Distribution-free junta testing

Xi Chen
Columbia University

Zhengyang Liu
Shangahi Jiao Tong University

Rocco Servedio
Columbia University

Ying Sheng
Columbia University → Stanford University

Jinyu Xie
Columbia University

Simons Institute Workshop: Boolean Devices
September 2018

Boolean Function

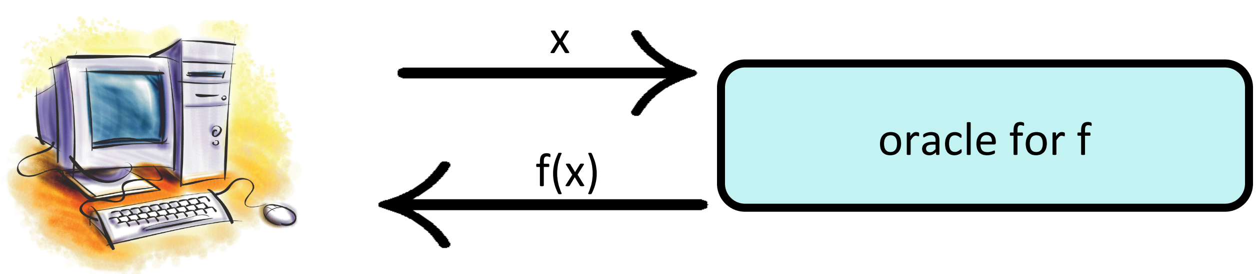
Property Testing

Boolean function property testing

"Property" of Boolean functions: a class **C** of Boolean functions
(= all n-variable functions that have the property)

Examples:
C = linear functions over GF(2) (parities)
C = degree-d GF(2) polynomials
C = halfspaces
C = monotone functions
C = s-term DNF formulas
etc.

Testing algorithm (randomized): Makes black-box queries to *arbitrary* f: $\{0,1\}^n \rightarrow \{0,1\}$

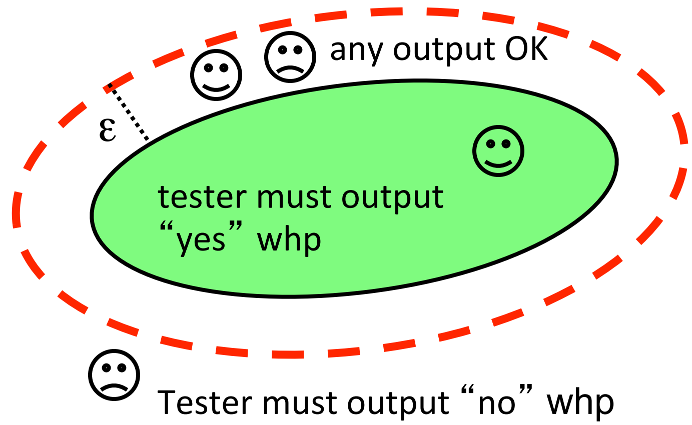


And eventually outputs "yes" or "no"

"Standard" (uniform-distribution) property testing

Tester for class **C** must (whp) output

- "yes" if f is in **C**
- "no" if f is ε-far from every function g in **C**



Standard model: Measure distance between f and g w.r.t. uniform distribution over $\{0,1\}^n$, i.e.

$$\Pr_{x \text{ uniform}}[f(x) \neq g(x)]$$

Main concern: information-theoretic # of queries required

Gold Standard: # queries required is *independent of n*

Much is known about testing various well-studied classes of Boolean functions in the standard (uniform) model:

- linear functions over GF(2) (parities) [BLR93, many others]
- degree-d GF(2) polynomials [AKKLR05]
- halfspaces [MORS09, MORS10, BBBY12, RS15]
- small-width OBDDs [G10, BMW11, RT12]
- monotone functions [DGLRRS99, GGLRS00, CST14, CDST15, KMS15, BB16, CWX17]
- dictators, conjunctions, monotone s-term DNF [PRS01]
- s-term DNF formulas, size-s decision trees, size-s Boolean formulas and circuits, s-sparse polynomials and algebraic circuits over GF(2), etc. [DLMORSW07]

This work:

- k-juntas: functions with $\leq$ k relevant variables [FKRSS04, CG04, AS07, B08, B09, RT11, BBR12, STW15, ABRW16, CSTWX17, BCEL18, LW18]

The class for this talk:
C = k-juntas

A function f: $\{0,1\}^n \rightarrow \{0,1\}$ is a k-junta if it only depends on k of the n input variables. (Think of k<<n.)

Example:

f(1 0 0 1 1 0 1 1 1 0 0 1 1 0 0 0 0 1 1 1 0 1 1 0 0 1 0 0 1 0 1 1) = 0

f(0 1 1 0 0 0 1 0 1 0 0 1 1 0 1 1 0 1 0 1 0 1 1 0 0 1 0 0 1 0 1 1) = 0

f(1 0 0 1 1 0 1 1 1 1 0 1 0 0 1 1 0 1 1 0 1 1 0 0 1 0 0 1 0 0 0 0) = 1

$f(x_1,\ldots,x_n) = \mathrm{MAJ}(x_6, x_{10}, x_{19})$ is a 3-junta

k-junta testing: well-understood!
(…in the standard model…)

Adaptive algorithms:

- $\tilde{O}(k)/\varepsilon$-query testing algorithm [Blais09]
- $\Omega(k)$-query lower bound for testing to constant accuracy [CG04]

Nonadaptive algorithms:

- $\tilde{O}(k^{3/2})/\varepsilon$-query algorithm [Blais08]
- $\tilde{\Omega}(k^{3/2})/\varepsilon$-query lower bound for testing to accuracy ε [CSTWX17]

So what is this talk about?

1-slide motivation for distribution-free testing model

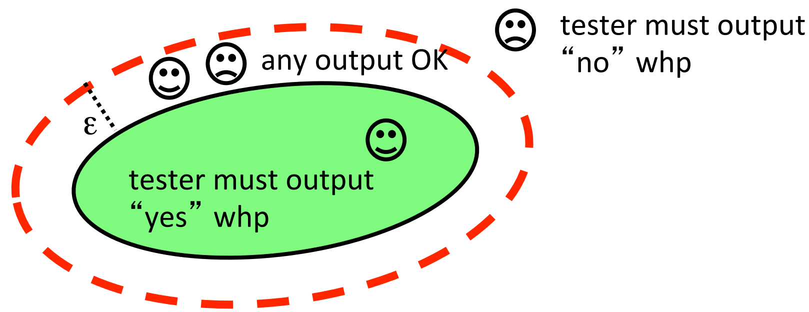
Juntas capture “this phenomenon depends on few causes”

Junta testing could conceivably be useful for real-world data analysis…

...but are your real-world data points really distributed according to uniform over $\{0,1\}^n$?

This work: Junta testing in the **distribution-free** model

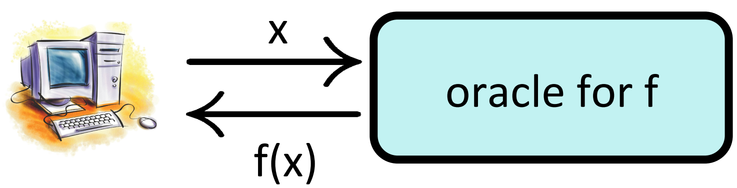
Distribution-free property testing:
same as before, but now



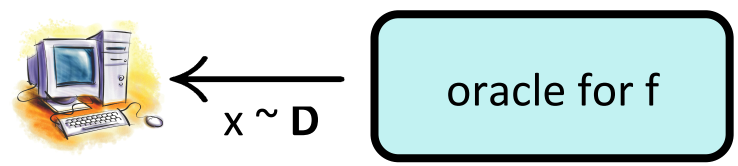
- There is an **unknown and arbitrary** distribution **D** over $\{0,1\}^n$ used to measure distance between f and g:

$$\Pr_{x \sim \mathbf{D}}\,[f(x) \neq g(x)]$$

- Tester can make black-box queries



and can *draw independent samples* from **D**



Some results on distribution-free testing of various classes

Class **C**	Standard model query complexity	Distribution-free query complexity
Conjunctions	$O(1/\varepsilon)$ [PRS01]	$\tilde{\Theta}(n^{1/3})$ [GS09,DR11, CX16]
Monotone functions	poly(n) [GGLRS00], [CDS14, KMS15, BB16, CWX17]	$2^{\Theta(n)}$ [HK05]
Linear threshold functions	$O(1/\varepsilon)$ [MORS09]	$\tilde{\Omega}(n^{1/5})$ [GS09]

Distribution-free testing
can be a lot harder
than standard testing!

Motivating question for this work:

Are k-juntas *easy* or *hard* to test in the distribution-free model?

The answer:

Sort of hard for non-adaptive algorithms…

but *surprisingly easy* for adaptive algorithms!

Non-Adaptive Distribution-Free Junta Testing: sort of hard

Theorem [FKRSS04, HK07,AW12]: The class of k-juntas is non-adaptively distribution-free testable using $O(2^k)/\varepsilon$ queries.

- Uniform distribution tester + “self-corrector” → distribution-free tester

Theorem [this work]: Any non-adaptive algorithm that distribution-free ε-tests k-juntas, for ε=0.49, must use $\Omega(2^{k/3})$ queries.

Adaptive Distribution-Free Junta Testing: surprisingly easy!

Theorem [this work]: There is an (adaptive) one-sided distribution-free ε-testing algorithm for the class of k-juntas that makes $\tilde{O}(k^2)/\varepsilon$ queries.

Sharp contrast with other classes (conjunctions, LTFs, monotone functions) where distribution-free testing *much harder* than standard testing

Rest of this talk

A few slides on the lower bound:

Theorem: Any non-adaptive algorithm that distribution-free ε-tests k-juntas, for ε=0.49, must use $\Omega(2^{k/3})$ queries.

Mostly about the upper bound:

Theorem: There is an (adaptive) one-sided distribution-free ε-testing algorithm for the class of k-juntas that makes $\tilde{O}(k^2)/\varepsilon$ queries.

The lower bound

Theorem: Any non-adaptive algorithm that distribution-free ε-tests k-juntas, for ε=0.49, must use $\Omega(2^{k/3})$ queries.

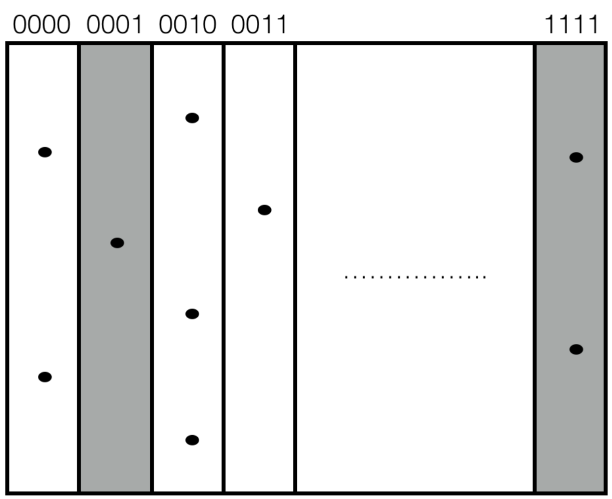
Proof is by the usual technique: **Yao's minimax principle**.

Suffices to exhibit two distributions D_{yes}, D_{no}, each over (distribution,function) pairs, such that no $2^{k/3}$-query *deterministic* algorithm can distinguish them.

D_{yes} over (distribution, function) pairs

The distribution:

- Puts equal weight on $2^k \log(n)$ many randomly chosen points



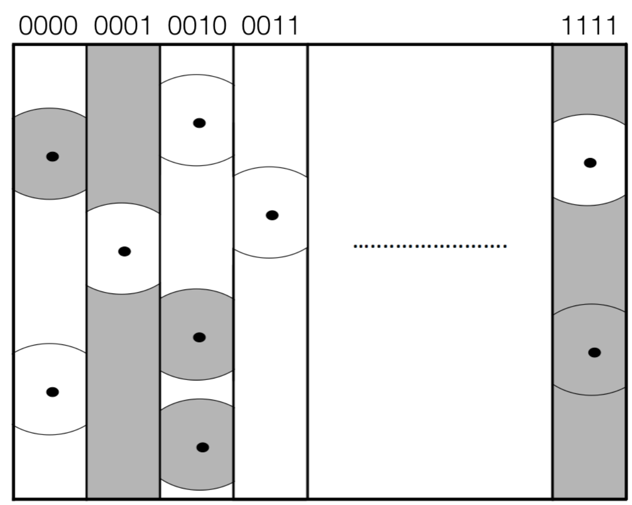
The function:

- Pick k variables at random (defines 2^k "sections")
- Pick random function over them

D_{no} over (distribution,function) pairs

The distribution:

- Same as in D_{yes}



The function:

- As before, but *toss new coin for each 0.4n-radius Hamming ball around each support point within each section*

The lower bound: Deterministic $2^{k/3}$ query **non-adaptive** algorithms can't tell D_{yes} from D_{no}

Non-adaptive algorithm first

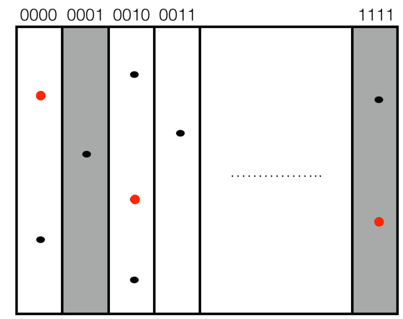
(1) gets $2^{k/3}$ samples from distribution, then

(2) makes $2^{k/3}$ non-adaptive queries.

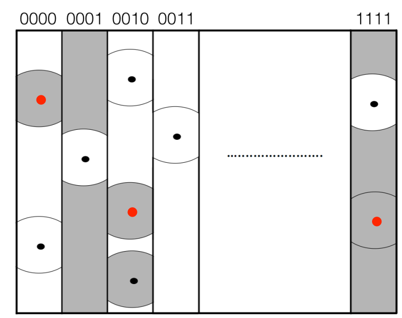
(1) Samples don't help: whp see $2^{k/3}$ different points in $2^{k/3}$ different sections, everything looks totally random in both cases.

(2) Do non-adaptive queries help?

What can non-adaptive queries do?



D_{yes}



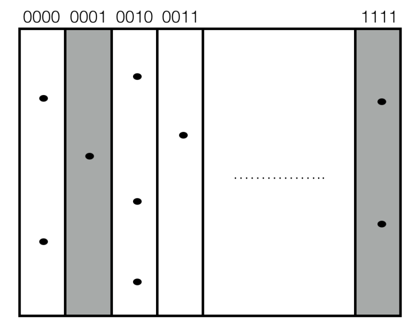
D_{no}

Intuition for the lower bound

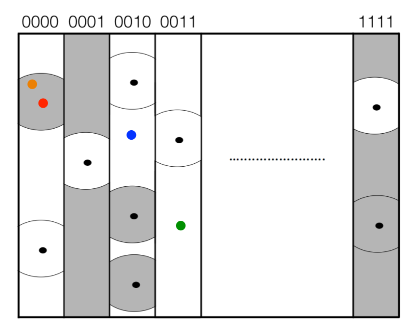
Intuition: To distinguish D_{no} from D_{yes}, must find two inputs in same "section" that are labeled differently.

Given a sampled point like •,

- if tester flips "too many" (>n/k) of its bits, whp it will end up in a different section (like • or •) ☹
- if tester flips "too few" (<n/k) of its bits, it won't escape the ball (like •) ☹



D_{yes}



D_{no}

Rest of talk:
An $\tilde{O}(k^2)/\varepsilon$-query adaptive algorithm

Theorem: There is an (adaptive) one-sided distribution-free ε-testing algorithm for the class of k-juntas that makes $\tilde{O}(k^2)/\varepsilon$ queries.

Let’s brainstorm:
How might we take advantage of adaptivity?

One of the all-time great adaptive algorithms:

Binary Search

The fastest known uniform-distribution testing algorithm [Blais09] makes crucial use of *binary search*

Let’s see (a simplified version of) this algorithm…

A ($k/\varepsilon + k*\log(n)$)-query **uniform-distribution** tester

Query complexity of actual [Blais09] algorithm is $\tilde{O}(k)/\varepsilon$.

Simplified version that makes $O(k/\varepsilon + k*\log(n))$ queries: Repeatedly try to grow a set R of *known-to-be-relevant* variables by 1 each time.

- If $|R|$ reaches k+1, output "not a k-junta"
- If $|R| < k+1$ after $100k/\varepsilon$ tries, output "is a k-junta"

Each attempt to grow R:

- Draw uniform random x in $\{0,1\}^n$, *rerandomize* coordinates in $[n]\backslash R$ to get y
- Query f at x and at y. If $f(x) \neq f(y)$, do **binary search** on coordinates in $[n]\backslash R$ to find a new relevant variable.

One attempt

R = known-to-be-relevant variables = {1,2,3}

Draw uniform x: x = 1 1 0 1 0 0 1 0 1 0 1 1 0 0 0 1 0 0

Rerandomize non-R coords: y = 1 1 0 1 0 0 1 0 1 0 0 0 1 1 1 0 1 1

Query x, y: $f(x)=1$, $f(y)=0$

Binary search to find a new relevant variable:

form z^1 = 1 1 0 1 0 0 1 0 1 0 0 0 1 1 0 1 0 0

Query z^1: $f(z^1)=1$:

form z^2 = 1 1 0 1 0 0 1 0 1 0 0 0 1 1 0 1 1 1

etc.

This simplified algorithm: log(n) queries to do one binary search
→ overall query complexity $k/\varepsilon + k*\log(n)$.

Blais's actual algorithm: at beginning forms poly(k/ε) many **randomly chosen blocks of variables.** Binary search over *relevant blocks*, not relevant variables → overall query complexity $\tilde{O}(k/\varepsilon)$.

Blais's (intricate) analysis heavily relies on *uniform distribution* (Efron-Stein / Fourier decomposition of functions over product spaces).

What to do in distribution-independent setting?

A $(k/\varepsilon + k^*\log(n))$-query **distribution-free** tester

Turns out that a simple tweak of the naïve binary-search-based uniform-distribution tester works in the distribution-free setting!

Everything is exactly as in uniform-distribution $(k/\varepsilon + k^*\log(n))$-query algorithm, but now each attempt to grow R works as follows:

- Draw x in $\{0,1\}^n$ **from distribution D**; *uniformly rerandomize coordinates in [n]\R* to get y

Key to (simple) analysis:

If f: $\{0,1\}^n \rightarrow \{0,1\}$ is ε-far from every k-junta with respect to **D**, and $|R| <= k$, then

$$\Pr_{x,y}[f(x) \neq f(y)] > \varepsilon/2.$$

From (k*log(n) + k/ε) queries to Õ(k^2/ε) queries?

Like [Blais09], we need to do binary search over blocks, not single variables.

Key to simple algorithm: ability to *uniformly rerandomize* coordinates in [n]\R (R = set of relevant variables)

- Draw x in $\{0,1\}^n$ **from distribution D;** *uniformly rerandomize* coordinates in [n]\R to get y

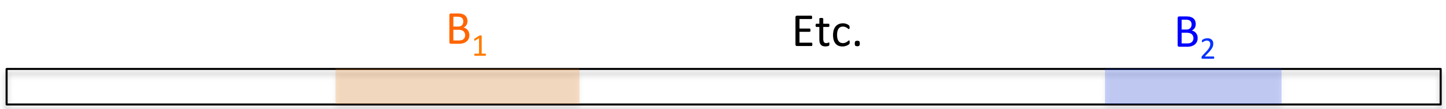
But seems we can't identify R without log(n) queries…

Setup for the real algorithm, and a crucial assumption

Our real algorithm:

- Maintains disjoint relevant blocks B_1, B_2, ...
- For each B_i, maintains two strings x^i, y^i differing only on B_i such that $f(x^i) \neq f(y^i)$. Let w^i = partial string which is common part of x^i, y^i.

Key assumption: for each i, f_{w^i} is close to a 1-junta over B_i *under the uniform distribution.*



x^1 = 1 1 0 1 0 0 1 0 1 0 1 1 0 **0** 0 1 0 0 0 1 1 0 0 1 0 0 0 1 1 0 0 0 0 1 0 1 0 1 1 1

y^1 = 1 1 0 1 0 0 1 0 1 1 1 0 1 **1** 0 0 0 0 0 1 1 0 0 1 0 0 0 1 1 0 0 0 0 1 0 1 0 1 1 1

x^2 = 0 0 1 0 1 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 1 0 1 0 0 0 1 1 1 1 0 1 0 0 1 **1** 0 1 0 0

y^2 = 0 0 1 0 1 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 1 0 1 0 0 0 1 1 1 1 0 0 0 1 1 **0** 0 1 0 0

Etc.

What the assumption enables

Our real algorithm:

- Maintains disjoint relevant blocks B_1, B_2, ...
- For each B_i, maintains two strings x^i, y^i differing only on B_i such that $f(x^i) \neq f(y^i)$. Let w^i = partial string which is common part of x^i, y^i.

Key assumption: for each i, f_{w^i} close to a 1-junta over B_i *under uniform distribution.*

Let R = set of all the 1-junta variables for B_1, B_2,...
(R is *unknown to the algorithm!*)

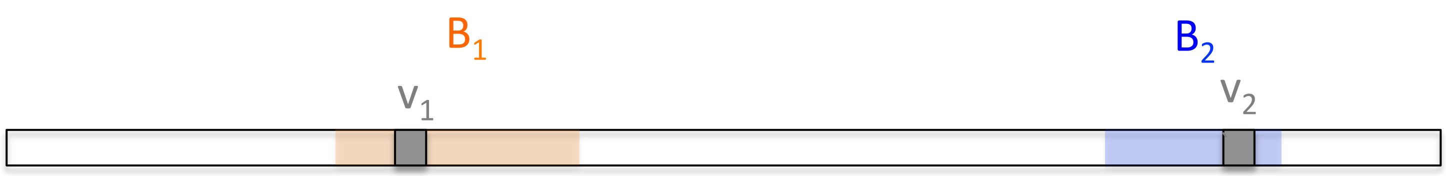
Key Fact: In the above scenario, we can uniformly rerandomize coordinates in $[n] \setminus R$ (even though we don't know what R is)!

Rerandomizing coordinates in [n]\R

For each block B_i, have strings x^i, y^i, differing only on B_i, $f(x^i) \neq f(y^i)$ and f_{w^i} close to 1-junta over B_i *under uniform distribution.*
Let v_i be the 1-junta variable in B_i, so R = {the 1-junta variables for B_1, B_2, …} = {v_1, v_2, …}

- Randomly split each B_i into two pieces; check which one v_i is in (easy), and let Q_i = other one.
- Let S = uniform random subset of variables outside B_1,B_2,…

Then **S U Q_1 U Q_2 U … is a uniform random subset of [n]\R** (equivalent to rerandomizing [n]\R).



S = random subset of white stuff.
Randomly split B_1; Q_1 is the piece not containing v_1 → Q_1 is a random subset of $B_1 \setminus \{v_1\}$
Likewise for B_2.

What about key assumption?

Recall **Key assumption:** for each i, f_{w^i} is close to 1-junta over B_i *under uniform distribution.*

We check this for each B_i using [Blais09] uniform-distribution 1-junta tester on f_{w^i}.

- If it holds, ☺
- If it *doesn't* hold: [Blais09] algorithm will *split B_i into* ***two*** relevant blocks → progress! ☺

Overall algorithm:

Algorithm maintains relevant blocks $(B_1, x^1, y^1), (B_2, x^2, y^2), \ldots, (B_t, x^t, y^t)$

- If some block B_i not (uniform-distribution) close to 1-junta, **split the block →** progress ([Blais09] 1-junta tester)
- If each block B_i is close to a 1-junta: try to add new relevant variables as in naive algorithm
  (R = relevant variables in these blocks; draw x ~ **D**, uniformly rerandomize variables outside R, do binary search over blocks to find new relevant block as before)
  - If $100k/\varepsilon$ tries didn't yield k+1 relevant blocks, output "junta"
  - If find k+1 relevant blocks within $100k/\varepsilon$ tries, output "not a junta"

Summary and future work

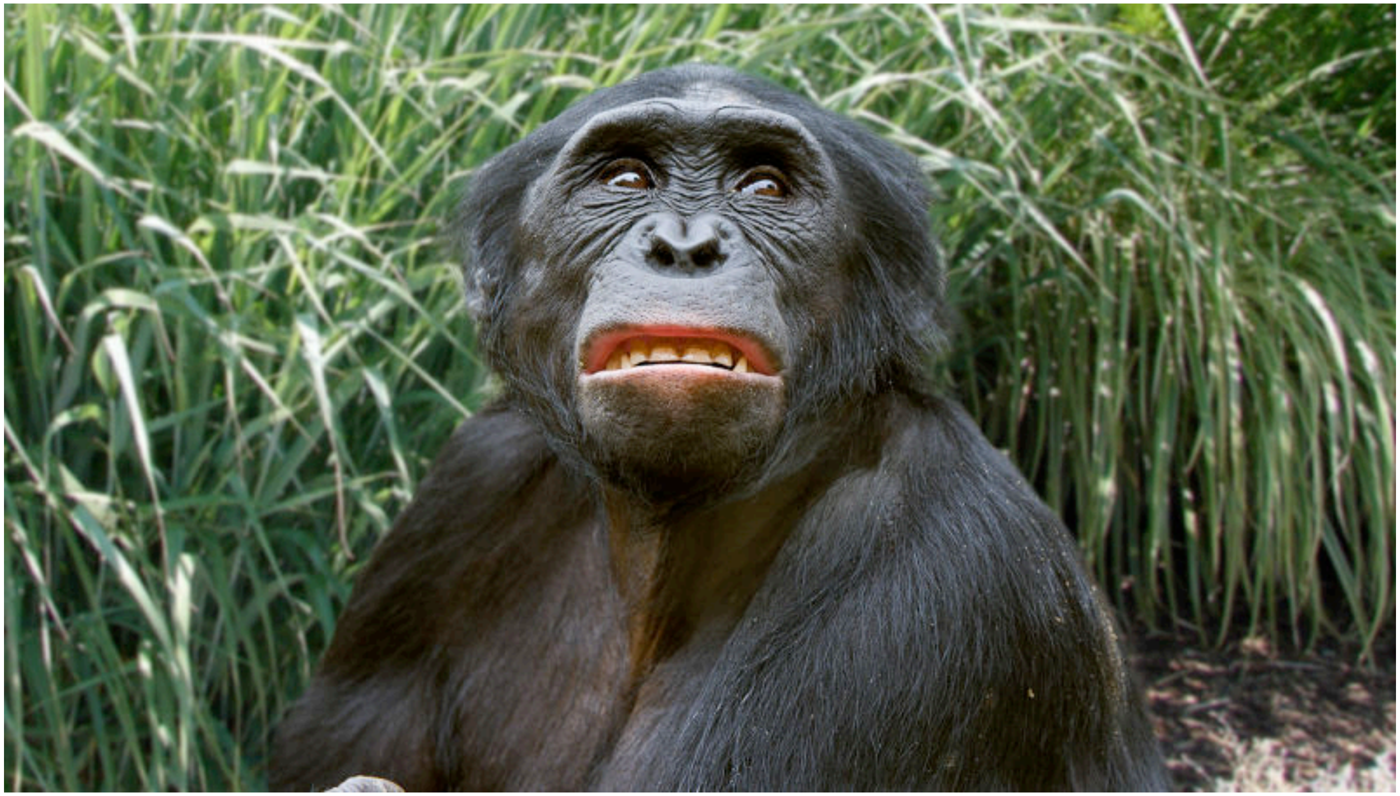
Summary of our results:

k-juntas can be (adaptively) ε-tested in distribution-free model with about k^2/ε queries.

Non-adaptive distribution-free testers need $2^{\Omega(k)}$ queries.

Future work:

- A k/ε-query algorithm? (Matching uniform distribution setting?)
- Tolerant / active / sample-based distribution-free testers?



Thank you!