Lecture Notes on Testing by Implicit Sampling

Oded Goldreich[*]

April 20, 2016

Summary: Building on the junta tester, we present a general methodology for constructing testers for properties of Boolean functions (of the form $f : \{0,1\}^\ell \to \{0,1\}$) that can be approximated by small juntas. This methodology yields testers of low query complexity for many natural properties, which contain functions that depend on relatively few relevant variables; specifically, the query complexity is related to the size of the junta and is independent of the length of the input to the function (i.e., ℓ).

These notes are based on the work of Diakonikolas, Lee, Matulef, Onak, Rubinfeld, Servedio and Wan [1]. The paradigm introduced in their work is often called *testing by implicit learning* (see, e.g., [4]), but I prefer the term "implicit sampling" for reasons that will be clarified later. This lecture builds on the lecture on testing juntas; thus, the latter lecture is a prerequisite to the current one.

1 Introduction

The natural interest in Boolean functions that have few relevant variables leads to an interest in functions of this type that have additional properties. We focus on the case that these additional properties are actually properties of the residual function, where the residual function of $f : \{0,1\}^\ell \to \{0,1\}$ that depends on the variable set $I \subseteq [\ell]$ is the function $f' : \{0,1\}^{|I|} \to \{0,1\}$ such that $f(x) = f'(x_I)$. In other words, we are interested in properties of the form Π such that there exists $\Pi' \subseteq \{f' : \{0,1\}^k \to \{0,1\}\}$ so that $f \in \Pi$ if an only if for some k-subset I and $f' \in \Pi'$ it holds that $f(x) = f'(x_I)$.

The study of testers for such properties leads to a technique that illustrates the usefulness of partial information of the type that is provided by property testers. We refer to information of the form "the set S contains no relevant variables" (of the function f). Specifically, given oracle access to f such that $f(x) = f'(x_I)$ for some small but unknown $I \subset [\ell]$, we show how to use partial information of the foregoing type in order to efficiently generate random pairs of the form $(z, f'(z))$. We stress that this generation is performed without knowing I and without trying to find it.

2 Testing subclasses of k-Juntas

Recall that that a function $f : \{0,1\}^\ell \to \{0,1\}$ is called a k-junta if there exists k indices $i_1, ..., i_k \in [\ell]$ and a Boolean function $f' : \{0,1\}^k \to \{0,1\}$ such that $f(x) = f'(x_{i_1} \cdots x_{i_k})$ for every $x =$

[*]Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel.

$x_1 \cdots x_\ell \in \{0,1\}^\ell$. Here, we assume, without loss of generality, that $i_1 < \cdots < i_k$. In other words, $f(x) = f'(x_I)$, where $I = \{i_1, ..., i_k\} \subseteq [\ell]$ such that $i_1 < \cdots < i_k$ and x_I denotes the k-bit long string $x_{i_1} \cdots x_{i_k}$. Natural subclasses of k-juntas arise when restricting f' to reside in a predeterimed set of functions. Specifically, we refer to the following definition.

Definition 1 ((k,Φ)-juntas): *Let Φ be a set of Boolean functions over $\{0,1\}^k$. A function $f : \{0,1\}^\ell \to \{0,1\}$ is called a* (k,Φ)-junta *if there exist k-subset I and a Boolean function $f' : \{0,1\}^k \to \{0,1\}$ in Φ such that $f(x) = f'(x_I)$ for every $x \in \{0,1\}^\ell$.*

Properties of this form (i.e., (k,Φ)-juntas) may be viewed as properties of functions that have only k relevant inputs (called "relevant attributes" in the machine learning literature). Hence, it is reasonable to hope that computational tasks related to these properties will have query complexity that does not depend on ℓ, and may only depend on $k \ll \ell$.

A natural way to test whether a function is a (k,Φ)-junta is to first check that it is a k-junta, then find the corresponding set I, and finally test whether the corresponding f' is in Φ. The point (of "testing by implicit sampling") is that we want to avoid finding the set I, since in general finding the set I requires more than $\log \binom{\ell}{k}$ queries (see Exercise 1), whereas we may wish the query complexity to be independent of ℓ. The paradigm of implicit sampling offers a way of skipping the second step (of finding I), and generating a random sample of labeled k-tuples that can be used for testing f'. Note, however, that the testing of f' is performed by samples only; that is, we invoke a tester for Φ that only uses f'-labeled samples (and makes no queries to f').[1] The relevant definition of such testers was briefly mentioned in the first lecture; they are called *sample-based*, and are defined as follows.

Definition 2 (sample-based tester for property Φ): *Let $\Phi = \cup_{n\in\mathbb{N}}\Phi_n$ such that Φ_n contains functions defined over $[n]$, and $s : \mathbb{N} \times (0,1] \to \mathbb{N}$. A* sample-based tester of (sample) complexity s for Φ *is a probabilistic machine, denoted T, that satisfies the following two conditions.*

1. T accepts inputs in Φ: *For every $n \in \mathbb{N}$ and $\epsilon > 0$, and for every $f \in \Phi_n$, it holds that $\mathbf{Pr}[T(n,\epsilon; ((i_1, f(i_1))..., (i_s, f(i_s))) = 1] \geq 2/3$, where $s = s(n,\epsilon)$ and $i_1, ..., i_s$ are drawn independently and uniformly in $[n]$.*
2. T rejects inputs that are ϵ-far from Φ: *For every $n \in \mathbb{N}$ and $\epsilon > 0$, and for every f with domain $[n]$ such that $\delta_\Phi(f) > \epsilon$, it holds that $\mathbf{Pr}[T(n,\epsilon; ((i_1, f(i_1))..., (i_s, f(i_s))) = 0] \geq 2/3$, where $\delta_\Phi(f)$ denotes the distance of f from Φ, and $i_1, ..., i_s$ are as in Item 1.*

If the first condition holds with probability 1, then we say that T has one-sided error.

The sequence $((i_1, f(i_1))..., (i_s, f(i_s)))$ is called an f-labeled sample of s points (in the domain of f). Recall that any class $\Phi = \cup_{n\in\mathbb{N}}\Phi_n$ can be tested by using a sample of size $O(\epsilon^{-1} \log |\Phi_n|)$, via reducing (sample-based) testing to (sample-based) proper learning (see the first lecture). Now, we are ready to state a general result that is obtained by the "implicit sampling" paradigm.

Theorem 3 (testing by implicit sampling): *Let Φ be a property of k-variate Boolean functions* (i.e., functions from $\{0,1\}^k$ to $\{0,1\}$) *such that Φ is invariant under permuting the bits of the argument to the function* (i.e., $f' \in \Phi$ if and only if for every permutation $\pi : [k] \to [k]$ it holds

[1]The reason that we cannot support queries will be clarified in the proof of Theorem 3.

that $f'_\pi(y) = f'(y_{\pi(1)}, ..., y_{\pi(k)})$ is in Φ). *Suppose that there exists a* sample-based *tester of sample complexity* $s_k : (0,1] \to \mathbb{N}$ *for* Φ *such that* $s_k(\epsilon) \geq 1/\epsilon$. *Then,* (k, Φ)*-juntas can be tested within* query *complexity* $q(n, \epsilon) = \text{poly}(k) \cdot \widetilde{O}(s_k(0.9\epsilon))^2$. *Furthermore, each of the queries made by this tester is uniformly distributed in* $\{0,1\}^\ell$.

Needless to say, this result is beneficial only when $k \ll \ell$ (since we can always find the junta within complexity $\widetilde{O}(k \log \ell/\epsilon)$; see Exercise 2). Note that all properties of ℓ-variate Boolean functions discussed in prior lectures are invariant in the foregoing sense (i.e., they are invariant under renaming of the variables; see Exercise 3). In contrast, properties that do not satisfy this condition refer to the identity of the variables (e.g., all Boolean functions that are influenced by their first variable), and seem less natural (especially in the current context).

Proof: Recall that we plan to test whether $f : \{0,1\}^\ell \to \{0,1\}$ is a (k, Φ)-junta by first testing whether f is a k-junta, which means that $f(x) = f'(x_I)$ for some k-subset I and $f' : \{0,1\}^k \to \{0,1\}$, and then testing whether f' is in Φ. We have seen a junta tester in a previous lecture, so the real challenge here is to test f' for membership in Φ while only having access to f. Recall that passing the k-junta test only assures us that f is close to being a k-junta (rather than actually being a k-junta). Nevertheless, let us assume for a moment that f is a k-junta. Furthermore, suppose that we are given a k-partition of $[\ell]$, denoted $(S_1, ..., S_k)$, such that each part has exactly one member of the junta (i.e., $|S_j \cap I| = 1$ for every $j \in [k]$).

In such a case, things would have been easy. We could have emulated a *standard* tester for Φ as follows. When the tester issues a query $y = y_1 \cdots y_k$, we would query f on the string z such that for every $j \in [k]$ and $i \in S_j$ it holds that $z_i = y_j$. This relies on the hypothesis that $f(x) = f'(x_I)$, which implies that z_I equals $y_{\pi(1)} \cdots y_{\pi(k)}$ for some permutation $\pi : [k] \to [k]$, and on the hypothesis that membership in Φ is invariant under permuting the bits of the argument to f'.

Unfortunately, the k-junta test only assures us that f is closed to being a k-junta, and so we cannot rely on the answers that f provides on the 2^k possible z's used in the foregoing construction. In other words, after verifying that $f : \{0,1\}^\ell \to \{0,1\}$ is close to being a k-junta, denoted g (such that $g(x) = g'(x_I)$ for some I and $g' : \{0,1\}^k \to \{0,1\}$), we can safely obtain g's values only at uniformly distributed points. We shall show that this suffices for generating g'-labeled samples (in the domain of g'), which is far from being obvious. (For this reason, we can emulate a *sample-based* tester, but not a tester that makes queries.)

The key question, indeed, is how can we generate these g'-labelled samples, without knowing I. Suppose that f is ϵ'-close to a k-junta g (such that $g(x) = g'(x_I)$ for some k-subset I), and suppose again that we are given a k-partition of $[\ell]$, denoted $(S_1, ..., S_k)$, such that each part has exactly one member of the junta (i.e., $|S_j \cap I| = 1$ for every $j \in [k]$). The difference, of course, is that this junta refers to g, not to f (which is only close to g). Now suppose that we pick $x \in \{0,1\}^\ell$ uniformly at random, and obtain $f(x)$ (which equals $g(x) = g'(x_I)$ with probability at least $1 - \epsilon'$). So we got the g'-label of x_I, but we don't know x_I (although we know x, since we don't know I). Actually, having $x_{S_1 \cap I} \cdots x_{S_k \cap I}$ is good enough, since we can consider testing $g'_\pi(z) = g'(z_{\pi(1)} \cdots z_{\pi(k)})$ (for a suitable π)[2] whose distance from Φ equals the distance of g' from Φ. Hence, for each $j \in [k]$, we wish to obtain $x_{S_j \cap I}$.

In other words, given $x \in \{0,1\}^\ell$ and $S = S_j \subset [\ell]$ such that exactly one bit-location in S influences the value of g, we wish to find out the value assigned to this bit-location in x. So we need to find out whether $S^0 = S \cap \{i : x_i = 0\}$ influences g, since if the answer is positive then

[2]If $I = \{i_1, .., i_k\}$ such that $i_1 < \cdots < i_k$, then π is defined such that $\{i_j\} = S_{\pi(j)} \cap I$. Hence, $z_{\pi(j)} = x_{S_{\pi(j)} \cap I} = x_{i_j}$.

$S^0 \cap I = S \cap I$ and $x_{S\cap I} = 0$, and otherwise $x_{S\cap I} = 1$ holds (since we have assumed that S does influence g). Furthermore, the influence of S on g is closely related to the influence of S on f (i.e., these influences differ by at most $2\epsilon'$), since g is close (i.e., ϵ'-close) to f. Finally, recall that we know how to test whether a set of locations influences a function; this is part of the junta tester (presented in the prior lecture). We review this part next.

Algorithm 3.1 (testing influence of a set of locations on a function f): *On input a set $S \subseteq [\ell]$ and a parameter m, and oracle access to $f : \{0,1\}^\ell \to \{0,1\}$, select uniformly m random pairs (r,s) such that r and s agree on bit positions $[\ell] \setminus S$* (i.e., $r_{\overline{S}} = s_{\overline{S}}$)*, and indicate that S* is influential *if and only if $f(r) \neq f(s)$ for any of these pairs (r,s). Actually, output the fraction of pairs (r,s) such that $f(r) \neq f(s)$ as an* estimate of the influence of *S.*

Recalling that the influence of S on f, denoted $\mathtt{I}_S(f)$, equal the probability that a single pair yields different values (i.e., $\mathbf{Pr}_{r,s:r_{\overline{S}}=s_{\overline{S}}}[f(r) \neq f(s)]$), it follows that S is deemed influential with probability $1-(1-\mathtt{I}_S(f))^m$, which equals $1-\exp(-\Theta(m\cdot\mathtt{I}_S(f)))$ if $\mathtt{I}_S(f) > 0$ (and zero otherwise). Furthermore, the estimate output by Algorithm 3.1 distinguishes, with success probability $1 - \exp(-\Omega(m \cdot \nu))$, between the case that $\mathtt{I}_S(f) \geq 2\nu$ and the case that $\mathtt{I}_S(f) \leq \nu$. This is done by ruling according to whether or not the said estimate (i.e., the fatction of pairs (r,s) such that $f(r) \neq f(s)$) exceeds 1.5ν.

Returning to the foregoing k-partition $(S_1, ..., S_k)$, we observe that a procedure for finding such a k-partition is also implicit in the k-junta tester we saw (in the prior lecture): It amounts to selecting a $O(k^2)$-partition at random, and testing whether more than k of the parts influence f. If the answer is positive, then we shall reject, and otherwise we can use this $O(k^2)$-partition for our purposes (either by merging the $O(k^2)$ parts into k sets such that each set contains at most one influential part or by just using the influential parts and ignoring the rest).

There is one problem with the forgoing suggestion. Taking a close look at the paragraph preceding Algorithm 3.1, note that we have assumed that each S_j contains a single influential variable; that is, we assumed that the singleton $I \cap S_j$ has positive influence on f. This is not necessarily the case. For starters, it may be that g is actually a $(k-1)$-junta. Moreover, even if g depends on all variables in I, it may be the case that some of these variables have negligible influence on g. Lastly, recall that we are estimating the influence of sets on F rather than on g, and the difference is not necessarily zero, although it is small. The reason that this is a problem is that if we determine the j^{th} bit in the k-bit sample (i.e., $x_{S_j \cap I}$) according to the influence of S_j^0 on f, then we may almost always set this value to 1 when $S_j \cap I$ has negligible influence on f. In such a case we shall end-up invoking the sample-based tester on a sample that is not uniformly distributed.

The foregoing problem is resolved by estimating the influence of S_j on f. If this influence is noticeable, then we set the j^{th} bit of the sample as suggested (i.e., we set it to 0 if and only if S_j^0 has positive influence of f). Otherwise (i.e., when S_j has negligible influence on F), we just set this bit at random (i.e., to be 0 with probability 1/2). The foregoing ideas yield the following algorithmic schema, which utilizes a sample-based tester of complexity s_k for Φ. In this schema the proximity parameters for the "tests of influence" (denoted ϵ_2 and ϵ_3) are set to a value that is smaller than ϵ (but related to it).

Algorithm 3.2 (testing (k, Φ)-juntas): *Let $c > 0$ be a sufficiently small constant* (e.g., $c = 0.01$). *On input parameters ℓ and ϵ, and oracle access to a function $f : \{0,1\}^\ell \to \{0,1\}$, the tester sets $t = \Theta(k^2)$, and proceeds as follows.*

1. *Select a random t-way partition of $[\ell]$, denoted $(R_1, ..., R_t)$, by assigning each $i \in [\ell]$ a uniformly selected $j \in [t]$, which means that i is assigned to R_j.*

2. *For each $j \in [t]$, check whether R_j influences f* (i.e., R_j has positive influence on f). *The aim is distinguishing, with success probability $1 - c/t$, between the case that $\mathtt{I}_{R_j}(f) = 0$ and the case that $\mathtt{I}_{R_j}(f) \geq \epsilon_2 \overset{\text{def}}{=} c/(2t \cdot k \cdot s_k(0.9\epsilon))$.*

   This is done by using Algorithm 3.1, while setting the parameter m to $m_2 = O(\epsilon_2^{-1} \log t)$, and asserting that R_j influences f if and only if the estimate output by the algorithm is positive.

   Let J denote the set of j's for which R_j was found to influence f. If $|J| > k$, then the algorithm rejects. Otherwise, assume, without loss of generality, that $|J| = k$, by possibly considering a k-superset of J. For notational simplicity, we assume that $J = [k]$.[3]

3. *For each $j \in J$, estimate the influence of R_j on f with the aim of distinguishing, with success probability $1 - c/k$, between the case that $\mathtt{I}_{R_j}(f) \geq 4\epsilon_3$ and the case that $\mathtt{I}_{R_j}(f) < 3\epsilon_3$, where $\epsilon_3 = 4t\epsilon_2 = 2c/(k \cdot s_k(0.9\epsilon)) \leq 2c \cdot \epsilon$.*

   This is done by using Algorithm 3.1, while setting the parameter m to $m_3 = O(\epsilon_3^{-1} \log k)$, and deciding based on the estimate that it outputs.

   Let $J' \subset J$ denote the set of j's for which the foregoing estimate exceeds $3.5\epsilon_3$.

4. Generate $s_k(0.9\epsilon)$ labelled samples for the (sample-based) tester of Φ, *where each labelled sample is generated as follows.*

   (a) *Select uniformly $x \in \{0,1\}^\ell$ and query f at x.*

   (b) *For every $j \in J'$, estimate the influence of R_j^x on f, where $R_j^x = \{i \in R_j : x_i = 0\}$. Here the aim is to distinguish, with success probability $1 - c/(k \cdot s_k(0.9\epsilon))$, between the case that $\mathtt{I}_{R_j^x}(f) \leq \epsilon_3$ and the case that $\mathtt{I}_{R_j^x}(f) \geq 2\epsilon_3$.*

   This is done by using Algorithm 3.1, while setting the parameter m to $m_4 = O(\epsilon_3^{-1} \log(k \cdot s_k(0.9\epsilon)))$, and asserting that R_j^x has high influence on f if and only if the output estimate exceeds $1.5\epsilon_3$. In the first case (i.e., R_j^x was asserted to have high influence), *set $y_j = 0$ and otherwise set $y_j = 1$.*

   (c) *For every $j \in J \setminus J'$, select y_j uniformly at random in $\{0,1\}$.*

   The labelled sample is $(y_1 \cdots y_k, f(x))$.

5. *Invoke the sample-based tester for Φ, while using proximity parameter 0.9ϵ, and assuming it has error probability at most c. Provide this tester with the $s_k(0.9\epsilon)$ labeled sample generated in Step 4, and output its verdict* (i.e., accept if and only if the latter tester has accepted).

We first note that each query made by Algorithm 3.2 is uniformly distributed in $\{0,1\}^\ell$. The query complexity of the algorithm is $t \cdot 2m_2 + k \cdot 2m_3 + s_k(0.9\epsilon) \cdot (1 + k \cdot 2m_4)$, where the first term is due to Step 2 the second term is due to Step 3, and the third term is due to Step 4. (We may ignore

[3]In general, one should use a one-to-one mapping $\phi : J \to [k]$. In this case, in Step 4b, for every $j \in J$, we set $y_{\phi(j)}$ according to R_j^x.

the second term since it is dominated by the second.) Using $m_2 = O(\epsilon_2^{-1} \log t) = \widetilde{O}(tk) \cdot s_k(0.9\epsilon)$ and $m_4 = O(\epsilon_3^{-1} \log(k \cdot s_k(0.9\epsilon))) = \widetilde{O}(s_k(0.9\epsilon) \cdot k)$, we obtain a complexity bound of

$$O(t \cdot m_2 + s_k(0.9\epsilon) \cdot k \cdot m_3) = \widetilde{O}(k^5 \cdot s_k(0.9\epsilon) + k^2 \cdot s_k(0.9\epsilon)^2).$$

We now turn to the analysis of Algorithm 3.2.

First, suppose that f is a (k, Φ)-junta. Let $f' \in \Phi$ be such that $f(x) = f'(x_I)$ for some k-subset I and all $x \in \{0,1\}^\ell$. Then, with probability at least $1 - c$ over the choice of the t-partition (selected in Step 1), it holds that $|R_j \cap I| \leq 1$ for each $j \in [t]$. In this case, with probability at least $1 - c$, the set J determined in Step 2 contains all j's such that $\mathtt{I}_{R_j}(f) \geq \epsilon_2$ (which implies that it contains contains all j's such that $\mathtt{I}_{R_j}(f) \geq 4\epsilon_3$). Likewise, with probability at least $1 - c$, the set J' determined in Step 3 satisfies

$$\{j \in [t] : \mathtt{I}_{R_j}(f) \geq 4\epsilon_3\} \subseteq J' \subseteq \{j \in [t] : \mathtt{I}_{R_j}(f) > 3\epsilon_3\}. \tag{1}$$

Now, for each x selected in Step 4 and for each $j \in J'$, with probability at least $1 - c/(k \cdot s_k(0.9\epsilon))$, the algorithm determines y_j such that $y_j = x_{R_j \cap I}$.[4] As for $j \in J \setminus J'$, with probability at least $1 - 4\epsilon_3$ (over the choice of x), it holds that replacing $x_{R_j \cap I}$ by y_j does not affect the value of f. Hence (using $\epsilon_3 = 2c/(ks_k(0.9\epsilon))$), with probability at least $(1-c)^3 \cdot (1 - c/(k \cdot s_k(0.9\epsilon)))^{k \cdot s_k(0.9\epsilon)} - k \cdot s_k(0.9\epsilon) \cdot 4\epsilon_3 > (1-c)^4 - 8c > 1 - 12c$, the sample-based tester for Φ is invoked with a uniformly distributed f'-labeled sample. It follows that f is accepted with probability at least $(1 - 12c) \cdot (1 - c) > 2/3$.

Next, we consider the case that f is ϵ-far from being a (k, Φ)-junta. As shown in the prior lecture, if f is $2t\epsilon_2$-far from being a k-junta, then it will be rejected in Step 2 (with high probability over the choice of the t-partition and the execution of Step 2). Hence, we focus on the case that f is $2t\epsilon_2$-close to a k-junta g, which in turn is $(\epsilon - 2t\epsilon_2)$-far from being a (k, Φ)-junta; that is, $g(x) = g'(x_I)$ for some $g' \notin \Phi$ and some k-subset I (and all $x \in \{0,1\}^\ell$). It follows that g' is $(\epsilon - 2t\epsilon_2)$-far from Φ. Now, as before, with probability at least $1 - c$ over the partition selected in Step 1, it holds that $|R_j \cap I| \leq 1$ for each $j \in [t]$. Furthermore, with probability at least $1 - c$, either Step 2 rejects or the set J' determined in Step 3 satisfies Eq. (1). Using the fact that *the influence of a set on the function g is within an additive distance of $2 \cdot 2t\epsilon_2$ from the influence of the same set on the function f* (see Exercise 4)) and $4t\epsilon_2 \leq \epsilon_3$, we have

$$\{j \in [t] : \mathtt{I}_{R_j}(g) \geq 5\epsilon_3\} \subseteq J' \subseteq \{j \in [t] : \mathtt{I}_{R_j}(g) > 2\epsilon_3\}. \tag{2}$$

Hence, for every $j \in J'$ it holds that $\mathtt{I}_{R_j}(g) > 2\epsilon_3$, whereas for every $i \in I \setminus \cup_{j \in J'} R_j$ it holds that $\mathtt{I}_{\{i\}}(g) \leq 5\epsilon_3$.

Now, note that, with probability at least $1 - s_k(0.9\epsilon) \cdot 2t\epsilon_2 > 1 - c$, it holds that $f(x) = g(x)$ (which equals $g'(x_I)$) for all x's generated in Step 4, since each x is uniformly distributed in $\{0,1\}^\ell$ (and f is $2t\epsilon_2$-close to g). Again, for each x generated in Step 4 and each $j \in J'$, with probability at least $1 - c/(k \cdot s_k(0.9\epsilon))$, the algorithm determines correctly the j^{th} bit of x_I. As before, the random setting of the bits in positions $J \setminus J'$ has lmited affect. Hence (using $\epsilon_3 = 2c/(ks_k(0.9\epsilon))$), with probability at least $(1-c)^3 \cdot (1-c) \cdot (1 - 0.1/(k \cdot s_k(0.9\epsilon)))^{k \cdot s_k(0.9\epsilon)} - k \cdot s_3(0.9\epsilon) \cdot 5\epsilon_3 > (1-c)^5 - 10c >$

[4] This description assumes, for notational simplicity, that $J = [k]$ and that $R_j \cap I = \{i_j\}$ where $I = \{i_1, ..., i_k\}$ and $i_1 < \cdots < i_k$. Eliminating the first assumption requires using $y_{\phi(j)}$ instead of y_j, where ϕ is as in Footnote 3. Eliminating the second assumption requires referring to f'_π (rather than to f') for an adequate permutation π over $[k]$ (i.e., π sorts the k-sequence $(R_j \cap I)_{j \in J}$), as in the motivating discussion. The same comment applies to the next couple of paragraphs (which deals with f that is ϵ-far from being a (k, Φ)-junta).

$1-15c$, either Step 2 rejects or the sample-based tester for Φ is invoked with a uniformly distributed g'-labeled sample. Since g' is $(\epsilon - 2t\epsilon_2)$-far from Φ and $\epsilon - 2t\epsilon_2 \geq \epsilon - c/s_k(0.9\epsilon) > 0.9\epsilon$, it follows that, in this case, f is rejected with probability at least $(1-15c)\cdot(1-c) > 2/3$. The theorem follows. ■

Applications. To illustrate the applicability of Theorem 3, we consider the problems of testing whether a function $f : \{0,1\}^k \to \{0,1\}$ is a (monotone and general) k-monomial, which were studied in a previous lecture. Clearly, the set of k-monomials is a subset of k-juntas, and testing that a Boolean function $f' : \{0,1\}^k \to \{0,1\}$ is a k-monomial is quite straightforward (since there are only 2^k such functions that are k-monomials (and a single monotone k-monomial)).[5] Hence, invoking Theorem 3, we get –

Corollary 4 (testing monotone and general k-monomials): *The following two properties of Boolean functions over $\{0,1\}^\ell$ can be tested within query complexity* $\text{poly}(k/\epsilon)$*:*

1. *The set of monotone k-monomials; that is, functions $f : \{0,1\}^\ell \to \{0,1\}$ such that for some k-subset $I \subseteq [\ell]$ it holds that $f(x) = \wedge_{i\in I} x_i$.*
2. *The set of k-monomials; that is, functions $f : \{0,1\}^\ell \to \{0,1\}$ such that for some k-subset $I \subseteq [\ell]$ and $\sigma = \sigma_1 \cdots \sigma_\ell \in \{0,1\}^\ell$ it holds $f(x) = \wedge_{i\in I}(x_i \oplus \sigma_i)$.*

Proof: Starting with the set of monotone k-monomials, let Φ denote the set of k-variate functions that are monotone k-monomials. Indeed, Φ is a singleton; that is, there is only one such function. Hence, testing whether $f' : \{0,1\}^k \to \{0,1\}$ is in Φ amounts to estimating the distance of f' from the unique monotone k-monomial, which can be done by using $O(1/\epsilon)$ random samples. Applying Theorem 3, Part 1 follows.

Turning to the set of k-monomials, let Φ denote the set of k-variate functions that are k-monomials. Indeed, Φ is of size 2^k, and each function in it evaluates to 1 on a single point (out of the 2^k possible points). Now, if $\epsilon > 3 \cdot 2^{-k}$, then every function f' that evaluates to 1 on more than an $\epsilon/3$ fraction of the domain is not in Φ and can be safely rejected. On the other hand, every function that evaluates to 1 on at most an $2\epsilon/3$ fraction of the domain is ϵ-close to Φ and can be safely accepted. Thus, in this case, using $O(1/\epsilon)$ random samples, we estimate the fraction of points on which the input function f' evaluates to 1, and accept if and only if this estimate is at most $\epsilon/2$. Lastly, if $\epsilon \leq 3 \cdot 2^{-k}$, then by using $O(k \cdot 2^k) = \widetilde{O}(1/\epsilon)$ random samples, we can reconstruct f' and check if it belongs to Φ. Now, applying Theorem 3, Part 2 follows. ■

3 Extension to properties approximated by subclasses of k-Juntas

In this section we extend the result of the previous section to properties that can be approximated by classes of (k,Φ)-juntas, for adequate choices of k and Φ. The notion of approximation is defined next.

Definition 5 (approximation of a property): *The property Π is δ-approximated by the property Π' if each function in Π is δ-close to some function in Π', and vice versa.*

[5]We shall use a better tester for the case of general monomials.

For example, the class of (monotone) k-monomials, considered in Corollary 4, is $2^{-k'}$-approximated by the class of (monotone) k'-monomials, which in turn is a subclass of k'-juntas. Specifically, any k-monomial can be replaced by a monomial that contains only $k' < k$ of the original literals. Note that in this case the approximation error decreases exponentially with k', whereas the query complexity increases polynomially with k'. Hence, for sufficiently large k', the approximation error is smaller than the reciprocal of the query complexity. This is the setting envisioned in the following general result.[6]

Theorem 6 (testing via an approximating property): *Let $\Pi = \cup_{n\in\mathbb{N}}\Pi_n$ such that Π_n contains functions defined over $[n]$ and $q' : \mathbb{N} \times (0,1] \to \mathbb{N}$ such that $q'(n,\epsilon) \geq 1/\epsilon$. Suppose that for every $\epsilon > 0$ there exists a property $\Pi^\epsilon = \cup_{n\in\mathbb{N}}\Pi_n^\epsilon$ such that*

1. *Π_n is $0.1/q'(n, 0.9\epsilon)$-approximated by Π_n^ϵ; and*
2. *Π_n^ϵ can be ϵ'-tested by using $q'(n,\epsilon')$ queries that are* each uniformly distributed in $[n]$.

Then, Π can be tested within query complexity $q(n,\epsilon) = O(q'(n, 0.9\epsilon))$.

Note that the transformation presented in the proof does not preserve one-sided error probability. Using Theorem 6 calls for presenting a sequence of parameterized properties (i.e., $(\Pi^\epsilon)_{\epsilon>0}$) such that the approximation distance (to Π) decreases with the parameter (.e., the parameter ϵ of the property Π^ϵ). It is likely that the query complexity increases with that parameter, and using Theorem 6 requires that the rate in which the query complexity increases is slower than the rate in which the approximation distance decreases. See further discussion following the proof.

Proof: On input parameters n,ϵ and oracle access to f, we invoke the guranateed tester for Π^ϵ, denoted T, providing it with the parameters n and 0.9ϵ as well as with access to f, and output whatever T does. The analysis of $T^f(n, 0.9\epsilon)$ is based on the observation that if f is δ-close to some function f', then

$$|\mathbf{Pr}[T^f(n, 0.9\epsilon) = 1] - \mathbf{Pr}[T^{f'}(n, 0.9\epsilon) = 1]| \leq q'(n, 0.9\epsilon)\cdot\delta, \tag{3}$$

since (by the hypothesis) each query is uniformly distributed in $[n]$.

Let $\delta = 0.1/q'(n, 0.9\epsilon)$, and suppose that $f \in \Pi_n$. Then, there exists $f' \in \Pi_n^\epsilon$ that is δ-close to f, and it follows that T accepts f' with probability at least 5/6. By Eq. (3), it follows that T accepts f with probability at least $2/3 - 0.1 > 0.55$, since $q'(n, 0.9\epsilon)\cdot\delta = 0.1$.

On the other hand, for f that is ϵ-far from Π_n, we observe that f must be $(\epsilon-\delta)$-far from Π_n^ϵ, because otherwise f is $(\epsilon-\delta)$-close to a function $g' \in \Pi_n^\epsilon$, which is δ-close to some $g \in \Pi_n$, which implies that f is $((\epsilon-\delta)+\delta)$-close to Π_n. Using $\epsilon - \delta = \epsilon - (0.1/q'(n, 0.9\epsilon)) \geq 0.9\epsilon$, where the inequality is due to the hypothesis $q(n,\epsilon) \geq 1/\epsilon$, it follows that f is 0.9ϵ-far from Π^ϵ, and so T must reject f with probability at least 2/3. Using error reduction, the theorem follows. ■

Applications. In the current context, we approximate a given property Π by a sequence of $(k,\cdot)$-junta properties such that the approximation distance to Π decreases with the junta-size parameter k. It is likely that the query complexity increases with k, and using Theorem 6 requires that the

[6]We chose not to state the approximation parameter as an explicit parameter but rather postulate that it upper bounded by $0.1/q'$, where q' is the query complexity.

rate in which the query complexity increases is slower than the rate in which the approximation distance decreases. In many cases (see examples in Diakonikolas *et al.* [1]), the approximation distance decreases exponentially with k, whereas the query complexity only grows polynomially with k. In such cases, we can apply Theorem 6.

As with Theorem 3, we shall illustrate this application by considering the class of functions that are (monotone or general) monomials, but this time we refer to monomials of unbounded arity. Clearly, the class of (monotone or general) monomials is 2^{-k}-approximated by the corresponding class of monomials of size at most k. The latter class is merely the union of k classes that are each easily testable (i.e., the classes of i-monomials, for $i \in [k]$). Hence, we get –

Corollary 7 (testing monotone and general monomials): *The following two properties of Boolean functions over* $\{0,1\}^\ell$ *can be tested withing query complexity* $\mathrm{poly}(1/\epsilon)$*:*

1. *The set of monotone monomials; that is, functions* $f : \{0,1\}^\ell \to \{0,1\}$ *such that for some set* $I \subseteq [\ell]$ *it holds that* $f(x) = \wedge_{i\in I} x_i$.
2. *The set of monomials; that is, functions* $f : \{0,1\}^\ell \to \{0,1\}$ *such that for some set* $I \subseteq [\ell]$ *and* $\sigma = \sigma_1 \cdots \sigma_\ell \in \{0,1\}^\ell$ *it holds* $f(x) = \wedge_{i\in I}(x_i \oplus \sigma_i)$.

Proof: As stated above, the relevant set of monomials, denoted Π, is 2^{-k}-approximated by the corresponding set of monomials of size at most k, denoted Π'. The latter set is the union over $i \in [k]$ of the sets of corresponding i-monomials. Hence, by Corollary 4 (and the closure of testability under unions)[7], testing Π' has query complexity $\mathrm{poly}(k/\epsilon)$. Setting $k = O(\log(1/\epsilon))$ and applying Theorem 6 , while noting that $2^{-k} < 0.1/\mathrm{poly}(k/\epsilon)$, the corollary follows. ■

More generally, combining Theorems 3 and 6, we get –

Corollary 8 (testing via an approximating $(\cdot,\cdot)$-juntas property): *Let* $\Pi = \cup_{n\in\mathbb{N}}\Pi_n$ *such that* Π_n *contains functions defined over* $[n]$. *Suppose that there exist a function* $\kappa : (0,1] \to \mathbb{N}$ *and a sequence of properties* $(\Phi_k)_{k\in\mathbb{N}}$ *such that* $\Phi_k \subseteq \{f' : \{0,1\}^k \to \{0,1\}\}$ *and it holds that*

1. *For every* $k \in \mathbb{N}$, *the property* Φ_k *is invariant under permuting the bits of the argument to the function*[8] *and* Φ_k *has a sample-based tester of sample complexity* $s_k(\epsilon)$, *where* s_k *is monotonically non-decreasing.*
2. Π_n *is* $\delta(\kappa(\epsilon))$*-approximated by the union over* $i \in [\kappa(\epsilon)]$ *of the sets of* (i,Φ_i)*-juntas and*

$$\delta(\kappa(\epsilon)) < \frac{0.1}{\mathrm{poly}(\kappa(\epsilon)) \cdot \widetilde{O}(s_{\kappa(\epsilon)}(0.8\epsilon))^2}.$$

Then, Π *can be tested within query complexity* $\mathrm{poly}(\kappa(\epsilon)) \cdot \widetilde{O}(s_{\kappa(\epsilon)}(0.8\epsilon))^2$.

Note that the two conditions correspond to the hypotheses in Theorems 3 and 6, respectively. In many cases, $s_k(\epsilon) = \mathrm{poly}(k/\epsilon)$ and $\delta(k) = \exp(k^{\Omega(1)})$, which allows setting $\kappa(\epsilon) = \mathrm{poly}(\log(1/\epsilon))$.

[7]See the first lecture.

[8]As in Theorem 3, this means that $f' \in \Phi_k$ if and only if for every permutation $\pi : [k] \to [k]$ it holds that $f'_\pi(y) = f'(y_{\pi(1)}, ..., y_{\pi(k)})$ is in Φ_k.

Proof: By Theorem 3 and the first hypothesis, for each i we can test (i,Φ_i)-juntas by a tester that makes $\text{poly}(i)\cdot\widetilde{O}(s_i(0.9\epsilon)^2)$ uniformly distributed queries. The same holds with respect to the union of the first k such properties; that is, it can be ϵ'-tested using $\text{poly}(k)\cdot\widetilde{O}(s_k(0.9\epsilon')^2)$ uniformly distributed queries).[9] Fixing any $\epsilon > 0$, let Π'_n be the union of the first $\kappa = \kappa(\epsilon)$ foregoing properties. Then, by the foregoing, Π'_n can be ϵ'-tested using $q'(\epsilon') = \text{poly}(\kappa)\cdot\widetilde{O}(s_\kappa(0.9\epsilon'))^2$ uniformly distributed queries. By the second hypothesis, Π_n is $\delta(\kappa)$-approximated by Π'_n, whereas

$$\begin{aligned}\delta(\kappa) &< \frac{0.1}{\text{poly}(\kappa)\cdot\widetilde{O}(s_\kappa(0.8\epsilon))^2}\\ &\le \frac{0.1}{\text{poly}(\kappa)\cdot\widetilde{O}(s_\kappa(0.9\cdot 0.9\epsilon))^2}\\ &= \frac{0.1}{q'(0.9\epsilon)},\end{aligned}$$

where the inequality uses the hypothesis that s_k is monotonically non-decreasing and the equality uses $q'(0.9\epsilon) = \text{poly}(\kappa)\cdot\widetilde{O}(s_\kappa(0.9\cdot 0.9\epsilon))^2$. Applying Theorem 6, it follows that Π_n can be tested within query complexity $q(\epsilon) = O(q'(0.9\epsilon)) = \text{poly}(\kappa)\cdot\widetilde{O}(s_\kappa(0.9^2\epsilon))^2$, and the corollary follows (recalling that $\kappa = \kappa(\epsilon)$ and using the hypothesis that s_k is monotonically non-decreasing). ■

4 Comments and Exercises

The "testing by implicit sampling" methodology originates in the work of Diakonikolas, Lee, Matulef, Onak, Rubinfeld, Servedio, and Wan [1], which presents numerous applications of it. In particular, their paper uses this methodology to derive testers for several natural properties including sets of functions computable by *bounded size* devices such as decision trees, branching programs, Boolean formulas, and Boolean circuits.

This methodology is often called *testing by implicit learning* (see, e.g., [4]), but we prefer the term "implicit sampling" for reasons that are closely related to the fact that our presentation of the said methodology differs from the one in [1] in several aspects. Firstly, we decouple the *reduction of testing a property* Π *to testing* $(\cdot,\cdot)$*-junta properties that approximate* Π from the actual *testing of* $(\cdot,\cdot)$*-junta properties*: The former reduction is captured by Theorem 6, which is actually more general, whereas the testing of $(\cdot,\cdot)$-junta properties is captured by Theorem 3.

Secondly, we reduce the testing of (k,Φ)-junta properties to *testing* Φ, which is a property of k-variate functions, where the testing is by sample-based testers. In contrast, Diakonikolas *et al.* [1] reduce the testing of (k,Φ)-junta properties to the *proper learning* of Φ (also via sample-based algorithms). Indeed, such a learning algorithm implies a sample-based tester of about the same sample complexity (see first lecture), but there is no reason to restrict the methodology to this special case (since sample-based testing may be easier than learning, see, e.g., [3]). For this reason we prefer to avoid a term that associates this methodology with learning. Furthermore, the core of the methodology is the technique of generating a labeled sample that refers to the (unknown) relevant variables, and it is nice to reflect this fact in the name of the methodology.

Exercises

Exercise 1 (on the complexity of finding the junta – lower bounds): *For each k-subset $I \subseteq [\ell]$,*

[9] See the first lecture (for the closure of testability under unions).

consider the function $f_I : \{0,1\}^\ell \to \{0,1\}$ *defined by* $f_I(x) = \oplus_{i \in I} x_i$. *Prove that finding* I *requires at least* $\log_2 \binom{\ell}{k} - 1$ *queries, when given access to an arbitrary* f_I, *even if one is allowed to fail with probability at most* $1/3$.

Guideline: consider first the case of deterministic algorithms. The computation of such an algorithm is captured by a decision tree in which the vertices correspond to queries, and the edges represent to the corresponding answers. Hence, a deterministic algorithm that finds the set I corresponds to a decision tree that has at least $\binom{\ell}{k}$ different leaves (which implies that its depth is at least $\log_2 \binom{\ell}{k}$). Turning to randomized algorithms, note that each such algorithm can be viewed as a distribution on such decision trees, and the distribution must contain a tree that corresponds to an algorithm that succeeds on at least a $2/3$ fraction of the possible functions (which means that this tree must have at least $\frac{2}{3} \cdot \binom{\ell}{k}$ different leaves).

Exercise 2 (on the complexity of finding the junta – upper bounds): *Present a randomized algorithm that when given access to a* k*-junta* $f : \{0,1\}^\ell \to \{0,1\}$ *in which each junta variable has influence at least* ϵ, *finds the junta with probability at least* $2/3$ *while making* $\widetilde{O}(k) \cdot (\log \ell)/\epsilon$ *queries.*

Guideline: On input f, for $t = O(k^2)$, we first select a random t-partition, $(R_1, ..., R_t)$, as in Step 1 of Algorithm 3.2, and find $J = \{j \in [\ell] : \mathtt{I}_{R_j}(f) \geq \epsilon\}$. Next, for each $j \in J$, we find $i \in R_j$ such that $\mathtt{I}_{\{i\}}(f) \geq \epsilon$ by a binary search, while using Algorithm 3.1 to estimate the influence of the various relevant subsets. This algorithm makes $(\widetilde{O}(t) + \widetilde{O}(k) \log \ell)/\epsilon$ queries, but the first step can be made more efficient.[10]

Exercise 3 (properties of Boolean functions that are invariant under renaming of variables): *Prove that all properties of* ℓ*-variate Boolean functions discussed in prior lectures are invariant under renaming of the variables; that is,* $f : \{0,1\}^\ell \to \{0,1\}$ *has the property for every permutation* $\pi : [\ell] \to [\ell]$ *it holds that* $f_\pi(x) = f(x_{\pi(1)}, ..., x_{\pi(\ell)})$ *has the property. Specifically, consider the following properties: linearity (and being a low degree polynomial), monotonicity, being a monotone dictatorship, being a (monotone or general) monomial, and being a* k*-junta.*

Exercise 4 (the influences of a set on close functions): *Prove that if* $f : \{0,1\}^\ell \to \{0,1\}$ *is* ϵ*-close to* $g : \{0,1\}^\ell \to \{0,1\}$, *then* $|\mathtt{I}_S(f) - \mathtt{I}_S(g)| \leq 2 \cdot \epsilon$ *for every* $S \subseteq [\ell]$.

Guideline: Fixing S, let D_f (resp., D_g) denote the set of pairs $(r,s) \in \{0,1\}^\ell \times \{0,1\}^\ell$ such that $r_{\overline{S}} = s_{\overline{S}}$ and $f(r) \neq f(s)$ (resp., $g(r) \neq g(s)$). Then, $|D_f| - |D_g| \leq |D_f \bigtriangledown D_g|$, where $A \bigtriangledown B$ denotes the symmetric difference between A and B. It follows that

$$\begin{aligned}
|\mathtt{I}_S(f) - \mathtt{I}_S(g)| &\leq \mathbf{Pr}_{r,s:r_{\overline{S}}=s_{\overline{S}}}[(r,s) \in D_f \bigtriangledown D_g] \\
&= \mathbf{Pr}_{r,s:r_{\overline{S}}=s_{\overline{S}}}[f(r) - f(s) \neq g(r) - g(s)] \\
&\leq \mathbf{Pr}_{r,s:r_{\overline{S}}=s_{\overline{S}}}[f(r) \neq g(r) \vee g(s) \neq g(s)].
\end{aligned}$$

Exercise 5 (an easy case of approximation): *Show that the class of functions that are* ϵ*-close to* Π *is* ϵ*-approximated by* Π.

[10] Place $R_1, ..., R_t$ at the t leaves of a balanced binary tree and let each internal vertex hold the union of the sets placed at its children. Now conduct a DFS from the root while continuing only on vertices that were found to hold an influential set.

Exercise 6 (an easy case of approximation): *Suppose that any two functions in* Π *are at distance at least* ϵ *of one another, and let* Π' *be the class of functions that are at distance approximately* $\epsilon/2$ *from some function in* Π (i.e., $\Pi = \{f : \delta_\Pi(f) \in [0.4\epsilon, 0.6\epsilon\})$. *Show that* Π' *is* 0.6ϵ*-approximated by* Π.

References

[1] I. Diakonikolas, H.K. Lee, K. Matulef, K. Onak, R. Rubinfeld, R.A. Servedio, and A. Wan. Testing for Concise Representations. In *48th IEEE Symposium on Foundations of Computer Science*, pages 549–557, 2007.

[2] O. Goldreich (ed.). *Property Testing: Current Research and Surveys.* Springer, LNCS, Vol. 6390, 2010.

[3] O. Goldreich and D. Ron. On Sample-Based Testers. In *6th Innovations in Theoretical Computer Science*, pages 337–345, 2015.

[4] R. Servedio. Testing by Implicit Learning: A Brief Survey. In [2].