Distribution Testing and Probabilistic Programming: A Match made in Heaven

Kuldeep S. Meel

School of Computing, National University of Singapore

Joint work with Sourav Chakraborty, Priyanka Golia, Yash Pote, and Mate Soos

Relevant publication: *AAAI-19, NeurIPS-20, NeurIPS-21, FMCAD-21, CP-22*

Probabilistic Programs and Distributions

```
program P
        X[1] = Bernoulli(p1);
        X[2] = Bernoulli(p2);
         ....
        X[n] = Bernoulli(pn);
        observe(X[2] + X[3] + X[7] > 2);
        return Y
```

- Semantics of discrete probabilistic programs: Distributions over $\{0,1\}^n$
- Distance Measures
  - Closeness
    - $d_\infty(P,Q) = \max_{x\in\{0,1\}^n} |P(x) - Q(x)|$
  - Farness
    - $d_{TV}(P,Q) = \frac{1}{2}\sum_{x\in\{0,1\}^n} |P(x) - Q(x)|$

The Road Ahead: Long and Winding

- Hardness of probabilistic program distance computation
- Grey-box Testing
- The Boon of Testers

Hardness of probabilistic program distance computation

Product Distributions

- Represented by list of probabilities: $\{p_1, p_2, \dots p_n\}$
- $P(x) = \prod_{x_i=1} p_i \prod_{x_i=0} (1 - p_i)$

Product Distributions

- Represented by list of probabilities: $\{p_1, p_2, \ldots p_n\}$
- $P(x) = \prod_{x_i=1} p_i \prod_{x_i=0} (1 - p_i)$

Question How hard it is to compute $d_{TV}(P, Q)$ when P and Q are product distributions

Product Distributions

- Represented by list of probabilities: $\{p_1, p_2, \dots p_n\}$
- $P(x) = \prod_{x_i=1} p_i \prod_{x_i=0} (1 - p_i)$

Question How hard it is to compute $d_{TV}(P, Q)$ when P and Q are product distributions

```
program P
        X[1] = Bernoulli(p1);
        X[2] = Bernoulli(p2);
        ....
        X[n] = Bernoulli(pn);
        return X;
```

```
program Q
        Y[1] = Bernoulli(q1);
        Y[2] = Bernoulli(q2);
        ....
        Y[n] = Bernoulli(qn);
        return Y
```

Hardness

Theorem

Given two product distributions P and Q, computation of $d_{TV}(P, Q)$ is #P-hard.

Hardness

Theorem

Given two product distributions P and Q, computation of $d_{TV}(P,Q)$ is $\#$P-hard.

- We reduce from $\#$Knapsack, i.e., we construct P and Q such that $\#\{x : P(x) = Q(x)\}$ will solve $\#$Knapsack

Hardness

Theorem

Given two product distributions P and Q, computation of $d_{TV}(P, Q)$ is #P-hard.

- We reduce from #Knapsack, i.e., we construct P and Q such that $\#\{x : P(x) = Q(x)\}$ will solve #Knapsack
- Now given P and Q, we construct P' and Q' where $p'_i = p_i$ and $q'_i = q_i$ for $i \leq n$ and $p'_{n+1} = \frac{1}{2} + \gamma$ and $q'_{n+1} = \frac{1}{2} - \gamma$ such that whenever $P(x) > Q(x)$ we have $P(x)(\frac{1}{2} - \gamma) > Q(x)(\frac{1}{2} + \gamma)$

$$
\begin{aligned}
d_{TV}(P', Q') = \sum_{x \in \{0,1\}^n} \bigg(&\left| P(x)(\frac{1}{2} + \gamma) - Q(x)(\frac{1}{2} - \gamma) \right| \\
&+ \left| P(x)(\frac{1}{2} - \gamma) - Q(x)(\frac{1}{2} + \gamma) \right| \bigg) \\
&= d_{TV}(P, Q) + 2\gamma \sum_{x : P(x) = Q(x)} P(x)
\end{aligned}
$$

Grey-box Testing

Basic Access

- We can run the program: sample!

What does Complexity Theory Tell Us

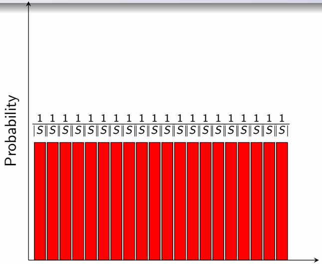


Figure: $\mathcal{U}$: Reference Distribution

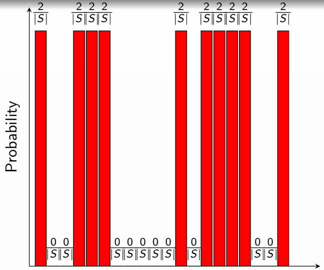


Figure: $\mathcal{A}$: 1/2-far from uniform

What does Complexity Theory Tell Us

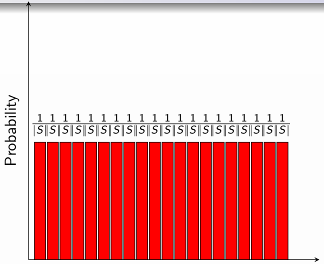


Figure: $\mathcal{U}$: Reference Distribution

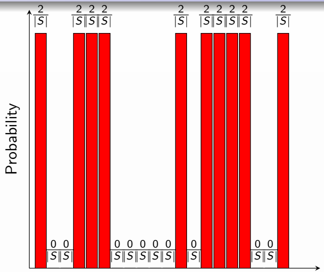


Figure: $\mathcal{A}$: 1/2-far from uniform

- If $< \sqrt{S}/100$ samples are drawn then with high probability you see only distinct samples from either distribution.

Theorem (Batu-Fortnow-Rubinfeld-Smith-White (JACM 2013))

Testing whether a distribution is ϵ-close to uniform has query complexity $\Theta(\sqrt{|S|}/\epsilon^2)$. [*Paninski (Trans. Inf. Theory 2008)*]

- If the output of a program is represented by 3 doubles, then $S > 2^{100}$

Definition (Conditional Sampling)

Given a distribution $\mathcal{A}$ on S one can

- *Specify a set $T \subseteq S$,*
- *Draw samples according to the distribution $\mathcal{A}|_T$, that is,*
  $\mathcal{A}$ under the condition that the samples belong to T.

Beyond Basic Access

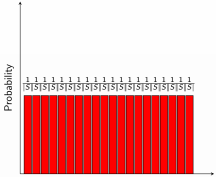
Definition (Conditional Sampling)

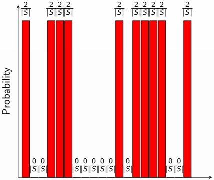
Given a distribution $\mathcal{A}$ on S one can

- *Specify a set $T \subseteq S$,*
- *Draw samples according to the distribution $\mathcal{A}|_T$, that is, $\mathcal{A}$ under the condition that the samples belong to T.*

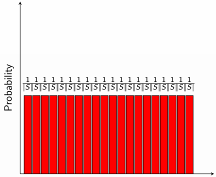
Conditional sampling is at least as powerful as drawing normal samples.
But how more powerful is it?

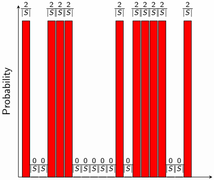
Testing Uniformity Using Conditional Sampling





Testing Uniformity Using Conditional Sampling

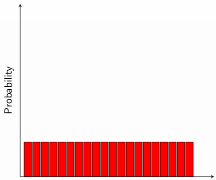


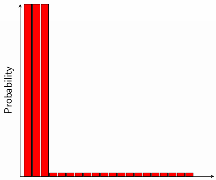


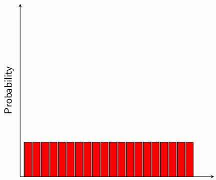
An algorithm for testing uniformity using conditional sampling:

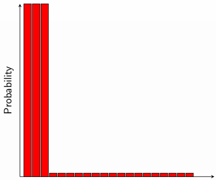
1. Draw two elements x and y uniformly at random from the domain. Let $T = \{x, y\}$.
2. In the case of the "far" distribution, with probability $\frac{1}{2}$, one of the two elements will have probability 0, and the other probability non-zero.
3. Note $\sqrt{|T|} = \sqrt{2}$ is a constant.
4. Now a constant number of conditional samples drawn from $\mathcal{A}|_T$ is enough to identify that it is not uniform.

What about other distributions?





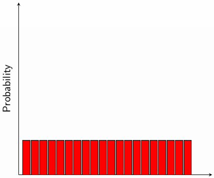


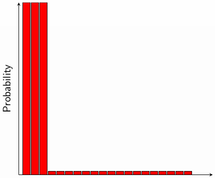


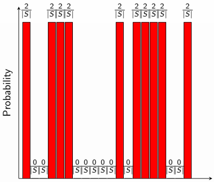
Previous algorithm fails in this case:

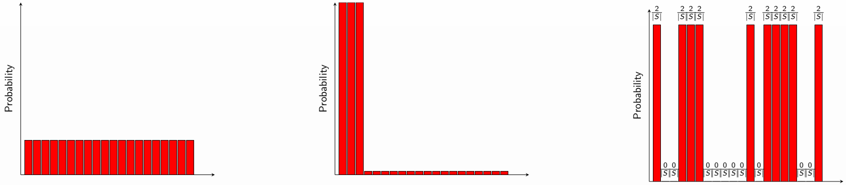
1. Draw two elements σ_1 and σ_2 uniformly at random from the domain. Let $T = \{\sigma_1, \sigma_2\}$.
2. In the case of the "far" distribution, with probability almost 1, both the two elements will have probability same, namely ϵ.
3. Probability that we will be able to distinguish the far distribution from the uniform distribution is very low.

Testing Uniformity Using Conditional Sampling









1. Draw σ_1 uniformly at random from the domain and draw σ_2 according to the distribution $\mathcal{A}$. Let $T = \{\sigma_1, \sigma_2\}$.
2. In the case of the "far" distribution, with constant probability, σ_1 will have "low" probability and σ_2 will have "high" probibility.
3. We will be able to distinguish the far distribution from the uniform distribution using constant number of samples from $\mathcal{A}|_T$.
4. The constant depend on the farness parameter.

Input: A Distribution under test $\mathcal{A}$, a reference uniform distribution $\mathcal{U}$, a tolerance parameter $\varepsilon > 0$, an intolerance parmaeter $\eta > \varepsilon$, a guarantee parameter δ
Output: ACCEPT or REJECT with the following guarantees:

- if the generator $\mathcal{A}$ is ε-close (in d_∞), i.e., $d_\infty(\mathcal{A},\mathcal{U}) \leq \varepsilon$ then Barbarik ACCEPTS with probability at least $(1 - \delta)$.
- If the generator $\mathcal{A}$ is η-far in (d_{TV}), i.e., $d_{TV}(\mathcal{A},\mathcal{U}) > \eta$, then Barbarik REJECTS with probability at least $1 - \delta$.

Sample complexity

Theorem

Given ε, η and δ, Barbarik need at most $K = \widetilde{O}(\frac{1}{(\eta-\varepsilon)^4})$ samples for any input formula φ, where the tilde hides a poly logarithmic factor of $1/\delta$ and $1/(\eta - \varepsilon)$.

- $\varepsilon = 0.6, \eta = 0.9, \delta = 0.1$
- Maximum number of required samples $K = 1.72 \times 10^6$
- Independent of the number of variables
- To Accept, we need K samples but rejection can be achieved with lesser number of samples.

Extensions to general case

Setting Given P and Q, test whether P and Q are ε-close (in d_∞) or η-far (in d_{TV})

- For all $\sigma_1, \sigma_2 \in 0, 1^n$, $\frac{P(\sigma_1)}{P(\sigma_2)}$ is known: A prior distribution conditioned by constraints: dependence on $\max_{\sigma_1, \sigma_2 \in 0,1^n} \frac{P(\sigma_1)}{P(\sigma_2)}$ (MPC20; PM21)
- Arbitrary P and Q (DCM22 – under submission)

The Boon of Testers

Where are the Samplers?

- Implicit representation of a set S: Set of all solutions of φ.
- Given a CNF formula φ, a Sampler $\mathcal{A}$, outputs a random solution of φ.

Definition

A CNF-Sampler, $\mathcal{A}$, is a randomized algorithm that, given a φ, outputs a random element of the set S, such that, for any $\sigma \in S$

$$\Pr[\mathcal{A}(\varphi) = \sigma] = \frac{1}{|S|},$$

Where are the Samplers?

- Implicit representation of a set S: Set of all solutions of φ.
- Given a CNF formula φ, a Sampler $\mathcal{A}$, outputs a random solution of φ.

Definition

A CNF-Sampler, $\mathcal{A}$, is a randomized algorithm that, given a φ, outputs a random element of the set S, such that, for any $\sigma \in S$

$$\Pr[\mathcal{A}(\varphi) = \sigma] = \frac{1}{|S|},$$

- Uniform sampling has wide range of applications in automated bug discovery, pattern mining, and so on.
- Several samplers available off the shelf: tradeoff between guarantees and runtime

Constrained Sampling

- Input formula: F over variables X
- Challenge: Conditional Sampling over $T = \{\sigma_1, \sigma_2\}$.
- Construct $G = F \wedge (X = \sigma_1 \vee X = \sigma_2)$

Constrained Sampling

- Input formula: F over variables X
- Challenge: Conditional Sampling over $T = \{\sigma_1, \sigma_2\}$.
- Construct $G = F \wedge (X = \sigma_1 \vee X = \sigma_2)$
- Most of the samplers enumerate all the points when the number of points in the Domain are small
- Need way to construct formulas whose solution space is large but every solution can be mapped to either σ_1 or σ_2.

Kernel

Input: A Boolean formula φ, two assignments σ_1 and σ_2, and desired number of solutions τ

Output: Formula $\hat{\varphi}$

1. $\tau = |\mathsf{Sol}(\hat{\varphi})|$
2. $Supp(\varphi) \subseteq \ Supp(\hat{\varphi})$
3. $z \in \mathsf{Sol}(\hat{\varphi}) \implies z_{\downarrow S} \in \{\sigma_1, \sigma_2\}$
4. $|\{z \in \mathsf{Sol}(\hat{\varphi}) \mid z_{\downarrow Supp(\varphi)} = \sigma_1\}| = |\{z \in \mathsf{Sol}(\hat{\varphi}) \mid z_{\downarrow Supp(\varphi)} \cap \sigma_2\}|$
5. φ and $\hat{\varphi}$ has similar structure

Non-adversarial Sampler

Let $(\hat{\varphi})$ obtained from $\mathit{kernel}(\varphi, \sigma_1, \sigma_2, N)$ such that there are only two set of assignments to variables in φ that can be extended to a satisfying assignment for $\hat{\varphi}$

Definition

The **non-adversarial sampler assumption** states that the distribution of the projection of samples obtained from $\mathcal{A}(\hat{\varphi})$ to variables of φ is same as the conditional distribution of $\mathcal{A}(\varphi)$ restricted to either σ_1 or σ_2

- If $\mathcal{A}$ is a uniform sampler for all the input formulas, it satisfies non-adversarial sampler assumption
- If $\mathcal{A}$ is not a uniform sampler for all the input formulas, it may not necessarily satisfy non-adversarial sampler assumption

Input: A sampler under test $\mathcal{A}$, a reference uniform sampler $\mathcal{U}$, a tolerance parameter $\varepsilon > 0$, an intolerance parmaeter $\eta > \varepsilon$, a guarantee parameter δ and a CNF formula φ
Output: ACCEPT or REJECT with the following guarantees:

- if the generator $\mathcal{A}$ is ε-close (in d_∞), i.e., $d_\infty(\mathcal{A},\mathcal{U}) \leq \varepsilon$ then Barbarik ACCEPTS with probability at least $(1-\delta)$.
- If the generator $\mathcal{A}$ is η-far in (d_{TV}), i.e., $d_{TV}(\mathcal{A},\mathcal{U}) > \eta$ and if non-adversarial sampler assumption holds then Barbarik REJECTS with probability at least $1-\delta$.

Experimental Setup

- Three state of the art (almost-)uniform samplers
  - UniGen2: Theoretical Guarantees of almost-uniformity
  - SearchTreeSampler: Very weak guarantees
  - QuickSampler: No Guarantees

- The study (in 2018) that proposed Quicksampler perform unsound statistical tests and claimed that all the three samplers are indistinguishable

Results-I

Instances	#Solutions	UniGen2		SearchTreeSampler	
		Output	#Samples	Output	#Samples
71	1.14×2^{59}	A	1729750	R	250
blasted_case49	1.00×2^{61}	A	1729750	R	250
blasted_case50	1.00×2^{62}	A	1729750	R	250
scenarios_aig_insertion1	1.06×2^{65}	A	1729750	R	250
scenarios_aig_insertion2	1.06×2^{65}	A	1729750	R	250
36	1.00×2^{72}	A	1729750	R	250
30	1.73×2^{72}	A	1729750	R	250
110	1.09×2^{76}	A	1729750	R	250
scenarios_tree_insert_insert	1.32×2^{76}	A	1729750	R	250
107	1.52×2^{76}	A	1729750	R	250
blasted_case211	1.00×2^{80}	A	1729750	R	250
blasted_case210	1.00×2^{80}	A	1729750	R	250
blasted_case212	1.00×2^{88}	A	1729750	R	250
blasted_case209	1.00×2^{88}	A	1729750	R	250
54	1.15×2^{90}	A	1729750	R	250

Results-II

Instances	#Solutions	UniGen2		QuickSampler	
		Output	#Samples	Output	#Samples
71	1.14×2^{59}	A	1729750	R	250
blasted_case49	1.00×2^{61}	A	1729750	R	250
blasted_case50	1.00×2^{62}	A	1729750	R	250
scenarios_aig_insertion1	1.06×2^{65}	A	1729750	R	250
scenarios_aig_insertion2	1.06×2^{65}	A	1729750	R	250
36	1.00×2^{72}	A	1729750	R	250
30	1.73×2^{72}	A	1729750	R	250
110	1.09×2^{76}	A	1729750	R	250
scenarios_tree_insert_insert	1.32×2^{76}	A	1729750	R	250
107	1.52×2^{76}	A	1729750	R	250
blasted_case211	1.00×2^{80}	A	1729750	R	250
blasted_case210	1.00×2^{80}	A	1729750	R	250
blasted_case212	1.00×2^{88}	A	1729750	R	250
blasted_case209	1.00×2^{88}	A	1729750	R	250
54	1.15×2^{90}	A	1729750	R	250

How can we use the availability of Barbarik to design a good sampler?

- Is it even possible ?

How can we use the availability of Barbarik to design a good sampler?

- Is it even possible ?

Wishlist

- Sampler should pass the Barbarik test.
- Sampler should be at least as fast as STS and QuickSampler.
- Sampler should perform good on real world applications.

CMSGen

- Exploits the flexibility CryptoMiniSat.

CMSGen

- Exploits the flexibility CryptoMiniSat.
- Pick polarities and branch on variables at random.
  - To explore the search space as evenly as possible.
  - To have samples over all the solution space.

CMSGen

- Exploits the flexibility CryptoMiniSat.
- Pick polarities and branch on variables at random.
  - To explore the search space as evenly as possible.
  - To have samples over all the solution space.
- Turn off all pre and inprocessing.
  - Processing techniques: bounded variable elimination, local search, or symmetry breaking, and many more.
  - Can change solution space of instances.

CMSGen

- Exploits the flexibility CryptoMiniSat.
- Pick polarities and branch on variables at random.
  - To explore the search space as evenly as possible.
  - To have samples over all the solution space.
- Turn off all pre and inprocessing.
  - Processing techniques: bounded variable elimination, local search, or symmetry breaking, and many more.
  - Can change solution space of instances.
- Restart at static intervals.
  - Helps to generate samples which are very hard to find.

Power of Test-Driven Development

- Test-Driven Development of CMSGen.
- Parameters of CMSGen are decided with the help of Barbarik
  - Iterative process.
  - Based on feedback from Barbarik, change the parameters.
- Uniform-like-sampler.
- Lack of theoretical analysis.

Testing of Samplers

- Samplers without guarantees (Uniform-like Samplers):
  - STS (Ermon, Gomes, Sabharwal, Selman,2012)
  - QuickSampler (Dutra, Laeufer, Bachrach, Sen, 2018)
- Sampler with guarantees:
  - UniGen3 (Chakraborty, Meel, and Vardi 2013, 2014,2015)

	QuickSampler	STS	UniGen3
ACCEPTs	0	14	50
REJECTs	50	36	0

Testing of Samplers

- Samplers without guarantees (Uniform-like Samplers):
  - STS (Ermon, Gomes, Sabharwal, Selman,2012)
  - QuickSampler (Dutra, Laeufer, Bachrach, Sen, 2018)
  - **CMSGen**
- Sampler with guarantees:
  - UniGen3 (Chakraborty, Meel, and Vardi 2013, 2014,2015)

	QuickSampler	STS	UniGen3	**CMSGen**
ACCEPTs	0	14	50	50
REJECTs	50	36	0	0

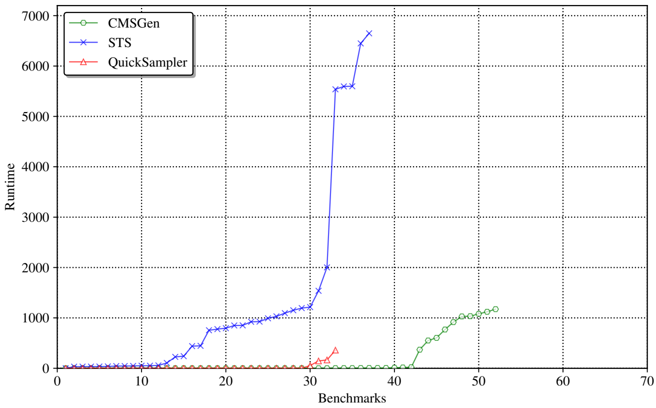
Wishlist

- Sampler should pass the Barbarik test. ✓
- Sampler should be at least as fast as STS and QuickSampler.
- Sampler should perform good on real world applications.

CMSGen vs. Other State-of-the-Art Samplers

- 70 Benchmarks arising from:
  - probabilistic reasoning, (Chakraborty, Fremont, Meel et al.,2015)
  - bounded model checking. (Clarke, Biere, Raimi, Zhu,2001)
  - bug synthesis. (Roy, Pandey, Dolan-Gavitt,Hu, 2018)
- Runtime evaluation to generate 1000 samples.
- Timeout: 7200 seconds.

CMSGen vs. Other State-of-the-Art Samplers (II)



QuickSampler	STS	CMSGen
33	37	52

Wishlist

- Sampler should pass the Barbarik test. ✓
- Sampler should be at least as fast as STS and QuickSampler. ✓
- Sampler should perform good on real world applications.

Combinatorial Testing

- A powerful paradigm for testing configurable system.
- Challenge: To generate test suites that maximizes t-wise coverage.

$$\text{t-wise coverage:} = \frac{\#\text{ of t-sized combinations in test suite}}{\text{all possible valid t-sized combinations}}$$

- To generate the test suites use constraint samplers.
- Uniform sampling to have high t-wise coverage (Plazar, Acher, Perrouin et al., 2019).

Combinatorial Testing

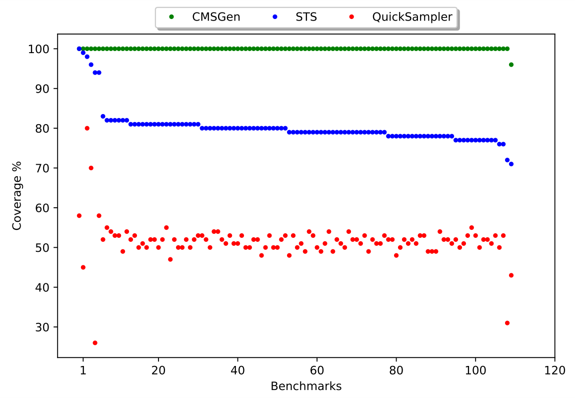
- A powerful paradigm for testing configurable system.
- Challenge: To generate test suites that maximizes t-wise coverage.

$$\text{t-wise coverage:} = \frac{\#\text{ of t-sized combinations in test suite}}{\text{all possible valid t-sized combinations}}$$

- To generate the test suites use constraint samplers.
- Uniform sampling to have high t-wise coverage (Plazar, Acher, Perrouin et al., 2019).
- Experimental Evaluations:
  - Generate 1000 samples (test cases).
  - 110 Benchmarks, Timeout: 3600 seconds
  - 2-wise coverage $t = 2$.

Combinatorial Testing: The Power of CMSGen

Higher is better



	QuickSampler	STS	CMSGen
Avg. Coverage	51.5%	80.15%	$\sim$ 100%

A Long and Winding Road

- Hardness of probabilistic program distance computation
- Distribution Testing
- The Boon of Testers

Open Source Tools

Barbarik `https://github.com/meelgroup/barbarik`

CMSGen `https://github.com/meelgroup/cmsgen`

A Long and Winding Road

- Hardness of probabilistic program distance computation
- Distribution Testing
- The Boon of Testers

Open Source Tools

Barbarik `https://github.com/meelgroup/barbarik`

CMSGen `https://github.com/meelgroup/cmsgen`

Where do we go from here?

- Exploring the tradeoffs: the cost of conditioning and query complexity
- Modular proofs
- Benchmark suite generation

These slides are available at `https://tinyurl.com/meel-talk`