Testing equivalence between distributions using conditional samples

(when testers get to be picky)

Clément CANONNE* Dana Ron† Rocco Servedio*

*Columbia University
†Tel-Aviv University

January 6, 2014
Plan of the talk

1. Introduction
2. Testing Uniformity and Identity
3. Tools and subroutines
4. Conclusion
Background and motivation
What is distribution testing?

Property testing
Given a big, hidden “object” $X$ one can only access by local, expensive inspections (e.g., oracle queries), and a property $\mathcal{P}$, the goal is to check in sublinear number of inspections if (a) $X$ has the property or (b) $X$ is “far” from all objects having the property.\(^1\)

\(^1\)wrt to some specified metric, and parameter $\varepsilon > 0$ given to the tester.
Background and motivation
What is distribution testing?

Property testing
Given a big, hidden “object” $X$ one can only access by local, expensive inspections (e.g., oracle queries), and a property $\mathcal{P}$, the goal is to check in sublinear number of inspections if (a) $X$ has the property or (b) $X$ is “far” from all objects having the property.\(^1\)

Testing distributions (standard model)
$X$ is an unknown probability distribution $D$ over some $N$-element set; the testing algorithm has blackbox sample access to $D$.

\(^1\)wrt to some specified metric, and parameter $\varepsilon > 0$ given to the tester.
Distribution testing (1)
In more detail.

Distance criterion: total variation distance ($\propto L_1$ distance)

$$d_{TV}(D_1, D_2) \overset{\text{def}}{=} \frac{1}{2} \|D_1 - D_2\|_1 = \frac{1}{2} \sum_{i \in [N]} |D_1(i) - D_2(i)|.$$ 

Definition (Testing algorithm)

Let $\mathcal{P}$ be a property of distributions over $[N]$, and $\text{ORACLE}_D$ be some type of oracle which provides access to $D$. A $q(\varepsilon, N)$-query ORACLE testing algorithm for $\mathcal{P}$ is a (randomized) algorithm $T$ which, given $\varepsilon, N$ as input parameters and oracle access to an ORACLE$_D$ oracle, and for any distribution $D$ over $[N]$, makes at most $q(\varepsilon, N)$ calls to ORACLE$_D$, and:

- if $D \in \mathcal{P}$ then, w.p. at least $2/3$, $T$ outputs ACCEPT;
- if $d_{TV}(D, \mathcal{P}) \geq \varepsilon$ then, w.p. at least $2/3$, $T$ outputs REJECT.
A few remarks

- “gray” area for $d_{TV}(D, P) \in (0, \varepsilon)$;
A few remarks

- “gray” area for $d_{TV}(D, P) \in (0, \varepsilon)$;
- $2/3$ is completely arbitrary;
A few remarks

- “gray” area for $d_{TV}(D, \mathcal{P}) \in (0, \varepsilon)$;
- $2/3$ is completely arbitrary;
- extends to several oracles and distributions;
A few remarks

- “gray” area for $d_{TV}(D, \mathcal{P}) \in (0, \varepsilon)$;
- $2/3$ is completely arbitrary;
- extends to several oracles and distributions;
- our measure is the **sample complexity** (*not* the running time).
Distribution testing (3)
Concrete example: testing uniformity

Property $\mathcal{P}$ (“being $\mathcal{U}$, the uniform distribution over $[N]$”) $\iff$ set $S_{\mathcal{P}}$ of distributions with this property ($S_{\mathcal{P}} = \{\mathcal{U}\}$)

Distance to $\mathcal{P}$:

$$d_{TV}(D, S_{\mathcal{P}}) = \min_{D' \in S_{\mathcal{P}}} d_{TV}(D, D') = d_{TV}(D, \mathcal{U})$$

General outline

1. Draw a bunch of samples from $D$;
2. “Process” them (for instance by counting the number of points drawn more than once: collision-based tester);
3. Output ACCEPT or REJECT based on the result.
Background and motivation
Well, it’s more or less settled.

Fact

In the standard sampling model, most (natural) properties are “hard” to test; that is, require a strong dependence on $N$ (at least $\Omega(\sqrt{N})$).
Background and motivation
Well, it’s more or less settled.

Fact
In the standard sampling model, most (natural) properties are “hard” to test; that is, require a strong dependence on $N$ (at least $\Omega(\sqrt{N})$).

Example
Testing uniformity has $\Theta(\sqrt{N}/\varepsilon^2)$ sample complexity [GR00, BFR$^+$10, Pan08], equivalence to a known distribution $\tilde{\Theta}(\sqrt{N}/\varepsilon^2)$ [BFF$^+$01, Pan08]; equivalence of two unknown distributions $\Omega(N^{2/3})$ [BFR$^+$10, Val11, CDVV13] (and essentially matching upperbound).…
Our model

More power to the tester

We consider a new model where the tester can specify a subset of the domain, and then get a draw conditioned on it landing in that subset. Models natural applications where a scientist/experimenter has some control over an ’experiment’ to restrict the range of possible outcomes – e.g., by tuning the conditions or the setting: *this is not captured by the SAMP model.*
Our model

More power to the tester

We consider a new model where the tester can specify a subset of the domain, and then get a draw conditioned on it landing in that subset. Models natural applications where a scientist/experimenter has some control over an ’experiment’ to restrict the range of possible outcomes – e.g., by tuning the conditions or the setting: this is not captured by the SAMP model.

Definition (COND oracle)

Fix a distribution $D$ over $[N]$. A COND oracle for $D$, denoted $\text{COND}_D$, is defined as follows: The oracle is given as input a query set $S \subseteq [N]$ that has $D(S) > 0$, and returns an element $i \in S$, where the probability that element $i$ is returned is $D_S(i) = D(i)/D(S)$, independently of all previous calls to the oracle.
Remark

- generalizes the SAMP oracle \( S = [N] \), but allows adaptiveness;
Our model

Remark
- generalizes the SAMP oracle ($S = [N]$), but allows adaptiveness;
- variants of the (general) COND oracle, which only allow some specific types of subsets to be queried: PCOND (either $[N]$ or sets $\{i, j\}$) and ICOND (only intervals);
Our model

Remark
- generalizes the SAMP oracle ($S = [N]$), but allows adaptiveness;
- variants of the (general) COND oracle, which only allow some specific types of subsets to be queried: PCOND (either $[N]$ or sets $\{i, j\}$) and ICOND (only intervals);
- not defined for sets $S$ with zero probability under $D$;
Our model

**Remark**

- generalizes the SAMP oracle ($S = [N]$), but allows adaptiveness;
- variants of the (general) COND oracle, which only allow some specific types of subsets to be queried: PCOND (either $[N]$ or sets $\{i, j\}$) and ICOND (only intervals);
- **not defined** for sets $S$ with zero probability under $D$;
- similar model independently introduced by Chakraborty et al. [CFGM13].
### Our model

**Remark**
- generalizes the SAMP oracle \((S = [N])\), but allows adaptiveness;
- variants of the (general) COND oracle, which only allow some specific types of subsets to be queried: PCOND (either \([N]\) or sets \(\{i,j\}\)) and ICOND (only intervals);
- **not defined** for sets \(S\) with zero probability under \(D\);
- similar model independently introduced by Chakraborty et al. [CFGM13].

### Question
Do COND oracles enable more efficient testing algorithms than SAMP oracles? And what does it reveal about testing distributions?
Our results

**Question**

Do COND oracles enable more efficient testing algorithms than SAMP oracles?
Our results

Question
Do COND oracles enable more efficient testing algorithms than SAMP oracles? Yes, they do.
Our results
Comparison of the COND and SAMP models on several testing problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>Our results</th>
<th>Standard model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is $D = D^<em>$ for a known $D^</em>$?</td>
<td>COND$_D$ $\tilde{O}\left(\frac{1}{\epsilon^4}\right)$</td>
<td>$\tilde{\Theta}\left(\frac{\sqrt{N}}{\epsilon^2}\right)$ [BFF$^+$01, Pan08]</td>
</tr>
<tr>
<td></td>
<td>PCOND$_D$ $\tilde{O}\left(\frac{\log^4 N}{\epsilon^4}\right)$</td>
<td>$\Omega\left(\sqrt{\frac{\log N}{\log \log N}}\right)$</td>
</tr>
<tr>
<td></td>
<td>$\Omega\left(\sqrt{\frac{\log N}{\log \log N}}\right)$</td>
<td></td>
</tr>
<tr>
<td>Are $D_1, D_2$ (both unknown) equivalent?</td>
<td>COND$_{D_1,D_2}$ $\tilde{O}\left(\frac{\log^5 N}{\epsilon^4}\right)$</td>
<td>$\Theta\left(\max\left(\frac{N^{2/3}}{\epsilon^{4/3}}, \frac{\sqrt{N}}{\epsilon^2}\right)\right)$ [BFR$^+$10, Val11, CDVV13]</td>
</tr>
<tr>
<td></td>
<td>PCOND$_{D_1,D_2}$ $\tilde{O}\left(\frac{\log^6 N}{\epsilon^{21}}\right)$</td>
<td></td>
</tr>
</tbody>
</table>

Table: Comparison between the COND model and the standard model for these problems. The upper bounds are for testing $d_{TV} = 0$ vs. $d_{TV} \geq \epsilon$. 
Plan for rest of talk:

- sketch of testing uniformity and testing $D$ vs. $D^*$ (with pairwise queries)
- introducing tools: Estimate-Neighborhood and Approx-Eval
- using them: testing equivalence of two unknown distributions
Testing Uniformity (1)
Special case of testing identity to $D^*$

Theorem (Testing Uniformity with PCOND)

There exists a $\tilde{O}(1/\varepsilon^2)$-query PCOND$_D$ tester for uniformity, i.e. it accepts w.p. at least 2/3 if $D = \mathcal{U}$ and rejects w.p. at least 2/3 if $d_{TV}(D, \mathcal{U}) \geq \varepsilon$. 
Testing Uniformity (1)
Special case of testing identity to $D^*$

**Theorem (Testing Uniformity with PCOND)**

There exists a $\tilde{O}(1/\varepsilon^2)$-query $\text{PCOND}_D$ tester for uniformity, i.e. it accepts w.p. at least $2/3$ if $D = \mathcal{U}$ and rejects w.p. at least $2/3$ if $d_{TV}(D, \mathcal{U}) \geq \varepsilon$.

**High-level idea**

Intuitively, if $D$ is $\varepsilon$-far from uniform, it must have (a) a lot of points “very light”; and (b) a lot of weight on points “very heavy”. Sampling $O(1/\varepsilon)$ points both uniformly and according to $D$, we obtain whp both light and heavy ones; and use PCOND to compare them.
Testing Uniformity (1)
Special case of testing identity to \( D^* \)

**Theorem (Testing Uniformity with PCOND)**

There exists a \( \tilde{O}(1/\varepsilon^2) \)-query \( \text{PCOND}_D \) tester for uniformity, i.e. it accepts w.p. at least 2/3 if \( D = \mathcal{U} \) and rejects w.p. at least 2/3 if \( d_{TV}(D, \mathcal{U}) \geq \varepsilon \).

**High-level idea**

Intuitively, if \( D \) is \( \varepsilon \)-far from uniform, it must have (a) a lot of points “very light”; and (b) a lot of weight on points “very heavy”. Sampling \( O(1/\varepsilon) \) points both uniformly and according to \( D \), we obtain whp both light and heavy ones; and use PCOND to compare them.

Not good enough \( (O(1/\varepsilon^4) \text{ queries}) \) \( \rightsquigarrow \) refine this approach to get \( \tilde{O}(1/\varepsilon^2) \).
Testing Uniformity (2) – generalizing to $D^*$

From uniform to arbitrary distribution: poly($1/\varepsilon$)-query algorithm

**Approach does not work for general $D^*$. . .**

The ratios can be arbitrarily big or small: e.g., if $D^*(x)/D^*(y) = \sqrt{N}$, need $\Omega(\sqrt{N})$ calls to PCOND$_D$($\{x, y\}$) to distinguish $D(x)/D(y) = \sqrt{N}$ from $D(x)/D(y) = 2\sqrt{N}$. . . but it can be adapted.

Idea: compare points with carefully chosen comparable sets $\Rightarrow$ $D(x)/D(y)$ instead of $D(x)/D(y)$.

However, cannot do this with PCOND (Lower bound: $\log_2 \Omega(1/N)$ samples)): a COND oracle is needed.
Testing Uniformity (2) – generalizing to $D^*$

From uniform to arbitrary distribution: \( \text{poly}(1/\varepsilon) \)-query algorithm

**Approach does not work for general $D^*$ . . .**

The ratios can be arbitrarily big or small: e.g., if $D^*(x)/D^*(y) = \sqrt{N}$, need $\Omega(\sqrt{N})$ calls to $\text{PCOND}_D(\{x, y\})$ to distinguish $D(x)/D(y) = \sqrt{N}$ from $D(x)/D(y) = 2\sqrt{N}$

. . . but it can be adapted.

Idea: compare points with carefully chosen *comparable sets* $\sim D(x)/D(Y)$ instead of $D(x)/D(y)$
Testing Uniformity (2) – generalizing to $D^*$

From uniform to arbitrary distribution: poly($1/\varepsilon$)-query algorithm

**Approach does not work for general $D^*$ . . .**

The ratios can be arbitrarily big or small: e.g., if $D^*(x)/D^*(y) = \sqrt{N}$, need $\Omega(\sqrt{N})$ calls to $\text{PCOND}_D(\{x, y\})$ to distinguish $D(x)/D(y) = \sqrt{N}$ from $D(x)/D(y) = 2\sqrt{N}$

. . . but it can be adapted.

Idea: compare points with carefully chosen *comparable sets* $\sim D(x)/D(Y)$ instead of $D(x)/D(y)$

However, cannot do this with PCOND (Lower bound: $\log^{\Omega(1)} N$ samples)): a COND oracle is needed.
Building tools (1)

- **COMPARE**
  Low-level procedure: compares the relative weight of disjoint sets $X$, $Y$, given some accuracy parameter $\eta$.

- **ESTIMATE-NEIGHBORHOOD**
  On input a point $i \in [N]$ and parameter $\gamma$, estimates the weight under $D$ of the $\gamma$-neighborhood of $i$ – that is, points with probability mass within a factor $(1 + \gamma)$ of $D(i)$.

- **APPROX-EVAL**
  Given $i \in [N]$ and accuracy parameter $\eta$, returns an approximation of $D(i)$ – succeeds whp for most points $i$. 
Building tools (2)
First tool: The low-level \text{COMPARE}

“Comparison is the death of joy.” – Mark Twain.

\[ \begin{align*}
\rho &\approx \frac{D(Y)}{D(X)} \\
D(X) &\ll D(Y) \\
D(X) &\approx D(Y) \\
D(X) &\gg D(Y)
\end{align*} \]
Building tools (3)

Second tool: **Estimate-Neighborhood** procedure

**Definition (γ-Neighborhood)**

\[
U_\gamma(x) \overset{\text{def}}{=} \left\{ y \in [N] : \frac{1}{1 + \gamma} D(x) \leq D(y) \leq (1 + \gamma) D(x) \right\}, \quad \gamma \in [0, 1]
\]
Building tools (3)

Second tool: Estimate-Neighborhood procedure

Definition ($\gamma$-Neighborhood)

\[ U_\gamma(x) \overset{\text{def}}{=} \left\{ y \in [N] : \frac{1}{1 + \gamma} D(x) \leq D(y) \leq (1 + \gamma) D(x) \right\}, \quad \gamma \in [0, 1] \]

Goal

Given a point $x \in [N]$ and a parameter $\gamma$, Estimate-Neighborhood gives a multiplicative approximation of $D(U_\gamma(x))$ – i.e., “how much weight does $D$ put on points like $x$?”
Building tools (4)

Third tool: \texttt{APPROXIMATE-EVAL} oracle

\begin{itemize}
\item A $\delta$-\textsc{EVAL}_D simulator for $D$ is a randomized procedure \textsc{ORACLE} such that w.p. $1 - \delta$ the output of \textsc{ORACLE} on input $i^* \in [N]$ is $D(i^*)$.
\end{itemize}
(Approximate) EVAL oracle

Ideally, an \((\varepsilon, \delta)\)-approximate EVAL\(_D\) simulator for \(D\) would be a randomized procedure ORACLE such that w.p. \(1 - \delta\) the output of ORACLE on input \(i^* \in [N]\) is a value \(\alpha \in [0, 1]\) such that \(\alpha \in [1 - \varepsilon, 1 + \varepsilon]D(i^*)\).
Building tools (4)

Third tool: **Approximate-EVAL oracle**

**(Approximate) EVAL oracle**

Actually, an \((\varepsilon, \delta)\)-approximate \(EVAL_D\) simulator for \(D\) is a randomized procedure \(\text{ORACLE}\) s.t for each \(\varepsilon\), there is a fixed set \(S^{(\varepsilon)} \subset [N]\) with 
\(D(S^{(\varepsilon)}) < \varepsilon\) for which the following holds. For all \(i^* \in [N]\), \(\text{ORACLE}(i^*)\) is either a value \(\alpha \in [0, 1]\) or Unknown, and furthermore:

(i) If \(i^* \notin S^{(\varepsilon)}\) then w.p. \(1 - \delta\) the output of \(\text{ORACLE}\) on input \(i^*\) is a value \(\alpha \in [0, 1]\) such that \(\alpha \in [1 - \varepsilon, 1 + \varepsilon]D(i^*)\);

(ii) If \(i^* \in S^{(\varepsilon)}\) then w.p. \(1 - \delta\) the procedure either outputs Unknown or outputs a value \(\alpha \in [0, 1]\) such that \(\alpha \in [1 - \varepsilon, 1 + \varepsilon]D(i^*)\).
Building tools (4)

Third tool: \textsc{Approximate-EVAL} oracle

\textbf{(Approximate) EVAL oracle}

Actually, an \((\varepsilon, \delta)-approximate\) \textsc{EVAL}_D simulator for \(D\) is a randomized procedure \textsc{Oracle} s.t for each \(\varepsilon\), there is a \textbf{fixed} set \(S^{(\varepsilon)} \subset \mathbb{N}\) with \(D(S^{(\varepsilon)}) < \varepsilon\) for which the following holds. For all \(i^* \in \mathbb{N}\), \textsc{Oracle}(\(i^*\)) is either a value \(\alpha \in [0, 1]\) or Unknown, and furthermore:

(i) If \(i^* \notin S^{(\varepsilon)}\) then w.p. \(1 - \delta\) the output of \textsc{Oracle} on input \(i^*\) is a value \(\alpha \in [0, 1]\) such that \(\alpha \in [1 - \varepsilon, 1 + \varepsilon]D(i^*)\);

(ii) If \(i^* \in S^{(\varepsilon)}\) then w.p. \(1 - \delta\) the procedure either outputs Unknown or outputs a value \(\alpha \in [0, 1]\) such that \(\alpha \in [1 - \varepsilon, 1 + \varepsilon]D(i^*)\).

The high-level blackbox \textbf{Approx-Eval}

There is an algorithm \textbf{Approx-Eval} which uses \(\tilde{O}\left(\frac{(\log N)^5 \cdot (\log(1/\delta))^2}{\varepsilon^3}\right)\) calls to \textsc{Cond}_D, and is an \((\varepsilon, \delta)-approximate\) \textsc{EVAL}_D simulator.
Approx-Eval_ε

COND_D

i* ∈ [N] → APPROX-EVAL_ε

i*不属于S(ε) → "Unknown"
or ̂D(i)

i* ∈ S(ε) → ̂D(i)
Building tools (5)

Third tool: \textsc{Approximate-Eval} oracle

\[ S_0 = [N] \]

\textit{Scan over heavy elements: } \textit{i not amongst them?}

\[ S_1 \]

\[ S_1' \]

\textit{Scan over heavy elements: } \textit{i not amongst them?}

\[ S_2 \]

\[ S_2' \]

\[ S_{k-1} \]

\[ S_k = \{i\} \]

\[ S_k' \]

\textbf{Figure:} Execution of \textsc{Approx-Eval} on some \( i \): scan over heavy elements, randomly partition the light ones, recurse; finally get an estimate of \( D(i) \) by multiplying estimates at each branching.
Applications

Testing equivalence of two unknown distributions $D_1$, $D_2$

Blackbox access to $D_1$ and $D_2$ (two oracles); distinguish $D_1 = D_2$ vs. $d_{TV}(D_1, D_2) \geq \varepsilon$. 
Testing equivalence of two unknown distributions $D_1, D_2$

Blackbox access to $D_1$ and $D_2$ (two oracles); distinguish $D_1 = D_2$ vs. $d_{TV}(D_1, D_2) \geq \varepsilon$. 

Two different approaches:
1. with PCOND and Estimate-Neighborhood – finding “representatives” points for both distributions;
2. with COND and Approx-Eval – adapting an EVAL algorithm from [RS09].

Other uses: estimating distance to uniformity (Estimate-Neighborhood), testing monotonicity (Approx-Eval). . .
Applications

Testing equivalence of two unknown distributions $D_1, D_2$

Blackbox access to $D_1$ and $D_2$ (two oracles); distinguish $D_1 = D_2$ vs. $d_{TV}(D_1, D_2) \geq \varepsilon$.

Two different approaches:

1. with PCOND and Estimate-Neighborhood – finding “representatives” points for both distributions;
2. with COND and Approx-Eval – adapting an EVAL algorithm from [RS09].

Other uses: estimating distance to uniformity (Estimate-Neighborhood), testing monotonicity\(^2\) (Approx-Eval). . .

\(^2\)(extension of the original results)
Applications
Testing $D_1 \equiv D_2$ with PCOND and \text{Estimate-Neighborhood}

Idea: get a \textit{succinct representation}

\begin{itemize}
  \item Get a \textit{"cover for $D_1"} in $\tilde{O}(\log N/\epsilon^2)$ representatives $r_1, \ldots, r_\ell$;
\end{itemize}
Applications
Testing $D_1 \equiv D_2$ with PCOND and Estimate-Neighborhood

Idea: get a succinct representation

- Get a “cover for $D_1$” in $\tilde{O}(\log N/\epsilon^2)$ representatives $r_1, \ldots, r_\ell$;
- If $D_1 = D_2$, cover perfect for $D_2$;
Applications

Testing $D_1 \equiv D_2$ with PCOND and Estimate-Neighborhood

Idea: get a succinct representation

- Get a "cover for $D_1$" in $\tilde{O}(\log N/\varepsilon^2)$ representatives $r_1, \ldots, r_\ell$;
- If $D_1 = D_2$, cover perfect for $D_2$; but
- If $d_{TV}(D_1, D_2) \geq \varepsilon$, then for one of the representatives $r^*$ (covering a set of points $R^*$ under $D_1$), either
  1. "many" $y \in R^*$ are not covered by $r^*$ under $D_2$ (mismatching representative); or
  2. $D_2(R^*)$ differs significantly from $D_1(R^*)$ (mismatching neighborhoods)
Applications

Testing $D_1 \equiv D_2$ with PCOND and Estimate-Neighborhood

Idea: get a succinct representation

- Get a “cover for $D_1$” in $\tilde{O}(\log N/\varepsilon^2)$ representatives $r_1, \ldots, r_\ell$;
- If $D_1 = D_2$, cover perfect for $D_2$; but
- If $d_{TV}(D_1, D_2) \geq \varepsilon$, then for one of the representatives $r^*$ (covering a set of points $R^*$ under $D_1$), either
  1. “many” $y \in R^*$ are not covered by $r^*$ under $D_2$ (mismatching representative); or
  2. $D_2(R^*)$ differs significantly from $D_1(R^*)$ (mismatching neighborhoods)

Both can be detected efficiently; try it for each $r_i \sim \text{poly}(\log N, 1/\varepsilon)$ sample and time complexity.
new model for studying probability distributions
arises naturally in a number of settings
allows significantly more query-efficient algorithms
generalizing to other structured domains? (e.g., the Boolean hypercube $\{0, 1\}^n$)
what about distribution learning in this framework
more properties? (entropy, independence, monotonicity\textsuperscript{†} . . . )
The end.

Thank you.

An extended version of this work [CRS12] is available online (arXiv:1211.2664).


______, *Testing closeness of discrete distributions*, Tech. Report abs/1009.5397, 2010, This is a long version of [BFR+00].


Algorithm 1: PCOND$_D$-TEST-UNIFORM

Set $t = \Theta(\log(\frac{1}{\epsilon}))$.

Select $q = \Theta(1)$ points $i_1, \ldots, i_q$ uniformly 

for $j = 1$ to $t$ do

Call the oracle $s_j = \Theta(2^j t)$ times to get $h_1, \ldots, h_{s_j} \sim D$ 

Draw $s_j$ points $\ell_1, \ldots, \ell_{s_j}$ uniformly from $[N]$ 

for all pairs $(x, y) = (i_r, h_r)$ and $(x, y) = (i_r, \ell_r)$ do

Get a good estimate of $D(x)/D(y)$.

Reject if the value is not in $[1 - 2^{j-5}\epsilon/4, 1 + 2^{j-5}\epsilon/4]$ 

end for

end for

Accept
Testing Uniformity (4)

Proof (Outline).

Sample complexity by the setting of \( t, q \) and the calls to \text{COMPARE}

Completeness unless \text{COMPARE} fails to output a correct value, no rejection

Soundness Suppose \( D \) is \( \varepsilon \)-far from \( U \); refinement of the previous approach by bucketing low and high points:

\[
H_j \overset{\text{def}}{=} \left\{ h \ \bigg| \ \left(1 + 2^j \frac{-1}{4} \varepsilon\right) \frac{1}{N} \leq D(h) < \left(1 + 2^j \frac{\varepsilon}{4}\right) \frac{1}{N} \right\}
\]

\[
L_j \overset{\text{def}}{=} \left\{ \ell \ \bigg| \ \left(1 - 2^j \frac{\varepsilon}{4}\right) \frac{1}{N} < D(\ell) \leq \left(1 - 2^j \frac{-1}{4} \varepsilon\right) \frac{1}{N} \right\}
\]

for \( j \in [t-1] \), with also \( H_0, L_0, H_t, L_t \) to cover everything; each loop iteration on l.3 “focuses” on a particular bucket.

+ Chernoff and union bounds.
The (slightly) higher-level subroutine `Estimate-Neighborhood`

Given as input a point $x$, parameters $\gamma, \beta, \eta \in (0, 1/2]$ and `PCOND_D` access, the procedure `Estimate-Neighborhood` outputs a pair $(\hat{w}, \alpha) \in [0, 1] \times (\gamma, 2\gamma)$ such that w.h.p

1. If $D(U_\alpha(x)) \geq \beta$, then $\hat{w} \in [1 - \eta, 1 + \eta] \cdot D(U_\alpha(x))$, and (\ldots)
2. If $D(U_\alpha(x)) < \beta$, then $\hat{w} \leq (1 + \eta) \cdot \beta$, and (\ldots)

`Estimate-Neighborhood` performs $\tilde{O}\left(\frac{1}{\gamma^2 \eta^4 \beta^3}\right)$ queries.
The (slightly) higher-level subroutine **Estimate-Neighborhood**

Given as input a point $x$, parameters $\gamma, \beta, \eta \in (0, 1/2]$ and $\text{PCOND}_D$ access, the procedure **Estimate-Neighborhood** outputs a pair $(\hat{w}, \alpha) \in [0, 1] \times (\gamma, 2\gamma)$ such that w.h.p

1. If $D(U_\alpha(x)) \geq \beta$, then $\hat{w} \in [1 - \eta, 1 + \eta] \cdot D(U_\alpha(x))$, and (…)
2. If $D(U_\alpha(x)) < \beta$, then $\hat{w} \leq (1 + \eta) \cdot \beta$, and (…)

**Estimate-Neighborhood** performs $\tilde{O}\left(\frac{1}{\gamma^2 \eta^4 \beta^3}\right)$ queries.

**Remark**

Does not estimate exactly $D(U_\gamma(x))$. 

---

Clément Canonne (Columbia University)  
Testing distributions with a COND oracle  
January 6, 2014  
29 / 31
Figure: (Rough) idea of the “binary descent” on $i$ for Approx-Eval: get an estimate of $D(i)$ by multiplying estimates at each branching.