# Almost Optimal Testers for Concise Representations

**Nader H. Bshouty**
Department of Computer Science, Technion, Haifa, Israel
bshouty@cs.technion.ac.il

## Abstract

We give improved and almost optimal testers for several classes of Boolean functions on $n$ variables that have concise representation in the uniform and distribution-free model. Classes, such as $k$-Junta, $k$-Linear, $s$-Term DNF, $s$-Term Monotone DNF, $r$-DNF, Decision List, $r$-Decision List, size-$s$ Decision Tree, size-$s$ Boolean Formula, size-$s$ Branching Program, $s$-Sparse Polynomial over the binary field and functions with Fourier Degree at most $d$.

The approach is new and combines ideas from Diakonikolas et al. [24], Bshouty [13], Goldreich et al. [32], and learning theory. The method can be extended to several other classes of functions over any domain that can be approximated by functions with a small number of relevant variables.

**2012 ACM Subject Classification** Mathematics of computing; Mathematics of computing → Discrete mathematics; Mathematics of computing → Probabilistic algorithms; Theory of computation → Probabilistic computation

**Keywords and phrases** Property Testing, Boolean function, Junta

**Digital Object Identifier** 10.4230/LIPIcs.APPROX/RANDOM.2020.5

**Category** RANDOM

**Related Version** A full version of the paper is available at `https://arxiv.org/abs/1904.09958`.

**Acknowledgements** I am extremely grateful to Oded Goldreich for very helpful discussions and comments and for allowing me to include his overview of the first tester in the paper (Section 2). I am also grateful to the reviewers of the earlier version of the manuscript for their helpful comments.

## 1 Introduction

Property testing of Boolean function was first considered in the seminal works of Blum, Luby and Rubinfeld [12] and Rubinfeld and Sudan [46] and has recently become a very active research area. See for example, [2, 4, 5, 6, 8, 9, 13, 15, 16, 17, 18, 19, 20, 21, 24, 27, 31, 33, 36, 37, 41, 40, 43, 47] and other works referenced in the surveys and books [29, 30, 44, 45].

A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is said to be $k$-junta if it depends on at most $k$ coordinates. The class $k$-Junta is the class of all $k$-juntas. The class $k$-Junta has been of particular interest to the computational learning theory community [10, 11, 14, 23, 34, 38, 42]. A problem closely related to learning $k$-Junta is the problem of learning and testing subclasses $C$ of $k$-Junta and classes $C$ of Boolean functions that can be approximated by $k$-juntas [9, 11, 25, 17, 24, 32, 33, 43]. In both testing and learning we are given black-box query access to a Boolean function $f$. In learning, for $f \in C$, we need to learn, with high probability, a hypothesis $h$ that is $\epsilon$-close to $f$. In testing, for any Boolean function $f$, we need to distinguish, with high probability, the case that $f$ is in $C$ versus the case that $f$ is $\epsilon$-far from every function in $C$.

In the *uniform-distribution property testing* (and learning) model, the distance between Boolean functions is measured with respect to the uniform distribution. In the *distribution-free property testing*, [32], (and learning [48]) the distance between Boolean functions is measured with respect to an arbitrary and unknown distribution $\mathcal{D}$ over $\{0,1\}^n$. In the


Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020).
Editors: Jarosław Byrka and Raghu Meka; Article No. 5; pp. 5:1–5:20
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

distribution-free model, the testing (and learning) algorithm is allowed (in addition to making black-box queries) to draw random $x \in \{0,1\}^n$ according to the distribution $\mathcal{D}$. This model is studied in [13, 22, 26, 28, 35, 39].

## 1.1 Results

In Table 1, we list all the previous results and our results in this paper. In the table, $\tilde{O}(T)$ stands for $O(T \cdot Poly(\log T))$, $U$ and $D$ stand for uniform and distribution-free models, resp., and Exp and Poly stand for exponential and polynomial time, resp.

It follows from the lower bounds of Saglam [47], that our query complexity is almost optimal (up-to log-factor) for the classes $k$-Junta, $k$-Linear, $k$-Term, $s$-Term DNF, $s$-Term Monotone DNF, $r$-DNF ($r$ constant), Decision List, $r$-Decision List ($r$ constant), size-$s$ Decision Tree, size-$s$ Branching Programs and size-$s$ Boolean Formula.

## 1.2 Notations

In this subsection, we give some notations that we use throughout the paper.

Denote $[n] = \{1, 2, \dots, n\}$. For $S \subseteq [n]$ and $x = (x_1, \dots, x_n)$ we denote $x(S) = \{x_i | i \in S\}$. For $X \subset [n]$ we denote by $\{0,1\}^X$ the set of all binary strings of length $|X|$ with coordinates indexed by $i \in X$. For $x \in \{0,1\}^n$ and $X \subseteq [n]$ we write $x_X \in \{0,1\}^X$ to denote the projection of $x$ over coordinates in $X$. We denote by $1_X$ and $0_X$ the all-one and all-zero strings in $\{0,1\}^X$, respectively. When $y$ is a variable then $z = (y)_X$ is the all $y$ string with coordinates indexed by $i \in X$, i.e., $z_i = y$ for all $i \in X$. When we write $x_I = 0$ we mean $x_I = 0_I$. For $X_1, X_2 \subseteq [n]$ where $X_1 \cap X_2 = \emptyset$ and $x \in \{0,1\}^{X_1}, y \in \{0,1\}^{X_2}$ we write $x \circ y$ to denote their concatenation, i.e., the string in $\{0,1\}^{X_1 \cup X_2}$ that agrees with $x$ over coordinates in $X_1$ and agrees with $y$ over coordinates in $X_2$. For $X \subseteq [n]$ we denote $\overline{X} = [n] \backslash X = \{x \in [n] | x \notin X\}$. For a function $f : \{0,1\}^k \to \{0,1\}$, $x \in \{0,1\}^n$ and $X = \{i_1, \dots, i_k\} \subseteq [n]$ where $i_1 < i_2 < \cdots < i_k$, when we write $f(x_X)$ we mean $f(x_{i_1}, \dots, x_{i_k})$.

Given $f, g : \{0,1\}^n \to \{0,1\}$ and a probability distribution $\mathcal{D}$ over $\{0,1\}^n$, we say that $f$ is *$\epsilon$-close to $g$ with respect to $\mathcal{D}$* if $\mathbf{Pr}_{x \in \mathcal{D}}[f(x) \neq g(x)] \leq \epsilon$, where $x \in \mathcal{D}$ means $x$ is chosen from $\{0,1\}^n$ according to the distribution $\mathcal{D}$. We say that $f$ is *$\epsilon$-far from $g$ with respect to $\mathcal{D}$* if $\mathbf{Pr}_{x \in \mathcal{D}}[f(x) \neq g(x)] \geq \epsilon$. For a class of Boolean functions $C$, we say that $f$ is *$\epsilon$-far from every function in $C$ with respect to $\mathcal{D}$* if for every $g \in C$, $f$ is $\epsilon$-far from $g$ with respect to $\mathcal{D}$. We will use $U$ to denote the uniform distribution over $\{0,1\}^n$ or over $\{0,1\}^X$ when $X$ in clear from the context.

For a distribution $\mathcal{D}$ over $\{0,1\}^n$ and $X \subset [n]$, we denote by $\mathcal{D}_X$ the distribution $\mathcal{D}$ projected on the coordinates $X$. That is, the distribution of $x_X$ when $x \in \mathcal{D}$.

For a Boolean function $f$ and $X \subset [n]$, we say that $X$ is an *influential set* of $f$ if there are $a, b \in \{0,1\}^n$ such that $f(a) \neq f(b_X \circ a_{\overline{X}})$. We call the pair $(a, b)$ (or just $a$ when $b = 0$) a *witness* of $f$ for the influential set $X$. When $X = \{i\}$ then we say that $x_i$ is an *influential variable* of $f$ and $a$ is *a witness* of $f$ for $x_i$. Obviously, if $X$ is influential set of $f$ then $x(X)$ contains at least one influential variable of $f$.

We say that the Boolean function $f : \{0,1\}^n \to \{0,1\}$ is a *literal* (*dictatorship* and *anti-dictatorship*, resp.) if $f \in \{x_1, \dots, x_n, \overline{x_1}, \dots, \overline{x_n}\}$ where $\overline{x}$ is the negation of $x$ ($f \in \{x_1, \dots, x_n\}$ and $f \in \{\overline{x_1}, \dots, \overline{x_n}\}$, resp.).

Let $C$ be a class of Boolean functions $f : \{0,1\}^n \to \{0,1\}$. We say that $C$ is *symmetric* if for every permutation $\pi : [n] \to [n]$ and every $f \in C$ we have $f_\pi \in C$ where $f_\pi(x) := f(x_{\pi(1)}, \cdots, x_{\pi(n)})$. We say that $C$ is *closed under zero projection* (resp. *closed under one projection*) if for every $f \in C$ and every $i \in [n]$, $f(0_{\{i\}} \circ x_{\overline{\{i\}}}) \in C$ (resp. $f(1_{\{i\}} \circ x_{\overline{\{i\}}}) \in C$). We say it is closed under zero-one projection if is closed under zero and one projection.

**Table 1** A table of the results. In the table, $\tilde{O}(T)$ stands for $O(T\cdot poly(\log T))$, $U$ and $D$ stand for uniform and distribution-free models, resp., and Exp and Poly stand for exponential and polynomial time, resp.

| Class of Functions | Model | #Queries | Time | Reference |
|---|---|---|---|---|
| $s$-Term Monotone DNF | U | $\tilde{O}(s^2/\epsilon)$ | Poly. | [43] |
| $s$-Term Unate DNF | U | $\tilde{O}(s/\epsilon^2)$ | Exp. | [17] |
| | U | $\tilde{O}(s/\epsilon)$ | Poly. | This Paper |
| $s$-Term Monotone $r$-DNF | U | $\tilde{O}(s/\epsilon^2)$ | Exp. | [17] |
| $s$-Term Unate $r$-DNF | U | $\tilde{O}(s/\epsilon)$ | Poly. | This Paper |
| | D | $\tilde{O}(s^2r/\epsilon)$ | Poly. | This Paper |
| $s$-Term DNF | U | $\tilde{O}(s^2/\epsilon)$ | Exp. | [24] |
| | U | $\tilde{O}(s/\epsilon^2)$ | Exp. | [17] |
| | U | $\tilde{O}(s/\epsilon)$ | Exp. | This Paper |
| $r$-DNF (Constant $r$) | U | $\tilde{O}(1/\epsilon)$ | Poly. | This Paper |
| Decision List | U | $\tilde{O}(1/\epsilon^2)$ | Poly. | [24] |
| | U | $\tilde{O}(1/\epsilon)$ | Poly. | This Paper |
| Length-$k$ Decision List | D | $\tilde{O}(k^2/\epsilon)$ | Poly. | This Paper |
| $r$-DL (Constant $r$) | U | $\tilde{O}(1/\epsilon)$ | Poly. | This Paper |
| $k$-Linear | U | $\tilde{O}(k/\epsilon)$ | Poly. | [7, 12] |
| | D | $\tilde{O}(k/\epsilon)$ | Poly. | This Paper |
| $k$-Term | U | $O(1/\epsilon)$ | Poly. | [43] |
| | U | $\tilde{O}(1/\epsilon)$ | Poly. | This Paper |
| | D | $\tilde{O}(k/\epsilon)$ | Poly. | This Paper |
| size-$s$ Decision Trees and | U | $\tilde{O}(s/\epsilon^2)$ | Exp. | [17] |
| size-$s$ Branching Programs | U | $\tilde{O}(s/\epsilon)$ | Exp. | This Paper |
| | D | $\tilde{O}(s^2/\epsilon)$ | Exp. | This Paper |
| size-$s$ Boolean Formulas | U | $\tilde{O}(s/\epsilon^2)$ | Exp. | [17] |
| | U | $\tilde{O}(s/\epsilon)$ | Exp. | This Paper |
| size-$s$ Boolean Circuit | U | $\tilde{O}(s^2/\epsilon^2)$ | Exp. | [17] |
| | U | $\tilde{O}(s^2/\epsilon)$ | Exp. | This Paper |
| Functions with | U | $\tilde{O}(2^{2d}/\epsilon^2)$ | Exp. | [17] |
| Fourier Degree $\le d$ | D | $\tilde{O}(2^d/\epsilon+2^{2d})$ | Poly. | This Paper |
| $s$-Sparse Polynomial | U | $poly(s/\epsilon)+\tilde{O}(2^{2d})$ | Poly. | [1, 25] |
| over $F_2$ of Degree $d$ | U | $\tilde{O}(s^2/\epsilon+2^{2d})$ | Poly. | This Paper+[1] |
| | U | $\tilde{O}(s/\epsilon+s2^d)$ | Poly. | This Paper |
| | D | $\tilde{O}(s^2/\epsilon+s2^d)$ | Poly. | This Paper |
| $s$-Sparse Polynomial | U | $\tilde{O}(s/\epsilon^2)$ | Exp. | [17] |
| over $F_2$ | U | $Poly(s/\epsilon)$ | Poly. | [25] |
| | U | $\tilde{O}(s^2/\epsilon)$ | Poly. | This Paper |

## 2 Overview of the Distribution-Free Tester

### 2.1 Preface

Our approach refers to testing properties that are (symmetric) sub-classes $C$ of $k$-juntas; that is, $f:\{0,1\}^n\to\{0,1\}$ has the property if there exists a function $f':\{0,1\}^k\to\{0,1\}$ that belongs to a predetermined class $C'$ of functions (over $k$-bit strings) such that $f(x)=f'(x_\Gamma)$

for some $k$-subset $\Gamma$. Our new approach builds upon the "testing by implicit sampling" approach of Diakonikolas et al. [24], while extending it from the case of uniform distribution to the case of arbitrary unknown distributions (i.e., the distribution-free model).

This allows us to present (almost optimal) *distribution-free* testers for classes of properties that are sub-classes of $k$-juntas, which correspond to classes of $k$-bit long Boolean functions.

While we follow Diakonikolas et al. [24] in considering learning algorithms for the underlying classes, our approach is also applicable to testing algorithms (see [30, Sec. 6.2]).

Let us again spell out our task. For a class $C$ of $n$-bit long Boolean functions and a proximity parameter $\epsilon$, given samples from an unknown distribution $\mathcal{D}$ and oracle access to a function $f : \{0,1\}^n \to \{0,1\}$, we wish to distinguish the case that $f \in C$ from the case that $f$ is $\epsilon$-far from $C$. Recall that $C$ is a (symmetric) class consisting of a symmetric subclass of $k$-juntas $C'$; that is, $f \in C$ if and only if there exists a $k$-subset $\Gamma \subset [n]$ and $f' \in C'$ such that $f(x) = f'(x_\Gamma)$, where $x_{\{i_1,...,i_k\}} = (x_{i_1}, \ldots, x_{i_k})$. Actually, we also assume that $C'$ is closed under zero projection.

## 2.2 A Bird's Eye View

The basic strategy is to consider a random partition of $[n]$ to $r = O(k^2)$ parts, denoted $(X_1, \ldots, X_r)$, while relying on the fact that, whp, each $X_i$ contains at most one influential variable. Assuming that $f \in C$, first we determine a set $I$ of at most $k$ indices such that $\cup_{i\in[n]\setminus I} X_i$ contains no "significantly influential" variables of $f$. Suppose that $f' : \{0,1\}^k \to \{0,1\}$, $f' \in C'$, is a function that corresponds to the tested function $f : \{0,1\}^n \to \{0,1\}$, and that $I \subset [n]$ is indeed the collection of all sets that contain influential variables. The crucial ingredient is devising a method that allows to generate samples of the form $(x', f'(x'))$, when given samples of the form $(x, f(x))$ (for $x \in \mathcal{D}$). We stress that we cannot afford to find the influential variables, and so this method works without determining these locations. Using this method, we can test whether $f'$ belongs to the underlying class $C'$; hence, we test $f$ by implicitly sampling the projection of $\mathcal{D}$ on the (unknown) influential variables.

The method employed by Diakonikolas et al. [24] only handles the uniform distribution (i.e., the case that $\mathcal{D}$ is uniform over $\{0,1\}^n$), and so it only yields testers for the standard testing model (rather than for the distribution-free testing model). Furthermore, their method as well as the identification of the set $I$ rely heavily on the notion of influence of sets, where the influence of a set $S$ of locations on the value of a function is defined as $\Pr_{x',x''\in\{0,1\}^n:x'_S=x''_S}[f(x') \neq f(x'')]$. However, this notion refers to the uniform distribution (over $\{0,1\}^n$) and does not seem adequate for the distribution-free context (e.g., for[1] $f(x) = x_1 + x_2$ we may get $\Pr_{x',x''\in\mathcal{D}:x'_1=x''_1}[f(x') \neq f(x'')] = 0$).

We use a different way of identifying the set $I$ and for generating samples for the underlying function $f'$. Loosely speaking, we identifies $I$ as the set of indices $i$ for which $f(1_{X_i} \circ 0_{\overline{X_i}}) \neq f(0^n)$, where (recall that) $1_S \circ 0_{\overline{S}}$ is a string that is 1 on the locations in $S$ and is 0 on other locations. (*Be warned that this description is an over-simplification*!) This means that for every $i \in I$ and $x \in \{0,1\}^n$, the value of $x$ at the influential variable in the set $X_i$ (a variable whose location is unknown to us!), equals $f(x') + f(0^n)$ where $x' = x_{X_i} \circ 0_{\overline{X_i}}$, i.e., $x'_j = x_j$ if $j \in X_i$ and $x'_j = 0$ otherwise.[2] Note that the foregoing holds when $f \in C$; in

[1] The addition operation in this paper is over the binary field $F_2$.

[2] Indeed, if $\tau(i) \in X_i$ is the index of the (unique) influential variable that resides in the set $X_i$, then

$$f(x') = x_{\tau(i)} \cdot f(1_{X_i} \circ 0_{\overline{X_i}}) + (x_{\tau(i)} + 1) \cdot f(0^n) = x_{\tau(i)} + f(0^n)$$

since $f(1_{X_i} \circ 0_{\overline{X_i}}) + f(0^n) = 1$.

general, we can test whether $x \mapsto f(x') + f(0^n)$ is close to a dictatorship (under the uniform distribution) and reject otherwise, whereas if the mapping is close to a dictatorship, we can self-correct it.

To sample the distribution $\mathcal{D}_\Gamma$, where $\Gamma$ is the influential variables in $X_I = \cup_{i\in I} X_i$, we sample $\mathcal{D}$ and determine the value of the influential variable in each set $X_i$, for $i \in I$. Queries to the function $f'$ are answered by querying $f$ such that the query $y = y_1 \cdots y_k$ is mapped to the query $\mathtt{ext}(y)$ such that[3] $\mathtt{ext}(y)_j = y_i$ if $j$ belongs to the $i^{\text{th}}$ set in the collection $I$ (and $\mathtt{ext}(y)_j = 0$ if $j \in [n] \setminus X_I$). Effectively, we query the function $F : \{0,1\}^n \to \{0,1\}$ defined as $F(x) = f(\mathtt{ext}(x_\Gamma))$, and this makes sense provided that $F$ is close to $f$ (under the distribution $\mathcal{D}$). To test the latter hypothesis condition, we sample $\mathcal{D}$ and for each sample point $x$ we compare $f(x)$ to $F(x)$, where here we again use the ability to determine the value of the influential variable in each set. Specifically, $\mathtt{ext}(x_\Gamma)$ is computed by determining the value of $x_\Gamma$ (without knowing $\Gamma$), and using our knowledge of $(X_i)_{i\in I}$.

We warn that the foregoing description presumes that we have correctly identified the collection $I$ of all sets containing an influential variable. This leaves us with two questions: The first question is, how do we identify the set $I$. (Note that the influence of a variable may be as low as $2^{-k}$, whereas we seek algorithms of poly($k$)-complexity.) The solution (to be presented in Section 2.3.1) will be randomized, and will have one-sided error; specifically, we may fail to identify some sets that contain influential variables, but will never include in our collection sets that have no influential variables. Consequently, $f(1_{X_i} \circ 0_{\overline{X_i}}) \neq f(0^n)$ may not hold for some $i \in I$, and (over-simplifying again) we shall seek instead some $v^{(i)} \in \{0,1\}^n$ such that $f(v^{(i)}) \neq f(w^{(i)})$, where $w^{(i)} = v^{(i)}_{\overline{X_i}} \circ 0_{X_i}$ (i.e., $w^{(i)}_j = v^{(i)}_j$ if $j \in [n] \setminus X_i$ and $w^{(i)}_j = 0$ otherwise). Second, as before, for every $i \in I$ and $x \in \{0,1\}^n$, we wish to determine the value in $x$ of the influential variable in the set $X_i$ (a variable whose location is unknown to us!). This is done by observing that if $f \in C$ then this value equals $f(x') + f(v^{(i)}) + 1$ where $x' = x_{X_j} \circ v^{(i)}_{\overline{X_j}}$ (i.e., $x'_j = x_j$ if $j \in X_i$ and $x'_j = v^{(i)}_j$ otherwise).[4] Again, we need to test whether $x \mapsto f(x') + f(v^{(i)}) + 1$ is a dictatorship, and use self-correction.

## 2.3 The Actual Tester

As warned, the above description is an over-simplification, and the actual way in which the set $I$ is identified and used is more complex.

We fix a random partition of $[n]$ to $r = O(k^2)$ parts, denoted $(X_1, \ldots, X_r)$. If $f \in C$, then, with high probability, each $X_i$ contains at most one influential variable, denoted $\tau(i)$. We assume that this is the case when providing intuition throughout this section.

### 2.3.1 Stage 1: Finding $I$ and corresponding $v^{(i)}$

Our goal is to find a collection $I$ of at most $k$ sets such that the function $h_I$ is $\epsilon/3$-close to $f$ (w.r.t distribution $\mathcal{D}$), where $h_I$ is defined as $h_I(x) = f(x_{X_I} \circ 0_{\overline{X_I}})$ and $X_I = \cup_{i\in I} X_i$. In addition, for each $i \in I$, we seek a witness $v^{(i)}$ for the fact that $f$ depends on some variable in $X_i$; that is, $f(v^{(i)}) \neq f(w^{(i)})$ for some $v^{(i)}$ that differ from $w^{(i)}$ only on $X_i$.

---

[3] Notice that $\mathtt{ext}(y) = 0_{\overline{X_I}} \circ \left( \underset{i\in I}{\circ} (y_i)_{X_i} \right)$ - Here $(y)_X = 1_X$ if $y = 1$ and $0_X$ if $y = 0$.

[4] Indeed, if $\tau(i) \in X_i$ is the index of the (unique) influential variable that resides in the set $X_i$, then

$$f(x') = x_{\tau(i)} \cdot f(v^{(i)}) + (x_{\tau(i)} + 1) \cdot f(w^{(i)}) = x_{\tau(i)} + f(v^{(i)}) + 1$$

since $f(v^{(i)}) + f(w^{(i)}) = 1$.

#### The procedure

We proceed in iterations, starting with $I = \emptyset$.

1. We sample $\mathcal{D}$ for $O(1/\epsilon)$ times, trying to find $u \in \mathcal{D}$ such that $f(u) \neq h_I(u)$.
(Note that if $I = \emptyset$, then $h_I(u) = f(0^n)$. In general, we seek $u$ such that $f(u) \neq f(u_{X_I} \circ 0_{\overline{X_I}})$.
If no such $u$ is found, then we set $h = h_I$ and proceed to Stage 2. In this case, we may assume that $h_I$ is $\epsilon/3$-close to $f$ (w.r.t $\mathcal{D}$).
2. Otherwise (i.e., $f(u) \neq h_I(u)$), we find an $i \in [m] \setminus I$ and $v^{(i)}$ such that $h_I(v^{(i)}) \neq h_{I\cup\{i\}}(v^{(i)})$, which means that $X_i$ contains an influential variable and $v^{(i)}$ is the witness for the sensitivity that we seek. We set $I \leftarrow I \cup \{i\}$ and proceed to the next iteration.
(We find this $i$ by binary search that seeks $i$ and $S$ such that $h_{I\cup S\cup\{i\}}(u) \neq h_{I\cup S}(u)$, which means that $v^{(i)}$ equals $u$ in locations outside $S$ and is zero on $S$.)[5]

Once the iterations are suspended (due to not finding $u$), we reject if $|I| > k$, and continue to the Stage 2 otherwise. Recall that in the latter case $h = h_I$ is $\epsilon/3$-close to $f$ (w.r.t $\mathcal{D}$).

Note that if $f \in C$, then $I$ contains only sets that contain variables of the $k$-junta, and so we never reject in this stage. In general, if $i \in I$, then $h_{I\setminus\{i\}}(v^{(i)}) \neq h_I(v^{(i)})$, which implies that $f(x') \neq f(x'')$, where $x'$ and $x''$ differ only on $X_i$ (e.g., $x''_{X_I} = v^{(i)}_{X_I}$ and $x''_j = 0$ if $j \notin X_I$).

## 2.3.2 Stage 2: Extracting the value of an influential variable

Given a collection $I$ as found in Stage 1 (and a sensitivity witness $v^{(i)}$ for each $i \in I$), let $h = h_I$ and recall that $h$ is close to $f$ w.r.t $\mathcal{D}$. For each $i \in I$, given $x \in \{0,1\}^n$, we wish to determine the value of $x$ at the influential variable that resides in $X_i$.

For each $i \in I$, we define $\nu_i : \{0,1\}^{|X_i|} \to \{0,1\}$ such that $\nu_i(z) = h_I(y)$, where $y_{X_i} = z$ and $y_{\overline{X_i}} = v^{(i)}_{\overline{X_i}}$. Suppose that $f \in C$, and recall that $\tau(i) \in X_i$ denotes the location of the influential variable in $X_i$. Let $\sigma(i)$ denote the index of $\tau(i)$ in $X_i$ (i.e., the $\sigma(i)^{\text{th}}$ element of $X_i$ is $\tau(i)$). Then, in this case, $\nu_i$ is either a dictatorship or an anti-dictatorship. In particular, if $\nu_i$ is a dictatorship, then $\nu_i(z) = z_{\sigma(i)}$ (and otherwise $\nu_i(z) = z_{\sigma(i)} + 1$).

For each $i \in I$, we test whether $\nu_i$ is a dictatorship or anti-dictatorship, where testing is w.r.t the uniform distribution over $\{0,1\}^{|X_i|}$. Note that we also check whether $\nu_i$ is a dictatorship or anti-dictatorship. If the tester (run with proximity parameter 0.1) fails, we reject. Otherwise (i.e., if we did not reject), we can compute $\nu_i$ via self-correction on $h_I$; that is, to compute $\nu_i$ at $z$, we select $u \in \{0,1\}^{|X_i|}$ at random, and return $\nu_i(z+u) + \nu_i(u)$, which (w.h.p.) equals $(z+u)_{\sigma(i)} + u_{\sigma(i)} = z_{\sigma(i)}$.

Hence, we always continue to Stage 3 if $f \in C$, and whenever we continue to Stage 3 we can compute all $\nu_i$ (for $i \in I$) via self-correction.

## 2.3.3 Stage 3: Emulating a tester of $C'$

Recall that when reaching this stage, we may assume that $h = h_I$ is $\epsilon/3$-close to $f$ (w.r.t $\mathcal{D}$). Also recall that $h_I(x)$ depends only on $x_{X_I}$, where $X_I = \cup_{i\in I} X_i$, and that by Stage 2 we may assume that $\nu_i(z) = z_{\sigma(i)}$ (for every $i \in I$ and almost all $z$). In light of the forgoing, we define $F : \{0,1\}^n \to \{0,1\}$ such that $F(x) = h(x')$ where $x'_{X_i} = (x_{\sigma(i)}, \dots, x_{\sigma(i)})$ (i.e.,

---

[5] By Step 1, we have $h_{S'\cup I}(u) \neq h_{S''\cup I}(u)$, for $S' = [n] \setminus I$ and $S'' = \emptyset$, and in each iteration we cut $S' \setminus S''$ by half while maintaining $h_{S'\cup I}(u) \neq h_{S''\cup I}(u)$.

$x'_j = (x_{X_i})_{\sigma(i)} = x_{\tau(i)}$ if $j \in X_i$)[6] and $x'_j = 0$ otherwise. (Indeed, if $f \in C$, then $F(x) = h(x)$, since $h(y)$ depends only on $(y_{\tau(i)})_{i \in I}$. Using hypothesis that $C'$ (and so $C$) is closed under zero projection, it follows that $F \in C$.)

We observe that if $F$ is $\epsilon/3$-close (w.r.t $\mathcal{D}$) to both $h$ and $C$, then $f$ must be $\epsilon$-close to $C$ (since $f$ is $\epsilon/3$-close to $h$). Hence, we test both these conditions. Specifically, using our ability to sample $\mathcal{D}$, query $f$, and determine the value of the influential variables in $X_I$, we proceed as follows:

1. Test whether $F = h$, where testing is w.r.t the distribution $\mathcal{D}$ and proximity parameter $\epsilon/3$.

   This is done by taking $O(1/\epsilon)$ samples of $\mathcal{D}$, and comparing the values of $F$ and $h$ on these sample points. Recall that $h(u) = h_I(u) = f(u_{X_I} \circ 0_{\overline{X_I}})$.

   The value of $F$ on $u$ is determined as follows.

   a. For every $i \in I$, if $\nu_i$ is a dictatorship, then set $v_i$ to equal the self-corrected value of $\nu_i(u_{X_i})$, where $\nu_i$ is as defined in Stage 2. Otherwise (i.e., when $\nu_i$ is an anti-dictatorship), we set $v_i$ to equal the self-corrected value of $\nu_i(u_{X_i}) + 1$.

   b. Return the value $h(u')$, where $u'_j = v_i$ if $j \in X_i$ and $u'_j = 0$ otherwise.

   Indeed, $F = h$ always passes this test, whereas $F$ that is $\epsilon/3$-far from $h$ (w.r.t $\mathcal{D}$) is rejected w.h.p.

2. Test whether $F$ is in $C$, where testing is w.r.t the distribution $\mathcal{D}$ and proximity parameter $\epsilon/3$. This is done by testing whether $F'$ is in $C$, where $F'(z) = F(x)$ such that $x_j = z_i$ if $j$ is in the $i^{\text{th}}$ set in the collection $I$, and $x_j = 0$ otherwise. Here we use a distribution-free tester, and analyze it w.r.t the distribution $\mathcal{D}_I$. Toward this end, we need to samples $\mathcal{D}_I$ as well as answer queries to $F'$, where both tasks can be performed as in the prior step. Recall that if $f \in C$, then $F \in C$, and this test will accept (w.h.p.), whereas if $F$ is $\epsilon/3$-far from $C$ the test will reject (w.h.p.).

We conclude that if we reached Stage 3 and $f \in C$ (resp., $f$ is $\epsilon$-far from $C$), then we accept (resp., reject) w.h.p.

## 2.4 Digest: Our approach vs the original one [24]

Our new approach differs from the original approach of Diakonikolas et al. [24] in two main aspects:

1. In [24], sets that contain influential variables are identified according to their influence, which is defined with respect to the uniform distribution. This definition seems inadequate when dealing with arbitrary distributions. Instead, we identify such a set by searching for two assignments that differ only on this set and yield different function values. The actual process is iterative and places additional constraints on these assignments (as detailed in Section 2.3.1).
2. In [24], given an assignment to the function, the value of the unique influential variable that resides in a given set $S$ is determined by approximating the influence of two subsets of $S$ (i.e., the subsets of locations assigned the value 0 and 1, respectively). In contrast, we determines this value by defining an auxiliary function, which depends on the unknown influential variable, and evaluating this function (via self-correction w.r.t the uniform distribution (!); see Section 2.3.2).

[6] In general, $\tau(i)$ denotes the location in $[n]$ of the $\sigma(i)^{\text{th}}$ element of $X_i$.

## 2.5 The Model

In this subsection, we define the testing and learning models.

In the testing model, we consider the problem of testing a Boolean function class $C$ in the uniform and distribution-free testing models. In the distribution-free testing model (resp. uniform model), the algorithm has access to a Boolean function $f$ via a black-box that returns $f(x)$ when a string $x$ is queried. We call this query *membership query* ($\mathrm{MQ}_f$ or just MQ). The algorithm also has access to unknown distribution $\mathcal{D}$ (resp. uniform distribution) via an oracle that returns $x \in \{0,1\}^n$ chosen randomly according to the distribution $\mathcal{D}$ (resp. according to the uniform distribution). We call this query *example query* ($\mathrm{ExQ}_{\mathcal{D}}$ (resp. ExQ or $\mathrm{ExQ}_U$)).

A *distribution-free testing algorithm*, [32], (resp. *testing algorithm*) $\mathcal{A}$ for $C$ is an algorithm that, given as input a distance parameter $\epsilon$ and the above two oracles to a Boolean function $f$,

1. if $f \in C$ then $\mathcal{A}$ outputs "accept" with probability at least $2/3$.
2. if $f$ is $\epsilon$-far from every $g \in C$ with respect to the distribution $\mathcal{D}$ (resp. uniform distribution) then $\mathcal{A}$ outputs "reject" with probability at least $2/3$.

We will also call $\mathcal{A}$ *a tester (or $\epsilon$-tester) for the class* $C$ and an algorithm for *$\epsilon$-testing* $C$.

We say that $\mathcal{A}$ is *one-sided* if it always accepts when $f \in C$; otherwise, it is called *two-sided* algorithm. The *query complexity of* $\mathcal{A}$ is the maximum number of queries $\mathcal{A}$ makes on any Boolean function $f$.

In the learning models, $C$ is a class of representations of Boolean functions rather than a class of Boolean functions. Therefore, we may have two different representations in $C$ that are logically equivalent. In this paper, we assume that this representation is verifiable; that is, given a representation $g$, one can decide in polynomial time on the length of this representation if $g \in C$.

A *distribution-free proper learning algorithm* (resp. proper learning algorithm under the uniform distribution) $\mathcal{A}$ for $C$ is an algorithm that, given as input an accuracy parameter $\epsilon$, a confidence parameter $\delta$ and an access to both $\mathrm{MQ}_f$ for the *target function* $f \in C$ and $\mathrm{ExQ}_{\mathcal{D}}$, with unknown $\mathcal{D}$, (resp. ExQ or $\mathrm{ExQ}_U$), with probability at least $1-\delta$, $\mathcal{A}$ returns $h \in C$ that is $\epsilon$-close to $f$ with respect to $\mathcal{D}$ (resp. with respect to the uniform distribution). This model is also called *proper PAC-learning with membership queries* under any distribution (resp. under the uniform distribution) [3, 48].

# 3 The Distribution-Free Tester

In this section, we sketch the proof of the tester from Section 2.

For a class $C$ of $n$-bit long Boolean functions and a set $Y = \{y_1,\ldots,y_q\}$ we define $C^*(Y)$ the class of all $q$-bit long Boolean functions $f(y_1,\ldots,y_q) = g(y_1,\ldots,y_q,0,\ldots,0)$ where $g \in C$. We define $C(Y) \subseteq C^*(Y)$ the class of $f \in C(Y)$ that depends on all the variables in $Y$. That is, all the variables in $Y$ are influential.

Our main result is

▶ **Theorem 1.** *Let $C$ be a class of $n$-bit long Boolean functions that is symmetric subclass of $k$-Junta and is closed under zero projection. Suppose for every $Y = \{y_1,\ldots,y_q\}$ with $q \le k$, there is a tester $T_Y$ for $q$-bit Boolean function $F$ such that*

1. *$T_Y$ is a polynomial time two-sided distribution-free (resp. uniform-distribution) adaptive $\epsilon$-tester*
2. *If $F \in C(Y)$ then, with probability at least $1-\delta$, $T_Y$ accepts.*

3. *If $F$ is $\epsilon$-far from every function in $C^*(Y)$ w.r.t $\mathcal{D}$ then, with probability at least $1-\delta$, $T_Y$ rejects.*
4. *$T_Y$ makes $M(\epsilon,\delta)$ MQs and $Q(\epsilon,\delta)$ ExQ$_{\mathcal{D}}$ (resp. ExQ$_U$).*

*Then*

1. *There is a polynomial time two-sided distribution-free adaptive algorithm for $\epsilon$-testing $C$ that makes*

$$\tilde{O}\left(M(\epsilon/12,1/24)+kQ(\epsilon/12,1/24)+\frac{k}{\epsilon}\right)$$

*queries.*

2. *(resp. There is a polynomial time two-sided uniform-distribution adaptive algorithm for $\epsilon$-testing $C$ that makes*

$$\tilde{O}\left(M(\epsilon/12,1/24)+Q(\epsilon/12,1/24)+\frac{k}{\epsilon}\right)$$

*queries.)*

Using Goldreich et. al [32], reduction of testing to proper learning we get

▶ **Theorem 2.** *Let $C$ be a class of $n$-bit long Boolean functions that is symmetric subclass of $k$-Junta and is closed under zero projection. Suppose for every $Y=\{y_1,\ldots,y_q\}$ with $q\le k$, there is a polynomial time proper learning algorithm that learns $C^*(Y)$ using $M(\epsilon,\delta)$ MQs and $Q(\epsilon,\delta)$ ExQ$_{\mathcal{D}}$s (resp. ExQ$_U$s). Then*

1. *There is a polynomial time two-sided distribution-free adaptive algorithm for $\epsilon$-testing $C$ that makes*

$$\tilde{O}\left(M(\epsilon/12,1/24)+kQ(\epsilon/12,1/24)+\frac{k}{\epsilon}\right)$$

*queries.*

2. *(resp. There is a polynomial time two-sided uniform-distribution adaptive algorithm for $\epsilon$-testing $C$ that makes*

$$\tilde{O}\left(M(\epsilon/12,1/24)+Q(\epsilon/12,1/24)+\frac{k}{\epsilon}\right)$$

*queries.)*

Before we give the proof sketch, we give some applications.

## 3.1 Some Applications

**$k$-Junta:** For the class $C=k$-Junta, $C^*(Y)=q$-Junta and since $f$ is $q$-bit Boolean function $f\in C^*(Y)$ and the tester $T_Y$ can just returns accept. Then $M=Q=0$ and we get a distribution-free tester for $k$-Junta that makes $\tilde{O}(k/\epsilon)$.

**$k$-Linear:** For the class $C=k$-Linear, the sum (over $F_2$) of at most $k$ variables, we have $C(Y)=\{y_1+y_2+\cdots+y_q\}$. If $f$ is $\epsilon$-far from $C^*(Y)$ then it is $\epsilon$-far from $y_1+y_2+\cdots,y_q$. We can distinguish between $g=y_1+y_2+\cdots+y_q$ and a function that is $\epsilon$-far from $g$ with $O(1/\epsilon)$ ExQ$_{\mathcal{D}}$. Then $M=0$ and $Q=O(1/\epsilon)$. So we get a distribution-free tester for $k$-Linear that makes $O(k/\epsilon)$ queries.

**$k$-Term:** For the class $C = k$-Term, the conjunction of at most $k$ literals, we have $C(Y) = \{z_1 \wedge z_2 \wedge \cdots \wedge z_q \mid z_i \in \{y_i, \overline{y_i}\}\}$. The tester $T_Y$ asks $O(1/\epsilon)$ $\mathrm{ExQ}_{\mathcal{D}}$. If for all the strings $f$ is zero then the tester accept. Otherwise, there is a string $a$ such that $f(a) = 1$. Then $z_i = y_i$ if $a_i = 1$ and $z_i = \overline{y_i}$ if $a_i = 0$. Then as above, it tests if $f$ is $\epsilon$-far from $z_1 \wedge z_2 \wedge \cdots \wedge z_q$ and we get a distribution-free tester for $k$-Term that makes $O(k/\epsilon)$ queries.

**$s$-term monotone $r$-DNF:** For the class $s$-term monotone $r$-DNF (a DNF that contains at most $s$ $r$-Terms), in the full paper, we give an algorithm that properly learns this class in polynomial time and makes $O(s/\epsilon)$ $\mathrm{ExD}_{\mathcal{D}}$ and $O(rs\log(ns))$ MQs. Since the number of influential variables in any $s$-term monotone $r$-DNF is at most $sr$ we have $q \le k = sr$. Therefore the class $C^*(Y)$ $(n = q \le sr)$ can be properly learned using $O(s/\epsilon)$ $\mathrm{ExD}_{\mathcal{D}}$ and $O(rs\log(rs))$ MQs. By Theorem 2 we get

▶ **Theorem 3.** *For any $\epsilon > 0$, there is a polynomial time two-sided distribution-free adaptive algorithm for $\epsilon$-testing $s$-Term Monotone $r$-DNF that makes $\tilde{O}(rs^2/\epsilon)$ queries.*

In the full paper, we also show that

▶ **Theorem 4.** *For any $\epsilon > 0$, there is a polynomial time two-sided distribution-free adaptive algorithm for $\epsilon$-testing $s$-Term Unate[7] $r$-DNF that makes $\tilde{O}(rs^2/\epsilon)$ queries.*

## 3.2 Proof Sketch

**Algorithm 1** A distribution-free tester for subclasses $C$ of $k$-Junta.

---

**Tester**$C(f, \mathcal{D}, \epsilon)$
*Input*: Oracle that accesses a Boolean function $f$ and $\mathcal{D}$.
*Output*: If any one of the procedures rejects
then "reject", otherwise, "accept"

1. $(X, V, I) \leftarrow$**ApproxTarget**$(f, \mathcal{D}, \epsilon, 1/3)$.
2. Define $h(x) = f(x_X \circ 0_{\overline{X}})$.
3. **TestSets**$(h, X, V, I)$.
4. Define $F$
5. **Close**$fF(f, \mathcal{D}, \epsilon, 1/15)$
6. Run $T_Y$ on $F$
7. Accept

---

We now sketch the proof of Theorem 1. See the full proof in [13]. The tester is **Tester**$C$ in Algorithm 1. First, **Tester**$C$ calls the procedure **ApproxTarget**, in Algorithm 2. This procedure executes the first stage of the tester. See Subsection 2.3.1. The reason that here the procedure is more complex is because, for learning classes of unate functions the procedure in Algorithm 1 returns witnesses for $h(x) = f(x_{X_I} \circ 0_{\overline{X_I}})$ and not for $f$ (as in Subsection 2.3.1). So those witnesses also give us the unateness of each variable in the function. This is, for example, how we get the result in Theorem 4.

Throughout the paper we denote $X = X_I$ and $h(x) = f(x_X \circ 0_{\overline{X}})$. In [13] we prove the following. The proof sketch is in Subsection 2.3.1.

---

[7] The function $f$ is unate if there is $a$ such that $f(x + a)$ is monotone.

**Algorithm 2** A procedure that finds influential sets $\{X_i\}_{i\in I}$ of $f$ and a witness $v^{(i)}$ for each influential set $X_i$ for $h := f(x_{X_I} \circ 0_{\overline{X_I}})$ where $X_I = \cup_{i\in I} X_i$. Also, whp, $h$ is $(\epsilon/3)$-close to the target.

---

**ApproxTarget**$(f, \mathcal{D}, \epsilon, c)$
*Input*: Oracle that accesses a Boolean function $f$ and
an oracle that draws $x \in \{0,1\}^n$ according to the distribution $\mathcal{D}$.
*Output*: Either "reject" or $(X_I, V, I)$

**Partition $[n]$ into $r$ sets**
1. Set $r = 2k^2$.
2. Choose uniformly at random a partition $X_1, X_2, \ldots, X_r$ of $[n]$

**Find a close function and influential sets**
3. Set $X_I = \emptyset$; $I = \emptyset$; $V = \emptyset$; $t(X_I) = 0$.
4. Repeat $M = 3k\ln(15k)/\epsilon$ times
5. Choose $u \in \mathcal{D}$.
6. $t(X_I) \leftarrow t(X_I) + 1$
7. If $f(u_{X_I} \circ 0_{\overline{X_I}}) \neq f(u)$ then
8. $W \leftarrow \emptyset$.
9. $(\ell, w^{(\ell)}) \leftarrow$ Binary Search to find a new influential set $X_\ell$
using $u$ and $u_{X_I} \circ 0_{\overline{X_I}}$ and a string
10. $w^{(\ell)} \in \{0,1\}^n$ such that $f(w^{(\ell)}) \neq f(w^{(\ell)}_{\overline{X_\ell}} \circ 0_{X_\ell})$;
11. $X_I \leftarrow X_I \cup X_\ell$; $I \leftarrow I \cup \{\ell\}$.
12. If $|I| > k$ then Output("reject").
13. $W = W \cup \{w^{(\ell)}\}$.
14. Choose $w^{(r)} \in W$.
15. If $f(w^{(r)}_{X_I} \circ 0_{\overline{X_I}}) \neq f(w^{(r)}_{X_I\backslash X_r} \circ 0_{\overline{X_I \cup X_r}})$ then
$W \leftarrow W\backslash\{w^{(r)}\}$; $v^{(r)} \leftarrow w^{(r)}_{X_I} \circ 0_{\overline{X_I}}$; $V \leftarrow V \cup \{v^{(r)}\}$;
If $W \neq \emptyset$ then Goto 14
16. Else If $f(w^{(r)}_{X_I} \circ 0_{\overline{X_I}}) \neq f(w^{(r)})$ then $u \leftarrow w^{(r)}$; Goto 9
17. Else $u \leftarrow w^{(r)}_{\overline{X_r}} \circ 0_{X_r}$; Goto 9
18. $t(X_I) = 0$.
19. If $t(X_I) = c\ln(15k)/\epsilon$ then Output$(X_I, V, I)$.

---

▶ **Lemma 5.** *Consider steps 1-2 in the **ApproxTarget**. If $f$ is a $k$-junta then, with probability at least $2/3$, for each $i \in [r]$, the set $x(X_i) = \{x_j | j \in X_i\}$ contains at most one influential variable of $f$.*

▶ **Lemma 6.** *If **ApproxTarget** does not reject then it outputs $(X, V, I)$ that satisfies*
1. *$q = |I| \le k$.*
2. *For every $\ell \in I$, $v^{(\ell)}_{\overline{X}} = 0$ and $f(v^{(\ell)}) \neq f(0_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$. That is, $v^{(\ell)} \in V$ is a witness of $h(x) = f(x_X \circ 0_{\overline{X}})$ for $X_\ell$ .*
3. *Each $x(X_\ell)$, $\ell \in I$, contains at least one influential variable of $h(x) = f(x_X \circ 0_{\overline{X}})$.*
4. *With probability at least $14/15$*

$$\boldsymbol{Pr}_{u\in\mathcal{D}}[h(x) \neq f(x)] \le \epsilon/3.$$

▶ **Lemma 7.** *If $f$ is $k$-junta and each $x(X_i)$ contains at most one influential variable of $f$ then*

1. ***ApproxTarget*** *outputs $(X,V,I)$.*
2. *Each $x(X_\ell)$, $\ell\in I$, contains exactly one influential variable in $h(x)=f(x_X\circ 0_{\overline{X}})$.*
3. *For every $\ell\in I$, $h(x_{X_\ell}\circ v^{(\ell)}_{\overline{X_\ell}})=f(x_{X_\ell}\circ v^{(\ell)}_{\overline{X_\ell}})$ is a literal.*

▶ **Lemma 8.** *The procedure* ***ApproxTarget*** *makes $O((k\log k)/\epsilon)=\tilde{O}(k/\epsilon)$ queries.*

The tester then defines $h(x)=f(x_X\circ 0_{\overline{X}})$. Now, since $\mathbf{Pr}_{u\in\mathcal{D}}[h(x)\neq f(x)]\le \epsilon/3$ it is enough to distinguish between $h$ is in $C$ and $h$ that is $(2\epsilon/3)$-far from every function in $C$ with respect to $\mathcal{D}$.

The tester then moves to the second stage. See Subsection 2.3.2. First, it calls the procedure **TestSets**. See Algorithm 3 in Appendix A. The procedure tests if every $h(x_{X_\ell}\circ v^{(\ell)}_{\overline{X_\ell}})$ is close to a literal. In the procedure, **UniformJunta**$(g,k,\epsilon,\delta)$ is Blais's uniform-distribution one-sided tester [7] for $k$-Junta. For $k=1$ it tests whether $g$ is a literal or constant function or $\epsilon$-far from any literal and constant function with respect to the uniform distribution.

The following is very easy to prove

▶ **Lemma 9.** *We have*

1. *If $h$ is $k$-junta and each $x(X_i)$ contains at most one influential variable of $f$ then* ***TestSets*** *returns "OK".*
2. *If for some $\ell\in I$, $h(x_{X_\ell}\circ v^{(\ell)}_{\overline{X_\ell}})$ is $(1/30)$-far from every literal with respect to the uniform distribution then, with probability at least $1-(1/15)$,* ***TestSets*** *rejects.*
3. *The procedure* ***TestSets*** *makes $O(k)$ queries.*

This test does not give $\tau(i)$ (the index of the influential variable in $X_i$) but the fact that $h(x_{X_i}\circ v^{(i)}_{\overline{X_i}})$ is close to $x_{\tau(i)}$ or $\overline{x_{\tau(i)}}$ can be used to find the value of $u_{\tau(i)}$ in every assignment $u\in\{0,1\}^n$ without knowing $\tau(i)$. The latter is done, whp, by the procedure **RelVarValues** that uses self-correction. See Algorithm 4 in Appendix A. We have

▶ **Lemma 10.** *We have*

1. *If $h$ is $k$-Junta and each $x(X_i)$ contains at most one influential variable of $h$ then* ***RelVarValues*** *outputs $z$ such that $z_\ell=w_{\tau(\ell)}$ where $h(x_{X_\ell}\circ 0_{\overline{X_\ell}})\in\{x_{\tau(\ell)},\overline{x_{\tau(\ell)}}\}$.*
2. *If for every $\ell\in I$ the function $h(x_{X_\ell}\circ v^{(\ell)}_{\overline{X_\ell}})$ is $(1/30)$-close to a literal in $\{x_{\tau(\ell)},\bar{x}_{\tau(\ell)}\}$ with respect to the uniform distribution, where $\tau(\ell)\in X_\ell$, and* ***RelVarValues*** *does not reject then, with probability at least $1-\delta$, we have: For every $\ell\in I$, $z_\ell=w_{\tau(\ell)}$.*
3. *The procedure* ***RelVarValues*** *makes $O(k\log(k/\delta))$ queries.*

**Proof.** Since $\nu_i(x)=h(x_{X_\ell}\circ v^{(\ell)}_{\overline{X_\ell}})$ is $(1/30)$-close to a literal in $\{x_{\tau(\ell)},\bar{x}_{\tau(\ell)}\}$ we have that for a uniform random string $z$, with probability at least $1/15$ we have $\nu(x+z)+\nu(z)=x_{\tau(\ell)}$. If we repeat this test $O(\log(k/\delta))$ times for every $\ell\in I$, we get a success probability $1-\delta$. Since $|I|\le k$, **RelVarValues** makes $O(k\log(k/\delta))$ queries. ◀

We collect all the above events that happens with high probability in the following

▶ **Assumption 11.** *For the rest of this section we assume*

1. *If $f\in C$ then*
   - *$h(x)=f(x_X\circ 0_{\overline{X}})\in C$.*
   - *Each $x(X_\ell)$, $\ell\in I$ contains exactly one influential variable.*
   - *For every $\ell\in I$, $f(x_{X_\ell}\circ v^{(\ell)}_{\overline{X_\ell}})$ is a literal in $\{x_{\tau(\ell)},\overline{x_{\tau(\ell)}}\}$.*

2. *If $f$ is $\epsilon$-far from every function in $C$ then*
   - *$h(x) = f(x_X \circ 0_{\overline{X}})$ is $(\epsilon/3)$-close to $f$ with respect to $\mathcal{D}$ and therefore $h(x)$ is $(2\epsilon/3)$-far from every function in $C$ with respect to $\mathcal{D}$.*
   - *For every $\ell \in I$, $f(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$ is $(1/30)$-close to a literal in $\{x_{\tau(\ell)}, \bar{x}_{\tau(\ell)}\}$ with respect to the uniform distribution.*
3. *For any $u$, we can get $(u_{\tau(\ell)})_{\ell\in I}$ using $\tilde{O}(k)$ queries.*

We are getting now to the third stage. See Subsection 2.3.3. For a function $f \in C$ we define $\mathrm{Rel}(f)$, the set of all influential variables of $f$. Let $q = |I|$ and $\Gamma := \{\tau(\ell_1), \ldots, \tau(\ell_q)\}$. Notice that, with the above assumption, if $h \in C$ then $\mathrm{Rel}(h) = \Gamma$. We define the class $C(\Gamma)$ (resp. $C^*(\Gamma)$), the set of functions in $C$ with $\mathrm{Rel}(f) = \{x_\gamma \mid \gamma \in \Gamma\}$ (resp. $\mathrm{Rel}(f) \subseteq \{x_\gamma \mid \gamma \in \Gamma\}$). Since $C$ is symmetric and closed under zero projection $C(\Gamma)$ (resp. $C^*(\Gamma)$) is the set of all functions $f(x_{\tau(\ell_1)}, \ldots, x_{\tau(\ell_q)}, 0, 0, \ldots, 0)$ where $f \in C$ and $\mathrm{Rel}(f) = \{x_1, \ldots, x_q\}$ (resp $\mathrm{Rel}(f) \subseteq \{x_1, \ldots, x_q\}$). We recall that, for $Y = \{y_1, \ldots, y_q\}$, $C(Y)$ (resp. $C^*(Y)$) are the set of all functions $f(y_1, \ldots, y_q, 0, 0, \ldots, 0)$ where $f \in C$ and $\mathrm{Rel}(f) = \{x_1, \ldots, x_q\}$ (resp. $\mathrm{Rel}(f) \subseteq \{x_1, \ldots, x_q\}$).

Let $F(y_1, \ldots, y_q) = h(z)(= f(z))$ where $z = (y_1)_{X_{\ell_1}} \circ \cdots \circ (y_q)_{X_{\ell_q}} \circ 0_{\overline{X}}$. That is, for every $\ell_i \in I$, $j \in X_{\ell_i}$ we have $z_j = y_i$ and for every $j \in \overline{X}$ we have $z_j = 0$. Then, by Assumption 11, it is easy to prove that (see Subsection 2.3.3)

▶ **Lemma 12.** *We have*
1. *If $h \in C$ then $F(x_{\tau(\ell_1)}, \ldots, x_{\tau(\ell_q)}) = h$.*
2. *If $h \in C$ then $F(y_1, \ldots, y_q) \in C(Y)$.*
3. *If $h$ is $2\epsilon/3$-far from every function in $C$ with respect to $D$ then either*
   - **a.** *$h$ is $\epsilon/3$-far from $F(x_{\tau(\ell_1)}, \ldots, x_{\tau(\ell_q)})$ with respect to $\mathcal{D}$,*
     *or,*
   - **b.** *$F(x_{\tau(\ell_1)}, \ldots, x_{\tau(\ell_q)})$ is $\epsilon/3$-far from every function in $C$ with respect to $\mathcal{D}$.*

Therefore, we need to do two tests. The first is to distinguish between $h = F$ and $h$ that is $\epsilon/3$-far from $F$ w.r.t $\mathcal{D}$. The second is to distinguish between $F \in C$ and $F$ that is $\epsilon/3$-far from every function in $C$ w.r.t $\mathcal{D}$.

The following result shows that we can query $F(y_1, \ldots, y_q)$ and for every $x \in \{0,1\}^n$ we can extract $x_\Gamma$. So, in particular, we can get a sample according to $\mathcal{D}_\Gamma$ and query $F(x_{\tau(\ell_1)}, \ldots, x_{\tau(\ell_q)})$. The proof is immediate (See Appendix A)

▶ **Lemma 13.** *For the function $F$ we have*
1. *Given $(y_1, \ldots, y_q)$, computing $F(y_1, \ldots, y_q)$ can be done with one query to $f$.*
2. *Given $x \in \{0,1\}^n$ and $\delta$, there is an algorithm that makes $O(k \log(k/\delta))$ queries and, with probability at least $1-\delta$, either discovers that $f \notin C$ and then reject or computes $x_\Gamma = (x_{\tau(\ell_1)}, \ldots, x_{\tau(\ell_q)})$ and $F(x_\Gamma)$.*
3. *There is a polynomial time algorithm that makes $O(k \log(k/\delta))$ queries and with probability at least $1-\delta$ returns a string $u$ in $\{0,1\}^q$ according to the distribution $\mathcal{D}_\Gamma$ and $F(u)$.*
4. *There is a polynomial time algorithm that makes one query and returns a string $u$ in $\{0,1\}^q$ according to the uniform distribution and $F(u)$.*

We now give the procedure **Close**$fF$ that tests whether $h(x) = f(x_X \circ 0_{\overline{X}})$ is $(\epsilon/3)$-far from $F$ with respect to $\mathcal{D}$. See Algorithm 5 and the proof in Appendix A.

▶ **Lemma 14.** *For any $\epsilon$, a constant $\delta$, and $(X, V, I)$ that satisfies Assumption 11, procedure* ***Close****$fF$ makes $O((k/\epsilon)\log(k/\epsilon)) = \tilde{O}(k/\epsilon)$ queries and*
1. *If $f \in C$ then* ***Close****$fF$ returns OK.*
2. *If $h(x)$ is $(\epsilon/3)$-far from $F$ with respect to $\mathcal{D}$ then, with probability at least $1-\delta$,* ***Close****$fF$ rejects.*

The second test is to distinguish between $F \in C$ and $F$ that is $\epsilon/3$-far from $C$ w.r.t the distribution $\mathcal{D}$.

Consider the tester $T_Y$ in Theorem 1. If $h(x) = F(x_{\tau(\ell_1)}, \ldots, x_{\tau(\ell_q)}) \in C$ then, by Assumption 11, since each $x(X_\ell)$, $\ell \in I$, contains exactly one influential variable, $F(x_{\tau(\ell_1)}, \ldots, x_{\tau(\ell_q)}) \in C(\Gamma)$. Therefore $F = F(y_1, \ldots, y_q) \in C(Y)$. If $F(x_{\tau(\ell_1)}, \ldots, x_{\tau(\ell_q)})$ is $\epsilon/3$-far from every function in $C$ w.r.t $\mathcal{D}$, then it is $\epsilon/3$-far from every function in $C^*(\Gamma)$ w.r.t $\mathcal{D}_\Gamma$. Therefore, $F = F(y_1, \ldots, y_q)$ is $\epsilon/3$-far from every function in $C^*(Y)$ w.r.t $\mathcal{D}_\Gamma$. Therefore, $T_Y$ can be used for the second test of $F(x_\Gamma) \in C$ vs. $\epsilon/3$ far from every function in $C$.

Now by, Lemma 13, every MQ to $F$ can be simulated with one MQ to $f$ and every $\text{ExQ}_{\mathcal{D}_\Gamma}$ can be simulated with $O(k \log(k/\delta'))$ $\text{ExQ}_{\mathcal{D}}$ queries and have success probability $1-\delta'$. Since $T_Y$ asks $Q(\epsilon, \delta)$ queries, we need $O(k \log(kQ(\epsilon, \delta)/\delta))$ $\text{ExQ}_{\mathcal{D}}$ to have success $1-\delta$ for all the queries. When $\mathcal{D}$ is uniform then, by Lemma 13, every $\text{ExQ}_{U_\Gamma}$ can be simulated with one query to MQ.

Notice that the success probability in all the procedures above is $1-\delta$ for any constant $\delta$. By choosing $\delta = 1/20$ for each procedure, we get a tester with confidence of at least 2/3. By Lemmas 8, 9 and the above analysis the query complexity of the tester is as stated in Theorem 1.

## 4 The Second Tester: Uniform-Distribution Tester for Classes that are Close to $k$-Junta w.r.t the Uniform Distribution

The second tester in this paper tests classes that are close to $k$-Junta w.r.t the uniform distribution, that is, for every $f \in C$ and every $\epsilon$, there is $k$ such that every function in $C$ is $\epsilon$-close to some function in $k$-Junta.

To understand the intuition behind the second tester, we demonstrate it for testing $s$-term DNF, the class of DNF with at most $s$ terms. This class is close to $(s \log(s/\epsilon))$-Junta. This is because, for every $s$-term DNF and every $\epsilon$, the function $g$ that results by removing the terms of size greater than or equal to $\log(s/\epsilon)$ in $f$ is $\epsilon$-close to $f$ and $g \in (s \log(s/\epsilon))$-Junta.

Let $f$ be an $s$-term DNF. Since, for $\epsilon$-testing, the variables that are influential in $f$ are variables in terms of size $d = O(\log(s/\epsilon))$, there are at most $k = sd = O(s \log(s/\epsilon))$ influential variables in $f$. Suppose we uniformly at random distribute the variables of $f$ into $r = 10k$ bins $X_1, \ldots, X_r$. The influential variables falls into at most $k$ bins. We call those bins influential bins. Terms in $f$ of size greater than $d$, with high probability, $d/2$ of their variables falls into the uninfluential bins.

We try as before to find the influential bins, but this time, we use uniform random strings in our search. This is because, when we use uniform random strings, with high probability, all the large terms in $f$ are equal zero for those strings, and therefore, no uninfluential bin is found by the search procedure.

We find enough influential bins such that if we substitute a random assignment in the variables of the uninfluential bins, w.h.p., we get a function $H$ that is $\epsilon/4$-close to $f$. The next key idea is: as we said before, if we have a large term in $f$, then with high probability, many of its variables fall into the uninfluential bins. So when we substitute a random assignment for the variables in the uninfluential bins, with high probability, the large terms in $f$ vanish in $H$. Therefore, with high probability, $H$ is $\epsilon/4$ close to $f$ and contains small terms (terms of size at most $d = O(\log(s/\epsilon))$. Since $H$ is $s$-term $d$-DNF, it is a function in $sd$-Junta, and we can use the first tester to test $H$.

See more details in Appendix B.

## References


1 Noga Alon, Tali Kaufman, Michael Krivelevich, Simon Litsyn, and Dana Ron. Testing low-degree polynomials over GF(2). In *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques, 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2003 and 7th International Workshop on Randomization and Approximation Techniques in Computer Science, RANDOM 2003, Princeton, NJ, USA, August 24-26, 2003, Proceedings*, pages 188–199, 2003. doi:10.1007/978-3-540-45198-3_17.

2 Noga Alon, Tali Kaufman, Michael Krivelevich, Simon Litsyn, and Dana Ron. Testing reed-muller codes. *IEEE Trans. Information Theory*, 51(11):4032–4039, 2005. doi:10.1109/TIT.2005.856958.

3 Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987.

4 Roksana Baleshzar, Meiram Murzabulatov, Ramesh Krishnan S. Pallavoor, and Sofya Raskhodnikova. Testing unateness of real-valued functions. *CoRR*, abs/1608.07652, 2016. arXiv:1608.07652.

5 Aleksandrs Belovs and Eric Blais. A polynomial lower bound for testing monotonicity. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 1021–1032, 2016. doi:10.1145/2897518.2897567.

6 Arnab Bhattacharyya, Swastik Kopparty, Grant Schoenebeck, Madhu Sudan, and David Zuckerman. Optimal testing of reed-muller codes. In *Property Testing - Current Research and Surveys*, pages 269–275. Springer, 2010. doi:10.1007/978-3-642-16367-8_19.

7 Eric Blais. Testing juntas nearly optimally. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 151–158, 2009. doi:10.1145/1536414.1536437.

8 Eric Blais, Joshua Brody, and Kevin Matulef. Property testing lower bounds via communication complexity. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity, CCC 2011, San Jose, California, USA, June 8-10, 2011*, pages 210–220, 2011. doi:10.1109/CCC.2011.31.

9 Eric Blais and Daniel M. Kane. Tight bounds for testing k-linearity. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, pages 435–446, 2012. doi:10.1007/978-3-642-32512-0_37.

10 Avrim Blum. Learning a function of r relevant variables. In *Computational Learning Theory and Kernel Machines, 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003, Proceedings*, pages 731–733, 2003. doi:10.1007/978-3-540-45167-9_54.

11 Avrim Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artif. Intell.*, 97(1-2):245–271, 1997. doi:10.1016/S0004-3702(97)00063-5.

12 Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993. doi:10.1016/0022-0000(93)90044-W.

13 Nader H. Bshouty. Almost optimal distribution-free junta testing. *CoRR*, abs/1901.00717, 2018.

14 Nader H. Bshouty and Areej Costa. Exact learning of juntas from membership queries. *Theor. Comput. Sci.*, 742:82–97, 2018. doi:10.1016/j.tcs.2017.12.032.

15 Deeparnab Chakrabarty and C. Seshadhri. A $o(n)$ monotonicity tester for boolean functions over the hypercube. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 411–418, 2013. doi:10.1145/2488608.2488660.

16 Deeparnab Chakrabarty and C. Seshadhri. A $\tilde{O}(n)$ non-adaptive tester for unateness. *CoRR*, abs/1608.06980, 2016. arXiv:1608.06980.

17 Sourav Chakraborty, David García-Soriano, and Arie Matsliah. Efficient sample extractors for juntas with applications. In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, pages 545–556, 2011. doi:10.1007/978-3-642-22006-7_46.

18 Xi Chen, Anindya De, Rocco A. Servedio, and Li-Yang Tan. Boolean function monotonicity testing requires (almost) $n^{1/2}$ non-adaptive queries. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 519–528, 2015. doi:10.1145/2746539.2746570.

19 Xi Chen, Rocco A. Servedio, and Li-Yang Tan. New algorithms and lower bounds for monotonicity testing. *CoRR*, abs/1412.5655, 2014. arXiv:1412.5655.

20 Xi Chen, Erik Waingarten, and Jinyu Xie. Beyond talagrand functions: new lower bounds for testing monotonicity and unateness. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 523–536, 2017. doi:10.1145/3055399.3055461.

21 Xi Chen, Erik Waingarten, and Jinyu Xie. Boolean unateness testing with $\widetilde{O}(n^{3/4})$ adaptive queries. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 868–879, 2017. doi:10.1109/FOCS.2017.85.

22 Xi Chen and Jinyu Xie. Tight bounds for the distribution-free testing of monotone conjunctions. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 54–71, 2016. doi:10.1137/1.9781611974331.ch5.

23 Peter Damaschke. Adaptive versus nonadaptive attribute-efficient learning. *Machine Learning*, 41(2):197–215, 2000. doi:10.1023/A:1007616604496.

24 Ilias Diakonikolas, Homin K. Lee, Kevin Matulef, Krzysztof Onak, Ronitt Rubinfeld, Rocco A. Servedio, and Andrew Wan. Testing for concise representations. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 549–558, 2007. doi:10.1109/FOCS.2007.32.

25 Ilias Diakonikolas, Homin K. Lee, Kevin Matulef, Rocco A. Servedio, and Andrew Wan. Efficiently testing sparse $GF(2)$ polynomials. *Algorithmica*, 61(3):580–605, 2011. doi:10.1007/s00453-010-9426-9.

26 Elya Dolev and Dana Ron. Distribution-free testing for monomials with a sublinear number of queries. *Theory of Computing*, 7(1):155–176, 2011. doi:10.4086/toc.2011.v007a011.

27 Eldar Fischer, Guy Kindler, Dana Ron, Shmuel Safra, and Alex Samorodnitsky. Testing juntas. In *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*, pages 103–112, 2002. doi:10.1109/SFCS.2002.1181887.

28 Dana Glasner and Rocco A. Servedio. Distribution-free testing lower bound for basic boolean functions. *Theory of Computing*, 5(1):191–216, 2009. doi:10.4086/toc.2009.v005a010.

29 Oded Goldreich, editor. *Property Testing - Current Research and Surveys*, volume 6390 of *Lecture Notes in Computer Science*. Springer, 2010. doi:10.1007/978-3-642-16367-8.

30 Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017. doi:10.1017/9781108135252.

31 Oded Goldreich, Shafi Goldwasser, Eric Lehman, Dana Ron, and Alex Samorodnitsky. Testing monotonicity. *Combinatorica*, 20(3):301–337, 2000. doi:10.1007/s004930070011.

32 Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998. doi:10.1145/285055.285060.

33 Parikshit Gopalan, Ryan O'Donnell, Rocco A. Servedio, Amir Shpilka, and Karl Wimmer. Testing fourier dimensionality and sparsity. *SIAM J. Comput.*, 40(4):1075–1100, 2011. doi:10.1137/100785429.

34 David Guijarro, Jun Tarui, and Tatsuie Tsukiji. Finding relevant variables in PAC model with membership queries. In *Algorithmic Learning Theory, 10th International Conference, ALT '99, Tokyo, Japan, December 6-8, 1999, Proceedings*, page 313, 1999. doi:10.1007/3-540-46769-6_26.

35 Shirley Halevy and Eyal Kushilevitz. Distribution-free property-testing. *SIAM J. Comput.*, 37(4):1107–1138, 2007. doi:10.1137/050645804.

36 Subhash Khot, Dor Minzer, and Muli Safra. On monotonicity testing and boolean isoperimetric type theorems. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 52–58, 2015. doi:10.1109/FOCS.2015.13.

37 Subhash Khot and Igor Shinkar. An $\widetilde{O}(n)$ queries adaptive tester for unateness. *CoRR*, abs/1608.02451, 2016. arXiv:1608.02451.

38 Richard J. Lipton, Evangelos Markakis, Aranyak Mehta, and Nisheeth K. Vishnoi. On the fourier spectrum of symmetric boolean functions with applications to learning symmetric juntas. In *20th Annual IEEE Conference on Computational Complexity (CCC 2005), 11-15 June 2005, San Jose, CA, USA*, pages 112–119, 2005. doi:10.1109/CCC.2005.19.

39 Zhengyang Liu, Xi Chen, Rocco A. Servedio, Ying Sheng, and Jinyu Xie. Distribution-free junta testing. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 749–759, 2018. doi:10.1145/3188745.3188842.

40 Kevin Matulef, Ryan O'Donnell, Ronitt Rubinfeld, and Rocco A. Servedio. Testing ±1-weight halfspace. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009, Berkeley, CA, USA, August 21-23, 2009. Proceedings*, pages 646–657, 2009. doi:10.1007/978-3-642-03685-9_48.

41 Kevin Matulef, Ryan O'Donnell, Ronitt Rubinfeld, and Rocco A. Servedio. Testing halfspaces. *SIAM J. Comput.*, 39(5):2004–2047, 2010. doi:10.1137/070707890.

42 Elchanan Mossel, Ryan O'Donnell, and Rocco A. Servedio. Learning functions of k relevant variables. *J. Comput. Syst. Sci.*, 69(3):421–434, 2004. doi:10.1016/j.jcss.2004.04.002.

43 Michal Parnas, Dana Ron, and Alex Samorodnitsky. Testing basic boolean formulae. *SIAM J. Discrete Math.*, 16(1):20–46, 2002. URL: http://epubs.siam.org/sam-bin/dbq/article/40744.

44 Dana Ron. Property testing: A learning theory perspective. *Foundations and Trends in Machine Learning*, 1(3):307–402, 2008. doi:10.1561/2200000004.

45 Dana Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205, 2009. doi:10.1561/0400000029.

46 Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996. doi:10.1137/S0097539793255151.

47 Mert Saglam. Near log-convexity of measured heat in (discrete) time and consequences. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 967–978, 2018. doi:10.1109/FOCS.2018.00095.

48 Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984. doi:10.1145/1968.1972.


## A The Procedures of the First Tester

### Proof of Lemma 13.

**Proof.** *1* is immediate since $F(y_1,\ldots,y_q)=f(z)$ where $z=(y_1)_{X_{\ell_1}}\circ\cdots\circ(y_q)_{X_{\ell_q}}\circ 0_{\overline{X}}$. To prove *2*. We run **RelVarValues**$(x,X,V,I,\delta)$. If it rejects then, by Lemma 10, $f\notin C$. Otherwise, by Lemma 10, with probability at least $1-\delta$, the procedure outputs $z$ where for every $\ell$, $z_\ell=x_{\tau(\ell)}$. Then using *1* we compute $F(z)$. Since by Lemma 10, **RelVarValue** makes $O(k\log(k/\delta))$ queries, the result follows. Now 3 and 4 follows immediately from 1 and 2. ◀

**Algorithm 3** A procedure that tests if for all $\ell \in I$, $h(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}})$ is (1/30)-close to some literal with respect to the uniform distribution.

---

**TestSets**$(h, X, V, I)$
*Input*: Oracle that accesses a Boolean function $f$ and $(X, V, I)$.
*Output*: Either "reject" or "OK"

1. For every $\ell \in I$ do
2. If **UniformJunta**$(h(x_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}}), 1, 1/30, 1/15)$="reject"
3. then Output("reject")
4. Choose $b \in \{0,1\}^n$ uniformly at random
5. If $h(b_{X_\ell} \circ v^{(\ell)}_{\overline{X_\ell}}) = h(\overline{b_{X_\ell}} \circ v^{(\ell)}_{\overline{X_\ell}})$ then Output("reject")
6. Return "OK"

---

**Algorithm 4** A procedure that takes as input $(X, V, I)$ and a string $w \in \{0,1\}^n$ and, with probability at least $1-\delta$, returns the values of $w_{\tau(i)}$, $i \in I$, where $h(x_{X_i} \circ v^{(i)}_{\overline{X_i}})$ is (1/30)-close to one of the literals in $\{x_{\tau(i)}, \overline{x_{\tau(i)}}\}$ with respect to the uniform distribution.

---

**RelVarValues**$(w, X, V, I, \delta)$
*Input*: Oracle that accesses a Boolean function $h$, $(X, V, I)$ and $w \in \{0,1\}^n$.
*Output*: Either "reject" or returns for every $\ell \in I$, the value, $z_\ell = w_{\tau(\ell)}$ where $x_{\tau(\ell)}$ is one of the influential variables of $h(x_X \circ 0_{\overline{X}})$ in $x(X_\ell)$

1. For every $\ell \in I$ do
2. For $\xi \in \{0,1\}$ set $Y_{\ell,\xi} = \{j \in X_\ell | w_j = \xi\}$.
3. Set $G_{\ell,0} = G_{\ell,1} = 0$;
4. Repeat $t = \ln(k/\delta)/\ln(4/3)$ times
5. Choose $b \in \{0,1\}^n$ uniformly at random;
6. If $h(b_{Y_{\ell,0}} \circ b_{Y_{\ell,1}} \circ v^{(\ell)}_{\overline{X_\ell}}) \neq h(\overline{b_{Y_{\ell,0}}} \circ b_{Y_{\ell,1}} \circ v^{(\ell)}_{\overline{X_\ell}})$ then $G_{\ell,0} \leftarrow G_{\ell,0} + 1$
7. If $h(b_{Y_{\ell,1}} \circ b_{Y_{\ell,0}} \circ v^{(\ell)}_{\overline{X_\ell}}) \neq h(\overline{b_{Y_{\ell,1}}} \circ b_{Y_{\ell,0}} \circ v^{(\ell)}_{\overline{X_\ell}})$ then $G_{\ell,1} \leftarrow G_{\ell,1} + 1$
8. If $(\{G_{\ell,0}, G_{\ell,1}\} \neq \{0, h\})$ then Output("reject")
9. If $G_{\ell,0} = t$ then $z_\ell \leftarrow 0$ else $z_\ell \leftarrow 1$
10. Output("$\{z_\ell\}_{\ell \in I}$")

---

**Algorithm 5** A procedure that tests whether $h(x)$ is $(\epsilon/3)$-far from $F$ with respect to $\mathcal{D}$.

---

**Close**$fF(f, \mathcal{D}, \epsilon, \delta)$
*Input*: Oracle that accesses a Boolean function $f$ and $\mathcal{D}$.
*Output*: Either "reject" or "OK"

1. Repeat $t = (3/\epsilon)\ln(2/\delta)$ times
2. Choose $u \in \mathcal{D}$.
3. $z \leftarrow$**RelVarValue**$(u, X, V, I, \delta/(2t))$ .
4. If $h(u) \neq F(z)$ then Output("reject")
5. Return "OK".

---

**Proof of Lemma 14.**

**Proof.** If $f \in C$ then, by Lemma 12, $F(u_\Gamma^{(i)}) = h(u)$ for every $i$. By Lemma 10 and Assumption 11, $z^{(i)} = u_\Gamma^{(i)}$ for all $i$, and therefore **Close**$fF$ returns OK.

Suppose now $h(x)$ is $(\epsilon/3)$-far from $F$ with respect to $\mathcal{D}$. By *2* in Lemma 13, **RelVarValue** makes $O(kt\log((kt)/\delta))$ queries and computes $F(u_\Gamma^{(i)})$, $i = 1, \ldots, t$, with failure probability at most $\delta/2$. Then the probability that it fails to reject is at most $(1-\epsilon/3)^t \le \delta/2$. This gives the result.

Therefore, **Close**$fF$ makes $O((k/\epsilon)\log(k/\epsilon))$ queries and satisfies *1* and *2*. ◀

## B Classes that are Close to $k$-Junta

We now give more details. The tester first runs the procedure **Approx**$C$ in Algorithm 6. This procedure is similar to the procedure **ApproxTarget**. It randomly uniformly partitions the variables to $r = 4c^2(c+1)s\log(s/\epsilon)$ disjoint sets $X_1, \ldots, X_r$ and finds influential sets $\{X_i\}_{i\in I}$. Here $c$ is a large constant. To find a new influential set, it chooses two random uniform strings $u, v \in \{0,1\}^n$ and verifies if $f(u_X \circ v_{\overline{X}}) \neq f(u)$ where $X$ is the union of the influential sets that it has found thus far. If $f(u_X \circ v_{\overline{X}}) \neq f(u)$ then the binary search finds a new influential set.

In the binary search for a new influential set, the procedure defines a set $X'$ that is equal to the union of half of the sets in $\{X_i\}_{i\notin I}$. Then either $f(u_{X\cup X'} \circ v_{\overline{X'}}) \neq f(u)$ or $f(u_{X\cup X'} \circ v_{\overline{X'}}) \neq f(u_X \circ v_{\overline{X}})$. Then it recursively does the above until it finds a new influential set $X_\ell$.

It is easy to see that if $f$ is $s$-term DNF then, whp, for all the terms $T$ in $f$ of size at least $c^2\log(s/\epsilon)$, for all the random uniform strings $u, v$ chosen in the algorithm and for all the strings $z$ generated in the binary search, $T(u_X \circ v_{\overline{X}}) = T(u) = T(z) = 0$. Therefore, when $f$ is $s$-term DNF, the procedure, whp, runs as if there are no terms of size greater than $c^2\log(s/\epsilon)$ in $f$. This shows that, whp, each influential set that the procedure finds contains at least one variable that belongs to a term of size at most $c^2\log(s/\epsilon)$ in $f$. Therefore, if $f$ is $s$-term DNF, the procedure, whp, does not generate more than $c^2 s\log(s/\epsilon)$ influential sets. If the procedure finds more than $c^2 s\log(s/\epsilon)$ influential sets then, whp, $f$ is not $s$-term DNF and therefore it rejects.

Let $R$ be the set of all the variables that belong to the terms in $f$ of size at most $c^2\log(s/\epsilon)$. The procedure returns $h(x) = f(x_X \circ w_{\overline{X}})$ for random uniform $w$ where $X$ is the union of the influential sets $X = \cup_{i\in I} X_i$ that is found by the procedure. If $f$ is $s$-term DNF then since $r = 4c^2(c+1)s\log(s/\epsilon)$ and the number of influential sets is at most $c^2 s\log(s/\epsilon)$, whp, at least $(1/2)c\log(s/\epsilon)$ variables in each term of $f$ that contains at least $c\log(s/\epsilon)$ variables not in $R$ falls outside $X$ in the partition of $[n]$. Therefore, for random uniform $w$, whp, terms $T$ in $f$ that contains at least $c\log(s/\epsilon)$ variables not in $R$ satisfies $T(x_X \circ w_{\overline{X}}) = 0$ and therefore, whp, are vanished in $H = f(x_X \circ w_{\overline{X}})$. Thus, whp, $H$ contains all the terms that contains variables in $R$ and at most $cs\log(s/\epsilon)$ variables not in $R$. Therefore, whp, $H$ contains at most $c(c+1)s\log(s/\epsilon)$ influential variables. From this, and using similar arguments as for the procedure **ApproxTarget** in the previous subsection, we prove that, **Approx**$C$ makes at most $\tilde{O}(s/\epsilon)$ queries and

1. If $f$ is $s$-term DNF then, whp, the procedure outputs $X$ and $w$ such that
   - $H = f(x_X \circ w_{\overline{X}})$ is $s$-term DNF[8].
   - The number of influential variables in $H = f(x_X \circ w_{\overline{X}})$ is at most $O(s\log(s/\epsilon))$.

[8] So in this case we need the class to be closed under zero-one projection.

2. If $f$ is $\epsilon$-far from every $s$-term DNF then the procedure either rejects or outputs $X$ and $w$ such that, whp, $H = f(x_X \circ w_{\overline{X}})$ is $(3\epsilon/4)$-far from every $s$-term DNF.

We can now run **Tester**$C$ (with $3\epsilon/4$) on $H$ from the previous subsection for testing $C^*$ where $C^*$ is the set of $s$-term DNF with $k = O(s \log(s/\epsilon))$ influential variables.

**Algorithm 6** A procedure that removes variables from $f$ that only appear in large size terms.

---

**Algorithm Approx**$C(f, \epsilon)$
*Input*: Oracle that accesses a Boolean function $f$ and $\epsilon$
*Output*: Either "$X \subseteq [n], w \in \{0,1\}^n$" or "reject"

**Partition $[n]$ into $r$ sets**

Set $r = 8sc \log(s/\epsilon)$.
Choose uniformly at random a partition $X_1, X_2, \ldots, X_r$ of $[n]$

**Find a close function and influential sets**

Set $X = \emptyset$; $I = \emptyset$; $t(X) = 0$; $k = 3ms$
Repeat $M = 400k \ln(100k)/\epsilon$ times
    Choose $w$ and $u$ uniformly at random from $\{0,1\}^n$;.
    $t(X) \leftarrow t(X) + 1$
    If $f(u_X \circ v_{\overline{X}}) \neq f(u)$ then
        Binary Search to find a new influential set $X_\ell$; $X \leftarrow X \cup X_\ell$; $I \leftarrow I \cup \{\ell\}$.
        If $|I| > k$ then output "reject" and halt.
        $t(X) = 0$.
    If $t(X) = 400 \ln(100k)/\epsilon$ then
            Sample $w$ uniformly at random from $\{0,1\}^n$;
            Output$(X, w, H = f(x_X \circ w_{\overline{X}}))$.

---