

# Testing by Implicit Learning

Rocco Servedio  
Columbia University

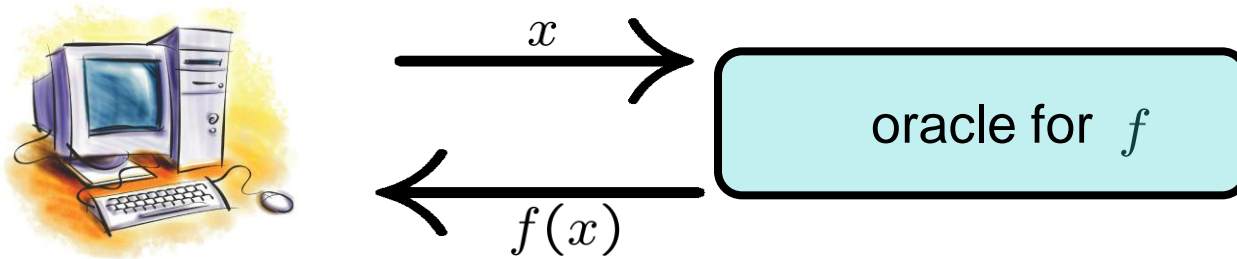
ITCS Property Testing Workshop  
Beijing  
January 2010

# What this talk is about

1. “Testing by Implicit Learning:” method for testing classes of Boolean functions
  - Combines learning theory ingredients with junta testing [FKRSS04]
  - Works for classes of functions that are “well approximated by juntas”
  - Gives new query-efficient testing results for many classes
2. Extension of basic method: **computationally** efficient testing for sparse GF(2) polynomials
3. Different extension of basic method: query-efficient testing for many classes with “low-dimensional Fourier spectrum”

# Basic framework for whole talk

Tester makes black-box queries to **arbitrary**  $f : \{0, 1\}^n \rightarrow \{0, 1\}$



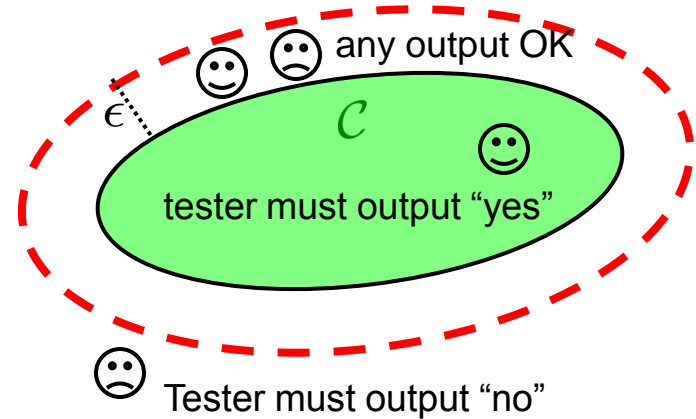
“Property” of Boolean functions  $\leftrightarrow$  class  $\mathcal{C}$  of all functions  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  that have the property

- Examples:
- $\mathcal{C} =$  linear functions over  $\text{GF}(2)$  (parities) [BLR93]
  - $\mathcal{C} =$  degree- $d$   $\text{GF}(2)$  polynomials [AKKSR03]
  - $\mathcal{C} =$  halfspaces [MORS09]
  - $\mathcal{C} =$  monotone functions [DGLRRS99, GGLRS00]
  - $\mathcal{C} =$   $s$ -term DNF formulas [DLMORSW07]
  - etc.

# Basic framework continued

Tester must output

- “yes” whp if  $f \in \mathcal{C}$
- “no” whp if  $f$  is  $\epsilon$ -far from every  $g \in \mathcal{C}$



Here & throughout talk, measure distance w.r.t. **uniform distribution** over domain  $\{0, 1\}^n$

Main concern: **information-theoretic** # of queries required (but computationally efficient algorithms are nice too...)

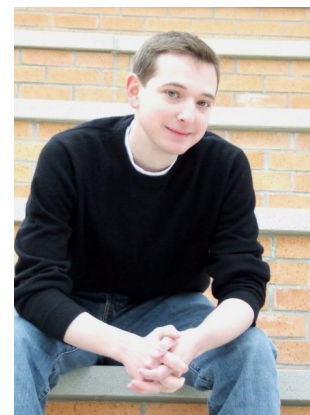
# 1. Testing by Implicit Learning



Ilias Diakonikolas



Homin Lee



Kevin Matulef



Krzysztof Onak



Ronitt Rubinfeld



Andrew Wan

# Learning a concept class $\mathcal{C}$

“PAC learning concept class  $\mathcal{C}$  under the uniform distribution”

**Setup:** Learner is given a sample of labeled examples

- **Target function**  $f \in \mathcal{C}$  is unknown to learner
- Each example  $x$  in sample is independent, uniform over  $\{0, 1\}^n$

$x$	$f(x)$
001001001001	1
100111011001	0
101011011101	0
011100010110	1
.....	...
011100110110	0

**Goal:** For every  $f \in \mathcal{C}$ , with probability  $\geq \frac{9}{10}$ , learner should output a hypothesis  $h : \{0, 1\}^n \rightarrow \{0, 1\}$  such that  $\Pr[f(x) \neq h(x)] \leq \epsilon$ .

# Proper Learning via “Occam’s Razor”

A learning algorithm for  $\mathcal{C}$  is **proper** if it outputs hypotheses from  $\mathcal{C}$ .

**Generic proper learning algorithm for any finite class  $\mathcal{C}$  :**

- Draw  $m = \frac{1}{\epsilon} \ln(10|\mathcal{C}|)$  labeled examples
- Output any  $h \in \mathcal{C}$  that is **consistent** with all  $m$  examples.

finding such an  $h$  may be  
**computationally hard...**

**Why it works:**

- Suppose true error rate of  $h' \in \mathcal{C}$  is  $> \epsilon$ .
- Then  $\Pr[h' \text{ consistent with } m \text{ random examples}] < (1 - \epsilon)^m \leq \frac{1}{10|\mathcal{C}|}$

So  $\Pr[\text{any “bad” } h \in \mathcal{C} \text{ is output}]$  is at most  $|\mathcal{C}| \cdot \frac{1}{10|\mathcal{C}|} = \frac{1}{10}$ .

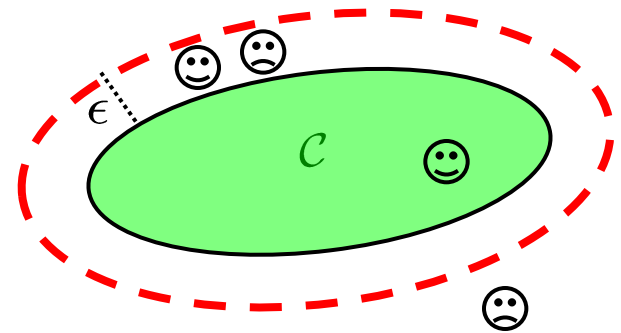
# Testing via proper learning

[GGR98]:  $\mathcal{C}$  **properly learnable**  $\rightarrow$   $\mathcal{C}$  **testable** with same # queries.

- Run learning algorithm to learn to error  $\frac{\epsilon}{2}$ ; hypothesis obtained is  $h$
- Draw  $O(\frac{1}{\epsilon})$  random examples, use them to check that  $\text{error}(h) \leq \frac{3\epsilon}{4}$

Why it works:

- $f \in \mathcal{C} \rightarrow h$  is  $\frac{\epsilon}{2}$ -accurate  
 $\rightarrow$  estimated error of  $h$  is  $< \frac{3\epsilon}{4}$
- $f$  is  $\epsilon$ -far from  $\mathcal{C} \rightarrow$  estimated error of  $h$  is  $> \frac{3\epsilon}{4}$  since  $h \in \mathcal{C}$  is  $\epsilon$ -far from  $f$





# It's not that easy

[GGR98]:  $\mathcal{C}$  properly learnable  $\rightarrow$   $\mathcal{C}$  testable with same # queries.

Occam's Razor: **Every**  $\mathcal{C}$  is properly learnable

Great! But...

- Occam's Razor proper learner uses many examples:

$$m = \frac{1}{\epsilon} \ln(10|\mathcal{C}|)$$

- Even for very simple classes of functions over  $n$  variables (like literals/dictators), **any** learning algorithm must use  $\Omega(\log n)$  examples...

and in testing, we want query complexity **independent of**  $n$ .

# Some known property testing results

Class of functions over $\{0, 1\}^n$	# of queries
parity functions [BLR93]	$O(1/\epsilon)$
deg- $d$ $GF(2)$ polynomials [AKK+03]	$O(4^d/\epsilon)$
literals [PRS02]	$O(1/\epsilon)$
conjunctions [PRS02]	$O(1/\epsilon)$
$J$ -juntas [FKRSS04, B08, B09]	$\tilde{O}(J^2/\epsilon), \tilde{O}(J/\epsilon)$
$s$ -term monotone DNF [PRS02]	$\tilde{O}(s^2/\epsilon)$
halfspaces [MORS09]	$\text{poly}(1/\epsilon)$

**Question:** [PRS02] what about **non-monotone**  $s$ -term DNF?

# “Testing via Implicit Learning” results

**Theorem:** [DLMORSW07]

The class of  $s$ -term DNF over  $\{0, 1\}^n$  is testable with  $\text{poly}(s/\epsilon)$  queries.

$s$ -leaf decision trees

size- $s$  branching programs

size- $s$  Boolean formulas (AND/OR/NOT gates)

size- $s$  Boolean circuits (AND/OR/NOT gates)

$s$ -sparse polynomials over  $\text{GF}(2)$

$s$ -sparse algebraic circuits over  $\text{GF}(2)$

$s$ -sparse algebraic computation trees over  $\text{GF}(2)$

All results follow from basic “testing by implicit learning” approach.

# Testing by Implicit Learning

## When it works:

Approach works for any class  $\mathcal{C}$  that is “well-approximated by juntas:”

for every  $f \in \mathcal{C}$  there exists an  $f' \in \mathcal{C}$  such that

- $f'$  is close to  $f$
- $f'$  depends on few variables  
( $\tau$ -close  $\rightarrow$  depends on  $\log(1/\tau)$  vars)

## How it works:

Running example:

testing whether  $f : \{0, 1\}^n \rightarrow \{0, 1\}$   
is an  $s$ -term DNF  
versus  
 $\epsilon$ -far from every  $s$ -term DNF

# $s$ -term DNF are well-approximated by juntas

Let  $f$  be any  $s$ -term DNF formula:

$$f = \dots (x_2 \bar{x}_4) \dots \vee \dots \vee (x_3 x_7) \vee (\bar{x}_1 \bar{x}_4) \vee (x_5 x_6)$$

There is an  $\tau$ -approximating DNF  $f'$  with  $\leq s$  terms where

- each term contains  $\leq \log(s/\tau)$  variables, and hence
- $f'$  is a  $s \log(s/\tau)$ -junta

- Any term with  $> \log(s/\tau)$  variables is satisfied with probability  $< \frac{\tau}{s}$
- Delete all (at most  $s$ ) such terms from  $f$  to get  $f'$

# Occam + approximation + [GGR98]?

Take  $\tau \ll \epsilon$ : makes  $f'$  so close to  $f$  that if we only use uniform random examples, we can pretend  $f' = f$ .

Given any  $s$ -term DNF  $f$ , there is a  $\tau$ -approximating DNF  $f'$  with  $\leq s$  terms where each term contains  $\leq \log(s/\tau)$  variables.

So can try to learn

$$\mathcal{C}' = \{\text{all } s\text{-term } \log(s/\tau)\text{-DNF over } \{0, 1\}^n\}$$

Now Occam requires

$$\frac{\ln |\mathcal{C}'|}{\epsilon} \approx \frac{\ln(n^{s \log(s/\tau)})}{\epsilon} = \frac{s \log(s/\tau) \log n}{\epsilon}$$

examples...better, but still depends on  $n$ .

# Getting rid of $n$ ?

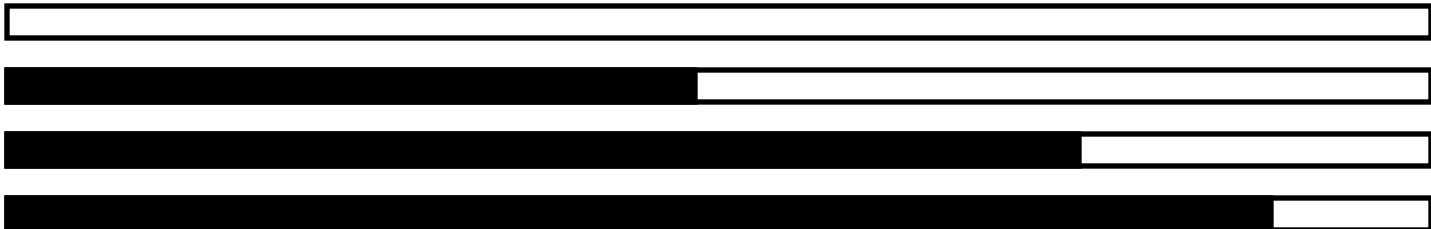
Each approximating DNF  $f'$  depends only on  $s \log(s/\tau)$  variables.

Suppose we knew those variables.

Then we'd have  $\mathcal{C}'' = \{\text{all } s\text{-term } \log(s/\tau)\text{-DNF over } \{0, 1\}^{s \log(s/\tau)}\}$

so Occam would need only  $\frac{\ln |\mathcal{C}''|}{\epsilon} \approx \frac{s^2 \log(s/\tau)}{\epsilon}$  examples,  
independent of  $n$ !

But, can't explicitly identify even **one** variable with  $o(\log n)$  examples...



# The fix: implicit learning

High-level idea: Learn the “structure” of  $f'$   
without explicitly identifying the relevant variables

Algorithm tries to find an approximator

$$h = (x_{\sigma(1)}x_{\sigma(2)}) \vee (\bar{x}_{\sigma(2)}x_{\sigma(3)}x_{\sigma(4)}) \vee (x_{\sigma(3)}x_{\sigma(5)}) \vee \dots$$

where  $\sigma : [s \log(s/\tau)] \rightarrow [n]$  is an **unknown** mapping.



# Implicit learning

How can we learn “structure” of  $f'$  without knowing relevant variables?

Need to generate  $\text{poly}(s/\epsilon)$  many correctly labeled random examples of  $f'$ :

the  $s$ -term  $\log(s/\tau)$ -DNF approximator for  $f$

$z$	$f'(z)$
001001001001	1
100111011001	0
101011011101	0
011100010110	1
.....	...
011100110110	0


each string  $z$  is  $\leq s \log(s/\tau)$  bits

Then can do Occam (brute-force search for consistent DNF).

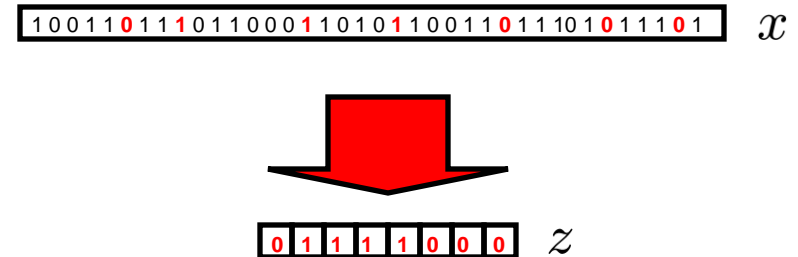
# Implicit learning cont

- Vars of  $z$  are the variables that have **high influence** in  $f'$ : flipping the bit is likely to change value of  $f'$
- setting of other variables almost always doesn't matter

$z$	$f'(z)$
001001001001	1
100111011001	0
.....	...
011100110110	0


 $\leq s \log(s/\tau)$  bits

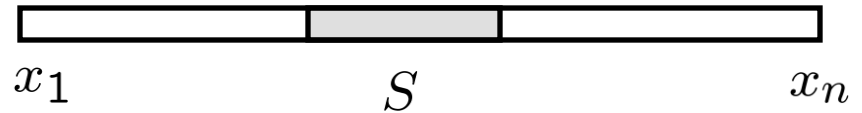
Given random  $n$ -bit labeled example  $(x, f(x))$ , want to construct  $s \log(s/\tau)$ -bit example  $(z, f'(z))$



Do this using techniques of [FKRSS02] "Testing Juntas"

# Use independence test of [FKRSS02]

Let  $S$  be a subset of variables.

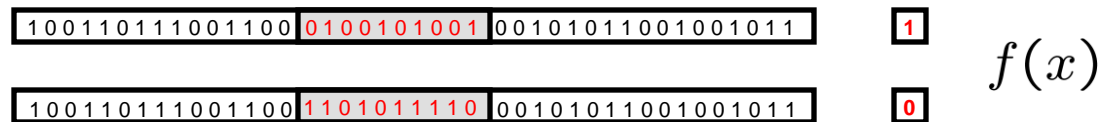


“Independence test” [FKRSS02]:

- Fix a random assignment to variables not in  $S$



- Draw **two independent settings** of variables in  $S$ , query  $f$  on these 2 points

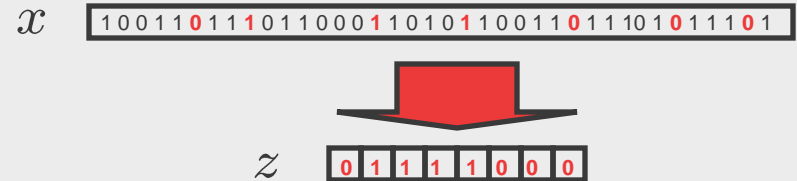


Intuition:

- if  $S$  has all low-influence variables, see same value for  $f$  whp
- if  $S$  has a high-influence variable, see different value sometimes

# Constructing our examples

Given random  $n$ -bit labeled example  $(x, f(x))$ , want to construct  $s \log(s/\tau)$ -bit example  $(z, f'(z))$



Follow [FKRSS02]:

- Randomly partition variables into blocks; run independence test on each block



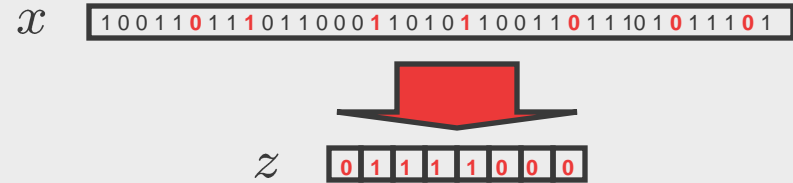
- Can determine which blocks have high-influence variables



- Each block should have **at most one** high-influence variable (birthday paradox)

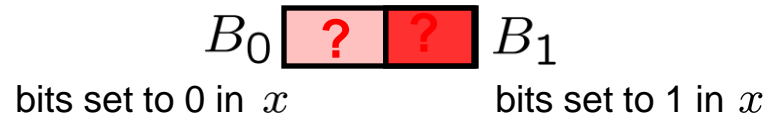
# Constructing our examples

Given random  $n$ -bit labeled example  $(x, f(x))$ , want to construct  $s \log(s/\tau)$ -bit example  $(z, f'(z))$



We know which blocks have high-influence variables; need to determine how the high-influence variable in the block is set.

Consider a fixed high-influence block  $B$ . String  $x$  partitions  $B$  into  $B_0 \cup B_1$ :



Run independence test on each of  $B_0, B_1$  to see which one has the high-influence variable.

Repeat for all high-influence blocks to get all bits of  $z$ .

# The test and its completeness

Suppose  $f$  is an  $s$ -term DNF.

- Then  $f$  is close to  $s$ -term  $\log(s/\tau)$ -DNF  $f'$
- Test constructs sample of random  $s \log(s/\tau)$ -bit examples that are all correctly labeled according to  $f'$  whp.
- Test checks all  $s$ -term  $\log(s/\tau)$ -DNFs over  $\{0, 1\}^{s \log(s/\tau)}$  for consistency with sample. Outputs “yes” if any consistent DNF found, otherwise outputs “no.”
- $f'$  is consistent, so **test outputs “yes”**

# Sketch of soundness of test

Suppose  $f$  is **far from** every  $s$ -term DNF

- If  $f$  far from every  $s \log(s/\tau)$ -junta, [FKRSS02] catches it (too many high-influence blocks)
- So suppose  $f$  close to an  $s \log(s/\tau)$ -junta  $f'$  and algorithm constructs sample of  $s \log(s/\tau)$ -bit examples labeled by  $f'$ .
- Then whp there exists no  $s$ -term  $\log(s/\tau)$ -DNF consistent with sample, so **test outputs “no”**
  - If there were such a DNF  $g$  consistent with sample, would have

$$\begin{array}{c}
 \text{Occam} \qquad \qquad \text{by assumption} \\
 \underbrace{\hspace{10em}} \\
 g \quad \text{close to} \quad f' \quad \text{close to} \quad f \\
 \underbrace{\hspace{10em}} \\
 \text{so } g \text{ close to } f \quad \text{-- contradiction}
 \end{array}$$

END OF  
SKETCH

# Testing by Implicit Learning

Can use Tbil approach for any class  $\mathcal{C}$  with the following property:

- $\forall f \in \mathcal{C} \exists f' \in \mathcal{C}$  such that
- $f'$  is an  $\epsilon$ -approximator for  $f$
  - $f'$  depends on few variables

Many classes have this property...



s-term DNF

size-s Boolean formulas (AND/OR/NOT gates)

size-s Boolean circuits (AND/OR/NOT gates)

s-sparse polynomials over GF(2) ( $\oplus$  of ANDs)

s-leaf decision trees

size-s branching programs

s-sparse algebraic circuits over GF(2)

s-sparse algebraic computation trees over GF(2)

All these classes are testable with  $\text{poly}(s/\epsilon)$  queries.



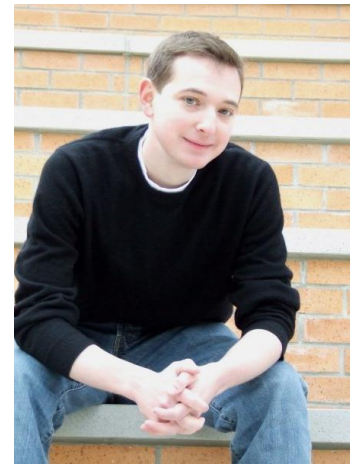
## 2. Extension of basic method: computationally efficient testing for s-sparse GF(2) polynomials



Ilias Diakonikolas



Homin Lee



Kevin Matulef



Andrew Wan

# Pros and cons of basic “Testing by Implicit Learning”

- ☺ Approach works for many classes of functions
- ☺ “One-size-fits-all” algorithm
- ☹ Computationally inefficient:
  - Running time is  $2^{\omega(s)}, 1/\epsilon^{\omega(1)}$
  - Running time bottleneck: brute-force search as proper learning algorithm (try all  $s$ -term DNFs over  $s \log(s/\tau)$  variables)

Can we use smarter learning algorithms to get (computationally) more efficient testers?

Yes (in at least one case)...but some complications ensue.

# $GF(2)$ Polynomials

$GF(2)$  polynomial  $p : \{0,1\}^n \rightarrow \{0,1\}$

*parity* (sum) of *monotone conjunctions* (monomials)

e.g.  $p(\mathbf{x}) = 1 + \mathbf{x}_1 \cdot \mathbf{x}_3 + \mathbf{x}_2 \cdot \mathbf{x}_3 + \mathbf{x}_1 \cdot \mathbf{x}_4 \cdot \mathbf{x}_5 \cdot \mathbf{x}_6 \cdot \mathbf{x}_8 + \mathbf{x}_2 \cdot \mathbf{x}_7 \cdot \mathbf{x}_8 \cdot \mathbf{x}_9 \cdot \mathbf{x}_{10}$

- “*sparsity*” = number of monomials
- Polynomial is *s-sparse* if it has at most  $s$  monomials

$\mathcal{C}$  = class of  $s$ -sparse  $GF(2)$  polynomials over  $\{0,1\}^n$

Well-studied from various perspectives:

- [BS90, FS92, **SS96**, B97, BM02] (learning)
- [K89, GKS90, RB9, EK89, KL93, LVW93] (approximation, approximate counting)

# Main Result

**Theorem :** There is a testing algorithm for the class  $\mathcal{C} = \{\text{all } s\text{-sparse } GF(2) \text{ polynomials}\}$  that uses  $\text{poly}(s, 1/\epsilon)$  queries and *runs in time*  $n \cdot \text{poly}(s, 1/\epsilon)$ .

(Matches  $\sqrt{s}$  lower bound on # queries required [DLMORSW07])

## Ingredients for main result:

- “*Testing by Implicit Learning*” framework [DLMORSW07]
- Efficient Proper Learning Algorithm [Schapire-Sellie’96]
- New Structural Theorem:
  - “*s-sparse polynomials simplify nicely under certain - carefully chosen - random restrictions*”

# Efficient Proper Learning of $s$ -sparse $GF(2)$ Polynomials

**Theorem** [SS'96]: There is a uniform distribution query algorithm that properly PAC learns  $s$ -sparse polynomials over  $\{0, 1\}^\ell$  to accuracy  $\epsilon$  in time (and query complexity)  $\text{poly}(\ell, s, 1/\epsilon)$ .

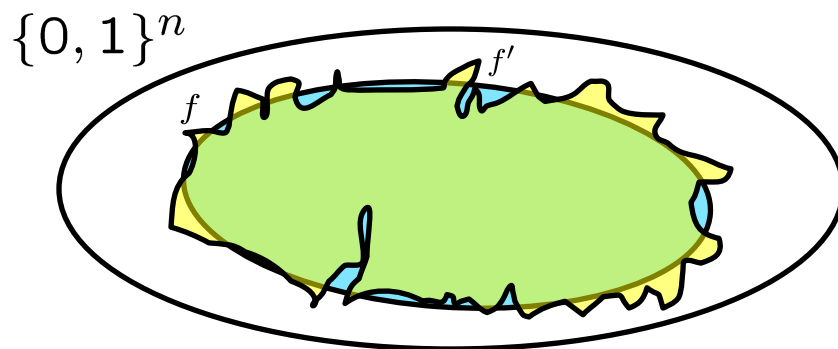
Great! But...

Learning Algorithm uses **black-box** (membership) **queries**.

Black-box queries to  $f'$  (an  $s \log(s/\tau)$ -junta which approximates  $f$ ) are not so easy to simulate as uniform random examples...

# Random Examples vs Queries

Let  $f: \{0,1\}^n \rightarrow \{0,1\}$  be a sparse polynomial and  $f'$  be *some*  $\tau$ -approximator to  $f$ .



- Assume  $1/\tau \gg$  number of **uniform random examples** required for Occam learning  $f'$ . Then, random examples for  $f$  are ok to use as random examples for  $f'$ .
- But a **black-box query** algorithm may make few queries, yet cluster those queries on the few inputs where  $f$  and  $f'$  disagree. No longer good enough for  $f'$  to be a high-accuracy approximator of  $f$ .

# The challenge

Let  $f: \{0,1\}^n \rightarrow \{0,1\}$  be an  $s$ -sparse polynomial. (presumably not a junta)

- Need to simulate **black-box queries** to a  $\tau$ -approximator  $f'$  which is an  $s$ -sparse polynomial and an  $s \log(s/\tau)$ -junta. (Must get answer of  $f'$  right on **every input!**)
- To make this work, need to define the approximating function  $f'$  carefully.

Roughly speaking,  $f'$  is obtained as follows:

1. Randomly partition variables in  $r = \text{poly}(s/\tau)$  subsets.
2.  $f' =$  restriction obtained from  $f$  by setting all variables in “low influence” subsets to 0.

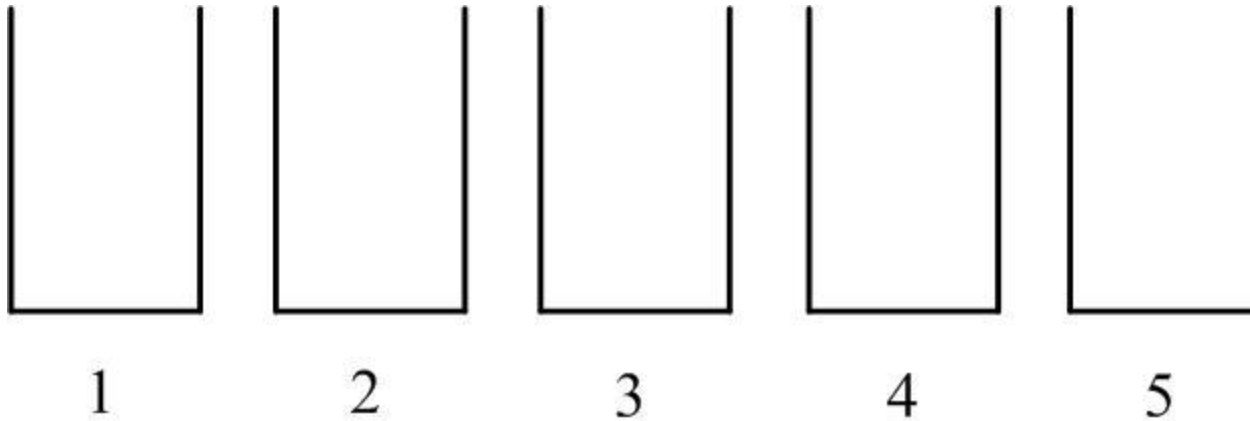
*Intuition:* Kill all “long” monomials.

# Illustration (I)

Suppose

$$p(\mathbf{x}) = 1 + \mathbf{x}_1 \cdot \mathbf{x}_3 + \mathbf{x}_2 \cdot \mathbf{x}_3 + \mathbf{x}_1 \cdot \mathbf{x}_4 \cdot \mathbf{x}_5 \cdot \mathbf{x}_6 \cdot \mathbf{x}_8 + \mathbf{x}_2 \cdot \mathbf{x}_7 \cdot \mathbf{x}_8 \cdot \mathbf{x}_9 \cdot \mathbf{x}_{10}$$

and  $r = 5$ .



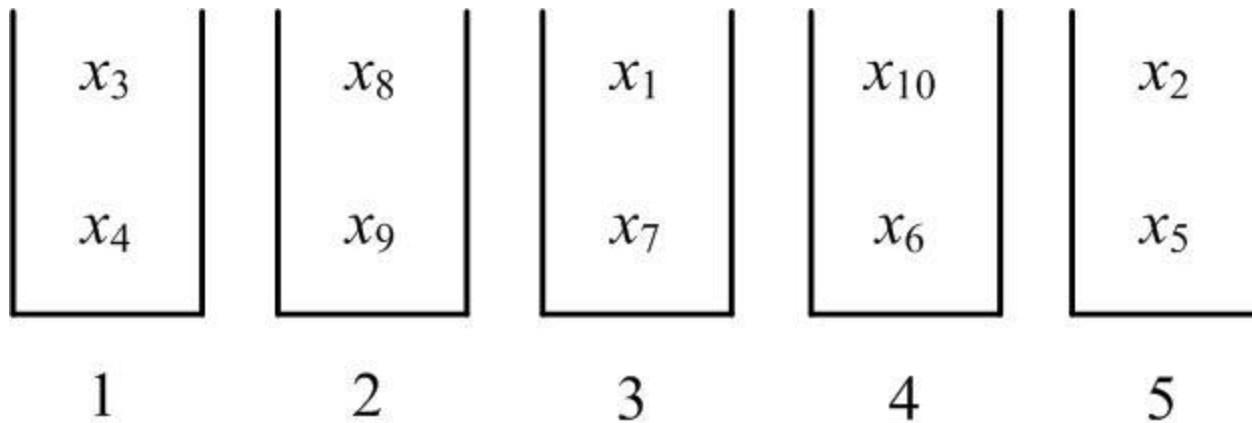


# Illustration (II)

Suppose

$$p(\mathbf{x}) = 1 + \mathbf{x}_1 \cdot \mathbf{x}_3 + \mathbf{x}_2 \cdot \mathbf{x}_3 + \mathbf{x}_1 \cdot \mathbf{x}_4 \cdot \mathbf{x}_5 \cdot \mathbf{x}_6 \cdot \mathbf{x}_8 + \mathbf{x}_2 \cdot \mathbf{x}_7 \cdot \mathbf{x}_8 \cdot \mathbf{x}_9 \cdot \mathbf{x}_{10}$$

and  $r = 5$ .



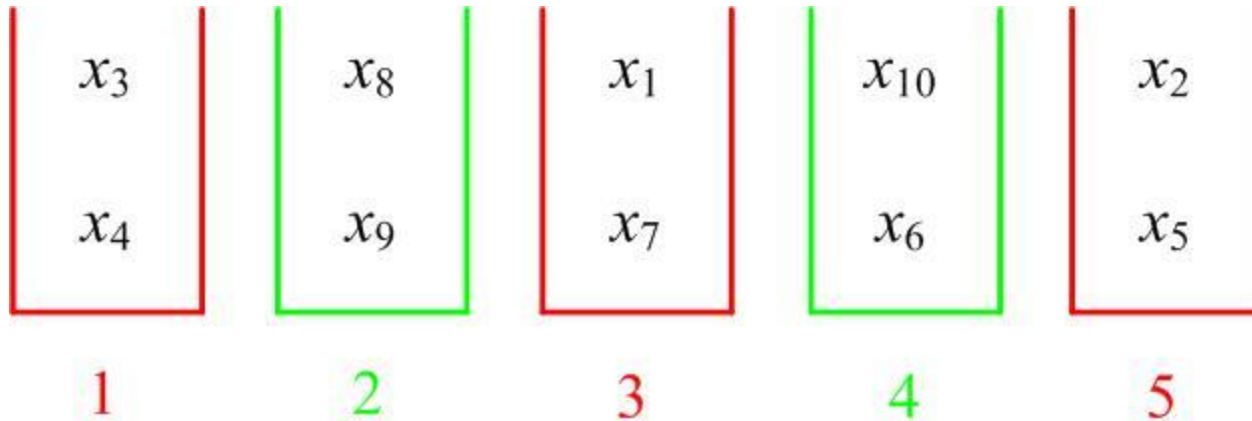
Randomly partition the variables into  $r$  subsets:

# Illustration (III)

Suppose

$$p(\mathbf{x}) = 1 + \mathbf{x}_1 \cdot \mathbf{x}_3 + \mathbf{x}_2 \cdot \mathbf{x}_3 + \mathbf{x}_1 \cdot \mathbf{x}_4 \cdot \mathbf{x}_5 \cdot \mathbf{x}_6 \cdot \mathbf{x}_8 + \mathbf{x}_2 \cdot \mathbf{x}_7 \cdot \mathbf{x}_8 \cdot \mathbf{x}_9 \cdot \mathbf{x}_{10}$$

and  $r = 5$ .



Check “influence” of each subset (independence test):

green subsets: low influence

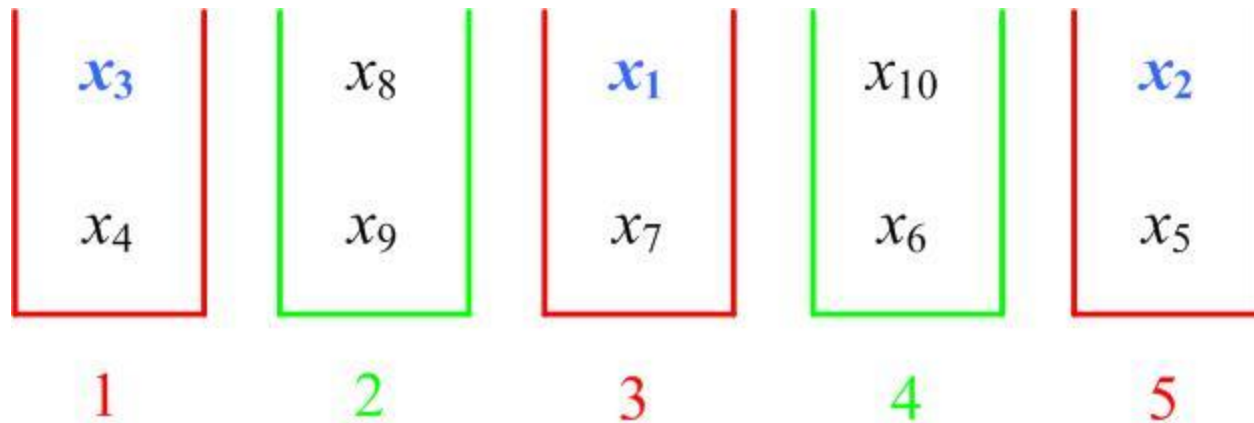
red subsets: high influence

# Illustration (IV)

Suppose

$$p(\mathbf{x}) = 1 + \mathbf{x}_1 \cdot \mathbf{x}_3 + \mathbf{x}_2 \cdot \mathbf{x}_3 + \mathbf{x}_1 \cdot \mathbf{x}_4 \cdot \mathbf{x}_5 \cdot \mathbf{x}_6 \cdot \mathbf{x}_8 + \mathbf{x}_2 \cdot \mathbf{x}_7 \cdot \mathbf{x}_8 \cdot \mathbf{x}_9 \cdot \mathbf{x}_{10}$$

and  $r = 5$ .



Zero out variables in low-influence subsets:

$$\begin{aligned} p'(\mathbf{x}) &= p(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, 0, \mathbf{x}_7, 0, 0, 0) \\ &= 1 + \mathbf{x}_1 \cdot \mathbf{x}_3 + \mathbf{x}_2 \cdot \mathbf{x}_3 \end{aligned}$$

# Testing Algorithm for $s$ -sparse GF(2) Polynomials

1. Partition the coordinates into  $[n]$  into  $r = \text{poly}(s / \tau)$  random subsets.
2. Distinguish subsets that contain a “high-influence” variable from subsets that do not.
3. Consider restriction  $f'$  obtained from  $f$  by “zeroing out” all the variables in “low-influence” subsets.
4. “Implicitly” run [SS'96] using a “simulated” black-box oracle for the function  $f'$ .
  - Do implicit learning, construct query strings similar to [DLMORSW07]

# Why it Works - Structural Theorem

**Theorem:** Let  $p: \{0,1\}^n \rightarrow \{0,1\}$  be an  $s$ -sparse polynomial.

Consider a random partition of the set  $[n]$  into  $r = \text{poly}(s/\tau)$  many subsets. Let  $p'$  be the restriction obtained by fixing all variables in “low-influence” subsets to 0. Then, whp the following are true:

1.  $p'$  has at most one of its relevant variables in each surviving subset;
2.  $p'$  consists of the monomials in  $p$  that consist entirely of high-influence variables;
3.  $p'$  is an  $s \log(s/\tau)$ -junta that is  $\tau$ -close to  $p$ .

Enables us to simulate black-box queries & thus “implicitly” run [SS96] learning algorithm.

### 3. Different extension: testing classes of functions with low Fourier dimension



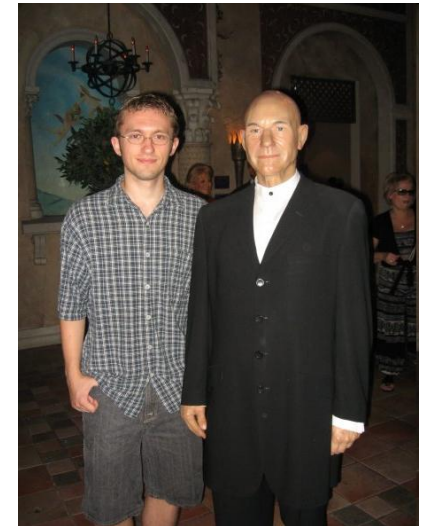
Parikshit Gopalan



Ryan O'Donnell



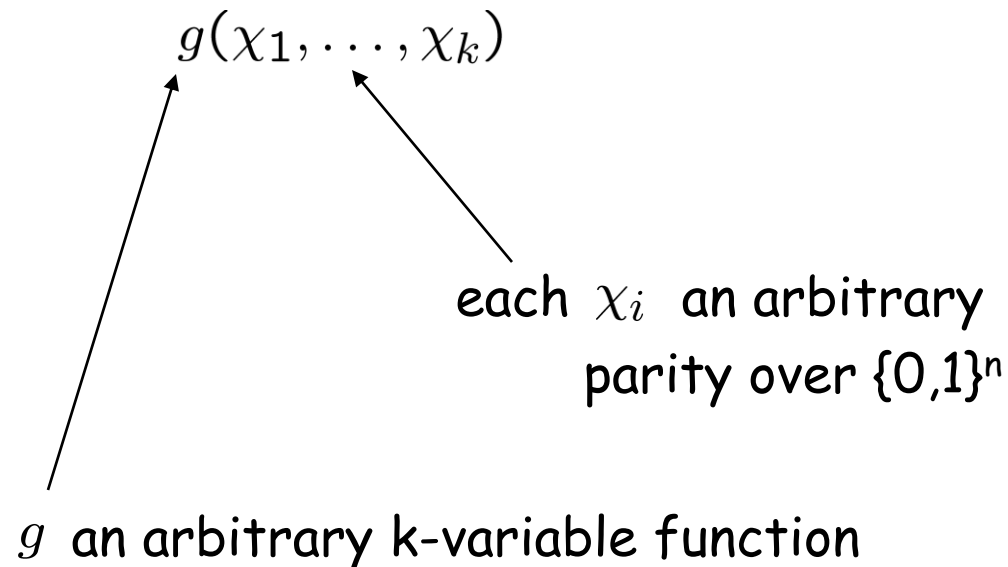
Amir Shpilka



Karl Wimmer (L)

# Fourier dimension

The class  $\mathcal{C} = \{\text{all Boolean functions with Fourier dimension } k\}$  is  
 $\mathcal{C} = \{\text{all "k-juntas-of-parities"}\}$ , i.e. functions of the form



# Example of Fourier dimension

$$f(x) = \text{sign}(7x_1x_2 + 9x_1x_3x_4 + \cdots - 21x_4x_5x_6)$$

view inputs as  $\pm 1$

$k$  monomials

$f$  has Fourier dimension at most  $k$

So  $k$ -sparse PTFs have Fourier dimension  $k$ .



## Third main result:

# Testing subclasses of $k$ -dimensional functions

$$\dim(f) = \min k \text{ s.t. } f \text{ is a } k\text{-junta of parities}$$

Let  $\mathcal{C}'$  be a subclass of all  $k$ -variable Boolean functions

The subclass of  $k$ -dimensional functions induced by  $\mathcal{C}'$  is the class

$$\mathcal{C} = \{g(x_1, \dots, x_k) : g \in \mathcal{C}'\} \text{ of functions over } \mathbb{F}_2^n$$

i.e.  $\mathcal{C} = \text{"}\mathcal{C}'\text{-of-parity"}$

Ex:  $\mathcal{C}' =$  all linear threshold functions over  $k$  variables

$\mathcal{C} =$  all  $k$ -sparse polynomial threshold functions over  $\{-1, 1\}^n$

# Main result:

## Testing subclasses of $k$ -dimensional functions

Let  $C'$  be a subclass of all  $k$ -variable Boolean functions

The subclass of  $k$ -dimensional functions induced by  $C'$  is the class

$$C = \text{"}C'\text{-of-parity"}$$

### Theorem:

Let  $C$  be any induced subclass of  $k$ -dimensional functions.

There is a nonadaptive  $\text{poly}(2^k/\epsilon)$ -query algorithm for testing  $C$ .

So can test, e.g.,

- $C = k$ -sparse PTFs over  $\{-1,1\}^n$  (take  $C' = \text{LTFs}$ )
- $C = \text{size-}k \text{ decision trees with parity nodes}$  (take  $C' = \text{decision trees of size } k$ )
- etc

# Very sketchy sketch

Parities play the role of “influential variables” in basic TbIL approach.

Can't explicitly identify them, but can determine the value they take in any given example.

Construct an “implicit truth table”:

$\chi_1$	$\chi_2$	$\cdots$	$\chi_{k-1}$	$\chi_k$	$f$
0	0	...	0	0	1
0	0	...	0	1	0
0	0	...	1	0	0
0	0	...	1	1	0
...	...	...	...	...	...
1	1	...	1	0	1
1	1	...	1	1	1

Check consistency with some  $g \in \mathcal{C}'$ .

(So “learning” is trivial - get whole truth table.)

# Summary

1. “Testing by Implicit Learning:” method for testing classes of Boolean functions
  - Combines learning theory ingredients with junta testing [FKRSS04]
  - Works for classes of functions that are “well approximated by juntas”
  - Gives new query-efficient testing results for many classes
2. Extension of basic method: **computationally** efficient testing for sparse GF(2) polynomials
  - Uses sophisticated black-box-query algorithm from learning theory [SS96]
  - Careful construction of junta approximator
3. Different extension of basic method: query-efficient testing for many classes with “low-dimensional Fourier spectrum”
  - Parities play role of variables in junta
  - Really brute-force learning (build whole truth table!)

# 1. Future Work: Testing by Implicit Learning

- What are the right **lower bounds** for testing classes like  $s$ -term DNF, size-  $s$  decision trees, formulas, circuits...?
  - Can get  $\approx \Omega(\log s)$  following [CG04], but feels like right bound is  $\Omega(\text{poly}(s))$ ?
- Any way to extend TbIL to distribution-independent testing framework?
  - Obvious problem: may be hard to approximate by juntas...

## 2. Future Work:

# Computationally Efficient TbIL for $s$ -sparse $GF(2)$ polynomials

- Get  $\text{poly}(s/\epsilon)$  runtime for more classes!
  - Computationally efficient proper learning algorithms would yield these, but these seem hard to come by
  - First step: extend  $GF(2)$  results to any finite field  $\mathbf{F}$ . Want runtime to be  $\text{poly}(s, |\mathbf{F}|, 1/\epsilon)$ .
    - Main bottleneck: need fast proper learning algorithms that only evaluate the poly being learned over inputs from  $\mathbf{F}^n$
    - [B97] requires  $s^{|\mathbf{F}|} \log |\mathbf{F}| \log n$  runtime to learn  $s$ -sparse polynomials over  $\mathbf{F}^n$

# 3. Future Work: Testing subclasses of $k$ -dimensional functions

- Any way to do it with “real learning” (implicitly generate just a sample rather than whole truth table)?
  - Might lead to  $\text{poly}(k)$  query bounds in some cases, rather than current  $2^k$  bound...

Thank

You

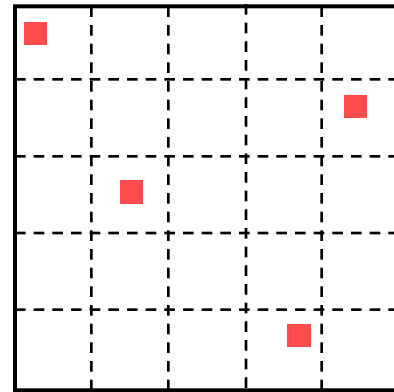


# Very sketchy sketch (1)

Extends Fourier sparsity test with “Testing by Implicit Learning” ideas.

Fourier sparsity test:  $\text{poly}(s/\varepsilon)$ -query test for  $C = s$ -sparse functions.

Works by hashing Fourier coefficients  
of  $f$  into random cosets:



If  $f$  is  $s$ -sparse and we hash into  $O(s^2)$  buckets, w.h.p. every coset gets at most one nonzero Fourier coefficient (birthday paradox).

So can “isolate” all the parities...