Testing by Implicit Learning

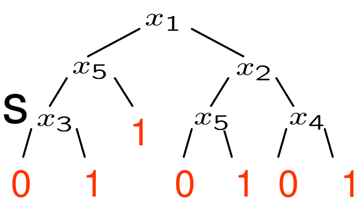
Ilias Diakonikolas
Columbia University

March 2009

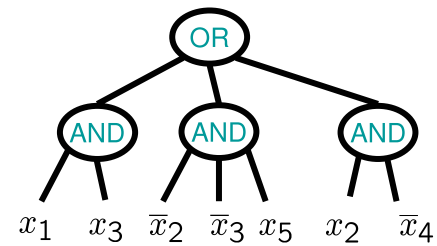
What this talk is about

Recent results on **testing** some natural types of functions:

– Decision trees



– DNF formulas, more general Boolean formulas



– Sparse polynomials over finite fields $3x^2y - 5xz + 4y^{20}z^{15}$

Exploiting **learning** techniques to do **testing**.

Based on joint works with:

Homin Lee (Columbia)

Rocco Servedio (Columbia)

Andrew Wan (Columbia)

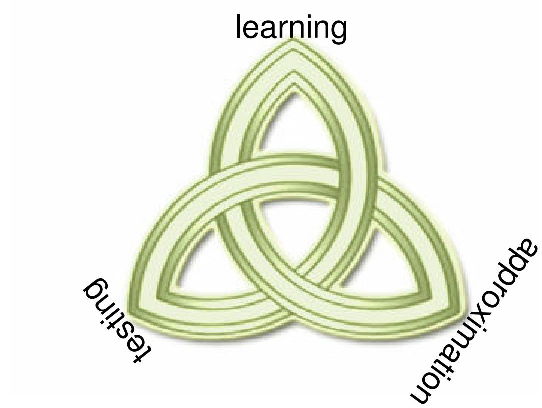
Kevin Matulef (MIT)

Krzysztof Onak (MIT)

Ronitt Rubinfeld (MIT and TAU)

Take-home message

there are close connections between these topics



Seems natural…

- Goal of learning is to produce an approximation to the function
- Goal of testing is to determine whether function "approximately" has some property

Overview of talk

0. Basics of learning, testing, approximation

1. A technique: "testing by implicit learning"

a little learning theory
+
a little approximation
+
testing ideas from [FKRSS04]

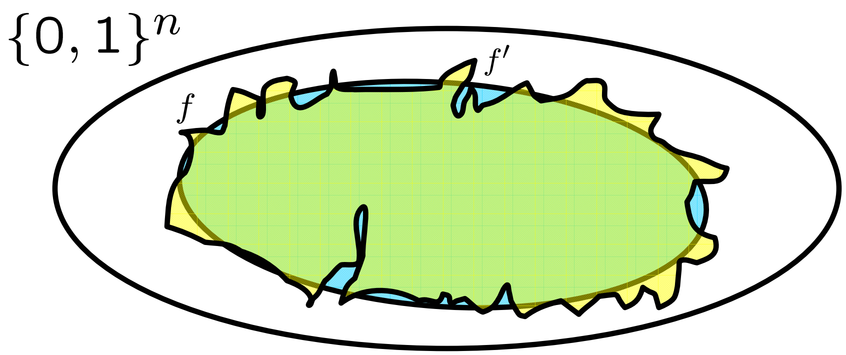
$\Big\} \longrightarrow$ new testing results for many classes of functions [DLMORSW07]

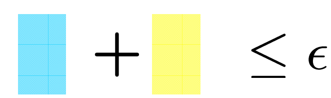
2. A specific class of functions: sparse polynomials

I. Approximation

Given a function $f : \{0,1\}^n \to \{0,1\}$, goal is to obtain a "simpler" function $f' : \{0,1\}^n \to \{0,1\}$ such that

$$\Pr[f(x) \neq f'(x)] \leq \epsilon.$$





- Measure distance between functions under uniform distribution.

Approximation – example

Let f be any s-term DNF formula:

$$f = (x_2x_4x_6x_8\ldots x_n) \ \vee \ (x_2\overline{x}_4) \ \vee \ (x_1x_2x_3x_4\ldots x_{\sqrt{n}}) \ \vee \ (x_3x_7) \ \vee \ (\overline{x}_1\overline{x}_4) \ \vee \ (x_5x_6)$$

There is an ϵ-approximating DNF f' with $\leq s$ terms where each term contains $\leq \log(s/\epsilon)$ variables [V88]

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

Approximation – example

Let f be any s-term DNF formula:

$$
\begin{aligned}
f &= (x_2x_4x_6x_8\ldots x_n) \vee (x_2\overline{x}_4) \vee (x_1x_2x_3x_4\ldots x_{\sqrt{n}}) \vee (x_3x_7) \vee (\overline{x}_1\overline{x}_4) \vee (x_5x_6)\\
f' &= (x_2\overline{x}_4) \vee (x_3x_7) \vee (\overline{x}_1\overline{x}_4) \vee (x_5x_6)
\end{aligned}
$$

There is an ϵ-approximating DNF f' with $\leq s$ terms where each term contains $\leq \log(s/\epsilon)$ variables [V88]

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

II. 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 $h : \{0,1\}^n \to \{0,1\}$ such that $\Pr[f(x) \neq h(x)] \leq \epsilon$.

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' consistent with m random examples] $\leq (1-\epsilon)^m \leq \frac{1}{10|\mathcal{C}|}$

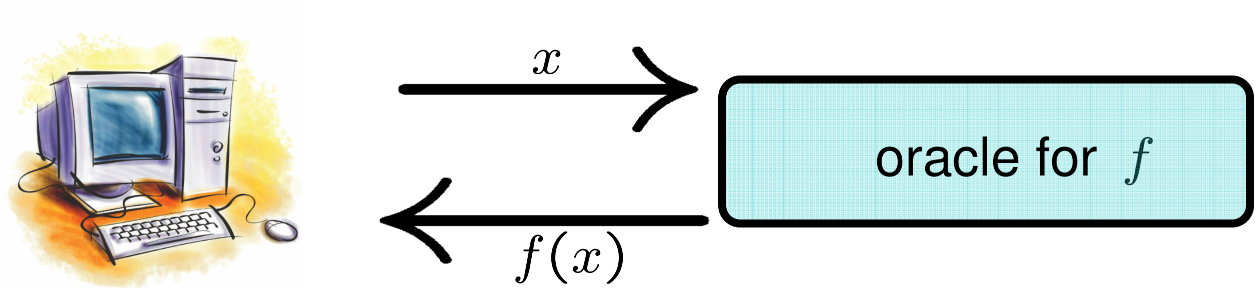
error $> \epsilon$

So Pr[any "bad" $h \in \mathcal{C}$ is output] < $|\mathcal{C}| \cdot \frac{1}{10|\mathcal{C}|}$.

III. Property testing

Goal: infer “global” property of function via few “local” inspections

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

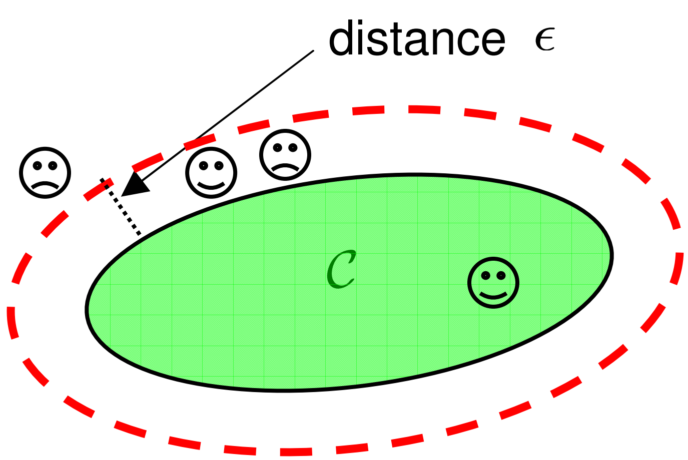


Tester must output

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

Usual focus: **information-theoretic**

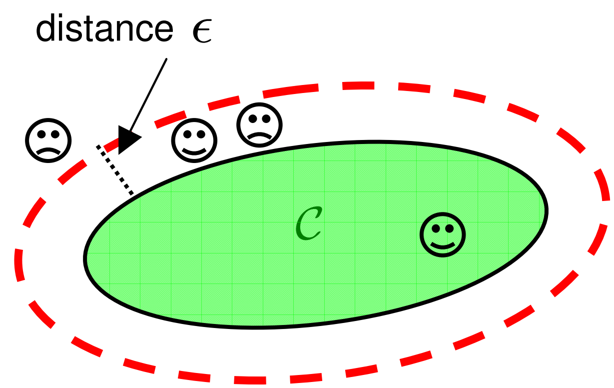
\# queries required



Testing via proper learning

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

- Run algorithm to learn to high accuracy; hypothesis obtained is h
- Draw random examples, use them to estimate $\mathtt{error}(h)$ to high accuracy



Why it works:

- $f \in \mathcal{C}$ → estimated error of h is small
- f is far from $\mathcal{C}$ → estimated error of h is large since $h \in \mathcal{C}$ is far from f

Great! But...

Even for very simple classes of functions over n variables (like literals), 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]	$\tilde{O}(J^2/\epsilon)$
s-term monotone DNF [PRS02]	$\tilde{O}(s^2/\epsilon)$

Different algorithm tailored for each of these classes.

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

New property testing results

Theorem: [DLMORSW07]

The class of s-term DNF over $\{0,1\}^n$ is testable with poly(s/ϵ) 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 GF(2)

s-sparse algebraic circuits over GF(2)

s-sparse algebraic computation trees over GF(2)

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

Overview of talk

0. Some basics ✔

1. A technique: "testing by implicit learning"

a little learning theory
+
a little approximation
+
testing ideas from [FKRSS04]

$\longrightarrow$ new testing results for many classes of functions [DLMORSW07]

Running example:

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

Straight-up testing by learning?

Recall

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

- Occam's Razor: can properly learn any $\mathcal{C}$ from $\approx \frac{1}{\epsilon} \ln |\mathcal{C}|$ examples

But for $\mathcal{C}$ = {all s-term DNF over $\{0,1\}^n$}, this is $O(ns/\epsilon)$ examples…

We want a $\text{poly}(s/\epsilon)$ -query algorithm.

Approximation to the rescue?

We also have approximation:

Take $\tau \ll \epsilon$: makes f' so close to f that we can pretend $f' = f$.

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

So can try to learn

$\mathcal{C}'$ = {all s-term $\log(s/\tau)$-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 $\mathcal{C}''$ = {all s-term $\log(s/\tau)$-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

$$\boxed{h = (x_{\sigma(1)} x_{\sigma(2)}) \vee (\overline{x}_{\sigma(2)} x_{\sigma(3)} x_{\sigma(4)}) \vee \cdots}$$

where $\sigma : [s \log(s/\tau)] \to [n]$ 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

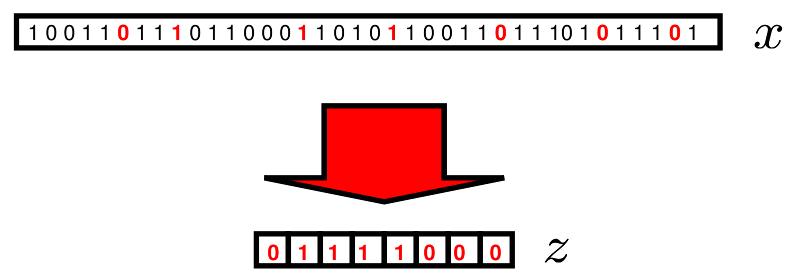
Vars of z are the variables that have **high influence** in f : flipping the bit is likely to change value of f

- setting of other variables almost always doesn't matter

z	$f'(z)$
001001001001	1
100111011001	0
$\cdots\cdots\cdots\cdots\cdots$	$\cdots$
011100110110	0

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

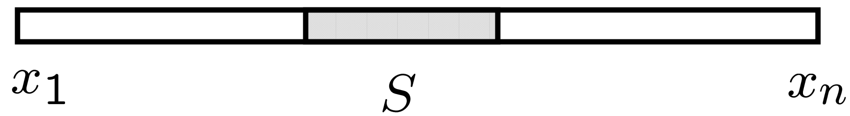
Given random n-bit labeled example $(x, f(x))$, want to construct $s\log(s/\tau)$-bit example $(z, f'(z))$



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

Use independence test of [FKRSS02]

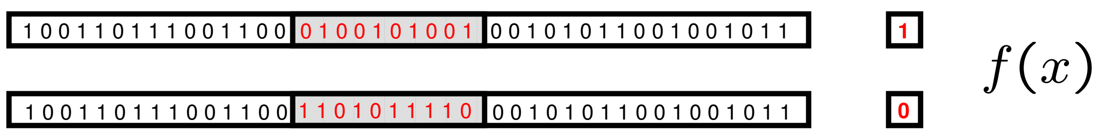
Let S be a subset of variables.



"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

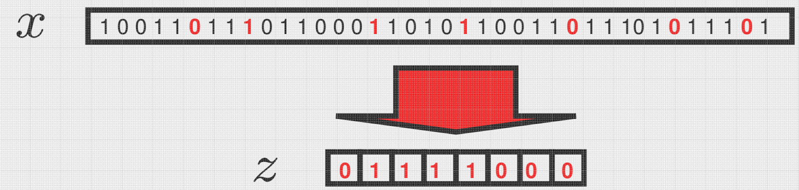


Intuition:

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

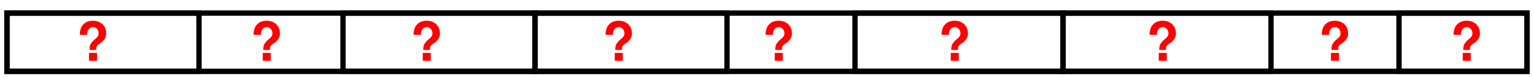
Constructing our examples

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

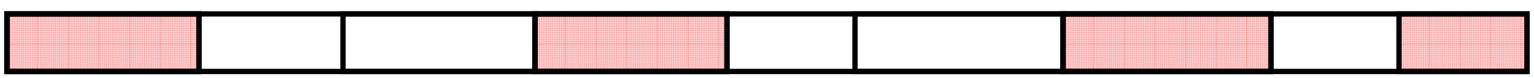


Follow [FKRSS02]:

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



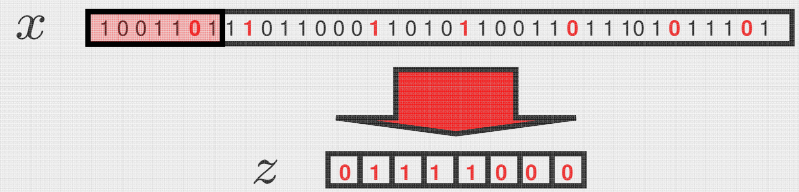
– Can determine which blocks have high-influence variables



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

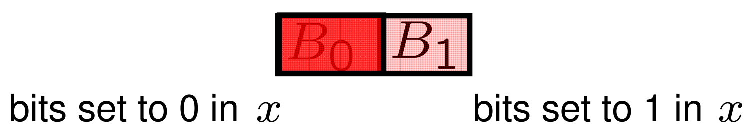
Constructing our examples

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.

Sketch of completeness of overall test

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.
  - 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 variables)

- 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

$$g \; \underbrace{\overbrace{\text{close to}}^{\text{Occam}} \; f' \; \overbrace{\text{close to}}^{\text{by assumption}}}_{\text{so } g \text{ close to } f} \; f \qquad \text{-- contradiction}$$

END OF SKETCH

Testing by Implicit Learning

Can use this 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 ϵ-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 poly(s/ϵ) queries.

Road map

0. Some basics

1. A technique: “testing by implicit learning”

a little learning theory
+
a little approximation
+
testing ideas from [FKRSS04]

new testing results for many classes of functions [DLMORSW07]

2. A specific class of functions: sparse polynomials

Testing *Efficiently*

testing

learning

approximation

Polynomials

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

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

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

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

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

Extensively studied from various perspectives:

[BS'90, FS'92, SS'96, Bsh'97, BM'02] (learning)

[Kar'89, GKS'90, RB'91; EK'89, KL'93, LVW'93] (approximation)

Efficiently Testing sparse poly's

Theorem **[DLMSW08]:** There is an ϵ-*testing* algorithm for the property of being an s-sparse $GF(2)$ polynomial that uses poly $(s, 1/\epsilon)$ queries and ***runs in time*** $n\cdot$ **poly** $(s, 1/\epsilon)$.

Ingredients:

- Main Technique:

  "*Testing by Implicit Learning*" Framework [DLM+07]

- 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 $\{0,1\}^r$ in time (and query complexity) poly $(r, s, 1/\epsilon)$.

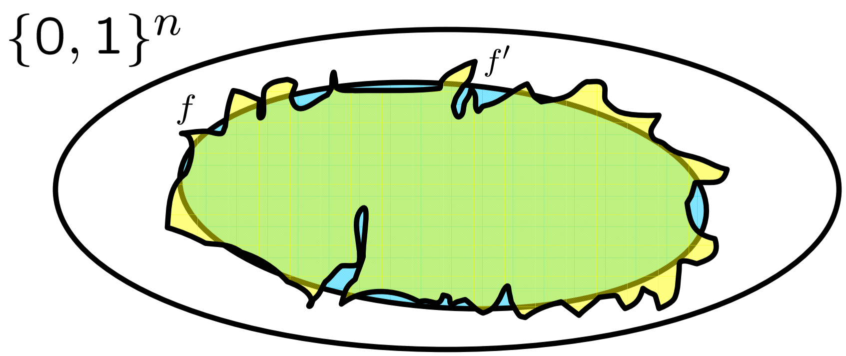
Great! But…

Learning Algorithm uses ***black-box queries***.

Cannot "implicitly simulate" the learning algorithm using random examples as before..

Random Examples vs Queries

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



- Assume $1/\tau \gg$ number of random examples required for Occam learning f'. Then, random examples for f are *ok*.
- A black-box algorithm may cluster its queries on the few inputs where f and f' disagree.

Difficulties

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

- Need to simulate queries to $f\,'$ having query access to f. And need to do this in a *query efficient* way.

- To make this work, need appropriate definition of the approximating function $f\,'$.

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

1. Randomly partition variables in r = poly $(s\,/\tau)$ subsets.
2. $f\,'$ = restriction obtained from f by setting all variables on "low influence" subsets to 0.

Intuition: "kill" all "long" monomials.

Illustration (I)

Suppose

$$p\,(x) = 1 + x_1 \cdot x_3 + x_2 \cdot x_3 + x_1 \cdot x_4 \cdot x_5 \cdot x_6 \cdot x_8 + x_2 \cdot x_7 \cdot x_8 \cdot x_9 \cdot x_{10}$$

and $r = 5$.

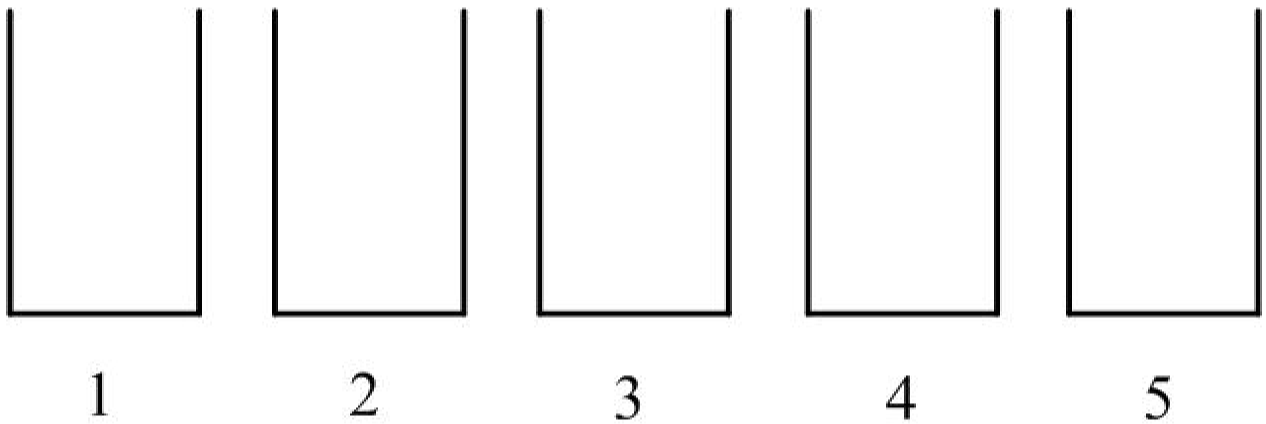


Illustration (II)

Suppose

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

and $r = 5$.

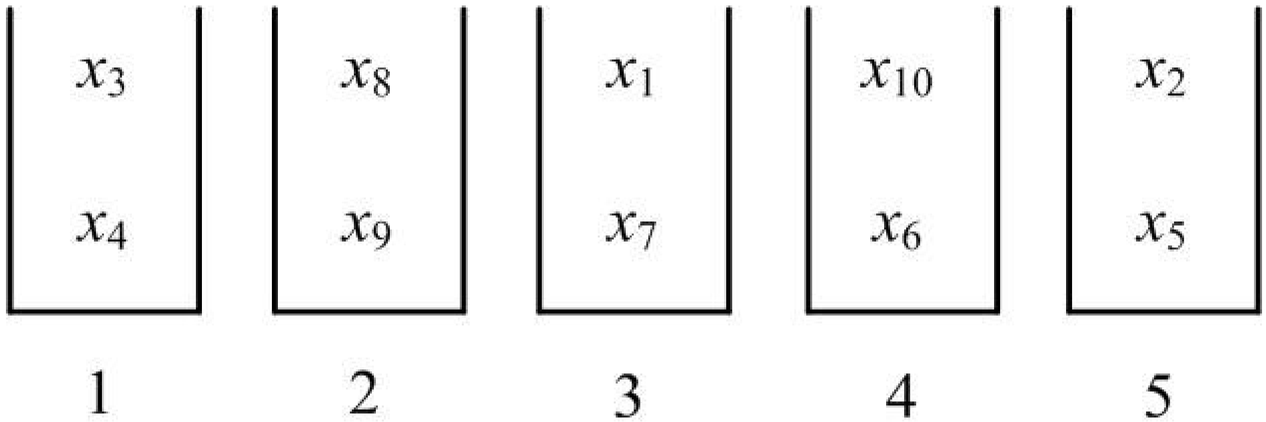


Illustration (III)

Suppose

$$p\,(x) = 1 + x_1 \cdot x_3 + x_2 \cdot x_3 + x_1 \cdot x_4 \cdot x_5 \cdot x_6 \cdot x_8 + x_2 \cdot x_7 \cdot x_8 \cdot x_9 \cdot x_{10}$$

and $r = 5$.

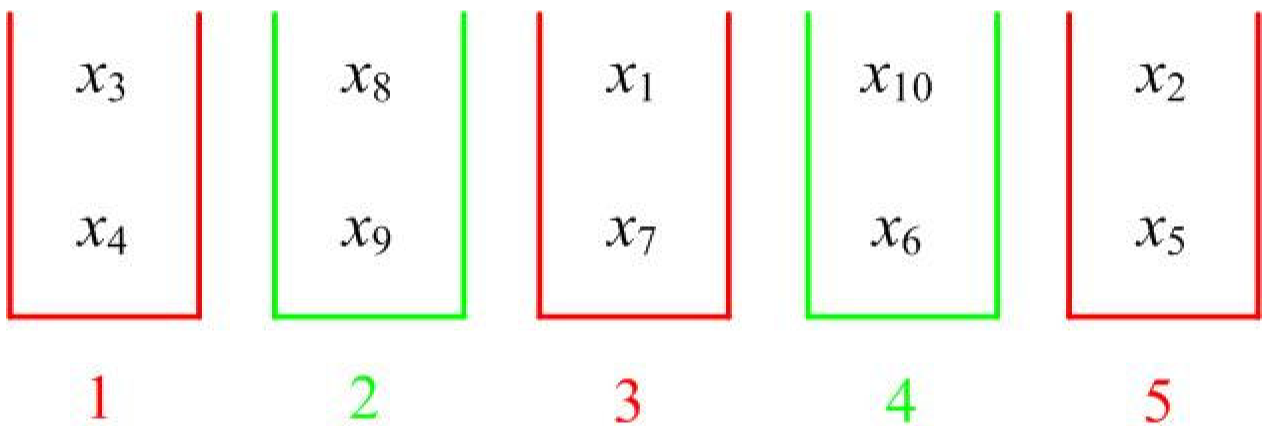
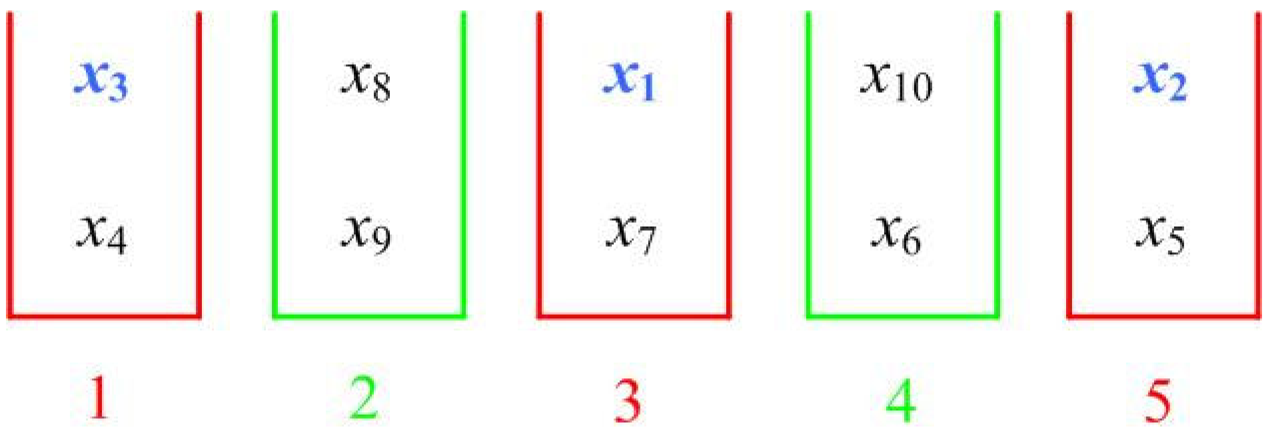


Illustration (IV)

Suppose

$$p\,(x) = 1 + x_1 \cdot x_3 + x_2 \cdot x_3 + x_1 \cdot x_4 \cdot x_5 \cdot x_6 \cdot x_8 + x_2 \cdot x_7 \cdot x_8 \cdot x_9 \cdot x_{10}$$

and $r = 5$.



$$p'\,(x_1, x_2, x_3) = 1 + x_1 \cdot x_3 + x_2 \cdot x_3$$

Algorithm Description

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. Run [SS'96] using the "simulated" membership query oracle for the junta f'.

Open Problems

- What are the right lower bounds for testing classes like s-term DNF, size-s decision trees?
  - Can get $\approx \Omega(\log s)$ following [CG04], but feels like right bound is $\Omega(\text{poly}(s))$?
- Can "testing by implicit learning"approach be modified to get testers that are more computationally efficient?
  - Ideally shoot for $\text{poly}(s/\epsilon)$ runtime to match query complexity…
  - Computationally efficient proper learning algorithms would yield these, but these seem hard to come by
- Better understanding of testability of boolean functions?

Big-picture question

Whole talk – uniform distribution.

What about **distribution-independent** {learning, testing, approximating}?

- Rich theory of distribution-independent (PAC) learning
- Less fully developed theory of distribution-independent testing [HK03,HK04,HK05,AC06]
- Things are much harder…what is doable?
  - [GS07] Any distribution-independent algorithm for testing whether f is a halfspace requires $\Omega(n^{1/5})$ queries.

Thank you for your attention