the case $S' = [n] \setminus S$, so that $w \sqcup w' \in \{0,1\}^n$ is a complete assignment (and $f(w \sqcup w')$ is well defined). Finally, for $x \in \{0,1\}^n$ and $S \subseteq [n]$, we let $x_{|S}$ denote the partial assignment $w \in \mathcal{A}(S)$ defined by $w_i = x_i$ for every $i \in S$, and $w_i = *$ for every $i \notin S$.

For the sake of conciseness, we shall use $\overline{S}$ as a shorthand for $[n] \setminus S$, whenever it is clear that $S \subseteq [n]$.

Variation. For a function $f : \{0,1\}^n \to \{1,-1\}$ and a subset $S \subset [n]$, we define the *variation* of f on S, denoted $\mathrm{Vr}_f(S)$, as the probability, taken over a uniform choice of $w \in \mathcal{A}(\overline{S})$ and $z_1, z_2 \in \mathcal{A}(S)$, that $f(w \sqcup z_1) \neq f(w \sqcup z_2)$. That is[2]:

$$\mathrm{Vr}_f(S) \stackrel{\mathrm{def}}{=} \Pr_{w \in \mathcal{A}(\overline{S}), z_1, z_2 \in \mathcal{A}(S)}[f(w \sqcup z_1) \neq f(w \sqcup z_2)] \,. \tag{4.6}$$

The simple but important observation is that if f does not depend on any variable x_i, where $i \in S$, then $\mathrm{Vr}_f(S) = 0$, and otherwise it must be nonzero (though possibly small). One useful property of variation is

[2] We note that in [71] a more general definition is given (for real-valued functions). For the sake of simplicity we give only the special case of $\{1,-1\}$-valued function, and we slightly modify the definition by removing a factor of 2.

that it is *monotone*. Namely, for any two subsets $S,T \subseteq [n]$,

$$\mathrm{Vr}_f(S) \leq \mathrm{Vr}_f(S \cup T). \tag{4.7}$$

Another property is that it is *subadditive*, that is, for any two subsets $S,T \subseteq [n]$,

$$\mathrm{Vr}_f(S \cup T) \leq \mathrm{Vr}_f(S) + \mathrm{Vr}_f(T). \tag{4.8}$$

As we show next, the variation can also be used to bound the distance that a function has to be a k-junta.

Lemma 4.7. Let $f : \{0,1\}^n \to \{1,-1\}$ and let $J \subset [n]$ be such that $|J| \leq k$ and $\mathrm{Vr}_f(\overline{J}) \leq \epsilon$. Then there exists a k-junta g that is dominated by J and is such that $\mathrm{dist}(f,g) \leq \epsilon$.

Proof. We define the function g as follows: for each $x \in \{0,1\}^n$ let

$$g(x) \stackrel{\mathrm{def}}{=} \mathrm{majority}_{u \in \mathcal{A}(\overline{J})}\{f(x_{|J} \sqcup u)\}\ . \tag{4.9}$$

That is, for each $w \in \mathcal{A}(J)$, the function g has the same value on all strings $x \in \{0,1\}^n = \mathcal{A}([n])$ such that $x_{|J} = w$, and this value is simply the majority value of the function f taken over all strings of this form.

We are interested in showing that $\Pr[f(x) = g(x)] \geq 1 - \epsilon$. That is,

$$\Pr_{w \in \mathcal{A}(J), z \in \mathcal{A}(\overline{J})}\big[f(w \sqcup z) = \mathrm{majority}_{u \in \mathcal{A}(\overline{J})}\{f(w \sqcup u)\}\big] \geq 1 - \epsilon. \tag{4.10}$$

Similarly to what was shown in the proof of Claim 3.5, this probability is lower bounded by $\Pr_{w \in \mathcal{A}(J), z_1, z_2 \in \mathcal{A}(\overline{J})}[f(w \sqcup z_1) = f(w \sqcup z_2)]$, which is simply $1 - \mathrm{Vr}_f(\overline{J}) \geq 1 - \epsilon$. □

4.2.2 An Algorithm for Testing Juntas

Here we describe an algorithm for testing k-juntas, which has one-sided error, is non-adaptive, and has query complexity $\tilde{O}(k^4/\epsilon)$. In [71] there are actually two algorithms with this complexity. We have chosen to describe the one on which the more efficient algorithms (mentioned previously) are based, and which also plays a role in the results described in Section 4.3. We assume that $k > 1$, since 1-juntas are

simply singletons, for which we already know that there is a testing algorithm.

Algorithm 4.2 (*k*-Junta Test).

(1) For $r = \Theta(k^2)$ select a random partition $\{S_1, \ldots, S_r\}$ of $[n]$ by assigning each $i \in [n]$ to a set S_j with equal probability.
(2) For each $j \in [r]$, perform the following dependence test at most $h = 4(\log(k+1)+4)r/\epsilon = \Theta(k^2 \log k/\epsilon)$ times:
   - Uniformly and independently select $w \in \mathcal{A}(\overline{S}_j)$ and $z_1, z_2 \in \mathcal{A}(S_j)$. If $f(w \sqcup z_1) \neq f(w \sqcup z_2)$ then declare that f depends on variables in S_j (and continue to $j+1$).
(3) If the number of subsets S_j that f was found to depend on is larger than k, then output reject, otherwise output accept.

Theorem 4.8. Algorithm 4.2 is a one-sided error testing algorithm for k-juntas. Its query complexity is $O(k^4 \log k/\epsilon)$.

The bound on the query complexity of the algorithm is $O(r \cdot h) = O(k^4 \log k/\epsilon)$. The dependence test declares that f depends on a set S_j only if it has found evidence of such a dependence and the algorithm rejects only if there are more than k disjoint sets for which such evidence is found. Therefore, the algorithm never rejects a k-junta. We hence turn to proving that if f is ϵ-far from a k-junta then it is rejected with probability at least $2/3$.

Let $\tau = (\log(k+1)+4)/h$ and note that by the definition of h, $\tau \leq \epsilon/(4r)$ (recall that r is the number of sets in the random partition selected by the algorithm and h is the number of applications of the dependence test). Define $J = J_\tau(f) \stackrel{\text{def}}{=} \{i \in [n] : \mathrm{Vr}_f(\{i\}) > \tau\}$. Thus $\overline{J}$ consists of all i such that $\mathrm{Vr}_f(\{i\}) \leq \tau$. We shall prove two lemmas:

Lemma 4.9. If $\mathrm{Vr}_f(\overline{J}) > \epsilon$ then Algorithm 4.2 rejects with probability at least $2/3$.

Lemma 4.10. If $|J| > k$ then Algorithm 4.2 rejects with probability at least 2/3.

By Lemma 4.7, if f is ϵ-far from any k-junta, then either $\mathrm{Vr}_f(\overline{J}) > \epsilon$ or $|J| > k$ (or both). By Lemmas 4.9 and 4.10 this implies that the algorithm rejects with probability at least 2/3. Both lemmas rely on the following claim regarding the dependence test.

Claim 4.11. For any subset S_j, if $\mathrm{Vr}_f(S_j) \geq \tau$, then the probability that Step 2 in Algorithm 4.2 declares that f depends on variables in S_j is at least $1 - 1/(e^4(k+1))$.

Proof. By the definition of the dependence test, the probability that a single application of the test finds evidence that f depends on S_j is exactly $\mathrm{Vr}_f(S_j)$. Since $\tau = (\log(k+1)+4)/h$, if $\mathrm{Vr}_f(S_j) \geq \tau$, the probability that the test fails to find such evidence in h independent applications is at most $(1-\tau)^h < \exp(-\tau h) < e^{-4}/(k+1)$, as claimed. □

We now prove Lemma 4.10, which is quite simple, and later sketch the proof of Lemma 4.9, which is more complex.

Proof of Lemma 4.10. First observe that if $|J| > k$, then the probability, over the choice of the partition, that there are fewer than $k+1$ sets S_j such that $S_j \cap J \neq \emptyset$, is $O(k^2/r)$. Since $r = ck^2$, where c is a constant, for an appropriate choice of c, this probability is at most 1/6. Assume from this point on that there are at least $k+1$ sets S_j such that $S_j \cap J \neq \emptyset$ (where we later take into account the probability that this is not the case).

By the monotonicity of the variation (Equation (4.7)) and since $\mathrm{Vr}_f(\{i\}) > \tau$ for each $i \in J$, if a set S_j satisfies $S_j \cap J \neq \emptyset$, then $\mathrm{Vr}_f(S_j) \geq \tau$. By Claim 4.11 and the union bound, the probability that the algorithm finds evidence of dependence for fewer than $k+1$ sets is less than 1/6. Summing this probability with the probability that there are fewer than $k+1$ sets S_j such that $S_j \cap J \neq \emptyset$, the lemma follows. □

Proof Sketch of Lemma 4.9: By the premise of the lemma, $\mathrm{Vr}_f(\overline{J}) > \epsilon$. Since the variation is subadditive (Equation (4.8)), for any partition $\{S_1, \ldots, S_r\}$, $\sum_{j=1}^{r} \mathrm{Vr}_f(S_j \cap \overline{J}) > \epsilon$. Since the subsets in the partition are equally distributed, we have that for each fixed choice of j, $\mathrm{Exp}[\mathrm{Vr}_f(S_j \cap \overline{J})] > \epsilon/r$. The main technical claim (whose proof we omit) is that with high probability $\mathrm{Vr}_f(S_j \cap \overline{J})$ is not much smaller than its expected value. To be precise, for each fixed choice of j, with probability at least $3/4$ (over the random choice of the partition), $\mathrm{Vr}_f(S_j \cap \overline{J}) \geq \epsilon/(4r)$. Recall that by the definition of τ (and of h as a function of r), we have that $\epsilon/(4r) \geq \tau$.

Using this claim, we now show how Lemma 4.9 follows. Recall that by monotonicity of the variation, $\mathrm{Vr}_f(S_j) \geq \mathrm{Vr}_f(S_j \cap \overline{J})$. We shall say that a subset S_j is *detectable*, if $\mathrm{Vr}_f(S_j) \geq \tau$. Thus, the expected number of detectable subsets is at least $(3/4)r$. Let α denote the probability that there are fewer than $r/8$ detectable subsets. Then $\alpha \leq 2/7$ (as the expected number of detectable subsets is at most $\alpha(r/4) + (1-\alpha)r$). Equivalently, with probability at least $5/7$, there are at least $r/8 = \Omega(k^2) > k+1$ detectable subsets. Conditioned on this event, by Claim 4.11 (and the union bound), the probability that the algorithm detects dependence for fewer than $J+1$ subsets is at most $1/e^4$. Adding this to the probability that there are fewer than $k+1$ detectable sets, the lemma follows. □

4.2.3 More Efficient Algorithms

By allowing the algorithm to be adaptive, it is possible to reduce the query complexity to $O(k^3 \log^3(k+1)/\epsilon)$, and by allowing the algorithm to have two-sided error, it can be reduced to $O(k^2 \log^3(k+1)/\epsilon)$ (without the need for adaptivity). Here we give the high-level ideas for the more efficient algorithms.

Both algorithms start by partitioning the variables into $r = \Theta(k^2)$ disjoint subsets $\{S_1, S_2, \ldots, S_r\}$ as done in Algorithm 4.2. The main idea used in the first improvement is to speed up the detection of subsets S_j that have non-negligible variation $\mathrm{Vr}_f(S_j)$, in the following manner of divide and conquer. Instead of applying the dependence test to each subset separately, it is applied to *blocks*, each of which is a union of

several subsets. If f is not found to depend on a block, then all the variables in the block are declared to be "variation free." Otherwise (some dependence is detected), the algorithm partitions the block into two equally sized sub-blocks, and continues the search on them.

The two-sided error test also applies the dependence test to blocks of subsets, only the blocks are chosen differently and in particular, may overlap. The selection of blocks is done as follows. For $s = \Theta(k \log r) = \Theta(k \log k)$, the algorithm picks s random subsets of coordinates $I_1 \ldots, I_s \subseteq [r]$ of size k independently, each by uniformly selecting (without repetitions) k elements of $[n]$. For each $1 \leq \ell \leq s$, block B_ℓ is defined as $B_\ell = \bigcup_{j \in I_\ell} S_j$. The dependence test is then applied h times to each block (where h is as in Algorithm 4.2). For each subset S_j, the algorithm considers the blocks that contain it. The algorithm declares that f depends on S_j, if it found that f depends on all blocks that contain S_j. If there are more than k such subsets, or if f depends on at least a half of the blocks, the algorithm rejects, otherwise, it accepts. For further details of the analysis, see [71].

4.3 Testing by Implicit Learning: General DNF, Decision Trees and More

In this subsection, we describe the results of Diakonikolas et al. [61]. They present a general method for testing whether a function has a concise representation (e.g., an s-term DNF or an s-node decision tree). Here we mostly focus on the Boolean case, though the technique in [61] extends to general domains and ranges. The query complexity is always polynomial in the size parameter s, and is quadratic in $1/\epsilon$. The running time grows exponentially[3] with s.

4.3.1 The Algorithm

The idea. The key observation behind the general algorithm of [61] is that many classes of functions that have a concise representation are "well-approximated" by small juntas that belong to the class. That is,

[3] In recent work [62] the dependence of the running time on s in the case of s-term polynomials over $GF(2)$ was reduced to polynomial.

every function in the class is close to some other function in the class that is a small junta. For example, for any choice of δ, every s-term DNF is δ-close to an s-term DNF that depends only at most $s\log(s/\delta)$ variables. This is true since by removing a term that has more than $\log(s/\delta)$ variables, the error incurred is at most δ/s (recall that the underlying distribution is uniform).

Given this observation, the algorithm works roughly as follows. It first finds a collection of subsets of variables such that each subset contains a single variable on which the function depends (in a non-negligible manner). If the number of such subsets is larger than some threshold k, then the algorithm rejects. Otherwise, the algorithm creates a sample of labeled examples, where the examples are points in $\{0,1\}^k$, that is, over the variables that the function depends on. It is important to stress that the algorithm creates this sample *without* actually identifying the relevant variables. Finally, the algorithm checks whether there exists a function of the appropriate form over the small set of variables that is consistent with the sample. This is the essence of the idea of "testing by implicit learning."

Before describing the algorithm in more detail, we give a central definition, and state the main theorem.

Definition 4.6. Let $\mathcal{F}$ be a class of Boolean functions over $\{0,1\}^n$. For $\delta > 0$, we say that a subclass $\mathcal{F}(\delta) \subseteq \mathcal{F}$ is a $(\delta, k(\delta))$-approximator for $\mathcal{F}$ if the following two conditions hold.

- The subclass $\mathcal{F}(\delta)$ is closed under permutations of the variables.
- For every function $f \in \mathcal{F}$ there is a function $f' \in \mathcal{F}(\delta)$ such that $\text{dist}(f', f) \leq \delta$ and f' is a $k(\delta)$-junta.

Returning to the case that $\mathcal{F}$ is the class of s-term DNF functions, we may take $\mathcal{F}(\delta)$ to be the subclass of $\mathcal{F}$ that consists of s-term DNF where each term is of size at most $\log(s/\delta)$, so that $k(\delta) = s\log(s/\delta)$. Note that $k(\delta)$ may be a function of other parameters determining the function class $\mathcal{F}$.

We shall use the notation $\widehat{\mathcal{F}}(\delta)$ for the subset of functions in $\mathcal{F}(\delta)$ that depend on the variables $x_1, \ldots, x_{k(\delta)}$. Moreover, we shall view these functions as taking only $k(\delta)$ arguments, that is, being over $\{0,1\}^{k(\delta)}$.

We now state the main theorem of [61] (for the Boolean case).

Theorem 4.12. Let $\mathcal{F}$ be a class of Boolean functions over $\{0,1\}^n$. For each choice of $\delta > 0$, let $\widehat{\mathcal{F}}(\delta) \subseteq \mathcal{F}$ be a $(\delta, k(\delta))$ approximator for $\mathcal{F}$. Suppose that for every $\epsilon > 0$ there is a δ satisfying

$$\delta \leq \frac{c\epsilon^2}{k^2(\delta) \cdot \log^2(k(\delta)) \cdot \log^2 |\widehat{\mathcal{F}}(\delta)| \cdot \log\log(k(\delta)) \cdot \log(\log |\widehat{\mathcal{F}}(\delta)|/\epsilon)}, \tag{4.11}$$

where c is a fixed constant. Let δ^* be the largest value of δ that satisfies Equation (4.11). Then there is a two-sided error testing algorithm for $\mathcal{F}$ that makes $\tilde{O}(k^2(\delta^*)\log^2 |\widehat{\mathcal{F}}(\delta^*)|/\epsilon^2)$ queries.

We note that Theorem 4.12 extends to function classes with domain Ω^n and any range, in which case there is a dependence on $\log|\Omega|$ in Equation (4.11) and in the query complexity of the algorithm.

All results from [61] that appear in Table 4.1 are obtained by applying Theorem 4.12. In all these applications, $k(\delta)$ grows logarithmically with $1/\delta$, and $\log|\widehat{\mathcal{F}}(\delta)|$ is at most polynomial in $k(\delta)$. This ensures that Equation (4.11) can be satisfied. The most typical case in the applications is that for a class $\mathcal{F}$ defined by a size parameter s, we have that $k(\delta) \leq \text{poly}(s)\log(1/\delta)$ and $\log|\widehat{\mathcal{F}}(\delta)| \leq \text{poly}(s)\text{polylog}(1/\delta)$. This yields $\delta^* = \tilde{O}(\epsilon^2)/\text{poly}(s)$, and so the query complexity of the algorithm is $\text{poly}(s)/\tilde{\Theta}(\epsilon^2)$.

In particular, returning to the case that $\mathcal{F}$ is the class of s-term DNF, we have that $k(\delta) = s\log(s/\delta)$ and $|\widehat{\mathcal{F}}(\delta)| \leq (2s\log(s/\delta))^{s\log(s/\delta)}$. This implies that $\delta^* = \tilde{O}(\epsilon^2/s^4)$, from which the upper bound of $\tilde{O}(s^4/\epsilon^2)$ on the query complexity follows. As another example, consider the case that $\mathcal{F}$ is the class of all decision lists. Then, for every δ, if we let $\widehat{\mathcal{F}}(\delta)$ be the subclass of decision lists with length $\log(1/\delta)$, and we set $k(\delta) = \log(1/\delta)$, then $\widehat{\mathcal{F}}(\delta)$ is a $(\delta, k(\delta))$-approximation for $\mathcal{F}$. Since $|\widehat{\mathcal{F}}(\delta)| \leq 2 \cdot 4^{\log(1/\delta)}(\log(1/\delta))!$, we get that $\delta^* = \tilde{O}(\epsilon^2)$, from which the bound of $\tilde{O}(1/\epsilon^2)$ on the query complexity follows.

The algorithm: The testing algorithm consists of three procedures. The first procedure, named Identify-Critical-Subsets, is a slight variant of the two-sided error junta test of [71] (described briefly in Subsection 4.2.3). This variant is executed with $k = k(\delta^*)$, where δ^* is as defined in Theorem 4.12 and with slightly larger constants than the original [71] algorithm. The main modification is that instead of returning accept in case of success, the procedure returns the at most $k(\delta^*)$ subsets of variables among $S_1, \ldots, S_r$ that the function f was found to depend on by the test. In case of failure, it outputs reject like the two-sided error junta test.

The analysis of the two-sided error test can be slightly modified so as to ensure the following. If $f \in \mathcal{F}$, so that it is δ^*-close to a $k(\delta^*)$-junta $f' \in \mathcal{F}(\delta^*)$, then with high probability, Identify-Critical-Subsets completes successfully and outputs $\ell \leq k(\delta^*)$ subsets of variables among $S_{i_1}, \ldots, S_{i_\ell}$. On the other hand, it is still true that if f is far from any $k(\delta^*)$-junta, then Identify-Critical-Subsets outputs reject with high probability. Moreover, if f is such that with probability at least $1/3$ the procedure completes successfully and outputs $\ell \leq k(\delta^*)$ subsets $S_{i_1}, \ldots, S_{i_\ell}$, then these subsets satisfy the following conditions with high probability. (1) For $\tau = \Theta(\epsilon/k(\delta^*))$, each variable x_i for which $\mathrm{Vr}_f(\{i\}) \geq \tau$ occurs in one of the subsets S_{i_j}, and each of these subsets contains at most one such variable; (2) The total variance of all other variables is $O(\epsilon/\log|\widehat{\mathcal{F}}(\delta^*)|)$.

We now turn to the second procedure, which is referred to as Construct-Sample. This procedure receives as input the subsets $S_{i_1}, \ldots, S_{i_\ell}$ that were output by Identify-Critical-Subsets. Assume that indeed the subsets satisfy the aforementioned conditions. For the sake of the discussion, let us make the stronger assumption that every variable has either non-negligible variance with respect to f or zero variance. This implies that each subset S_{i_j} output by Identify-Critical-Subsets contains exactly one relevant variable (and there are no other relevant variables).

Given a point $z \in \{0,1\}^n$, we would like to find the restriction of z to its $\ell \leq k(\delta^*)$ relevant variables (without actually determining these variables). Consider a subset S_{i_j} output by Identify-Critical-Subsets, and let x_p, for $p \in S_{i_j}$, denote the relevant variable in S_{i_j}. We would like to

know whether $z_p = 0$ or $z_p = 1$. To this end, we partition the variables in S_{i_j} into two subsets: $S^0_{i_j}(z) = \{q \in S_{i_j} : z_q = 0\}$, and $S^1_{i_j}(z) = \{q \in S_{i_j} : z_q = 1\}$. Now we run the dependence test (as defined in Algorithm 4.2) sufficiently many times so as to ensure (with high probability) that we determine whether $p \in S^0_{i_j}(z)$ (so that $z_p = 0$), or $p \in S^1_{i_j}(z)$ (so that $z_p = 1$). The pseudo-code for the procedure appears next.

Procedure Construct-Sample: Let $m = \Theta(\log|\widehat{\mathcal{F}}(\delta^*)|/\epsilon)$. For $t = 1,\ldots,m$ construct a labeled example (x^t, y^t), where $x^t \in \{0,1\}^{k(\delta^*)}$ and $y^t \in \{0,1\}$, as follows:

(1) Uniformly select $z^t \in \{0,1\}^n$, and let $y^t = f(z^t)$.
(2) For $j = 1,\ldots,\ell$ do:
   (a) For $b \in \{0,1\}$, let $S^b_{i_j}(z^t) = \{q \in S_{i_j} : z^t_q = b\}$.
   (b) For $g = \Theta(k(\delta^*)\log(m \cdot k(\delta^*))/\epsilon) = \Theta((k(\delta^*)/\epsilon)\log(\log|\mathcal{F}(\delta^*)|k(\delta^*)/\epsilon))$, run the dependence test on $S^0_{i_j}(z^t)$ and on $S^1_{i_j}(z^t)$, g times (each).
   (c) If there is evidence that f depends on both $S^0_{i_j}(z^t)$ and $S^1_{i_j}(z^t)$, then output reject (and exit). If there is evidence that f depends on $S^b_{i_j}(z^t)$ for $b = 0$ or $b = 1$, then set $x^t_j = b$. Otherwise set x^t_j uniformly at random to be either 0 or 1.
(3) For $j = \ell + 1,\ldots,k(\delta^*)$, set x^t_j uniformly at random to be either 0 or 1.

The third procedure, Check-Consistency, is given as input the sample output by Construct-Sample. If some function $f' \in \widehat{\mathcal{F}}(\delta^*)$ is consistent with the sample, then the procedure outputs accept. Otherwise it outputs reject.

Proof Sketch of Theorem 4.12: Consider first the case that $f \in \mathcal{F}$, so that it is δ^*-close to some function $f' \in \widehat{\mathcal{F}}(\delta^*)$, where, f' is a $k(\delta^*)$-junta. The parameter δ^* is selected to be sufficiently small so that we can essentially assume that $f = f'$. Thus, we shall make this assumption in this proof sketch. For $\tau = \Theta(\epsilon/k(\delta^*))$, each variable x_i such that $\mathrm{Vr}_{f'}(\{i\}) \geq \tau$ will be referred to as *highly relevant*. As discussed previously, with high probability, the procedure Identify-Critical-Subsets

outputs $\ell \leq k(\delta^*)$ subsets $S_{i_1}, \ldots, S_{i_\ell}$ that satisfy the following conditions: (1) each highly relevant variable occurs in one of these subsets; (2) each of the subsets contains at most one highly relevant variable of f' (in fact, exactly one relevant variable of f'); (3) all other variables are "very irrelevant" (have small total variance).

Assuming the subsets output by Identify-Critical-Subsets are as specified above, consider the construction of $x^t \in \{0,1\}^{k(\delta^*)}$ for any $1 \leq t \leq m$. Since each S_{i_j} contains exactly one relevant variable, if this variable is highly relevant, then the following holds with high probability: one of the executions of the dependence test finds evidence that either this variable is in $S^0_{i_j}(z^t)$ or that it is in $S^1_{i_j}(z^t)$, and x^t_i is set accordingly. If the variable is not highly relevant, then either x^t_i is set correctly, as in the highly relevant case, or x^t_i is set randomly to 0 or 1. Since the total variation of all nonhighly-relevant variables is small, with high probability $f'(x^t_i) = y^t$ (recall that $y^t = f(z^t)$). Thus, with high probability, we get a random sample of points in $\{0,1\}^{k(\delta^*)}$ that is labeled by the $k(\delta^*)$-junta f'. Since $f' \in \widehat{\mathcal{F}}(\delta^*)$, in such a case the procedure Check-Consistency will output accept, as required (recall that $\widehat{\mathcal{F}}(\delta^*)$ is closed under permutations of the variables).

We now turn to the case that f is ϵ-far from $\mathcal{F}$. If it is also $(\epsilon/2)$-far from every $k(\delta^*)$-junta, then Identify-Critical-Subsets detects this with high probability, and rejects. Otherwise, f is $(\epsilon/2)$-close to a $k(\delta^*)$-junta. Note that f can still be rejected by either Identify-Critical-Subsets or by Create-Sample. If this occurs with high probability, then we are done. Otherwise, by the properties of these two procedures, with high probability there would not be any function in $\widehat{\mathcal{F}}(\delta^*)$ that is consistent with the sample output by Create-Sample (based on the subsets output by Identify-Critical-Subsets). This is true since otherwise it would imply that there is a function $f'' \in \widehat{\mathcal{F}}(\delta^*) \subseteq \mathcal{F}$ that is $(\epsilon/2)$-close to a $k(\delta^*)$-junta f' such that $\text{dist}(f, f') \leq \epsilon/2$. But this would contradict the fact that f is ϵ-far from $\mathcal{F}$. □

4.4 Testing Linear Threshold Functions

One of the function classes most extensively studied in the learning theory literature is the class of *linear threshold functions*. A *lin-*

ear threshold function (*LTF*) is a Boolean function of the form $f(x) = \text{sgn}(w_1 x_1 + \cdots + w_n x_n - \theta)$ (where $\text{sgn}(y) = 1$ for $y \geq 0$, and $\text{sgn}(y) = -1$ for $y < 0$). In recent work, Matulef et al. [117] give an algorithm for testing LTFs when the domain is $\{1, -1\}^n$, whose query complexity is $\text{poly}(1/\epsilon)$. This is in contrast to the corresponding learning problem that requires $\Omega(n/\epsilon)$ queries (this can be shown to follow from, e.g., [111]). The algorithm of Matulef et al. is quite complex, and here we only mention that to obtain their result they first consider the case that the domain is $\Re^n$ and the underlying distribution is Gaussian. They give an algorithm with $\text{poly}(1/\epsilon)$ for this case, which they later modify for the case that the domain is $\{1, -1\}^n$ and the underlying distribution is uniform.

5

Other Models of Testing

In this section, we consider distribution-free testing (with queries), and learning from uniformly distributed random examples (i.e., without queries) under the uniform distribution. The main results discussed in this section are summarized in Table 5.1.

Table 5.1 Results in other models of testing. As shown in Section 5.1, the first result generalizes to any function class that has a self-corrector. The result for uniform examples extends to size-s decision trees over $[0,1]^d$ and certain neural networks over $[0,1]^d$ but the results are weaker.

Model	Function class	Queries/Examples	Reference
dist-free	deg-d polynomials $\lvert F\rvert = \Omega(d)$	same as standard testing	[95]
dist-free	monotone $f : \Sigma^n \to R$	$O((2\log\lvert\Sigma\rvert)^n/\epsilon)$ $\exp(\Omega(n))$, $\lvert\Sigma\rvert = \lvert R\rvert = 2$	[95]
dist-free	monomials, decision-lists and linear thresh. functions	$\Omega((n/\log n)^{1/5})$	[79]
examples (uniform)	s-interval functions (rej. boundary s/ϵ)	$O(\sqrt{s}/\epsilon^{3/2})$	[105]

5.1 Distribution-Free Testing

The notion of distribution-free testing (with or without queries) was introduced in [82]. However, in that paper it was only observed (as shown in Proposition 2.1) that distribution-free (proper) learning implies distribution-free testing. Other than that, in [82], there were only negative results about distribution-free testing of graph properties, which have very efficient standard testing algorithms (that is, that work under the uniform distribution).

The first positive results for distribution-free testing (with queries) were given by Halevy and Kushilevitz [92, 95]. They describe a distribution-free testing algorithm for degree-d multivariate polynomials (over large fields) with query complexity linear in $1/\epsilon$ and polynomial in d (like the standard testing algorithm of [128]), and a distribution-free monotonicity testing algorithm for functions $f : \Sigma^n \to R$ with query complexity $O((2\log|\Sigma|)^n/\epsilon)$. As we shall discuss later in this subsection, the first result is actually more general, and gives certain sufficient conditions for obtaining distribution-free testing algorithms from standard testing algorithms.

As for the second result for monotonicity, the complexity of the algorithm has exponential dependence on the dimension n of the input. This is in contrast to standard testing algorithms [63, 81], where the dependence on n is linear (to be precise, the complexity is $O(n\log|\Sigma|\log|R|/\epsilon)$, where $|R|$ is the effective size of the range of the function, that is, the number of distinct values of the function). In a further investigation of distribution-free testing of monotonicity [94, 95], Halevy and Kushilevitz showed that the exponential dependence on n is unavoidable even in the case of Boolean functions over the Boolean hypercube (that is, $|\Sigma| = |R| = 2$).

Motivated by positive results for standard testing of several classes of Boolean functions (as described in Section 4) Glasner and Servedio [79] asked whether these results can be extended to the distribution-free model of testing. Specifically, they consider monotone and general monomials (conjunction), decisions lists, and linear threshold functions. They prove that for these classes, in contrast to standard testing, where the query complexity does not depend on n, every distribution-free

testing algorithm must make $\Omega((n/\log n)^{1/5})$ queries (for constant ϵ). While there is still a gap between this lower bound and the upper bound implied by learning these classes, a strong dependence on n is unavoidable in the distribution-free case.

Finally, we note that Halevy and Kushilevitz [93] also study distribution-free testing of graph properties in sparse graphs, and give an algorithm for distribution-free testing of connectivity, with similar complexity to the standard testing algorithm for this property.

We next describe the general result for obtaining distribution-free testing algorithms from standard testing algorithms when the function class has a self-corrector. The algorithm for distribution-free testing of monotonicity is briefly discussed in Section 6.1 (as part of a discussion on results for testing monotonicity). Since the lower bound constructions are somewhat complex, we do not include them in this survey and refer the interested reader to the respective papers mentioned earlier.

5.1.1 Distribution-free Testing of Properties with Self-correctors

Halevy and Kushilevitz introduce the notion of a *property self corrector*, which generalizes the notion of a self-corrector, introduced by Blum et al. [42] (and which was already mentioned earlier in this survey, e.g., in Subsection 3.1.1).

Definition 5.1. A γ-self-corrector for a class of functions $\mathcal{F}$ is a probabilistic oracle machine M, which is given oracle access to an arbitrary function $f : X \to R$ and satisfies the following conditions (where M^f denotes the execution of M when given oracle access to f):

- If $f \in \mathcal{F}$ then $\Pr[M^f(x) = f(x)] = 1$ for every $x \in X$.
- If there exists a function $g \in \mathcal{F}$ such that $\text{dist}(f, g) \leq \gamma$, then $\Pr[M^f(x) = g(x)] \geq 2/3$ for every $x \in X$.

In this definition, the distance (i.e., the measure $\text{dist}(\cdot, \cdot)$) is defined with respect to the uniform distribution. However, it will be useful for distribution-free testing (when the distance is measured with respect

to some fixed but unknown distribution). Observe that the second condition in Definition 5.1 implies that either g is unique, or, if there is more than one function in $\mathcal{F}$ that is ϵ-close to f, then M^f is consistent with one such function (on at least 2/3 of the inputs x).

Theorem 5.1. Let $\mathcal{F}$ be a class of functions that has a standard testing algorithm T and a γ-self-corrector M. Let $Q_T(\cdot)$ be the query complexity of T (as a function of the distance parameter ϵ) and let Q_M be the query complexity of M (that is, the number of queries required in order to determine $M^f(x)$). Then there exists a distribution-free testing algorithm for $\mathcal{F}$ with query complexity $O(Q_T(\min\{\epsilon,\gamma\}) + Q_M/\epsilon)$.

We now describe the distribution-free testing algorithm referred to in Theorem 5.1. We assume that the distance parameter ϵ is smaller than γ (or else we set ϵ to γ).

Algorithm 5.1 (Distribution-free Test Based on Self-correction).

(1) Run the standard testing algorithm T on f, 24 (independent) times with the distance parameter ϵ. If T outputs reject in at least half of these executions then halt and output reject.

(2) Repeat $2/\epsilon$ times:

   (a) Sample a point $x \in X$ according to the underlying distribution D.

   (b) Repeat twice: Compute $M^f(x)$ and query $f(x)$. If $M^f(x) \neq f(x)$ then output reject (and exit).

(3) If no iteration caused rejection then output accept.

Proof of Theorem 5.1: Clearly the query complexity of Algorithm 5.1 is as stated in Theorem 5.1. Hence we turn to proving its correctness. Consider first the case that $f \in \mathcal{F}$. In such a case the standard testing algorithm T should accept with probability at least 2/3, and the probability that it rejects in at least half of its 24 independent executions is less than 1/3. Assume that such an event did not occur. By the first

condition in Definition 5.1, for every $x \in X$, we have that $M^f(x) = f(x)$ with probability 1. Hence, the second step of the algorithm never causes rejection. It follows that the algorithm accepts with probability at least 2/3. (Note that if T has one-sided error then so does Algorithm 5.1.)

In what follows, in order to distinguish between the case that distance is measured with respect to the uniform distribution, and the case that it is measured with respect to the underlying distribution D, we shall use the terms (ϵ, U)-close (or far) and (ϵ, D)-close (or far), respectively. Assume now that f is (ϵ, D)-far from $\mathcal{F}$. If f is also (ϵ, U)-far from $\mathcal{F}$ then it is rejected by T with probability at least 2/3, and is therefore rejected by the algorithm in its first step with probability at least 2/3. Hence assume that f is (ϵ, U)-close to $\mathcal{F}$.

In such a case, by the second condition in Definition 5.1, for every $x \in X$, $\Pr[M^f(x) = g(x)] \geq 2/3$, where g is a fixed function in $\mathcal{F}$ that is (γ, U)-close to f and the probability is taken over the internal coin flips of M (recall that $\epsilon \leq \gamma$ so such a function g exists). In particular, for any point x such that $f(x) \neq g(x)$ we have that $\Pr[M^f(x) \neq f(x)] \geq 2/3$. Thus, if in one of the $(2/\epsilon)$ iterations of the second step of the algorithm we obtain such a point x, then the algorithm rejects with probability at least $1 - (1/3)^2 = 8/9$ (since it computes $M^f(x)$ twice). But since f is (ϵ, D)-far from $\mathcal{F}$, for every function $h \in \mathcal{F}$, we have that $\Pr_{x \sim D}[f(x) \neq h(x)] > \epsilon$, and in particular this is true for g. Hence the probability that the algorithm does not obtain any point x for which $f(x) \neq g(x)$ is at most $(1 - \epsilon)^{2/\epsilon} < \exp(-2) < 1/6$. It follows that the algorithm rejects with probability at least $1 - (1/9 + 1/6) > 2/3$, as required. □

In particular, Theorem 5.1 can be applied to obtain distribution-free property testing algorithms for all algebraic properties described in Section 3, as well as singletons (since they are a subclass of the class of linear functions). This is also true for the class of k-juntas, since they are a subclass of degree-k multivariate polynomials.

5.2 Testing From Random Examples

Similarly to the case of distribution-free testing, the notion of testing from random examples was considered in [82]. For this testing model

too, the paper includes negative results for testing graph properties from random example only but does not include any positive results for testing algorithms with complexity strictly smaller than that required for learning. The only positive results we are aware of in this model are those presented by Kearns and Ron [105], which are described next. The functions studied in [105] are decision trees over $[0,1]^d$, and a special case of neural networks. We first extend the notion of property testing by allowing the relaxation of the rejection criteria. We focus in this extension on testing from random examples distributed according to the uniform distribution.

Definition 5.2 (Testing with a Rejection Boundary). Let $\mathcal{F}$ be a class of functions from domain X to range R, let $\mathcal{F}' \supseteq \mathcal{F}$, and let $0 < \epsilon \leq 1$. A testing algorithm for membership in $\mathcal{F}$ with rejection boundary $(\mathcal{F}', \epsilon)$ is given access to uniformly distributed examples labeled according to an unknown function $f : X \to R$.

- If $f \in \mathcal{F}$ then the algorithm should accept with probability at least $2/3$;
- If $\text{dist}(f, \mathcal{F}') > \epsilon$ then the algorithm should reject with probability at least $2/3$.

5.2.1 Interval Functions

Here we describe and analyze a testing algorithm for the class of *interval functions*. This is a special case of decision trees (which are defined precisely in Subsection 5.2.2) and the study of this simple class gives some of the flavor of the other results in [105].

For any size s, the class of interval functions with at most s intervals, denoted INT_s, is defined as follows. Each function $f \in \text{INT}_s$ is defined by $t \leq s - 1$ *switch points*, $a_1 < \cdots < a_t$, where $a_i \in (0,1)$. The value of f is fixed in each *interval* that lies between two switch points, and alternates between 0 and 1 when going from one interval to the next.

It is not hard to verify that learning the class INT_s requires $\Omega(s)$ examples (even when the underlying distribution is uniform). In fact,

$\Omega(s)$ is also a lower bound on the number of membership queries necessary for learning this class. As we show below, the complexity of *testing* under the uniform distribution is much lower — it suffices to observe $O(\sqrt{s})$ random examples. We also note that if the algorithm is allowed queries then the number of queries that suffice for testing is independent of s and linear in $1/\epsilon$.

Theorem 5.2. For any integer $s > 0$ and $\epsilon \in (0, 1/2]$, the class of interval functions INT_s is testable with rejection boundary $\mathrm{INT}_{s/\epsilon}$ under the uniform distribution using $O(\sqrt{s}/\epsilon^{1.5})$ examples. The running time of the testing algorithm is linear in the number of examples used.

The basic property of interval functions that the testing algorithm exploits is that *most* pairs of *close* points belong to the same interval, and thus have the same label. The algorithm scans the sample for such close pairs and accepts only if the fraction of pairs in which both points have the same label is above a certain threshold. In the proof below we quantify the notion of closeness, and analyze its implications both on the rejection boundary for testing and on the number of examples needed. Intuitively, there is the following tradeoff: as the distance between the points in a pair becomes smaller, we are more confident that they belong to the same interval (in the case that $f \in \mathrm{INT}_s$); but the probability that we observe such pairs of points in the sample becomes smaller, and the class $\mathcal{F}'$ in the rejection boundary becomes larger.

Proof. We first describe the testing algorithm. Let $s' = s/\epsilon$, and consider the partition of the domain $[0, 1]$ imposed by a one-dimensional grid with s' equal-size *cells* (intervals) $c_1, \ldots, c_{s'}$. Given a uniformly selected sample S of size $m = \Theta(\sqrt{s'}/\epsilon)$ $(=\Theta(\sqrt{s}/\epsilon^{1.5}))$, we partition the examples $x_1, \ldots, x_m$ into bins, $B_1, \ldots, B_{s'}$, where the bin B_j contains points belonging to the cell c_j. Within each (nonempty) bin B_j, let $x_{i_1}, x_{i_2}, \ldots, x_{i_t}$ be the examples in B_j, ordered according to their appearance *in the sample*, and let us pair the points in each such bin according to this order (thus, x_{i_1} is paired with x_{i_2}, x_{i_3} with x_{i_4}, and so on). We call these pairs the *close* pairs, and we further call a pair

pure if it is close *and* both points have the same label. The algorithm accepts f if the fraction of pure pairs (among all close pairs) is at least $1 - 3\epsilon/4$; otherwise it rejects.

The first central observation is that by the choice of m, with high probability the number m'' of close pairs is at least $m' = \Theta(1/\epsilon)$. To obtain this lower bound on m'', assume that we restricted our choice of pairs by breaking the random sample into $4m'$ random subsamples, each of size $2\sqrt{s'}$, and considered only close pairs that belong to the same subsample. We claim that by the well-known *Birthday Paradox*, for each subsample, the probability that the subsample contains a close pair is at least $1/2$. To see why this is true, think of each subsample S' as consisting of two parts, S'_1 and S'_2, each of size $\sqrt{s'}$. We consider two cases: In the first case, S'_1 already contains two examples that belong to a common cell and we are done. Otherwise, each example in S'_1 belongs to a different cell. Let this set of $\sqrt{s'}$ cells be denoted C and recall that all cells have the same probability mass $1/s'$. Thus, the probability that S'_2 does not contain any example from a cell in C is

$$\left(1 - |C| \cdot \frac{1}{s'}\right)^{|S'_2|} = \left(1 - \frac{1}{\sqrt{s'}}\right)^{\sqrt{s'}} < e^{-1} < 1/2 \tag{5.1}$$

as claimed. Hence, with very high probability, at least a fourth of the subsamples (that is, at least m') will contribute a close pair, in which case $m'' \geq m'$. Since the close pairs are equally likely to fall in each cell c_j and are uniformly distributed within each cell, the correctness of the algorithm when using examples reduces to the correctness of the following algorithm, which is given query access to f. The algorithm uniformly and independently selects $m' = O(1/\epsilon)$ of the grid cells, uniformly draws a pair of points in each cell chosen, and queries f on these pairs of points. The acceptance criteria is as in the original algorithm. We next establish the correctness of the latter algorithm (that performs queries).

Case 1: $f \in \mathrm{INT}_s$. For $t = 1 \ldots, m'$, let χ_t be a random variable that is 0 if the t^{th} close pair is pure, and 1 otherwise. Thus χ_t is determined by a two-stage process: (1) the choice of the t^{th} grid cell c_t; (2) the selection of the two points inside that cell. When c_t is a subinterval of some interval of f, then the points always have the same label, and

otherwise they have a different label with probability at most $1/2$. Since f has at most s intervals, the number of cells that intersect intervals of f (that is, are *not* subintervals of fs intervals) is at most s, and since there are s/ϵ grid cells, the probability of selecting such a cell is at most ϵ. It follows that for each t,

$$\mathrm{Exp}[\chi_t] \leq \epsilon \cdot (1/2) + (1-\epsilon) \cdot 0 = \epsilon/2. \tag{5.2}$$

By a multiplicative Chernoff bound (see Appendix A), with probability at least $2/3$, the average of the χ_ts (which is just the fraction of close pairs that are not pure), is at most $3\epsilon/4$, as required.

Case 2: $\mathrm{dist}(f, \mathrm{INT}_{s'}) > \epsilon$. In order to prove that in this case the algorithm rejects with probability at least $2/3$ we prove the contrapositive: if the algorithm accepts with probability greater than $1/3$, then there exists a function $f' \in \mathrm{INT}_{s'}$ that is ϵ-close to f.

Let $f' \in \mathrm{INT}_{s'}$ be the (equally spaced) s'-interval function that gives the majority label according to f to each grid cell. We claim that if f is accepted with probability greater than $1/3$ then $\mathrm{dist}(f, f') \leq \epsilon$. Assume, contrary to the claim, that $\mathrm{dist}(f, f') > \epsilon$. For each grid cell c_j, let $\epsilon_j \in [0, 1/2]$ be the probability mass of points in c_j that have the minority label of f among points in c_j. Thus, $\mathrm{dist}(f, f') = \mathrm{Exp}_j[\epsilon_j]$, and so, by our assumption, $\mathrm{Exp}_j[\epsilon_j] > \epsilon$. On the other hand, if we define χ_t as in Case 1, then we get that

$$\mathrm{Exp}[\chi_t] = \mathrm{Exp}_j\left[2\epsilon_j(1-\epsilon_j)\right] \geq \mathrm{Exp}_j[\epsilon_j], \tag{5.3}$$

where the second inequality follows from $\epsilon_j \leq 1/2$. By our assumption on f, $\mathrm{Exp}[\chi_t] > \epsilon$, and by applying a multiplicative Chernoff bound, with probability greater than $2/3$, the average over the χ_ts is greater than $3\epsilon/4$ (which causes the algorithm to reject). □

It is also proven in [105] that the dependence on $\sqrt{s}$ is unavoidable. Specifically, it is shown that distinguishing with probability at least $2/3$ between a function selected randomly from a certain subclass of INT_s and a completely random function over $[0,1]$ requires $\Omega(\sqrt{s})$ examples. This implies the same lower bound for testing with rejection boundary $\mathrm{INT}_{s/\epsilon}$ for constant ϵ, since for every s, with very high probability, a random function will be $\Omega(1)$-far from any interval function in $\mathrm{INT}_{s'}$ for $s' = O(s)$.

5.2.2 Decision Trees and Aligned Voting Networks

A *Decision Tree* over $[0,1]^d$ is given as input $x = x_1, \ldots, x_d$. The (binary) decision at each node of the tree is whether $x_i \geq a$ for some $i \in \{1, \ldots, d\}$ and $a \in [0,1]$. The labels of the leaves of the decision tree are in $\{0,1\}$. We define the *size* of such a tree to be the number of leaves. Thus, every tree of size s over $[0,1]^d$ determines a partition of the domain $[0,1]^d$ into at most s axis aligned rectangles, each of dimension d (the leaves of the tree), where all points belonging to the same rectangle have the same label.

Aligned Voting Networks are a restricted class of neural networks over $[0,1]^d$. These are essentially neural networks in which the hyperplane defining each hidden unit is constrained to be parallel to some coordinate axis, and the output unit takes a majority vote of the hidden units. To be precise, an *aligned hyperplane* over $[0,1]^d$ is a function $h : [0,1]^d \to \{1,-1\}$ of the form $h(x) = \text{sgn}(x_i - a)$ for some dimension $i \in \{1, \ldots, d\}$ and some $a \in [-1,1]$. An aligned voting network over $[0,1]^d$ is a function $f : [0,1]^d \to \{1,-1\}$ of the form:

$$f(x) = \text{sgn}\left(\sum_{j=1}^{s} h_j(x)\right), \tag{5.4}$$

where each $h_j(x)$ is an aligned hyperplane over $[0,1]^d$. The size of f is the number of voting hyperplanes s.

An alternative way of viewing an aligned voting network f is as a *constrained* labeling of the cells of a rectilinear partition of $[0,1]^d$. For each dimension i, we have *positions* $a_i^j \in [0,1]$ and *orientations* $u_i^j \in \{+1,-1\}$. The hyperplanes $x_i = a_i^j$ define the rectilinear partition, and f is constant over each cell c: for any x, we define

$$\#(x) = \sum_{i=1}^{d} \sum_{j=1}^{s_i} \text{sgn}(x_i - u_i^j a_i^j), \tag{5.5}$$

(where s_i is the number of aligned hyperplanes that project on dimension i), and then $f(x) = \text{sgn}(\#(x))$. By extension, for each cell c of the partition, we define $\#(c)$ as the constant value of $\#(x)$ for all $x \in c$, and $f(c)$ as the constant value of $f(x)$ for all $x \in c$.

The algorithms for testing decision trees and aligned voting networks apply a similar high level idea as the algorithm for interval functions. Roughly speaking, they decide whether to accept or reject the function f by pairing "nearby" points, and checking that such pairs have the same label according to f. A common theme in the analysis is that the class in question, which we denote by $\mathcal{F}_s$, can be "approximated by" a bounded number of fixed partitions of the domain. More precisely, if we consider the class of functions $\mathcal{H}$ defined by these partitions (when we allow all labelings of the cells of the partitions), then for every function in $\mathcal{F}_s$ there exists a function in $\mathcal{H}$ that approximates it. Furthermore, these partition functions can be implemented by a class $\mathcal{F}_{s'}$, where $s' \geq s$. The testing algorithms essentially perform the same task: for each fixed partition, pairs of points that belong to a common cell of the partition are considered, and if there is a sufficiently strong bias among these pairs toward having a common label, then the tested function is accepted.

However, the algorithms (which are somewhat more complex than the one for testing intervals), give weaker results in the sense of the rejection boundary $(\mathcal{F}_{s'}, \epsilon)$ they work for. This is both in terms of the relation between s' and s, and in terms of the ϵ they work for. Namely, as opposed to the case of interval functions, where the algorithm can work with any ϵ, for the more complex classes, the algorithm works only for certain settings of ϵ that are bounded away from $1/2$ by a function that depends exponentially on d. Thus these results can be seen as "weak testing," analogously to "weak learning."

5.2.3 Testing and Weak Learning

In fact, the relation between weak learning and a certain form of weak testing is formalized in [105] as described next.

Definition 5.3. Let $\mathcal{F}$ be a class of functions over a domain X, and let D be a distribution over X. We say that $\mathcal{F}$ is testable against a random function in m examples with respect to D if there is an algorithm T such that:

- If T is given m examples drawn according to D and labeled by any $f \in \mathcal{F}$, then T accepts with probability at least $2/3$.

- If T is given m examples drawn according to D and labeled randomly, then T rejects with probability at least 2/3. The probability here is taken both over the choice of examples and their random labels.

Recall that Proposition 2.1 established that proper learning implies testing. The proposition can be easily generalized to show that if a function class $\mathcal{F}$ is learnable using a hypothesis class $\mathcal{H} \supseteq \mathcal{F}$ with accuracy ϵ using m examples, then for every ϵ' the class $\mathcal{F}$ is testable with rejection boundary $(\mathcal{H}, \epsilon + \epsilon')$ using $m + O(1/(\epsilon')^2)$ examples.

Below we give a proposition concerning the reverse direction — namely, any class that is efficiently testable against a random function (as defined in Definition 5.3) is efficiently weakly learnable. Observe that testing against a random function (with respect to a particular distribution D) is no harder than testing with respect to a certain rejection boundary class $\mathcal{F}'$ whenever (with respect to D), a random function is far from any function in the class $\mathcal{F}'$ (with high probability over the choice of the random function).

Proposition 5.3. Let $\mathcal{F}$ be a class of functions over domain X and let D be a distribution over X. If $\mathcal{F}$ is testable against a random function in m examples with respect to D, then $\mathcal{F}$ is weakly learnable with respect to D with advantage $\Omega(1/m)$ and constant confidence in $\tilde{O}(m^2)$ examples.

Proof. Let T be the testing algorithm that distinguishes between functions in $\mathcal{F}$ and a random function. We start by using a standard technique first applied in the cryptography literature [87]. Let us fix any function $f \in \mathcal{F}$, and consider the behavior of the algorithm when it is given a random sample drawn according to D and labeled *partly by f and partly randomly*. More precisely, for $i = 0, \ldots, m$, let p_i be the probability, taken over a random sample $x_1, \ldots, x_m$ drawn according to D, and a vector $\vec{r}$ uniformly chosen vector in $\{0,1\}^{m-i}$, that the test T accepts when given as input $\langle x_1, f(x_1)\rangle, \ldots, \langle x_i, f(x_i)\rangle$, $\langle x_{i+1}, r_1\rangle\rangle, \ldots, \langle x_m, r_{m-i}\rangle\rangle$.

Since $p_m \geq 2/3$, while $p_0 \leq 1/3$, there must exist an index $1 \leq i \leq m$ such that $p_i - p_{i-1} = \Omega(1/m)$. Thus, by observing $\tilde{O}(m^2)$ examples (and generating the appropriate number of random labels) we can find an index i such that T has significant sensitivity to whether the i^{th} example is labeled by f or randomly. From this it can be shown [104] that by taking another $\tilde{O}(m^2)$ examples, we can find a *fixed* sequence S_1 of i examples labeled according to f, and a fixed sequence S_2 of $m - i$ examples having an arbitrary (but fixed) 0/1 labeling such that the difference between the probability that T accepts when given as input $S_1, \langle x, f(x)\rangle, S_2$ and the probability that it accepts when given as input $S_1, \langle x, \neg f(x)\rangle, S_2$, is $\Omega(1/m)$, where now the probability is taken only over the draw of x. Let $h(x)$ be the following probabilistic function. If $T(S_1, \langle x, 0\rangle, S_2) = T(S_1, \langle x, 1\rangle, S_2)$, then h outputs the flip of a fair coin. If for $b \in \{0,1\}$, $T(S_1, \langle x, b\rangle, S_2) = accept$ and $T(S_1, \langle x, \neg b\rangle, S_2) = reject$, then h outputs b. Then from the preceding arguments, h has an advantage of $\Omega(1/m)$ over a random coin in predicting f. □

6

Other Results

In this section we mention, quite briefly, several additional directions of research within the area of property testing and the corresponding known results.

6.1 Monotonicity

A very basic property of functions, which plays a role in learning theory, is *monotonicity*. For a partially ordered set (poset) X (e.g., $X = \{0,1\}^n$) and a totally ordered domain R, we say that a function $f : X \to R$ is *monotone* if for every $x, y \in X$, if $x < y$ then $f(x) \leq f(y)$.

Boolean functions over the Boolean hypercube. In this case, that is when $X = \{0,1\}^n$ (so that the partial order is the natural lexicographic order over strings), and $R = \{0,1\}$, it is possible to test monotonicity (in the standard model, that is, under the uniform distribution and with queries) using $O(n/\epsilon)$ queries [81]. The algorithm simply selects, uniformly and at random, $\Theta(n/\epsilon)$ pairs of points that differ on a single bit and checks whether the function violates monotonicity on any of the selected pairs. Clearly, every monotone function is accepted with

probability 1. The heart of the analysis is in showing that the probability that a function f violates monotonicity on such a pair of neighboring points is at least the distance of the function to monotonicity, divided by n. This is proved by showing that if the fraction of violating pairs is at most $\delta_M(f)$, then it is possible to "fix" the function f and make it a monotone function by modifying the value of f on at most an $(n\delta_M(f))$-fraction of the points.

The aforementioned testing algorithm makes essential use of queries. Goldreich et al. [81] show that this is no coincidence — any monotonicity tester that works under the uniform distribution, but is not allowed queries, must have much higher complexity. Specifically, they prove a lower bound of $\Omega(\sqrt{2^n/\epsilon})$ on the number of examples, for every $\epsilon = O(n^{-3/2})$. They also show that this lower bound is tight. Namely, there exists a tester for monotonicity that only utilizes random examples and uses at most $O(\sqrt{2^n/\epsilon})$ examples. As noted in [81], this tester is significantly faster than any learning algorithm for the class of all monotone concepts when the allowed error is $O(1/\sqrt{n})$: Learning (under the uniform distribution) requires $\Omega(2^n/\sqrt{n})$ examples (and at least that many queries) [104]. In contrast, "weak learning" [106] is possible in polynomial time. Specifically, the class of monotone concepts can be learned in polynomial time with error at most $1/2 - \Omega(1/\sqrt{n})$ [39] (though no polynomial-time learning algorithm can achieve an error of $1/2 - \omega(\log(n)/\sqrt{n})$) [39]).

Non-Boolean functions. Returning to standard testing, Ergun et al. [64] considered the case that $X = \Sigma$, where Σ is a totally ordered finite set and R is any totally ordered set. They referred to the testing problem as "Spot checking of sorting" and gave an algorithm whose query complexity is $O(\log n/\epsilon)$. Ergun et al. [64] also gave a lower bound for non adaptive testing algorithms, which, combined with a result of Fischer [67], implies a lower bound of $\Omega(\log|\Sigma|)$ for any testing algorithm (when ϵ is constant).

Batu et al. [30] extended the algorithm of Ergun et al. [64] to higher dimensions, that is, for $X = \Sigma^n$ (and any range R), at an exponential cost in the dimension n. The complexity of their algorithm is $O((2\log|\Sigma|)^n/\epsilon)$. Halevy and Kushilevitz [92] reduced the complexity

(for sufficiently large Σ) to $O(n4^n \log|\Sigma|/\epsilon)$, and Ailon and Chazelle [1] further improved this bound to $O(n2^n \log|\Sigma|/\epsilon)$.

Dodis et al. [63] showed that it is possible to reduce the dependence on the dimension n to linear, and obtain query complexity $O(n \log|\Sigma| \log|R|/\epsilon)$, where the dependence on $\log|R|$ can be replaced by $n \log|\Sigma|$.

General Posets. Fischer et al. [72] considered the case in which X is a general poset. They showed that testing monotonicity of Boolean functions over general posets is equivalent to the problem of testing 2CNF assignments (namely, testing whether a given assignment satisfies a fixed 2CNF formula or is far from any such assignment). They also showed that for every poset it is possible to test monotonicity over the poset with a number of queries that is sublinear in the size of the domain poset; specifically, the complexity grows like a square root of the size of the poset. Finally, they give some efficient algorithms for several special classes of posets (e.g., posets that are defined by trees).

Distribution-free testing. As mentioned in Section 5.1, Halevy and Kushilevitz [95] gave a distribution-free monotonicity testing algorithm for functions $f : \Sigma^n \to R$ with query complexity $O((2\log|\Sigma|)^n/\epsilon)$. They also showed that the exponential dependence on n is unavoidable even in the case of Boolean functions over the Boolean hypercube (that is, $|\Sigma| = |R| = 2$).

Related properties. The related properties of whether a one-dimensional function $f : [n] \to \Re$ is convex, and whether a two dimensional function, or $n \times n$ matrix, is submodular, are studied in [123].

6.2 Clustering

Alon et al. [5] consider the problem of testing whether a set of points can be clustered into a given number k of clusters with a given bound on the cost of the clustering for a fixed cost measure. To be precise, for a set X of points in $\Re^d$, the set X is (k, b)-*clusterable* if X can be partitioned into k subsets (*clusters*) so that the diameter (alternatively, the radius) of each cluster is at most b. For $\beta > 0$ and $0 \leq \epsilon \leq 1$, we say that X

is ϵ-far from being $(k,(1+\beta)b)$-clusterable if more than $\epsilon \cdot |X|$ points should be removed from X so that it becomes $(k,(1+\beta)b)$-clusterable.

Alon et al. [5] describe and analyze algorithms that by sampling from a set X, distinguish between the case that X is (k,b)-clusterable and the case that X is ϵ-far from being $(k,(1+\beta)b)$-clusterable for any given $0 < \epsilon \leq 1$ and for various values of $0 \leq \beta \leq 1$. The algorithms run in time *independent* of $n = |X|$, and use a sample from X that has size polynomial in k and ϵ. Specifically:

(1) For the radius cost, and when the underlying distance between points is the Euclidean distance, the algorithm works for $\beta = 0$ and the sample size is $\tilde{O}(d \cdot k/\epsilon)$.
(2) For the diameter cost and the Euclidean distance, the sample is of size $\tilde{O}\left(\frac{k^2}{\epsilon} \cdot \left(\frac{2}{\beta}\right)^{2d}\right)$. A dependence on $1/\beta$, as well as an exponential dependence on the dimension, are unavoidable: there is a lower bound of $\Omega(\beta^{-(d-1)/4})$ on the size of the sample required for testing, for $k = 1$ and a constant ϵ.
(3) If the underlying distance is a general metric, then there is an algorithm that works for $\beta = 1$, and for both the radius cost and the diameter cost, the sample selected is of size $O(k/\epsilon)$. Interestingly, any algorithm for testing diameter clustering for $\beta < 1$ under a general metric requires a sample of size $\Omega(\sqrt{n/\epsilon})$.

The running time of the third algorithm is $O(k^2/\epsilon)$. The first two algorithms work by running an exact clustering procedure on the selected sample. Since the corresponding clustering problems are $\mathcal{NP}$-hard (when an exact solution is required), we do not know of procedures that run in less than exponential time in k and d.

The algorithms can also be used to find *approximately good* clusterings. Namely, these are clusterings of all but an ϵ-fraction of the points in X that have optimal (or close to optimal) cost. The benefit of the algorithms is that they construct an *implicit representation* of such clusterings in time independent of $|X|$. That is, without actually having to partition all points in X, the implicit representation can be used to answer queries concerning the cluster any given point belongs to.

These results belong to a family of sublinear approximation algorithms for various cost measures (*cf.* [57, 97, 118]).

6.3 Properties of Distributions

The following types of problems were considered in several works. Given access to samples drawn from an unknown distribution D, the goal is to determine, with high success probability, whether D has a particular property (e.g., it is uniform over its domain), or possibly compute an approximation to a certain measure on distributions (e.g., entropy).[1] Similarly, the algorithm may have access to samples drawn from a pair of distributions, D_1 and D_2, and the property/measure in question is for pairs of distributions (e.g., their statistical distance). The goal is to perform the task by observing a number of samples that is sublinear in the size of the domain over which the distribution(s) is (are) defined. In what follows, the running times of the algorithms mentioned are linear (or almost linear) in their respective sample complexities.

Testing that distributions are close. Batu et al. [28] consider the problem of determining whether the distance between a pair of distributions over n elements is small (less than $\max\left\{\frac{\epsilon}{4\sqrt{n}}, \frac{\epsilon^2}{32n^{1/3}}\right\}$), or large (more than ϵ) according to the L_1 distance. They give an algorithm for this problem that takes $O(n^{2/3}\log n/\epsilon^4)$ independent samples from each distribution. This result is based on testing closeness according to the L_2 distance, which can be performed using $O(1/\epsilon^4)$ samples. This in turn is based on estimating the deviation of a distribution from uniform [84].

In recent work, Valiant [135] shows that $\Omega(n^{2/3})$ samples are also necessary for this testing problem (with respect to the L_1 distance). For the more general problem of distinguishing between the case that the two distributions are ϵ_1-close and the case that they are ϵ_2-far, where ϵ_1 and ϵ_2 are both constants, Valiant [135] proves an almost linear (in n) lower bound.

One can also consider the problem of testing whether a distribution D_1 is close to a fixed and known distribution D_2, or is far from it (letting

[1] An alternative model may allow the algorithm to obtain the probability that the distribution assigns to any element of its choice. We shall not discuss this variant.

D_2 be the uniform distribution is a special case of this problem). Batu et al. [27] showed that it is possible to distinguish between the case that the distance in L_2 norm is $O\left(\frac{\epsilon^3}{\sqrt{n}\log n}\right)$ and the case that the distance is greater than ϵ using $\tilde{O}(\sqrt{n}\cdot\text{poly}(1/\epsilon))$ samples from D_1.

Testing random variables for independence. Batu et al. [27] also showed that it is possible to test whether a distribution over $[n] \times [m]$ is independent or is ϵ-far from any independent joint distribution, using a sample of size $\tilde{O}(n^{2/3}m^{1/3}\text{poly}(1/\epsilon))$.

Approximating the entropy. A very basic and important measure of distributions is their (binary) entropy. The main result of Batu et al. [25] is an algorithm that computes a γ-multiplicative approximation of the entropy using a sample of size $O(n^{(1+\eta)/\gamma^2}\log n)$ for distributions with entropy $\Omega(\gamma/\eta)$, where n is the size of the domain of the distribution and η is an arbitrarily small positive constant. They also show that $\Omega(n^{1/(2\gamma^2)})$ samples are necessary. A lower bound that matches the upper bound of Batu et al. [25] is proved in [135].

Approximating the support size. Another natural measure for distributions is their support size. To be precise, consider the problem of approximating the support size of a distribution when each element in the distribution appears with probability at least $\frac{1}{n}$. This problem is closely related to the problem of approximating the number of distinct elements in a sequence of length n. For both problems, there is a lower bound on the sample complexity that is nearly linear in n, which is applicable even for approximation with *additive* error [126].

A unifying approach to testing symmetric properties of distributions. Valiant [135] obtains the lower bounds mentioned in the foregoing discussion as part of a general study of testing symmetric properties of distributions (or pairs of distributions). That is, he considers properties of distributions that are preserved under renaming of the elements in the domain of the distributions. Roughly speaking, his main finding is that for every such property, there exists a threshold such that elements whose probability weight is below the threshold "do not matter" in terms of the task of testing. This implies that such properties have a "canonical tester" that bases its decision on its estimate of the probability weight of elements that appear sufficiently often in the sample

("heavy elements"), and essentially ignores those elements that do not appear sufficiently often. In the other direction, lower bounds can be derived by constructing pairs of distributions on which the decision of the tester should be different, but that give the same probability weight to the heavy elements (and may completely differ on all light elements).

Other results. Other works on testing properties of distributions include [4, 29, 128].

6.4 Testing Membership in Regular Languages, Branching Programs, and Other Languages

Alon et al. [13] consider the following problem of testing membership in a regular language. For a predetermined regular language $L \subseteq \{0,1\}^*$, the tester for membership in L should accept every word $w \in L$ with probability at least 2/3, and should reject with probability at least 2/3 every word w that differs from any $w' \in L$ on more than $\epsilon|w|$ bits. We stress that the task is not to decide whether a language is regular (which can be seen as a relaxation of learning a regular language), but rather the language is predetermined, and the test is for membership in the language.

The query complexity and running time of the testing algorithm for membership in a regular language is $\tilde{O}(1/\epsilon)$, that is, independent of the length n of w. (The running time is dependent on the size of the (smallest) finite automaton accepting L, but this size is considered to be a fixed constant with respect to n.) Alon et al. [13] also show that a very simple context free language (of all strings of the form vv^Ru, where v^R denotes the reversal of v), cannot be tested using $o(\sqrt{n})$ queries.

Newman [121] extended the result of Alon et al. [13] for regular languages and gave an algorithm that has query complexity $\text{poly}(1/\epsilon)$ for testing whether a word w is accepted by a given constant-width oblivious read-once branching program. (It is noted in [46] that the result can be extended to the nonoblivious case.) On the other hand, Fischer et al. [75] showed that testing constant width oblivious read-*twice* branching programs requires $\Omega(n^\delta)$ queries, and Bol-

lig [46] shows that testing read-once branching programs of quadratic size (with no bound on the width) requires $\Omega(n^{1/2})$ queries (improving on [47]).

In both [75] and [46] lower bounds for membership in sets defined by CNF formulae are also obtained, but the strongest result is in [36]: an $\Omega(n)$ lower bound for 3CNF (over n variables). This should be contrasted with an $O(\sqrt{n})$ upper bound that holds for 2CNF [72]. More generally, Ben-Sasoon et al. [36] provide sufficient conditions for linear properties to be hard to test, where a property is linear if its elements from a linear space.

6.5 Testing Graph Properties

One of the main focuses of property testing has been on testing graph properties. Here we only mention, quite briefly, some of the results on testing graph properties in order to give the flavor of research in this sub-area of property testing. The results are partitioned according to the model they are obtained in.

6.5.1 The Adjacency Matrix Model

The first model, introduced in [82], is the *adjacency-matrix* model. In this model the algorithm may perform queries of the form: "is there an edge between vertices u and v in the graph?" That is, the algorithm may probe the adjacency matrix representing the graph. We refer to such queries as *vertex-pair* queries. The notion of distance is also linked to this representation: A graph is said to be ϵ-far from having property $\mathcal{P}$ if more than ϵn^2 edge modifications should be performed on the graph so that it obtains the property, where n is the number of vertices in the graph. In other words, ϵ measures the fraction of entries in the adjacency matrix of the graph that should be modified. This model is most suitable for *dense* graphs in which the number of edges m is $\Theta(n^2)$. For this reason, we shall also refer to it as the *dense-graphs* model.

The algorithms described in [82] are for testing a variety of properties, amongst them: bipartiteness, k-colorability, having a large cut, having a large clique, and a generalized partition property that includes

the former properties as special cases. The query complexity of all algorithms is polynomial in $1/\epsilon$ and independent of n. With the exception of the algorithm for bipartiteness, whose running time is polynomial in $1/\epsilon$, the other algorithms have running time that is exponential in $1/\epsilon$. This is not surprising given the fact that for all the properties but bipartiteness the exact decision problem is $\mathcal{NP}$-hard.[2] Alon and Krivelevich [11] improved the results in [82] for bipartiteness and k-colorability by applying a more sophisticated analysis.

Alon et al. [7] gave algorithms for the class of *first order* graph properties. These are properties that can be formulated by first order expressions about graphs. This covers a large class of graph properties (in particular coloring and subgraph-freeness properties). As in [82] their algorithms do not have a dependence on n, but since they build on the Regularity Lemma of Szemerédi [132], the dependence on $1/\epsilon$ is quite high. Interestingly, Alon [3] proved that for subgraph freeness, if the subgraph is not bipartite, then the dependence on $1/\epsilon$ must be superpolynomial.

A sequence of works by Alon and Shapira [16, 17, 18], together with the work of Fischer and Newman [74] culminated in a characterization of all graph properties that are testable using a number of queries that is independent of n [9]. As the title of the paper says: "It's all about regularity." To be a little more precise, the characterization says (roughly) that a graph property $\mathcal{P}$ is testable using a number of queries that is independent of n *if and only if* testing $\mathcal{P}$ can be reduced to testing the property of satisfying one of a finitely many Szemerédi-partitions [132]. A different characterization was proved independently by Borgs et al. [48].

Other results in the dense-graphs model (also for directed graphs) include [8, 15, 37, 45, 58, 68, 69, 86, 88]. Extensions to hypergraphs can be found in [14, 19, 59, 73, 110].

[2] Interestingly, a testing algorithm for k-colorability whose query complexity is independent of n is implicit in the earlier work of Alon et al. [6]. They build on a constructive version of the Regularity Lemma of Szemerédi [132] which they prove, and the query complexity of the implied testing algorithm is a tower of poly$(1/\epsilon)$ exponents.

6.5.2 The Bounded-degree Incidence-lists Model

The second model, introduced in [85], is the *bounded-degree incidence-lists* model. In this model, the algorithm may perform queries of the form: "who is the i^{th} neighbor of vertex v in the graph?" That is, the algorithm may probe the incidence lists of the vertices in the graph, where it is assumed that all vertices have degree at most d for some fixed degree-bound d. We refer to these queries as *neighbor* queries. Here too the notion of distance is linked to the representation: A graph is said to be ϵ-far from having property $\mathcal{P}$ if more than ϵdn edge modifications should be performed on the graph so that it obtains the property. In this case ϵ measures the fraction of entries in the incidence lists representation (among all dn entries) that should be modified. This model is most suitable for graphs with $m = \Theta(dn)$ edges; that is, whose maximum degree is of the same order as the average degree. In particular, this is true for *sparse* graphs that have *constant degree*.

Goldreich and Ron [85] gave algorithms in this model for connectivity and more generally k-edge-connectivity, cycle-freeness, being a Eulerian graph, and subgraph freeness. The complexity of all algorithms is polynomial in $1/\epsilon$ and independent of n. On the other hand, they showed that for some properties a dependence on n is unavoidable. In particular, they proved lower bounds of $\Omega(\sqrt{n})$ (for constant ϵ) on testing bipartiteness and expansion of bounded degree graphs. In [83], they gave an almost matching bound for testing bipartiteness using algorithms that perform random walks. In [84], they proposed (but did not prove the correctness of) an algorithm for testing expansion using $\tilde{O}(\sqrt{n}\cdot\text{poly}(1/\epsilon))$ queries, and recently there has been quite a bit of progress on this problem [60, 100, 120].

Czumaj, Shapira and Sohler [56] give a general result for testing bounded-degree graphs: hereditary properties can be testing on non-expanding bounded-degree graphs using a number of queries that is independent of n.

The bounded-degree model was also considered for the study of properties of directed graphs, and in particular acyclicity [37].

6.5.3 A General Model

In [122], it was first suggested to decouple the questions of representation and type of queries allowed from the definition of distance to having a property. Specifically, it was suggested that distance be measured simply with respect to the number of edges, denoted m, in the graph (or an upper bound on this number). Namely, a graph is said to be ϵ-far from having a property, if more than ϵm edge modifications should be performed so that it obtains the property. In [122], (where the main focus was on sparse graphs), the algorithm is allowed the same type of queries as in the bounded-degree incidence-lists model, and it can also query the degree of any given vertex. The properties studied in that paper are having a bounded-size diameter and connectivity.

The main advantage of the [122] model over the bounded-degree incidence-lists model is that it is suitable for sparse graphs whose degrees may vary significantly. More generally, when the graph is not necessarily sparse (and not necessarily dense), we may allow vertex-pair queries in addition to neighbor queries and degree queries. This model was first studied by Krivelevich et al. [101]. Their focus was on the property of bipartiteness, which exhibits the following interesting phenomenon. As noted previously, for dense graphs there is an algorithm whose query complexity is $\text{poly}(1/\epsilon)$ [11, 82]. In contrast, for bounded-degree graphs there is a lower bound of $\Omega(\sqrt{n})$ [85] (and an almost matching upper bound [83]). The question Krivelevich et al. asked is: what is the complexity of testing bipartiteness in *general* graphs (using the general model)?

They answer this question by describing and analyzing an algorithm for testing bipartiteness in general graphs whose query complexity (and running time) is $O(\min(\sqrt{n}, n^2/m) \cdot \text{poly}(\log n/\epsilon))$. Recall that m is the number of edges in the graph (or an upper bound on this number, with respect to which distance is measure). Thus, as long as the average degree of the graph is $O(\sqrt{n})$, the running time (in terms of the dependence on n) is $\tilde{O}(\sqrt{n})$, and once the average degree goes above this threshold, the running time starts decreasing. They also present an almost matching lower bound of $\Omega(\min(\sqrt{n}, n^2/m))$ (for a constant ϵ). This bound holds for all testing algorithms (that is, for those which are

allowed a two-sided error and are adaptive). Furthermore, the bound holds for regular graphs.

Another property that was studied in the general model is testing triangle-freeness (and more generally, subgraph-freeness) [10]. As noted previously, for this property there is an algorithm in the dense-graphs model whose complexity depends only on $1/\epsilon$ [7], and the same is true for constant-degree graphs [85]. Here too the question is what is the complexity of testing the property in general graphs. In particular this includes graphs that are sparse (that is, $m = O(n)$), but do not have constant degree. The main finding of Alon et al. [10] is a lower bound of $\Omega(n^{1/3})$ on the necessary number of queries for testing triangle-freeness that holds whenever the average degree d is upper-bounded by $n^{1-\nu(n)}$, where $\nu(n) = o(1)$. Since when $d = \Theta(n)$ the number of queries sufficient for testing is independent of n [7], we observe an abrupt, *threshold-like* behavior of the complexity of testing around n. Additionally, they provide sub-linear upper bounds for testing triangle-freeness that are at most quadratic in the stated lower bounds.

Finally, a study of the complexity of testing k-colorability (for $k \geq 3$) is conducted by Ben-Eliezer et al. [34]. For this property there is an algorithm with query complexity $\text{poly}(1/\epsilon)$ in the dense-graphs model [11, 82] (where the algorithm uses only vertex-pair queries), and there is a very strong lower bound of $\Omega(n)$ for testing in the bounded-degree model [44] (where the algorithm uses neighbor queries). Ben-Eliezer et al. [34] consider the complexity of testing k-colorability as a function of the average degree d in models that allow different types of queries (and in particular may allow only one type of query). In particular, they show that while for vertex-pair queries, testing k-colorability requires a number of queries that is a monotone decreasing function in the average degree d, the query complexity in the case of neighbor queries remains roughly the same for every density and for large values of k. They also study a new, stronger, query model, which is related to the field of Group Testing.

6.6 Tolerant Testing and Distance Approximation

Two natural extensions of property testing, first explicitly studied in [124], are *tolerant testing* and *distance approximation*. A tolerant property testing algorithm is required to accept objects that are ϵ_1-close to having a given property $\mathcal{P}$ and reject objects that are ϵ_2-far from having property $\mathcal{P}$, for $0 \leq \epsilon_1 < \epsilon_2 \leq 1$. Standard property testing refers to the special case of $\epsilon_1 = 0$. Ideally, a tolerant testing algorithm should work for any given $\epsilon_1 < \epsilon_2$, and have complexity that depends on $\epsilon_2 - \epsilon_1$. However, in some cases the relation between ϵ_1 and ϵ_2 may be more restricted (e.g., $\epsilon_1 = \epsilon_2/2$). A closely related notion is that of *distance approximation* where the goal is to obtain an estimate of the distance that the object has to a property. In particular, we would like the estimate to have an additive error of at most δ for a given error parameter δ, or we may also allow a multiplicative error.[3]

In [124], it was first observed that some earlier works imply results in these models. In particular, this is true for coloring and other partition problems on dense graphs [82], connectivity of sparse graphs [52], edit distance between strings [26] and L_1 distance between distributions [28] (which was discussed in Section 6.3). The new results obtained in [124] were for monotonicity of functions $f : [n] \to R$, and clusterability of a set of points. The first result was later improved in [2] and extended to higher dimensions in [65]. In [70], it is shown that there exist properties of Boolean functions for which there exists a test that makes a constant number of queries, yet there is no such tolerant test. In contrast, in [74] it is shown that *every* property that has a testing algorithm in the dense-graphs model whose complexity is only a function of the distance parameter ϵ, has a distance approximation algorithm with an additive error δ in this model, whose complexity is only a function of δ.[4] Distance approximation in sparse graphs is studied in [116]. Guruswami and Rudra [91] presented tolerant testing algorithms for several constructions of locally testable codes.

[3] We note that if one does not allow an additive error (that is, $\delta = 0$), but only allows a multiplicative error, then a dependence on the distance that the object has to the property must be allowed.

[4] The dependence on δ may be quite high (a tower of height polynomial in $1/\delta$), but there is *no* dependence on the size of the graph.

Acknowledgments

The author would like to thank a reviewer for many helpful detailed comments, and in particular for contributing to Section 1.3.

A

The (Multiplicative) Chernoff Bound

Throughout the survey, we have applied the following theorem, which we referred to as the "multiplicative Chernoff bound" (to distinguish it from the additive form, usually attributed to Hoeffding [96]).

Theorem A.1 ([53]). Let $\chi_1, \chi_2, \ldots, \chi_m$ be m independent random variables where $\chi_i \in [0,1]$. Let $p \stackrel{\text{def}}{=} \frac{1}{m}\sum_i \text{Exp}[\chi_i]$. Then, for every $\gamma \in [0,1]$, the following bounds hold:

$$\Pr\left[\frac{1}{m}\sum_{i=1}^{m}\chi_i \;>\; (1+\gamma)p\right] < \exp\left(-\gamma^2 pm/3\right)$$

and

$$\Pr\left[\frac{1}{m}\sum_{i=1}^{m}\chi_i \;<\; (1-\gamma)p\right] < \exp\left(-\gamma^2 pm/2\right) .$$

References

[1] N. Ailon and B. Chazelle, "Information theory in property testing and monotonicity testing in higher dimensions," *Information and Computation*, vol. 204, pp. 1704–1717, 2006.

[2] N. Ailon, B. Chazelle, S. Comandur, and D. Liue, "Estimating the distance to a monotone function," in *Proceedings of the Eight International Workshop on Randomization and Computation* (*RANDOM*), pp. 229–236, 2004.

[3] N. Alon, "Testing subgraphs of large graphs," *Random Structures and Algorithms*, vol. 21, pp. 359–370, 2002.

[4] N. Alon, A. Andoni, T. Kaufman, K. Matulef, R. Rubinfeld, and N. Xie, "Testing k-wise and almost k-wise independence," in *Proceedings of the Thirty-Ninth Annual ACM Symposium on the Theory of Computing*, pp. 496–505, 2007.

[5] N. Alon, S. Dar, M. Parnas, and D. Ron, "Testing of clustering," *SIAM Journal on Discrete Math*, vol. 16, no. 3, pp. 393–417, 2003.

[6] N. Alon, R. A. Duke, H. Lefmann, V. Rodl, and R. Yuster, "The algorithmic aspects of the regularity lemma," *Journal of Algorithms*, vol. 16, pp. 80–109, 1994.

[7] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy, "Efficient testing of large graphs," *Combinatorica*, vol. 20, pp. 451–476, 2000.

[8] N. Alon, E. Fischer, and I. Newman, "Testing of bipartite graph properties," *SIAM Journal on Computing*, vol. 37, pp. 959–976, 2007.

[9] N. Alon, E. Fischer, I. Newman, and A. Shapira, "A combinatorial characterization of the testable graph properties: It's all about regularity," in *Proceedings of the Thirty-Eighth Annual ACM Symposium on the Theory of Computing*, pp. 251–260, 2006.

[10] N. Alon, T. Kaufman, M. Krivelevich, and D. Ron, "Testing triangle freeness in general graphs," in *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*), pp. 279–288, 2006.

[11] N. Alon and M. Krivelevich, "Testing k-colorability," *SIAM Journal on Discrete Math*, vol. 15, no. 2, pp. 211–227, 2002.

[12] N. Alon, M. Krivelevich, T. Kaufman, S. Litsyn, and D. Ron, "Testing Reed-Muller codes," *IEEE Transactions on Information Theory*, vol. 51, no. 11, pp. 4032–4038, 2005. (An extended abstract of this paper appeared under the title: Testing Low-Degree Polynomials over GF(2), in the proceedings of RANDOM 2003).

[13] N. Alon, M. Krivelevich, I. Newman, and M. Szegedy, "Regular languages are testable with a constant number of queries," *SIAM Journal on Computing*, pp. 1842–1862, 2001.

[14] N. Alon and A. Shapira, "Testing satisfiability," *Journal of Algorithms*, vol. 47, pp. 87–103, 2003.

[15] N. Alon and A. Shapira, "Testing subgraphs in directed graphs," *Journal of Computer and System Sciences*, vol. 69, pp. 354–482, 2004.

[16] N. Alon and A. Shapira, "A characterization of easily testable induced subgraphs," *Combinatorics Probability and Computing*, vol. 15, pp. 791–805, 2005.

[17] N. Alon and A. Shapira, "A characterization of the (natural) graph properties testable with one-sided error," in *Proceedings of the Forty-Sixth Annual Symposium on Foundations of Computer Science* (*FOCS*), pp. 429–438, 2005.

[18] N. Alon and A. Shapira, "Every monotone graph property is testable," in *Proceedings of the Thirty-Seventh Annual ACM Symposium on the Theory of Computing* (*STOC*), pp. 129–137, 2005. (To appear in SICOMP).

[19] N. Alon and A. Shapira, "Linear equations, arithmetic progressions and hypergraph property testing," *Theory of Computing*, vol. 1, pp. 177–216, 2005.

[20] D. Angluin, "Queries and concept learning," *Machine Learning*, vol. 2, pp. 319–342, 1988.

[21] S. Arora and S. Safra, "Probabilistic checkable proofs: A new characterization of NP," *Journal of the ACM*, vol. 45, no. 1, pp. 70–122, 1998. A preliminary version appeard in Proc. 33rd FOCS, 1992.

[22] S. Arora and M. Sudan, "Improved low-degree testing and its applications," in *Proceedings of the Thirty-Second Annual ACM Symposium on the Theory of Computing* (*STOC*), pp. 485–495, 1997.

[23] L. Babai, L. Fortnow, L. Levin, and M. Szegedy, "Checking computations in polylogarithmic time," in *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing* (*STOC*), pp. 21–31, 1991.

[24] L. Babai, L. Fortnow, and C. Lund, "Non-deterministic exponential time has two-prover interactive protocols," *Computational Complexity*, vol. 1, no. 1, pp. 3–40, 1991.

[25] T. Batu, S. Dasgupta, R. Kumar, and R. Rubinfeld, "The complexity of approximating the entropy," *SIAM Journal on Computing*, vol. 35, no. 1, pp. 132–150, 2005.

[26] T. Batu, F. Ergun, J. Kilian, A. Magen, S. Raskhodnikova, R. Rubinfeld, and R. Sami, "A sublinear algorithm for weakly approximating edit distance," in

Proceedings of the Thirty-Fifth Annual ACM Symposium on the Theory of Computing (*STOC*), pp. 316–324, 2003.

[27] T. Batu, E. Fischer, L. Fortnow, R. Kumar, and R. Rubinfeld, "Testing random variables for independence and identity," in *Proceedings of the Forty-Second Annual Symposium on Foundations of Computer Science* (*FOCS*), pp. 442–451, 2001.

[28] T. Batu, L. Fortnow, R. Rubinfeld, W. Smith, and P. White, "Testing that distributions are close," in *Proceedings of the Forty-First Annual Symposium on Foundations of Computer Science* (*FOCS*), pp. 259–269, 2000.

[29] T. Batu, R. Kumar, and R. Rubinfeld, "Sublinear algorithms for testing monotone and unimodal distributions," in *Proceedings of the Thirty-Sixth Annual ACM Symposium on the Theory of Computing* (*STOC*), pp. 381–390, 2004.

[30] T. Batu, R. Rubinfeld, and P. White, "Fast approximate PCPs for multidimensional bin-packing problems," *Information and Computation*, vol. 196, no. 1, pp. 42–56, 2005.

[31] A. Beimel, F. Bergadano, N. Bshouty, E. Kushilevitz, and S. Varricchio, "Learning functions represented as multiplicity automata," *Journal of the ACM*, vol. 47, no. 5, pp. 506–530, 2000.

[32] M. Bellare, D. Coppersmith, J. Håstad, M. Kiwi, and M. Sudan, "Linearity testing over characteristic two," *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 1781–1795, 1996.

[33] S. Ben-David, "Can finite samples detect singularities of real-valued functions?," in *Proceedings of the Twenty-Fourth Annual ACM Symposium on the Theory of Computing* (*STOC*), pp. 390–399, 1992.

[34] I. Ben-Eliezer, T. Kaufman, M. Krivelevich, , and D. Ron, "Comparing the strength of query types in property testing: The case of testing k-colorability," in *Proceedings of the Ninteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*), 2008.

[35] M. Ben-Or and P. Tiwari, "A deterministic algorithm for sparse multivariate polynomial interpolation," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing* (*STOC*), pp. 301–309, 1988.

[36] E. Ben-Sasson, P. Harsha, and S. Raskhodnikova, "3CNF properties are hard to test," *SIAM Journal on Computing*, vol. 35, no. 1, pp. 1–21, 2005.

[37] M. Bender and D. Ron, "Testing properties of directed graphs: Acyclicity and connectivity," *Random Structures and Algorithms*, pp. 184–205, 2002.

[38] L. Bisht, N. Bshouty, and H. Mazzawi, "An optimal learning algorithm fur multiplicity automata," in *Proceedings of the Ninteenth Annual ACM Conference on Computational Learning Theorey* (*COLT*), pp. 184–198, 2006.

[39] A. Blum, C. Burch, and J. Langford, "On learning monotone Boolean functions," in *Proceedings of the Thirty-Ninth Annual Symposium on Foundations of Computer Science* (*FOCS*), pp. 408–415, 1998.

[40] A. Blum, L. Hellerstein, and N. Littlestone, "Learning in the presence of finitely or infinitely many irrelevant attributes," *Journal of Computer and System Sciences*, vol. 50, no. 1, pp. 32–40, 1995.

[41] A. Blum and M. Singh, "Learning functions of k terms," in *Proceedings of the Third Annual Workshop on Computational Learning Theorey* (*COLT*), pp. 144–153, 1990.

[42] M. Blum, M. Luby, and R. Rubinfeld, "Self-Testing/Correcting with applications to numerical problems," *Journal of the ACM*, vol. 47, pp. 549–595, 1993.

[43] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Occam's razor," *Information Processing Letters*, vol. 24, no. 6, pp. 377–380, April 1987.

[44] A. Bogdanov, K. Obata, and L. Trevisan, "A lower bound for testing 3-colorability in bounded-degree graphs," in *Proceedings of the Forty-Third Annual Symposium on Foundations of Computer Science* (*FOCS*), pp. 93–102, 2002.

[45] A. Bogdanov and L. Trevisan, "Lower bounds for testing bipartiteness in dense graphs," in *Proceedings of the Nintheenth Computationa Complexity Conference* (*CCC*), 2004.

[46] B. Bollig, "Property testing and the branching program size," in *Proceedings of FCT, Lecture notes in Computer Science 3623*, pp. 258–269, 2005.

[47] B. Bollig and I. Wegener, "Functions that have read-once branching programs of quadratic sizze are not necessarily testable," *Information Processing Letters*, vol. 87, no. 1, pp. 25–29, 2003.

[48] C. Borgs, J. Chayes, L. Lovász, V. T. Sós, B. Szegedy, and K. Vesztergombi, "Graph limits and parameter testing," in *Proceedings of the Thirty-Eighth Annual ACM Symposium on the Theory of Computing*, pp. 261–270, 2006.

[49] N. Bshouty, "On learning multivariate polynomials under the uniform distribution," *Information Processing Letters*, vol. 61, pp. 303–309, 1996.

[50] N. Bshouty, J. Jackson, and C. Tamon, "More efficient PAC-learning of DNF with membership queries under the uniform distribution," in *Proceedings of the Twelveth Annual ACM Conference on Computational Learning Theorey* (*COLT*), pp. 286–295, 1999.

[51] N. Bshouty and Y. Mansour, "Simple learning algorithms for decision trees and multivariate polynomials," *SIAM Journal on Computing*, vol. 31, no. 6, pp. 1909–1925, 2002.

[52] B. Chazelle, R. Rubinfeld, and L. Trevisan, "Approximating the minimum spanning tree weight in sublinear time," in *Automata, Languages and Programming: Twenty-Eighth International Colloquium* (*ICALP*), pp. 190–200, 2001.

[53] H. Chernoff, "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations," *Annals of the Mathematical Statistics*, vol. 23, pp. 493–507, 1952.

[54] H. Chockler and D. Gutfreund, "A lower bound for testing juntas," *Information Processing Letters*, vol. 90, no. 6, pp. 301–305, 2006.

[55] M. Clausen, A. Dress, J. Grabmeier, and M. Karpinski, "On zero-testing and interpolation of k-sparse multivariate polynomials over finite fields," *Theoretical Computer Science*, vol. 84, no. 2, pp. 151–164, 1991.

[56] A. Czumaj, A. Shapira, and C. Sohler, "Testing hereditary properties of nonexpanding bounded-degree graphs," Technical Report TR07-083, Electronic Colloquium on Computational Complexity (ECCC), 2007. Available from http://www.eccc.uni-trier.de/eccc/.

[57] A. Czumaj and C. Sohler, "Sublinear-time approximation for clustering via random sampling," in *Automata, Languages and Programming: Thirty-First International Colloquium* (*ICALP*), pp. 396–407, 2004.

[58] A. Czumaj and C. Sohler, "Abstract combinatorial programs and efficient property testers," *SIAM Journal on Computing*, vol. 34, no. 3, pp. 580–615, 2005.

[59] A. Czumaj and C. Sohler, "Testing hypergraph colorability," *Random Structures and Algorithms*, vol. 331, no. 1, pp. 37–52, 2005.

[60] A. Czumaj and C. Sohler, "Testing expansion in bounded-degree graphs," in *Proceedings of the Forty-Eighth Annual Symposium on Foundations of Computer Science* (*FOCS*), pp. 570–578, 2007.

[61] I. Diakonikolas, H. K. Lee, K. Matulef, K. Onak, R. Rubinfeld, R. A. Servedio, and A. Wan, "Testing for concise representations," in *Proceedings of the Forty-Eighth Annual Symposium on Foundations of Computer Science* (*FOCS*), pp. 549–557, 2007.

[62] I. Diakonikolas, H. K. Lee, K. Matulef, R. A. Servedio, and A. Wan, "Efficient testing sparse GF(2) polynomials," in *Automata, Languages and Programming: Thirty-Fifth International Colloquium* (*ICALP*), pp. 502–514, 2008.

[63] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky, "Improved testing algorithms for monotonocity," in *Proceedings of the Third International Workshop on Randomization and Approximation Techniques in Computer Science* (*RANDOM*), pp. 97–108, 1999.

[64] F. Ergun, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan, "Spot-checkers," *Journal of Computer and System Sciences*, vol. 60, no. 3, pp. 717–751, 2000.

[65] S. Fattal and D. Ron, "Approximating the distance to monotonicity in high dimensions," Manuscript, 2007.

[66] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, "Approximating Clique is almost NP-complete," *Journal of the ACM*, pp. 268–292, 1996.

[67] E. Fischer, "On the strength of comparisons in property testing," *Information and Computation*, vol. 189, no. 1, pp. 107–116, 2004.

[68] E. Fischer, "The difficulty of testing for isomorphism against a graph that is given in advance," *SIAM Journal on Computing*, vol. 34, pp. 1147–1158, 2005.

[69] E. Fischer, "Testing graphs for colorability properties," *Random Structures and Algorithms*, vol. 26, no. 3, pp. 289–309, 2005.

[70] E. Fischer and L. Fortnow, "Tolerant versus intolerant testing for Boolean properties," *Theory of Computing*, vol. 2, pp. 173–183, 2006.

[71] E. Fischer, G. Kindler, D. Ron, S. Safra, and S. Samorodnitsky, "Testing juntas," *Journal of Computer and System Sciences*, vol. 68, no. 4, pp. 753–787, 2004.

[72] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, and A. Samrodnitsky, "Monotonicity testing over general poset domains," in *Proceedings of the Thirty-Fourth Annual ACM Symposium on the Theory of Computing* (*STOC*), pp. 474–483, 2002.

[73] E. Fischer, A. Matsliah, and A. Shapira, "Approximate hypergraph partitioning and applications," in *Proceedings of the Forty-Eighth Annual Symposium on Foundations of Computer Science* (*FOCS*), pp. 579–588, 2007.

[74] E. Fischer and I. Newman, "Testing versus estimation of graph properties," *SIAM Journal on Computing*, vol. 37, no. 2, pp. 482–501, 2007.

[75] E. Fischer, I. Newman, and J. Sgall, "Functions that have read-twice constant width branching programs are not necessarily testable," *Random Structures and Algorithms*, vol. 24, no. 2, pp. 175–193, 2004.

[76] P. Fischer and H. U. Simon, "On learning ring-sum expansions," *SIAM Journal on Computing*, vol. 21, no. 1, pp. 181–192, 1992.

[77] K. Friedl and M. Sudan, "Some improvements to total degree tests," in *Proceedings of the 3rd Annual Israel Symposium on Theory of Computing and Systems*, pp. 190–198, 1995. Corrected version available online at http://theory.lcs.mit.edu/~madhu/papers/friedl.ps.

[78] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson, "Self testing/correcting for polynomials and for approximate functions," in *Proceedings of the Thirty-Second Annual ACM Symposium on the Theory of Computing* (*STOC*), pp. 32–42, 1991.

[79] D. Glassner and R. A. Servedio, "Distribution-free testing lower bounds for basic Boolean functions," in *Proceedings of the Eleventh International Workshop on Randomization and Computation* (*RANDOM*), pp. 494–508, 2007.

[80] O. Goldreich, "Short locally testable codes and proofs (a survey)," Technical Report TR05-014, Electronic Colloquium on Computational Complexity (ECCC), 2005. Available from http://www.eccc.uni-trier.de/eccc/.

[81] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samordinsky, "Testing monotonicity," *Combinatorica*, vol. 20, no. 3, pp. 301–337, 2000.

[82] O. Goldreich, S. Goldwasser, and D. Ron, "Property testing and its connection to learning and approximation," *Journal of the ACM*, vol. 45, no. 4, pp. 653–750, 1998.

[83] O. Goldreich and D. Ron, "A sublinear bipartite tester for bounded degree graphs," *Combinatorica*, vol. 19, no. 3, pp. 335–373, 1999.

[84] O. Goldreich and D. Ron, "On testing expansion in bounded-degree graphs," *Electronic Colloqium on Computational Complexity*, vol. 7, no. 20, 2000.

[85] O. Goldreich and D. Ron, "Property testing in bounded degree graphs," *Algorithmica*, pp. 302–343, 2002.

[86] O. Goldreich and L. Trevisan, "Three theorems regarding testing graph properties," *Random Structures and Algorithms*, vol. 23, no. 1, pp. 23–57, 2003.

[87] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[88] M. Gonen and D. Ron, "On the benefits of adaptivity in property testing of dense graphs," in *Proceedings of the Eleventh International Workshop on Randomization and Computation* (*RANDOM*), pp. 525–537, 2007.

[89] D. Y. Grigoriev, M. Karpinski, and M. F. Singer, "Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields," *SIAM Journal on Computing*, vol. 19, no. 6, pp. 1059–1063, 1990.

[90] D. Guijarro, J. Tarui, and T. Tsukiji, "Finding relevant variables in PAC model with membership queries," in *Proceedings of Algorithmic Learning Theory, 10th International Conference*, pp. 313–322, 1999.

[91] V. Guruswami and A. Rudra, "Tolerant locally testable codes," in *Proceedings of the Ninth International Workshop on Randomization and Computation* (*RANDOM*), pp. 306–317, 2005.

[92] S. Halevy and E. Kushilevitz, "Distribution-free property testing," in *Proceedings of the Seventh International Workshop on Randomization and Approximation Techniques in Computer Science* (*RANDOM*), pp. 341–353, 2003.

[93] S. Halevy and E. Kushilevitz, "Distribution-free connectivity testing," in *Proceedings of the Eight International Workshop on Randomization and Computation* (*RANDOM*), pp. 393–404, 2004.

[94] S. Halevy and E. Kushilevitz, "A lower bound for distribution-free monotonicity testing," in *Proceedings of the Ninth International Workshop on Randomization and Computation* (*RANDOM*), pp. 330–341, 2005.

[95] S. Halevy and E. Kushilevitz, "Distribution-free property testing," *SIAM Journal on Computing*, vol. 37, no. 4, pp. 1107–1138, 2007.

[96] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 13–30, March 1963.

[97] P. Indyk, "A sublinear-time approximation scheme for clustering in metric spaces," in *Proceedings of the Fortieth Annual Symposium on Foundations of Computer Science* (*FOCS*), pp. 154–159, 1999.

[98] J. Jackson, "An efficient membership-query algorithm for learning DNF with respect to the uniform distribution," *Journal of Computer and System Sciences*, vol. 55, pp. 414–440, 1997.

[99] C. S. Jutla, A. C. Patthak, A. Rudra, and D. Zuckerman, "Testing low-degree polynomials over prime fields," in *Proceedings of the Forty-Fifth Annual Symposium on Foundations of Computer Science* (*FOCS*), 2004.

[100] S. Kale and C. Seshadhri, "Testing expansion in bounded degree graphs," Technical Report TR07-076, Electronic Colloquium on Computational Complexity (ECCC), 2007. Available from http://www.eccc.uni-trier.de/eccc/.

[101] T. Kaufman, M. Krivelevich, and D. Ron, "Tight bounds for testing bipartiteness in general graphs," *SIAM Journal on Computing*, vol. 33, no. 6, pp. 1441–1483, 2004.

[102] T. Kaufman, S. Litsyn, and N. Xie, "Breaking the ϵ-soundness bound of the linearity test over GF(2)," Technical Report TR07-098, Electronic Colloquium on Computational Complexity (ECCC), 2007. Available from http://www.eccc.uni-trier.de/eccc/.

[103] T. Kaufman and D. Ron, "Testing polynomials over general fields," *SIAM Journal on Computing*, vol. 35, no. 3, pp. 779–802, 2006.

[104] M. Kearns, M. Li, and L. Valiant, "Learning Boolean formulae," *Journal of the ACM*, vol. 41, no. 6, pp. 1298–1328, 1995.

[105] M. Kearns and D. Ron, "Testing problems with sub-learning sample complexity," *Journal of Computer and System Sciences*, vol. 61, no. 3, pp. 428–456, 2000.

[106] M. Kearns and L. Valiant, "Cryptographic limitations on learning boolean formulae and finite automata," *Journal of the ACM*, vol. 41, no. 1, pp. 67–95, 1994.

[107] M. J. Kearns, R. E. Schapire, and L. M. Sellie, "Toward efficient agnostic learning," *Machine Learning*, vol. 17, nos. 2–3, pp. 115–141, 1994.
[108] J. C. Kiefer, *Introduction to Statistical Inference*. Springer Verlag, 1987.
[109] A. Klivans and R. A. Servedio, "Boosting and hard-core sets," in *Proceedings of the Fortieth Annual Symposium on Foundations of Computer Science* (*FOCS*), pp. 624–633, 1999.
[110] Y. Kohayakawa, B. Nagle, and V. R#odl, "Efficient testing of hypergraphs," in *Automata, Languages and Programming: Twenty-Ninth International Colloquium* (*ICALP*), pp. 1017–1028, 2002.
[111] S. Kulkarni, S. Mitter, and J. Tsitsiklis, "Active learning using arbitrary binary valued queries," *Machine Learning*, vol. 11, pp. 23–35, 1993.
[112] S. R. Kulkarni and O. Zeitouni, "On probably correct classification of concepts," in *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theorey* (*COLT*), pp. 111–116, 1993.
[113] E. K. Lehman and J. P. Romano, *Testing Statistical Hypotheses*. Springer Verlag, third edition, 2005.
[114] N. Littlestone, "Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm," *Machine Learning*, vol. 2, no. 4, pp. 285–318, 1987.
[115] Y. Mansour, "Randomized interpolation and approximation of sparse polynomials," *SIAM Journal on Computing*, vol. 24, no. 2, pp. 357–368, 1995.
[116] S. Marko and D. Ron, "Distance approximation in bounded-degree and general sparse graphs," in *Proceedings of the Tenth International Workshop on Randomization and Computation* (*RANDOM*), pp. 475–486, 2006. (To appear in *Transactions on Algorithms*).
[117] K. Matulef, R. O'Donnell, R. Rubinfed, and R. A. Servedio, "Testing Halfspaces," Report number 128 in the Electronic Colloqium on Computational Complexity (ECCC), 2007.
[118] N. Mishra, D. Oblinger, and L. Pitt, "Sublinear time approximate clustering," in *Proceedings of the Twelvth Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*), pp. 439–447, 2001.
[119] E. Mossel, R. O'Donnell, and R. A. Servedio, "Learning juntas," *Journal of Computer and System Sciences*, vol. 69, no. 3, pp. 421–434, 2004.
[120] A. Nachmias and A. Shapira, "Testing the expansion of a graph," Technical Report TR07-118, Electronic Colloquium on Computational Complexity (ECCC), 2007. Available from http://www.eccc.uni-trier.de/eccc/.
[121] I. Newman, "Testing membership in languages that have small width branching programs," *SIAM Journal on Computing*, vol. 31, no. 5, pp. 1557–1570, 2002.
[122] M. Parnas and D. Ron, "Testing the diameter of graphs," *Random Structures and Algorithms*, vol. 20, no. 2, pp. 165–183, 2002.
[123] M. Parnas, D. Ron, and R. Rubinfeld, "On testing convexity and submodularity," *SIAM Journal on Computing*, vol. 32, no. 5, pp. 1158–1184, 2003.
[124] M. Parnas, D. Ron, and R. Rubinfeld, "Tolerant property testing and distance approximation," *Journal of Computer and System Sciences*, vol. 72, no. 6, pp. 1012–1042, 2006.

[125] M. Parnas, D. Ron, and A. Samorodnitsky, "Testing basic Boolean formulae," *SIAM Journal on Discrete Math*, vol. 16, no. 1, pp. 20–46, 2002.

[126] S. Raskhodnikova, D. Ron, R. Rubinfeld, and A. Smith, "Strong lower bonds for approximating distributions support size and the distinct elements problem," in *Proceedings of the Forty-Eighth Annual Symposium on Foundations of Computer Science* (*FOCS*), pp. 559–568, 2007.

[127] R. Roth and G. Benedek, "Interpolation and approximation of sparse multivariate polynomials over GF(2)," *SIAM Journal on Computing*, vol. 20, no. 2, pp. 291–314, 1991.

[128] R. Rubinfeld and M. Sudan, "Robust characterization of polynomials with applications to program testing," *SIAM Journal on Computing*, vol. 25, no. 2, pp. 252–271, 1996.

[129] R. E. Schapire and L. M. Sellie, "Learning sparse multivariate polynomials over a field with queries and counterexamples," *Journal of Computer and System Sciences*, vol. 52, no. 2, pp. 201–213, 1996.

[130] A. Shpilka and A. Wigderson, "Derandomizing homomorphism testing in general groups," in *Proceedings of the Thirty-Sixth Annual ACM Symposium on the Theory of Computing* (*STOC*), pp. 427–435, 2004.

[131] M. Sudan, "Efficient checking of polynomials and proofs and the hardness of approximation problems," PhD thesis, UC Berkeley, 1992. Also appears as Lecture Notes in Computer Science, Vol. 1001, Springer, 1996.

[132] E. Szemeŕedi, "Regular partitions of graphs," in *Proceedings, Colloque Inter. CNRS*, pp. 399–401, 1978.

[133] R. Uehara, K. Tsuchida, and I. Wegener, "Optimal attribute-efficient learning of disjunction, parity and threshold functions," in *Proceedings of the 3rd European Conference on Computational Learning Theory*, pp. 171–184, 1997.

[134] L. G. Valiant, "A theory of the learnable," *CACM*, vol. 27, no. 11, pp. 1134–1142, November 1984.

[135] P. Valiant, "Testing symmetric properties of distributions," in *Proceedings of the Fourtieth Annual ACM Symposium on the Theory of Computing*, pp. 383–392, 2008.

[136] A. Van der Vaart, *Asymptotic Statistics*. Cambridge University Press, 1998.

[137] K. Yamanishi, "Probably almost discriminative learning," *Machine Learning*, vol. 18, pp. 23–50, 1995.

[138] R. Zippel, "Probabilistic algorithms for sparse polynomials," in *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pp. 216–226, 1978.

[139] R. Zippel, "Interpolating polynomials from their values," *Journal of Symbolic Computation*, vol. 9, pp. 375–403, 1990.