Distribution Testing: The New Frontier for Formal Methods

Kuldeep S. Meel

University of Toronto

A joint adventure with **Sourav Chakraborty**, Arnab Bhattacharyya, Sutanu Gayen, Priyanka Golia, Dimitrios Myrisiotis, A. Pavan, Yash Pote, Mate Soos, and N. V. Vinodchandran

Relevant Papers: AAAI-19, FMCAD-21, CP-22, NeurIPS-21, NeurIPS-22, IJCAI-23

Distribution Testing: The New Frontier for Formal Methods

Kuldeep S. Meel

University of Toronto

A joint adventure with **Sourav Chakraborty**, Arnab Bhattacharyya, Sutanu Gayen, Priyanka Golia, Dimitrios Myrisiotis, A. Pavan, Yash Pote, Mate Soos, and N. V. Vinodchandran

Relevant Papers: AAAI-19, FMCAD-21, CP-22, NeurIPS-21, NeurIPS-22, IJCAI-23

Summary: A new problem space with opportunities for exciting theory, algorithms, and systems with practical impact.

A Brief (and Biased) History of Computing

Church - Turing Thesis, 1930's: The notion of computability

von Neumann Architecture, 1945: Early hardware implementations

A Brief (and Biased) History of Computing

Church - Turing Thesis, 1930's: The notion of computability

von Neumann Architecture, 1945: Early hardware implementations

Scott-Rabin, 1958: Finite Automaton; the notion of non-determinism in automata

Floyd; Hoare, 1968-69 Assigning meanings to programs; $\{P\}C\{Q\}$

A Brief (and Biased) History of Computing

Church - Turing Thesis, 1930's: The notion of computability

von Neumann Architecture, 1945: Early hardware implementations

Scott-Rabin, 1958: Finite Automaton; the notion of non-determinism in automata

Floyd; Hoare, 1968-69 Assigning meanings to programs; $\{P\}C\{Q\}$

Pneulii, 1977: Introduction of Linear Temporal Logic to CS

Clarke-Emerson; Quielle and Sifakis, 1981 : Birth of Model Checking

A Brief (and Biased) History of Computing

Church - Turing Thesis, 1930's: The notion of computability

von Neumann Architecture, 1945: Early hardware implementations

Scott-Rabin, 1958: Finite Automaton; the notion of non-determinism in automata

Floyd; Hoare, 1968-69 Assigning meanings to programs; $\{P\}C\{Q\}$

Pneulii, 1977: Introduction of Linear Temporal Logic to CS

Clarke-Emerson; Quielle and Sifakis, 1981 : Birth of Model Checking

Fundamental Aspect: Every execution of the program must satisfy the specification

- A single (or constantly many) execution suffices as witness for falsifiability

The Start of Automated Reasoning Revolution: BDDs, SAT, and Beyond SAT

Boolean Satisfiability

Boolean Satisfiability (SAT); Given a Boolean expression, using "and" ($\wedge$) "or", ($\vee$) and "not" ($\neg$), *is there a satisfying solution* (an assignment of 0's and 1's to the variables that makes the expression equal 1)?
Example:

$$(\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4) \wedge (x_3 \vee x_1 \vee x_4)$$

Solution: $x_1 = 0$, $x_2 = 0$, $x_3 = 1$, $x_4 = 1$

The Story of SAT Revolution

The Progress over the years

- Late-90s: Few hundreds of variables and clauses
- Now: **Millions** of variables and clauses

Theoretical Advances + Algorithmic Engineering + Software Development

Knuth, 2016: "The story of satisfiability is a tale of the triumph of software engineering blended with rich doses of beautiful mathematics."

Many Industrial Applications: Hardware and Software verification, Security, Planning, Compliance, Telecom Feature Subscription, Bioinformatics, · · ·

B. Cook, 2022: **Virtuous cycle in Automated Reasoning**: ...application areas drives more investment in foundational tools, while improvements in the foundational tools drive further applications. Around and around.

The Story of SAT Revolution

The Progress over the years

- Late-90s: Few hundreds of variables and clauses
- Now: **Millions** of variables and clauses

Theoretical Advances + Algorithmic Engineering + Software Development

Knuth, 2016: "The story of satisfiability is a tale of the triumph of software engineering blended with rich doses of beautiful mathematics."

Many Industrial Applications: Hardware and Software verification, Security, Planning, Compliance, Telecom Feature Subscription, Bioinformatics, $\cdots$

B. Cook, 2022: **Virtuous cycle in Automated Reasoning**: ...application areas drives more investment in foundational tools, while improvements in the foundational tools drive further applications. Around and around.

Beyond Non-determinism: Power of Randomization

Erdos, 1959: Probabilistic Method in Graph Theory

Solovay and Strassen; Rabin, 1976: Checking primality of a number

Gill, 1977: Computational Complexity of Probabilistic Turing Machines

Beyond Non-determinism: Power of Randomization

Erdos, 1959: Probabilistic Method in Graph Theory

Solovay and Strassen; Rabin, 1976: Checking primality of a number

Gill, 1977: Computational Complexity of Probabilistic Turing Machines

Carter-Wegman, 1977: Strongly Universal Hash Functions

Morris, 1978: Probabilistic Counting

Beyond Non-determinism: Power of Randomization

Erdos, 1959: Probabilistic Method in Graph Theory

Solovay and Strassen; Rabin, 1976: Checking primality of a number

Gill, 1977: Computational Complexity of Probabilistic Turing Machines

Carter-Wegman, 1977: Strongly Universal Hash Functions

Morris, 1978: Probabilistic Counting

And then everything changed in 1980's and world was never the same

Randomization as a Core Ingredient: Distributed Computing, Cryptography, Testing, Streaming, and Machine Learning

With Prevalence comes the opportunity for Formal Methods

How do we test and verify randomness?

- How do we know python's implementation of random is correct?
- How do we know constrained samplers used in testing are generating from desired distributions?

With Prevalence comes the opportunity for Formal Methods

How do we test and verify randomness?

- How do we know python's implementation of random is correct?
- How do we know constrained samplers used in testing are generating from desired distributions?

What is different from traditional model checking

- A single (even, constants many) execution do not suffice as witness for falsifiability.
- Simple verification problems for probabilistic systems are #P-hard, compared to NP-hardness for (non)-deterministic programs [BGMMPV22]

With Prevalence comes the opportunity for Formal Methods

How do we test and verify randomness?

- How do we know python's implementation of random is correct?
- How do we know constrained samplers used in testing are generating from desired distributions?

What is different from traditional model checking

- A single (even, constants many) execution do not suffice as witness for falsifiability.
- Simple verification problems for probabilistic systems are #P-hard, compared to NP-hardness for (non)-deterministic programs [BGMMPV22]

Is there any hope?

With Prevalence comes the opportunity for Formal Methods

How do we test and verify randomness?

- How do we know python's implementation of random is correct?
- How do we know constrained samplers used in testing are generating from desired distributions?

What is different from traditional model checking

- A single (even, constants many) execution do not suffice as witness for falsifiability.
- Simple verification problems for probabilistic systems are #P-hard, compared to NP-hardness for (non)-deterministic programs [BGMMPV22]

Is there any hope?

Yes; We can build on the progress in the subfield of distribution testing in theoretical CS community

Distribution Testing: A "subfield, at the junction of property testing and Statistics, is concerned with studying properties of probability distributions." [Canonne, 2020]

Outline

Q1 What do distributions look like in the real world?

Q2 What properties matter to the practitioners?

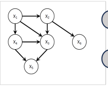
Q3 Theory and Practice of Distirbution Testing

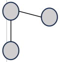
- Entropy Estimation [Part I]
- Complexity of Distance Estimation [Part II]
- Greybox Testing: Constrained Samplers [Part III]

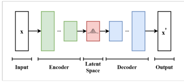
Q4 Can distribution testing influence the design of systems ? [Part IV]

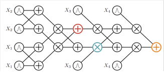
Q1: Distributions in Real World

Generative Probabilistic Models









Q1: Distributions in Real World: II

Constrained Random Simulation: Test Vector Generation

- Dominant methodology to test hardware systems
- Use a formula φ to encode the verification scenarios
- A Constrained Sampler $\mathcal{A}$ takes φ as input and returns $\sigma \in \mathsf{Sol}(\varphi)$, and ideally ensures

$$\Pr[\sigma \leftarrow \mathcal{A}(\varphi)] = \frac{1}{|\mathsf{Sol}(\varphi)|}$$

Constrained Samplers

- Even finding just a single satisfying assignment is NP-hard
- A well-studied problem by theoreticians and practitioners alike for nearly 40 years
- Only in 2010's, we could have samplers with theoretical guarantees and "reasonable" performance
  - Well, not really reasonable from practical perspective
- Design of *practical* samplers based on MCMC, random walk, local search etc.

Goal: Develop sound procedures to distinguish samplers (if possible).

Outline

Q1 What do distributions look like in the real world?

Q2 What properties matter to the practitioners?

Q3 How to develop practical scalable testers for distributions?

Q4 Can distribution testing influence the design of systems ?

Q2: Properties that Matter

(Approximate) Equivalence Checking

White-box Setting

- Is a given probabilitic generative model closer to a desired model?
- Consider a probablistic program $\mathcal{P}$ and say a compiler transforms $\mathcal{P}$ into $\mathcal{Q}$:
  - Is $\mathcal{Q}$ close to $\mathcal{P}$?

```
var x = sample(RandomInteger({n: 2**n}));
return x;
```

Listing: Program 1

```
var eps = 0.3;
var test = sample(Bernoulli({p: (1 - eps) / 2}));
if (test == 1) {
  var x = sample(RandomInteger({n: 2**(n-1)}));
} else {
  var y = sample(RandomInteger({n: 2**(n-1)}));
  var x = y + 2**(n-1);
}
return x;
```

Listing: Program 2, which is close to Program 1

Q2: Properties that Matter

Grey-box Setting

- (Fast) Sampler $\mathcal{A}$ and a reference (but, often slow) sampler $\mathcal{U}$
- Reference sampler $\mathcal{U}$ is certified to produce samples according to desired distribution but is slow.
- Is the distribution generated by $\mathcal{A}$, denoted by $\mathcal{A}_\varphi$, close to that of $\mathcal{U}_\varphi$?

How to Measure Equivalence

Consider two distribution $\mathcal{P}$ and $\mathcal{Q}$ over $\{0,1\}^n$.

Two Notions of Distance

- d_∞ distance: $\max_{\sigma \in \{0,1\}^n} |\mathcal{P}(\sigma) - \mathcal{Q}(\sigma)|$
  - The most commonly seen behavior where a developer wants to approximate $\mathcal{P}$ with another distribution $\mathcal{Q}$
  - Almost-uniform sampling in the context of constrained random simulation

How to Measure Equivalence

Consider two distribution $\mathcal{P}$ and $\mathcal{Q}$ over $\{0,1\}^n$.

Two Notions of Distance

- d_∞ distance: $\max_{\sigma \in \{0,1\}^n} |\mathcal{P}(\sigma) - \mathcal{Q}(\sigma)|$
  - The most commonly seen behavior where a developer wants to approximate $\mathcal{P}$ with another distribution $\mathcal{Q}$
  - Almost-uniform sampling in the context of constrained random simulation

- Total Variation Distance (d_{TV}) or L_1 distance: $\frac{1}{2} \sum_{\sigma \in \{0,1\}^n} |\mathcal{P}(\sigma) - \mathcal{Q}(\sigma)|$
  - Consider any arbitrary program $\mathcal{A}$ that uses samples from a distribution: there is a probability distribution over output of $\mathcal{A}$.
  - Consider a Bad event over the output of $\mathcal{A}$: such as not catching a bug.
  - Let's say $\mathcal{A}$ samples from $\mathcal{P}$.
  - Folklore: If we were to replace $\mathcal{P}$ with $\mathcal{Q}$ then the probability of Bad event would increase/decrease at most by $d_{TV}(P, Q)$.

How to Measure Equivalence

Consider two distribution $\mathcal{P}$ and $\mathcal{Q}$ over $\{0,1\}^n$.

Two Notions of Distance

- d_∞ distance: $\max_{\sigma \in \{0,1\}^n} |\mathcal{P}(\sigma) - \mathcal{Q}(\sigma)|$
  - The most commonly seen behavior where a developer wants to approximate $\mathcal{P}$ with another distribution $\mathcal{Q}$
  - Almost-uniform sampling in the context of constrained random simulation
- Total Variation Distance (d_{TV}) or L_1 distance: $\frac{1}{2} \sum_{\sigma \in \{0,1\}^n} |\mathcal{P}(\sigma) - \mathcal{Q}(\sigma)|$
  - Consider any arbitrary program $\mathcal{A}$ that uses samples from a distribution: there is a probability distribution over output of $\mathcal{A}$.
  - Consider a Bad event over the output of $\mathcal{A}$: such as not catching a bug.
  - Let's say $\mathcal{A}$ samples from $\mathcal{P}$.
  - Folklore: If we were to replace $\mathcal{P}$ with $\mathcal{Q}$ then the probability of Bad event would increase/decrease at most by $d_{TV}(P, Q)$.

Therefore, measure closeness with respect to d_∞ and farness with respect to d_{TV}

- Checker should return Accept if two distributions are close in d_∞-distance and return Reject if two distributions are far in d_{TV}.

Outline

Theory and Practice of Distirbution Testing

Probabilistic Generative Models

- Complexity of Distance Estimation for Probabilistic Generative Models [Topic I]

Constrained Samplers

- Greybox Testing: Constrained Samplers [Topic II]
- Can distribution testing influence the design of systems ? [Topic III]

Probability Basics (I)

- A random process: Tossing a coin
  - $p = 0.4$ (probability of Heads)

- A random variable: assign a numerical value for outcome of random process
  - $X = +1$ if Heads and 0 if Tails

- Expectation $\mu = E[X]$

Probability Basics (I)

- A random process: Tossing a coin
  - $p = 0.4$ (probability of Heads)

- A random variable: assign a numerical value for outcome of random process
  - $\mathrm{X} = +1$ if Heads and 0 if Tails

- Expectation $\mu = \mathrm{E}[X]$
  - $\mathrm{E}[X] = 1 \cdot p + 0 \cdot (1 - p) = p$

- Variance $\sigma^2[X] = \mathrm{E}[(X - \mu)^2] = \mathrm{E}[X^2] - \mu^2$

Probability Basics (I)

- A random process: Tossing a coin
  - $p = 0.4$ (probability of Heads)

- A random variable: assign a numerical value for outcome of random process
  - X $= +1$ if Heads and 0 if Tails

- Expectation $\mu = \mathrm{E}[X]$
  - $\mathrm{E}[X] = 1 \cdot p + 0 \cdot (1 - p) = p$

- Variance $\sigma^2[X] = \mathrm{E}[(X - \mu)^2] = \mathrm{E}[X^2] - \mu^2$
  - $\sigma^2[X] = p - p^2$
- Two variables X and Y are independent, if $\Pr[X|Y] = \Pr[X]$

Probability Basics (II)

Hoeffding Bound Let $X_1, X_2, \cdots X_n$ be independent and identically distributed variables such that $p = \mathrm{E}[X_i]$ and let $X = (\sum_i X_i)/n$. Then

$$\mu = \mathrm{E}[X] = p$$

$$\Pr[|X - \mu| \geq \beta\mu] \leq e^{-\beta^2 \mu/3}$$

Probability Basics (II)

Hoeffding Bound Let $X_1, X_2, \cdots X_n$ be independent and identically distributed variables such that $p = \mathrm{E}[X_i]$ and let $X = (\sum_i X_i)/n$. Then

$$\mu = \mathrm{E}[X] = p$$

$$\Pr[|X - \mu| \geq \beta\mu] \leq e^{-\beta^2 \mu/3}$$

$\varepsilon - \delta$ approximation A random variable Z is a (ε, δ)-approximation of a quantity k if the following holds:

$$\Pr[Z \in (1 \pm \varepsilon)k] \geq 1 - \delta$$

Probability Basics (II)

Hoeffding Bound Let $X_1, X_2, \cdots X_n$ be independent and identically distributed variables such that $p = \mathrm{E}[X_i]$ and let $X = (\sum_i X_i)/n$. Then

$$\mu = \mathrm{E}[X] = p$$

$$\Pr[|X - \mu| \geq \beta\mu] \leq e^{-\beta^2 \mu/3}$$

$\varepsilon - \delta$ approximation A random variable Z is a (ε, δ)-approximation of a quantity k if the following holds:

$$\Pr[Z \in (1 \pm \varepsilon)k] \geq 1 - \delta$$

Problem Suppose we are given a coin and we want to estimate it's bias (i.e., the probability of heads). How many samples (i.e., tosses) do we need?

Probability Basics (II)

Hoeffding Bound Let $X_1, X_2, \cdots X_n$ be independent and identically distributed variables such that $p = \mathrm{E}[X_i]$ and let $X = (\sum_i X_i)/n$. Then

$$\mu = \mathrm{E}[X] = p$$

$$\Pr[|X - \mu| \geq \beta\mu] \leq e^{-\beta^2 \mu/3}$$

$\varepsilon - \delta$ approximation A random variable Z is a (ε, δ)-approximation of a quantity k if the following holds:

$$\Pr[Z \in (1 \pm \varepsilon)k] \geq 1 - \delta$$

Problem Suppose we are given a coin and we want to estimate it's bias (i.e., the probability of heads). How many samples (i.e., tosses) do we need?

Zero-One Estimator Let X be a random variable with $\mu = \mathrm{E}[X]$, then $O\left(\frac{1}{\varepsilon^2 \mu} \log(1/\delta)\right)$ samples are sufficient to estimate μ within (ε, δ)-factor.

Probability Basics (III): Couplings

- Given P and Q on a common domain, a **coupling** C is a distribution on pairs (X, Y) such that X distributed as P and Y distributed as Q.

Example

- Suppose $P = \mathrm{Bernoulli}(2/3)$ and $Q = \mathrm{Bernoulli}(1/3)$. $d_{TV}(P, Q)$

Probability Basics (III): Couplings

- Given P and Q on a common domain, a **coupling** C is a distribution on pairs (X, Y) such that X distributed as P and Y distributed as Q.

  Example

  - Suppose $P = \mathrm{Bernoulli}(2/3)$ and $Q = \mathrm{Bernoulli}(1/3)$. $d_{TV}(P, Q) = 1/3$
  - If $X \sim P$ and $Y \sim Q$ independently, then $\Pr[X \neq Y] = \frac{2}{3} \cdot \frac{2}{3} + \frac{1}{3} \cdot \frac{1}{3} = \frac{5}{9}$
  - If $X \sim P$ and $Y = 1 - X$, then $\Pr[X \neq Y] = 1$
  - $X \sim P$ and $Y = \mathrm{Bernoulli}(1/2)$ if $X = 1$ else $Y = 0$, then $\Pr[X \neq Y] = \frac{1}{2} \cdot \frac{2}{3} = \frac{1}{3}$

Outline

Theory and Practice of Distirbution Testing

Probabilistic Generative Models

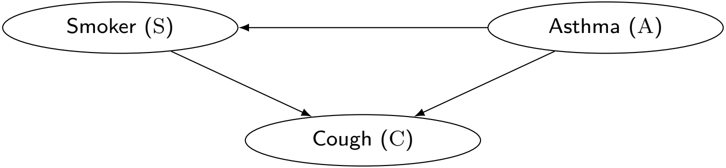
- **Complexity of Distance Estimation for Probabilistic Generative Models**

Constrained Samplers

- Greybox Testing: Constrained Samplers
- Can distribution testing influence the design of systems ?

Examples of Distributions

Bayesian Networks



Product Distributions

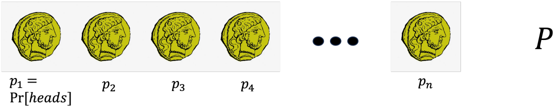


Figure: A network with no dependencies

TV Distance Computation

Consider P and Q as distributions on $\{0,1\}^n$.

$$d_{TV}(P,Q) = \frac{1}{2}||P-Q||_1 = \sum_{\sigma} |P(\sigma) - Q(\sigma)|$$

How hard is to compute TV Distance?

Theorem (BGMMPV-23)

TV Distance computation between two product distribution is #P-hard

- Given two circuits, checking their equivalence is just NP-hard
- #P-hard contains entire polynomial hierarchy

Technical Overview

#SubsetProd: Given integers $a_1, \ldots a_n$ and T, find

$$\left|\left\{ S \subseteq [n] : \prod_{i \in S} a_i = T \right\}\right|$$

#PMFEquals: Given $p_1, \ldots p_n, v \in [0,1]$ where $p_1, \ldots p_n$ are the parameters of a product distribution P, find

$$|\{x \in \{0,1\}^n : P(x) = v\}|$$

$$\#\text{SubsetProd} \leq \#\text{PMFEquals} \leq d_{TV}$$

#PMFEquals $\leq d_{TV}$

#PMFEquals: Given $p_1, \dots p_n, v \in [0,1]$ where $p_1, \dots p_n$ are the parameters of a product distribution P, find

$$|\{x \in \{0,1\}^n : P(x) = v\}|$$

For simplicity, assume $v \leq 2^{-n}$

$\#\text{PMFEquals} \leq d_{TV}$

#PMFEquals: Given $p_1, \ldots p_n, v \in [0, 1]$ where $p_1, \ldots p_n$ are the parameters of a product distribution P, find

$$|\{x \in \{0, 1\}^n : P(x) = v\}|$$

For simplicity, assume $v \leq 2^{-n}$

Define distributions $\hat{P}$ and $\hat{Q}$ on $n + 1$ bits

- $\hat{p}_i = p_i$ for $i \leq n$ and $\hat{p}_{n+1} = 1$
- $\hat{q}_i = 1/2$ for $i \leq n$ and $\hat{q}_{n+1} = v2^n$

#PMFEquals $\leq d_{TV}$

#PMFEquals: Given $p_1, \ldots p_n, v \in [0,1]$ where $p_1, \ldots p_n$ are the parameters of a product distribution P, find

$$|\{x \in \{0,1\}^n : P(x) = v\}|$$

For simplicity, assume $v \leq 2^{-n}$

Define distributions $\hat{P}$ and $\hat{Q}$ on $n+1$ bits

- $\hat{p}_i = p_i$ for $i \leq n$ and $\hat{p}_{n+1} = 1$
- $\hat{q}_i = 1/2$ for $i \leq n$ and $\hat{q}_{n+1} = v2^n$

Define Distributions P' and Q' on $n+2$ bits

- $p'_i = p_i$ for $i \leq n$, $p'_{n+1} = 1$ and $p'_{n+2} = \frac{1}{2} + \beta$
- $q'_i = q_i$ for $i \leq n$, $q'_{n+1} = 1$ and $q'_{n+2} = \frac{1}{2} + \beta$

where β is very small.

$\#$PMFEquals $\leq d_{TV}$

$\#$PMFEquals: Given $p_1, \ldots p_n, v \in [0,1]$ where $p_1, \ldots p_n$ are the parameters of a product distribution P, find

$$|\{x \in \{0,1\}^n : P(x) = v\}|$$

For simplicity, assume $v \leq 2^{-n}$

Define distributions $\hat{P}$ and $\hat{Q}$ on $n+1$ bits

- $\hat{p}_i = p_i$ for $i \leq n$ and $\hat{p}_{n+1} = 1$
- $\hat{q}_i = 1/2$ for $i \leq n$ and $\hat{q}_{n+1} = v2^n$

Define Distributions P' and Q' on $n+2$ bits

- $p'_i = p_i$ for $i \leq n$, $p'_{n+1} = 1$ and $p'_{n+2} = \frac{1}{2} + \beta$
- $q'_i = q_i$ for $i \leq n$, $q'_{n+1} = 1$ and $q'_{n+2} = \frac{1}{2} + \beta$

where β is very small.
Claim:

$$|\{x : P(x) = v\}| = \frac{d_{TV}(P', Q') - d_{TV}(\hat{P}, \hat{Q})}{2\beta v}$$

$\#\text{PMFEquals} \leq d_{TV}$

Define distributions $\hat{P}$ and $\hat{Q}$ on $n+1$ bits

- $\hat{p}_i = p_i$ for $i \leq n$ and $\hat{p}_{n+1} = 1$
- $\hat{q}_i = 1/2$ for $i \leq n$ and $\hat{q}_{n+1} = v2^n$

#PMFEquals $\leq d_{TV}$

Define distributions $\hat{P}$ and $\hat{Q}$ on $n+1$ bits

- $\hat{p}_i = p_i$ for $i \leq n$ and $\hat{p}_{n+1} = 1$
- $\hat{q}_i = 1/2$ for $i \leq n$ and $\hat{q}_{n+1} = v2^n$

$$d_{TV}(\hat{P}, \hat{Q}) = \sum_{x:P(x)>v} P(x) - v$$

#PMFEquals $\leq d_{TV}$

Define distributions $\hat{P}$ and $\hat{Q}$ on $n+1$ bits

- $\hat{p}_i = p_i$ for $i \leq n$ and $\hat{p}_{n+1} = 1$
- $\hat{q}_i = 1/2$ for $i \leq n$ and $\hat{q}_{n+1} = v2^n$

$$d_{TV}(\hat{P}, \hat{Q}) = \sum_{x:P(x)>v} P(x) - v$$

Define Distributions P' and Q' on $n+2$ bits

- $p'_i = p_i$ for $i \leq n$, $p'_{n+1} = 1$ and $p'_{n+2} = \frac{1}{2} + \beta$
- $q'_i = q_i$ for $i \leq n$, $q'_{n+1} = 1$ and $q'_{n+2} = \frac{1}{2} - \beta$

where β is very small.

$$d_{TV}(P', Q') = 2\beta v \cdot |\{x : P(x) = v\}| + \sum_{x:P(x)>v} P(x) - v$$

$$|\{x : P(x) = v\}| = \frac{d_{TV}(P', Q') - d_{TV}(\hat{P}, \hat{Q})}{2\beta v}$$

What about Approximation?

Theorem: The (ε, δ)-approximation of TV distance between two product distributions P and Q can be accomoplished in $O(\frac{n^2}{\varepsilon^2} \log(1/\delta))$ time.

- Algorithm boils down to a simple but clever Monte Carlo estimator
- Based on dual characterization of TV distance in terms of couplings.

Couplings

- Given P and Q on a common domain, a **coupling** C is a distribution on pairs (X, Y) such that X distributed as P and Y distributed as Q.

Example

- Suppose $P = \mathrm{Bernoulli}(2/3)$ and $Q = \mathrm{Bernoulli}(1/3)$. $d_{TV}(P, Q)$

Couplings

- Given P and Q on a common domain, a **coupling** C is a distribution on pairs (X, Y) such that X distributed as P and Y distributed as Q.

  Example

  - Suppose $P = \mathrm{Bernoulli}(2/3)$ and $Q = \mathrm{Bernoulli}(1/3)$. $d_{TV}(P, Q) = 1/3$
  - If $X \sim P$ and $Y \sim Q$ independently, then $\Pr[X \neq Y] = \frac{2}{3} \cdot \frac{2}{3} + \frac{1}{3} \cdot \frac{1}{3} = \frac{5}{9}$
  - If $X \sim P$ and $Y = 1 - X$, then $\Pr[X \neq Y] = 1$
  - $X \sim P$ and $Y = \mathrm{Bernoulli}(1/2)$ if $X = 1$ else $Y = 0$, then $\Pr[X \neq Y] = \frac{1}{2} \cdot \frac{2}{3} = \frac{1}{3}$
- An optimal coupling O satisifes:

$$\Pr_{(X,Y)\sim O}[X \neq Y] = d_{TV}(P, Q)$$

- In an optimal coupling O, for any w,

$$\Pr_O[X = Y = w] = \min(P(w), Q(w))$$

Coupling between Product Distributions

- Consider P and Q product distributions on $\{0,1\}^n$. Coupling between them is a distribution on $(\{0,1\}^n)^2$
- Let O_i be optimal coupling between i-th marginals, P_i and Q_i. Then, $C = O_1 \otimes O_2 \otimes \ldots \otimes O_n$ is a local coupling

  Example:

  - Suppose $P = \mathrm{Bernoulli}(2/3) \otimes \mathrm{Bernoulli}(2/3)$ and $Q = \mathrm{Bernoulli}(1/2) \otimes \mathrm{Bernoulli}(1/3)$.
  - $d_{TV}(P, Q)$

Coupling between Product Distributions

- Consider P and Q product distributions on $\{0,1\}^n$. Coupling between them is a distribution on $(\{0,1\}^n)^2$
- Let O_i be optimal coupling between i-th marginals, P_i and Q_i. Then, $C = O_1 \otimes O_2 \otimes \ldots \otimes O_n$ is a local coupling

  Example:
  - Suppose $P = \mathrm{Bernoulli}(2/3) \otimes \mathrm{Bernoulli}(2/3)$ and $Q = \mathrm{Bernoulli}(1/2) \otimes \mathrm{Bernoulli}(1/3)$.
  - $d_{TV}(P,Q) = 1/3$
  - If (X,Y) form a local coupling, then $\Pr[X \neq Y] = \frac{5}{9}$
- Local coupling may not be an optimal coupling
- But, it's easy to sample from local coupling

The Power of Local Coupling

Observation If C is local coupling, then
$\Pr_C[X \neq Y] \leq \sum_i \Pr_C(X_i \neq Y_i) \leq \sum_i d_{TV}(P_i, Q_i)$.

The Power of Local Coupling

Observation If C is local coupling, then
$\Pr_C[X \neq Y] \leq \sum_i \Pr_C(X_i \neq Y_i) \leq \sum_i d_{TV}(P_i, Q_i)$.

Key Lemma For Product distributions P and Q, we have

$$\max_i d_{TV}(P_i, Q_i) \leq d_{TV}(P, Q) \leq \sum_i d_{TV}(P_i, Q_i)$$

The Power of Local Coupling

Observation If C is local coupling, then
$\Pr_C[X \neq Y] \leq \sum_i \Pr_C(X_i \neq Y_i) \leq \sum_i d_{TV}(P_i, Q_i)$.

Key Lemma For Product distributions P and Q, we have

$$\max_i d_{TV}(P_i, Q_i) \leq d_{TV}(P, Q) \leq \sum_i d_{TV}(P_i, Q_i)$$

Let $\alpha = \frac{d_{TV}(P,Q)}{\Pr_C[X \neq Y]} = \frac{\Pr_O[X \neq Y]}{\Pr_C[X \neq Y]}$

We have

$$\frac{1}{n} \leq \alpha \leq 1$$

The Power of Local Coupling

Observation If C is local coupling, then
$\Pr_C[X \neq Y] \leq \sum_i \Pr_C(X_i \neq Y_i) \leq \sum_i d_{TV}(P_i, Q_i)$.

Key Lemma For Product distributions P and Q, we have

$$\max_i d_{TV}(P_i, Q_i) \leq d_{TV}(P, Q) \leq \sum_i d_{TV}(P_i, Q_i)$$

Let $\alpha = \frac{d_{TV}(P,Q)}{\Pr_C[X \neq Y]} = \frac{\Pr_O[X \neq Y]}{\Pr_C[X \neq Y]}$

We have

$$\frac{1}{n} \leq \alpha \leq 1$$

Key Idea The denominator $\Pr_C[X \neq Y]$ is easy to compute.

$$\Pr_C[X \neq Y] = 1 - \Pr_C(X = Y) = 1 - \prod_i (1 - d_{TV}(P_i, Q_i))$$

$$\alpha = \frac{\Pr_O[X \neq Y]}{\Pr_C[X \neq Y]}$$

Estimator for $\boldsymbol{\alpha}$

Define $f(w) = P(w) - \min(P(w), Q(w))$.

Define $g(w) = P(w) - \prod_i \min\big(P_i(w_i), Q_i(w_i)\big)$.

Define π to be distribution with mass function proportional to g.

Note that $0 \leq f(w) \leq g(w)$ for all w.

$$\alpha = \frac{\Pr_O[X \neq Y]}{\Pr_C[X \neq Y]}$$

Define $f(w) = P(w) - \min(P(w), Q(w))$.

Define $g(w) = P(w) - \prod_i \min\big(P_i(w_i), Q_i(w_i)\big)$.

Define π to be distribution with mass function proportional to g.

- $\sum_w f(w) = \Pr_O[X \neq Y]$.
- $\sum_w g(w) = \Pr_C[X \neq Y]$.
- Therefore,

$$\alpha = \frac{\sum_w f(w)}{\sum_w g(w)} = \mathbb{E}_\pi\left[\frac{f(w)}{g(w)}\right].$$

- Sampling from π reduces to computing $\sum_w g(w)$ which we know how to do efficiently.

Outline

Theory and Practice of Distirbution Testing

Probabilistic Generative Models

- Complexity of Distance Estimation for Probabilistic Generative Models [✓]

Constrained Samplers

- **Greybox Testing: Constrained Samplers**
- Can distribution testing influence the design of systems ?

Problem Setting

- A Boolean formula φ
- Reference Sampler $\mathcal{U}$
  - With rigorous theoretical guarantees but often slower
- Sampler Under Test: A sampler $\mathcal{A}$ that claims to be close to uniform sampler for formulas in benchmark set
  - Superior runtime performance but often no theoretical analysis
- Closeness and farness parameters: ε and η

Task: Determine whether distributions $\mathcal{A}_\varphi$ and $\mathcal{U}_\varphi$ are ε-close or η-far

Limitations of Black-Box Testing

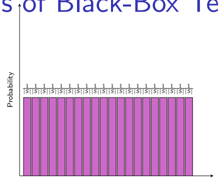


Figure: $\mathcal{U}_\varphi$: Uniform Distribution

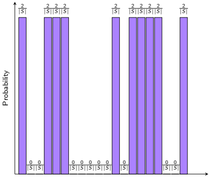


Figure: $\mathcal{A}_\varphi$: 1/2-far from uniform

SAMP: Allows you to draw samples from a distribution

Limitations of Black-Box Testing

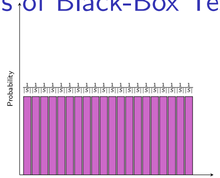


Figure: $\mathcal{U}_\varphi$: Uniform Distribution

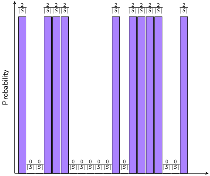


Figure: $\mathcal{A}_\varphi$: 1/2-far from uniform

SAMP: Allows you to draw samples from a distribution

- If $< \sqrt{|\mathsf{Sol}(\varphi)|}/100$ samples are drawn then with high probability you see only distinct samples from either distribution.

Theorem The above bound is optimal. [BFRSW 98; Pan 08]

Limitations of Black-Box Testing

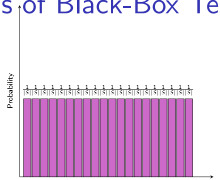


Figure: $\mathcal{U}_\varphi$: Uniform Distribution

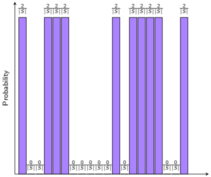


Figure: $\mathcal{A}_\varphi$: 1/2-far from uniform

SAMP: Allows you to draw samples from a distribution

- If $< \sqrt{|\mathsf{Sol}(\varphi)|}/100$ samples are drawn then with high probability you see only distinct samples from either distribution.

Theorem The above bound is optimal. [BFRSW 98; Pan 08]

Greybox Testing: Inspired by Distribution Testing Literature

COND $(\mathcal{P}, T)$

$$\Pr[\sigma \leftarrow \mathsf{COND}(\mathcal{P}, T)] = \begin{cases} \frac{\mathcal{P}(\sigma)}{\sum\limits_{\rho \in T} \mathcal{P}(\rho)} & \sigma \in T \\ 0 & \text{otherwise} \end{cases}$$

When $T = \{0,1\}^n$, then $\mathsf{COND}(\mathcal{P}, T) = \mathsf{SAMP}$

The Power of COND model

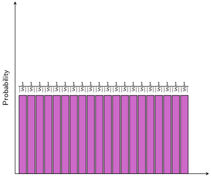


Figure: $\mathcal{U}_\varphi$: Uniform Distribution

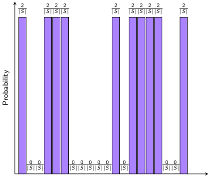


Figure: $\mathcal{A}_\varphi$: $1/2$-far from uniform

The Power of COND model

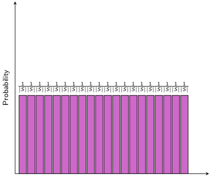


Figure: $\mathcal{U}_\varphi$: Uniform Distribution

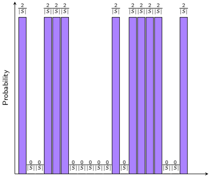


Figure: $\mathcal{A}_\varphi$: 1/2-far from uniform

An algorithm for testing uniformity using conditional sampling:

- Sample σ_1, σ_2 from $\mathcal{U}_\varphi$. Let $T = \{\sigma_1, \sigma_2\}$.
- In the case of the "far" distribution, with constant probability, $\mathcal{A}_\varphi(\sigma_1) \ll \mathcal{A}_\varphi(\sigma_2)$
- We will be able to distinguish the far distribution from the uniform distribution using constant number of samples from COND($\mathcal{A}_\varphi$, T).
- The constant depend on the farness parameter.

What about other distributions?

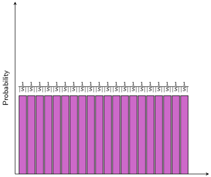


Figure: $\mathcal{U}_\varphi$: Uniform Distribution

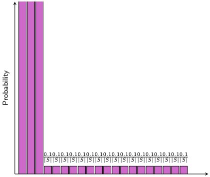


Figure: $\mathcal{A}_\varphi$: Far Distribution

What about other distributions?

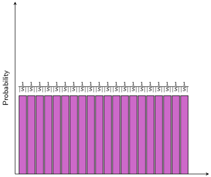


Figure: $\mathcal{U}_\varphi$: Uniform Distribution

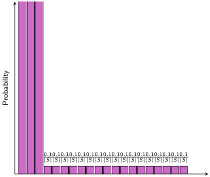


Figure: $\mathcal{A}_\varphi$: Far Distribution

Previous algorithm fails in this case:

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

Testing Uniformity Using Conditional Sampling

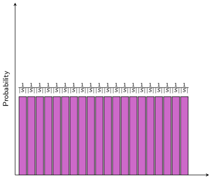
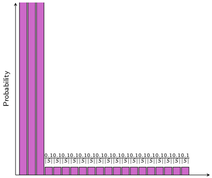
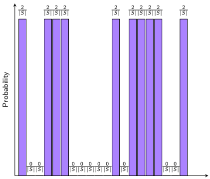


Figure: $\mathcal{U}_\varphi$: Uniform Distribution





Testing Uniformity Using Conditional Sampling

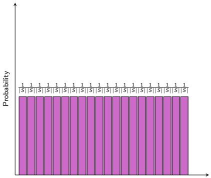
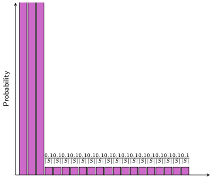
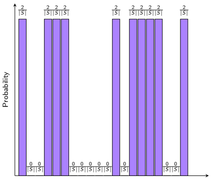


Figure: $\mathcal{U}_\varphi$: Uniform Distribution





An algorithm for testing uniformity using conditional sampling:

- Sample σ_1 from $\mathcal{U}_\varphi$ and σ_2 from $\mathcal{A}_\varphi$. Let $T = \{\sigma_1, \sigma_2\}$.
- In the case of the "far" distribution, with constant probability, $\mathcal{A}_\varphi(\sigma_1) \ll \mathcal{A}_\varphi(\sigma_2)$
- We will be able to distinguish the far distribution from the uniform distribution using constant number of samples from COND($\mathcal{A}_\varphi$, T).
- The constant depend on the farness parameter.

From Theory to Practice: Realizing COND Model

Challenge: How do we ask sampler for Conditional samples over $T = \{\sigma_1, \sigma_2\}$.

- Construct $\hat{\varphi} = \varphi \wedge (X = \sigma_1 \vee X = \sigma_2)$

From Theory to Practice: Realizing COND Model

Challenge: How do we ask sampler for Conditional samples over $T = \{\sigma_1, \sigma_2\}$.

- Construct $\hat{\varphi} = \varphi \wedge (X = \sigma_1 \vee X = \sigma_2)$

Almost all the constrained samplers just enumerate all the solutions when the number of solutions is small

From Theory to Practice: Realizing COND Model

Challenge: How do we ask sampler for Conditional samples over $T = \{\sigma_1, \sigma_2\}$.

- Construct $\hat{\varphi} = \varphi \wedge (X = \sigma_1 \vee X = \sigma_2)$

Almost all the constrained samplers just enumerate all the solutions when the number of solutions is 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 $\hat{\varphi}$

- $\tau = |\mathsf{Sol}(\hat{\varphi})|$
- $z \in \mathsf{Sol}(\hat{\varphi}) \implies z_{\downarrow X} \in \{\sigma_1, \sigma_2\}$
- $|\{z \in \mathsf{Sol}(\hat{\varphi}) \mid z_{\downarrow X} = \sigma_1\}| = |\{z \in \mathsf{Sol}(\hat{\varphi}) \mid z_{\downarrow X} \cap \sigma_2\}|$
- φ and $\hat{\varphi}$ has similar structure

Kernel

Input: A Boolean formula φ, two assignments σ_1 and σ_2, and desired number of solutions τ
Output: Formula $\hat{\varphi}$

- $\tau = |\mathsf{Sol}(\hat{\varphi})|$
- $z \in \mathsf{Sol}(\hat{\varphi}) \implies z_{\downarrow X} \in \{\sigma_1, \sigma_2\}$
- $|\{z \in \mathsf{Sol}(\hat{\varphi}) \mid z_{\downarrow X} = \sigma_1\}| = |\{z \in \mathsf{Sol}(\hat{\varphi}) \mid z_{\downarrow X} \cap \sigma_2\}|$
- φ and $\hat{\varphi}$ has similar structure

Non-adversarial Sampler Assumption: The distribution of the projection of samples obtained from $\mathcal{A}_{\hat{\varphi}}$ to variables of φ is same as $\mathsf{COND}(\mathcal{A}_{\varphi}, \{\sigma_1, \sigma_2\})$.

Implications:

- If $\mathcal{A}$ is a uniform sampler for every Boolean formula, it satisfies non-adversarial sampler assumption
- If $\mathcal{A}$ is not a uniform sampler, it may not necessarily satisfy non-adversarial sampler assumption

Kernel

Input: A Boolean formula φ, two assignments σ_1 and σ_2, and desired number of solutions τ
Output: Formula $\hat{\varphi}$

- $\tau = |\mathsf{Sol}(\hat{\varphi})|$
- $z \in \mathsf{Sol}(\hat{\varphi}) \implies z_{\downarrow X} \in \{\sigma_1, \sigma_2\}$
- $|\{z \in \mathsf{Sol}(\hat{\varphi}) \mid z_{\downarrow X} = \sigma_1\}| = |\{z \in \mathsf{Sol}(\hat{\varphi}) \mid z_{\downarrow X} \cap \sigma_2\}|$
- φ and $\hat{\varphi}$ has similar structure

Non-adversarial Sampler Assumption: The distribution of the projection of samples obtained from $\mathcal{A}_{\hat{\varphi}}$ to variables of φ is same as $\mathsf{COND}(\mathcal{A}_{\varphi}, \{\sigma_1, \sigma_2\})$.

Implications:

- If $\mathcal{A}$ is a uniform sampler for every Boolean formula, it satisfies non-adversarial sampler assumption
- If $\mathcal{A}$ is not a uniform sampler, it may not necessarily satisfy non-adversarial sampler assumption

Non-adversarial assumption allows us to use the theory of COND query model

Barbarik

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 $1-\delta$.

Observe: Complexity independent of $|\mathsf{Sol}(\varphi)|$ in contrast to black box's approach's dependence on $\sqrt{|\mathsf{Sol}(\varphi)|}$

Barbarik

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 $1-\delta$.

Observe: Complexity independent of $|\mathsf{Sol}(\varphi)|$ in contrast to black box's approach's dependence on $\sqrt{|\mathsf{Sol}(\varphi)|}$

Empirical Evaluation

Experimental Evaluation over three state of the art (almost-)uniform samplers

- UniGen3: Theoretical Guarantees of almost-uniformity
- SearchTreeSampler: Very weak guarantees
- QuickSampler: No Guarantees

The study (in 2018) that proposed Quicksampler could only perform unsound statistical tests, and therefore, could not distinguish the three samplers

Results-I

Instances	#Solutions	UniGen3		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	UniGen3		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

Recap: Outline

Q1 What do distributions look like in the real world?

Q2 What properties matter to the practitioners?

Q3 Theory and Practice of Distirbution Testing

- Complexity of Distance Estimation
- Greybox Testing: Constrained Samplers

Q4 Can distribution testing influence the design of systems ?

Q1: Distributions in Real World: II

Constrained Random Simulation: Test Vector Generation

- Dominant methodology to test hardware systems
- Use a formula φ to encode the verification scenarios
- A Constrained Sampler $\mathcal{A}$ takes φ as input and returns $\sigma \in \mathsf{Sol}(\varphi)$, and ideally ensures

$$\Pr[\sigma \leftarrow \mathcal{A}(\varphi)] = \frac{1}{|\mathsf{Sol}(\varphi)|}$$

Probabilistic Programs

- Typical programs augmented with ability to *sample* and *condition*
- $X \leftarrow \text{Sample}(\mathcal{N}, 100, 10)$
  - Sample from Gaussian with $\mu = 100$ and $\sigma^2 = 10$
- $\text{Observe}(X < 10)$
  - The compiler must ensure that the value of X is less than 10.
  - Allows conditioning of the distributions

Semantics: A probabilistic program P describes distribution

Who cares about Probabilistic Programs?

Facebook (HackPPL), Google(Tensorflow-probability), Uber (Pyro)
" *Probabilistic programming aims to make (probabilistic) modeling more accessible to developers*" (Facebook, 2016)

Q2: Properties that Matter

Grey-box Setting

- (Fast) Sampler $\mathcal{A}$ and a reference (but, often slow) sampler $\mathcal{U}$
- Reference sampler $\mathcal{U}$ is certified to produce samples according to desired distribution but is slow.
- Is the distribution generated by $\mathcal{A}$, denoted by $\mathcal{A}_\varphi$, close to that of $\mathcal{U}_\varphi$?

How to Measure Equivalence

Consider two distribution $\mathcal{P}$ and $\mathcal{Q}$ over $\{0,1\}^n$.

Two Notions of Distance

- d_∞ distance: $\max_{\sigma \in \{0,1\}^n} |\mathcal{P}(\sigma) - \mathcal{Q}(\sigma)|$
  - The most commonly seen behavior where a developer wants to approximate $\mathcal{P}$ with another distribution $\mathcal{Q}$
  - Almost-uniform sampling in the context of constrained random simulation

How to Measure Equivalence

Consider two distribution $\mathcal{P}$ and $\mathcal{Q}$ over $\{0,1\}^n$.

Two Notions of Distance

- d_∞ distance: $\max_{\sigma \in \{0,1\}^n} |\mathcal{P}(\sigma) - \mathcal{Q}(\sigma)|$
  - The most commonly seen behavior where a developer wants to approximate $\mathcal{P}$ with another distribution $\mathcal{Q}$
  - Almost-uniform sampling in the context of constrained random simulation
- Total Variation Distance (d_{TV}) or L_1 distance: $\frac{1}{2}\sum_{\sigma \in \{0,1\}^n} |\mathcal{P}(\sigma) - \mathcal{Q}(\sigma)|$
  - Consider any arbitrary program $\mathcal{A}$ that uses samples from a distribution: there is a probability distribution over output of $\mathcal{A}$.
  - Consider a Bad event over the output of $\mathcal{A}$: such as not catching a bug.
  - Let's say $\mathcal{A}$ samples from $\mathcal{P}$.
  - Folklore: If we were to replace $\mathcal{P}$ with $\mathcal{Q}$ then the probability of Bad event would increase/decrease at most by $d_{TV}(P, Q)$.

How to Measure Equivalence

Consider two distribution $\mathcal{P}$ and $\mathcal{Q}$ over $\{0,1\}^n$.

Two Notions of Distance

- d_∞ distance: $\max_{\sigma \in \{0,1\}^n} |\mathcal{P}(\sigma) - \mathcal{Q}(\sigma)|$
  - The most commonly seen behavior where a developer wants to approximate $\mathcal{P}$ with another distribution $\mathcal{Q}$
  - Almost-uniform sampling in the context of constrained random simulation
- Total Variation Distance (d_{TV}) or L_1 distance: $\frac{1}{2}\sum_{\sigma \in \{0,1\}^n} |\mathcal{P}(\sigma) - \mathcal{Q}(\sigma)|$
  - Consider any arbitrary program $\mathcal{A}$ that uses samples from a distribution: there is a probability distribution over output of $\mathcal{A}$.
  - Consider a Bad event over the output of $\mathcal{A}$: such as not catching a bug.
  - Let's say $\mathcal{A}$ samples from $\mathcal{P}$.
  - Folklore: If we were to replace $\mathcal{P}$ with $\mathcal{Q}$ then the probability of Bad event would increase/decrease at most by $d_{TV}(P, Q)$.

Therefore, measure closeness with respect to d_∞ and farness with respect to d_{TV}

- Checker should return Accept if two distributions are close in d_∞-distance and return Reject if two distributions are far in d_{TV}.

Recap: Outline

Theory and Practice of Distirbution Testing

Probabilistic Generative Models

- Complexity of Distance Estimation for Probabilistic Generative Models

Constrained Samplers

- Greybox Testing: Constrained Samplers
- Can distribution testing influence the design of systems ?
  - Constrained Samplers
  - Binomial Sampler in Python

Product Distributions

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

Product Distributions

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

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

Product Distributions

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

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

```
program P
X[1] = Bernoulli(p_1);
X[2] = Bernoulli(p_2);
....
X[n] = Bernoulli(p_n);
return X;
```

```
program Q
Y[1] = Bernoulli(q_1);
Y[2] = Bernoulli(q_2);
....
Y[n] = Bernoulli(q_n);
return Y
```

Testing Uniformity Using Conditional Sampling

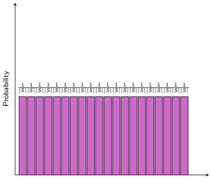
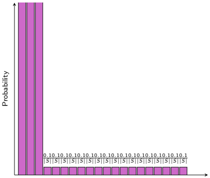
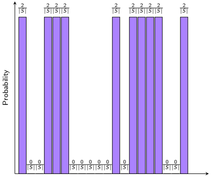


Figure: $\mathcal{U}_\varphi$: Uniform Distribution





Testing Uniformity Using Conditional Sampling

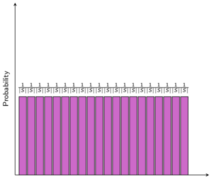
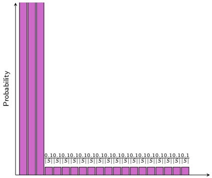
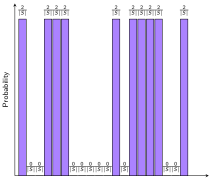


Figure: $\mathcal{U}_\varphi$: Uniform Distribution





An algorithm for testing uniformity using conditional sampling:

- Sample σ_1 from $\mathcal{U}_\varphi$ and σ_2 from $\mathcal{A}_\varphi$. Let $T = \{\sigma_1, \sigma_2\}$.
- In the case of the "far" distribution, with constant probability, $\mathcal{A}_\varphi(\sigma_1) \ll \mathcal{A}_\varphi(\sigma_2)$
- We will be able to distinguish the far distribution from the uniform distribution using constant number of samples from COND($\mathcal{A}_\varphi$, T).
- The constant depend on the farness parameter.

Barbarik

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 $1-\delta$.

Observe: Complexity independent of $|\mathsf{Sol}(\varphi)|$ in contrast to black box's approach's dependence on $\sqrt{|\mathsf{Sol}(\varphi)|}$

Barbarik

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 $1-\delta$.

Observe: Complexity independent of $|\mathsf{Sol}(\varphi)|$ in contrast to black box's approach's dependence on $\sqrt{|\mathsf{Sol}(\varphi)|}$

Results-I

Instances	#Solutions	UniGen3		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	UniGen3		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

Barbarik for Probabilistic Programs

- Conditioning is just inserting Observe statements!
- Input: A program under test $\mathcal{A}$, a reference program generating 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 program $\mathcal{A}$ specifies ε-additive uniform distribution then Barbarik ACCEPTS with probability at least $(1-\delta)$.
  - if $\mathcal{A}$ is η-far from a uniform generator holds then Barbarik REJECTS with probability at least $1-\delta$.
- Preliminary experiments in progress

Outline

Theory and Practice of Distirbution Testing

Probabilistic Generative Models

- Complexity of Distance Estimation for Probabilistic Generative Models [✓]

Constrained Samplers

- Greybox Testing: Constrained Samplers [✓]
- **Can distribution testing influence the design of systems ?**
  - Constrained Samplers
  - Binomial Sampler in Python

Can distribution testing influence the design of systems ?

Wishlist

- Sampler should be at least as fast as STS and QuickSampler.
- Sampler should by accepted by Barbarik.
- Sampler should have impact on downstream (real world) applications.

CMSGen

- Exploits the flexibility CryptoMiniSat.

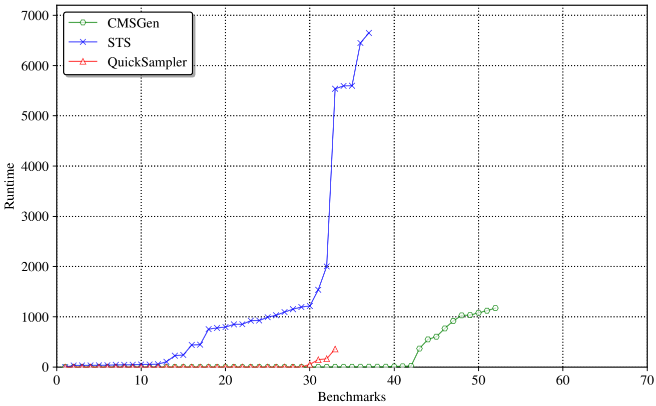
CMSGen

- 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.
- 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 Distribution Testing-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
  - We have very little idea about why SAT solvers work?
  - Much less about what happens when you tweak them to make them samplers

Runtime Performance



QuickSampler	STS	CMSGen
33	37	52

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 [EGSS12]
  - QuickSampler [DLBS18]
  - **CMSGen**
- Sampler with guarantees:
  - UniGen3 [CMV13, CMV14, SGM20]

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

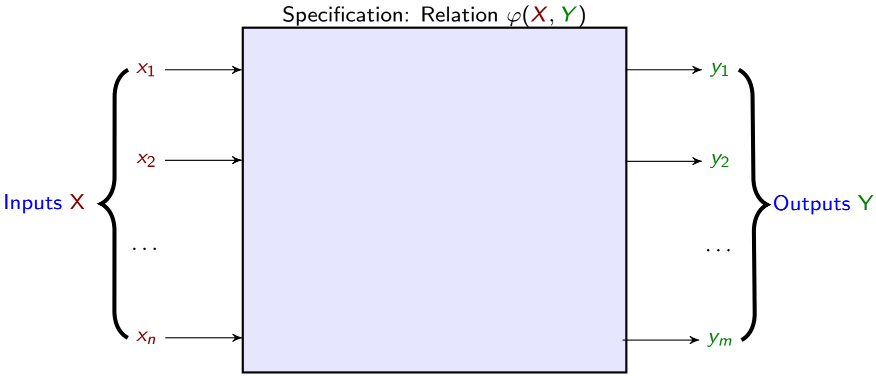
Outline

Wishlist

- Sampler should be at least as fast as STS and QuickSampler. ✓
- Sampler should by accepted by Barbarik. ✓
- Sampler should have impact on downstream (real world) applications.

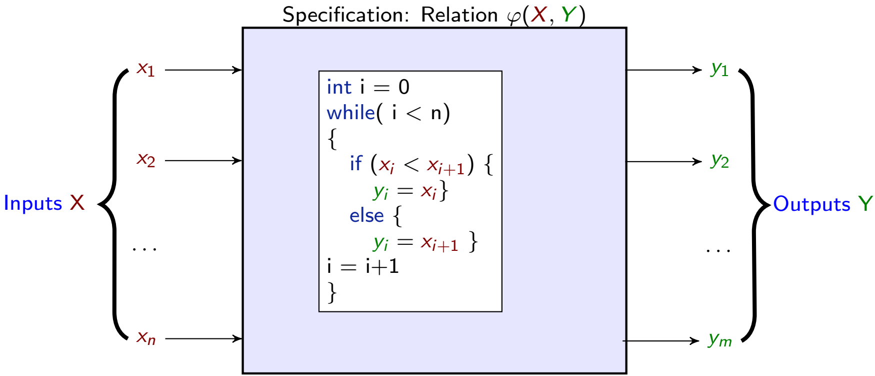
Application I: Functional Synthesis

Holy Grail of Programming: *The user states the problem, the computer solves it* (Freuder, 1996)



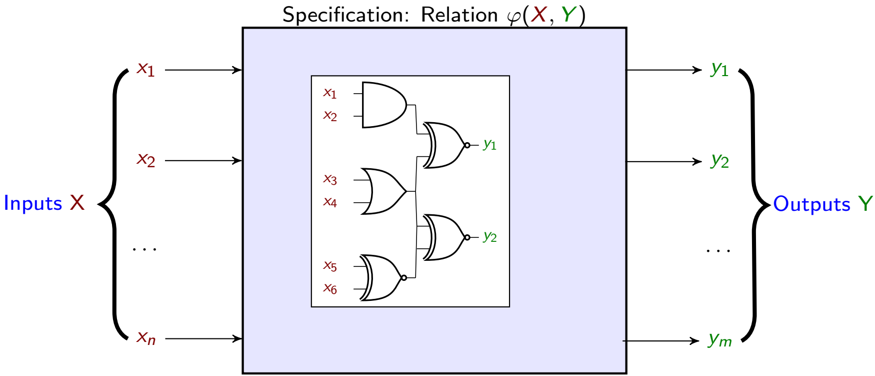
Application I: Functional Synthesis

Holy Grail of Programming: *The user states the problem, the computer solves it* (Freuder, 1996)



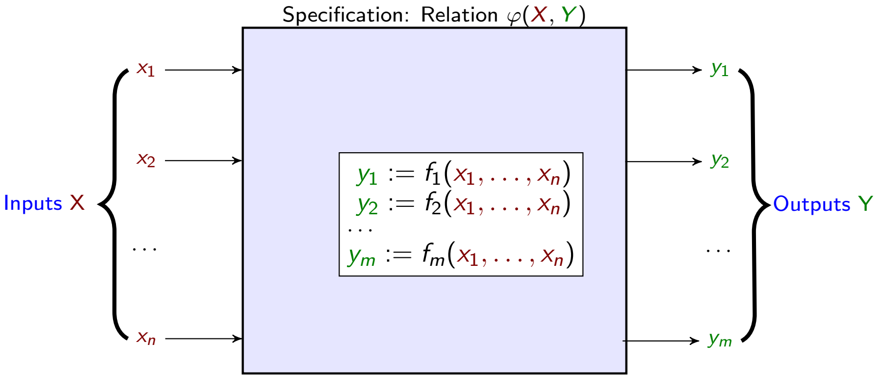
Application I: Functional Synthesis

Holy Grail of Programming: *The user states the problem, the computer solves it* (Freuder, 1996)



Application I: Functional Synthesis

Holy Grail of Programming: *The user states the problem, the computer solves it* (Freuder, 1996)



Application I: Functional Synthesis

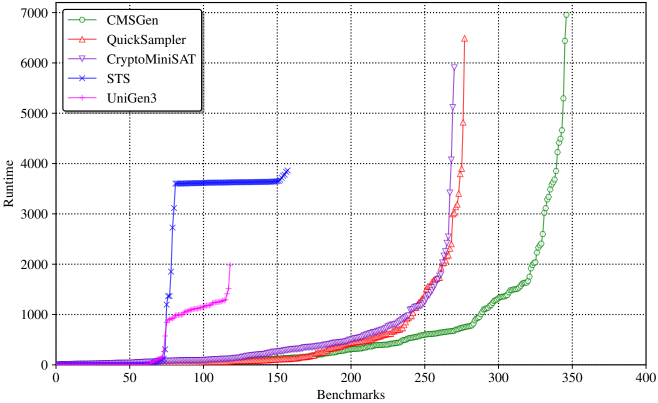
State of the art approach: Manthan

Sampling + Machine Learning + Counter-example guided repair

Application I: Functional Synthesis

State of the art approach: Manthan

Sampling + Machine Learning + Counter-example guided repair



Application II: 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.

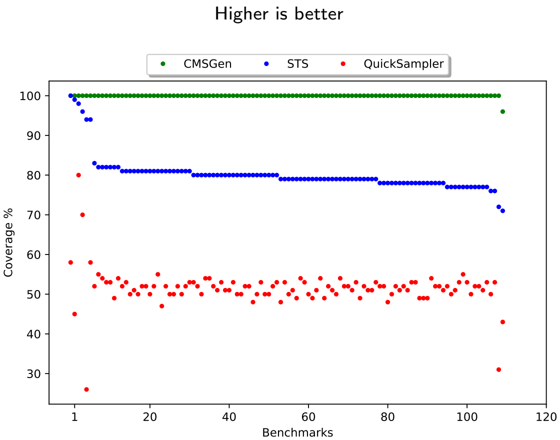
Application II: Combinatorial Testing

- 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.
- Experimental Evaluations:
  - Generate 1000 samples (test cases).
  - 110 Benchmarks, Timeout: 3600 seconds
  - 2-wise coverage $t = 2$.

Combinatorial Testing: The Power of CMSGen



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

Outline

Can distribution testing influence the design of systems?

Wishlist

- Sampler should be at least as fast as STS and QuickSampler. ✓
- Sampler should by accepted by Barbarik. ✓
- Sampler should have impact on downstream (real world) applications. ✓

Outline

Theory and Practice of Distirbution Testing

Probabilistic Generative Models

- Complexity of Distance Estimation for Probabilistic Generative Models [✓]

Constrained Samplers

- Greybox Testing: Constrained Samplers [✓]
- **Can distribution testing influence the design of systems ?**
  - Constrained Samplers [✓]
  - Binomial Sampler in Python

Binomial Distribution in Python

```
> numpy.random.binomimial(2**63, 0.1)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "numpy/random/mtrand.pyx", line 3455, in numpy.random.mtrand.RandomState.binomial
OverflowError: Python int too large to convert to C long
```

Figure: Code snippet (Numpy version: 1.26.1 and Python version 3.9.6)

Inverse Transform Sampling

$$\hat{H}^{-1}(u) = \left(\frac{2a}{(1/2 - |u|)} + b\right) u + c, \quad \hat{h}^{-1}(u) = \frac{1}{\hat{h}(u)} = \frac{a}{(1/2 - |u|)^2} + b$$

$$\alpha = (2.83 + 5.1/b)\sqrt{np(1-p)}$$

Algorithm 1: Binomial Transformed Rejection Sampling

```
Input  : Binomial Distribution B_{n,p}
Output : Sample k from B_{n,p}
Initialize inverse distribution Ĥ^{-1}(·), ĥ^{-1}(·) ( according to Equation 1);
Initialize rejection ratio α (according to Equation 2);
m ← ⌊(n + 1)p⌋;
l_m ← log m!;
l_nm ← log (n − m)!;
while True do
    generate uniform random variates u, v;
    k ← Ĥ^{-1}(u);
    l_k ← log k!, l_nk ← log (n − k)!;
    if log v ≤ l_m + l_mk − l_k − l_nk + (n − k) log (p/q) + log ĥ^{-1}(u) − log α then
        return k
```

Inverse Transform Sampling

$$\hat{H}^{-1}(u) = \left(\frac{2a}{(1/2 - |u|)} + b\right) u + c, \quad \hat{h}^{-1}(u) = \frac{1}{\hat{h}(u)} = \frac{a}{(1/2 - |u|)^2} + b$$

$$\alpha = (2.83 + 5.1/b)\sqrt{np(1-p)}$$

Algorithm 1: Binomial Transformed Rejection Sampling

Input : Binomial Distribution $\mathcal{B}_{n,p}$
Output : Sample k from $\mathcal{B}_{n,p}$

Initialize inverse distribution $\hat{H}^{-1}(\cdot), \hat{h}^{-1}(\cdot)$ (according to Equation 1);
Initialize rejection ratio α (according to Equation 2);
$m \leftarrow \lfloor (n+1)p \rfloor$;
$l_m \leftarrow \log m!$;
$l_{nm} \leftarrow \log (n-m)!$;
while *True* **do**
  generate uniform random variates u, v;
  $k \leftarrow \hat{H}^{-1}(u)$;
  $l_k \leftarrow \log k!, \ l_{nk} \leftarrow \log (n-k)!$;
  if $\log v \leq l_m + l_{mk} - l_k - l_{nk} + (n-k)\log\left(\frac{p}{q}\right) + \log \hat{h}^{-1}(u) - \log \alpha$ **then**
    return k

Just Implement all operations with arbitrary precision arithmetic

Inverse Transform Sampling

$$\hat{H}^{-1}(u) = \left(\frac{2a}{(1/2 - |u|)} + b\right) u + c, \quad \hat{h}^{-1}(u) = \frac{1}{\hat{h}(u)} = \frac{a}{(1/2 - |u|)^2} + b$$

$$\alpha = (2.83 + 5.1/b)\sqrt{np(1-p)}$$

Algorithm 1: Binomial Transformed Rejection Sampling

Input : Binomial Distribution $\mathcal{B}_{n,p}$
Output : Sample k from $\mathcal{B}_{n,p}$

Initialize inverse distribution $\hat{H}^{-1}(\cdot), \hat{h}^{-1}(\cdot)$ (according to Equation 1);
Initialize rejection ratio α (according to Equation 2);
$m \leftarrow \lfloor (n+1)p \rfloor$;
$l_m \leftarrow \log m!$;
$l_{nm} \leftarrow \log (n-m)!$;
while *True* **do**
  generate uniform random variates u, v;
  $k \leftarrow \hat{H}^{-1}(u)$;
  $l_k \leftarrow \log k!, l_{nk} \leftarrow \log (n-k)!$;
  if $\log v \leq l_m + l_{mk} - l_k - l_{nk} + (n-k)\log\left(\frac{p}{q}\right) + \log \hat{h}^{-1}(u) - \log \alpha$ **then**
    return k

Just Implement all operations with arbitrary precision arithmetic

- Factorials need approximation, for runtime efficiency
- But approximation introduces errors

Le Cam's Theorem

Le Cam's Theorem:

$$\sum_{k=0}^{\infty} \left| \Pr[\mathcal{B}_{n,p} = k] - \frac{\lambda^k e^- \lambda}{k!} \right| < 2np^2$$

where $\lambda = np$. In other words,

$$d_{TV}(\mathcal{B}_{n,p}, \text{Pois}(np)) \leq np^2$$

For certain range of parameters (n and p), sampling from Poisson distribution is closer in total variation distance and is more efficient

Proposal for New Interface

Sample from $\mathcal{B}_{n,p}$

- Find the closest distribution from which we should sample to balance total variation distance and runtime
- Report the total variation distance (i.e., error)

Not merely return a sample but also return total variation distance

Runtime Performance Improvement

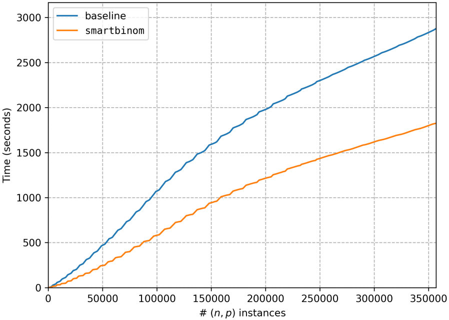


Figure: Comparison of the time taken by `smartBinom` and Baseline across 350,000 calls to (n, p) instances.

Quality of Error

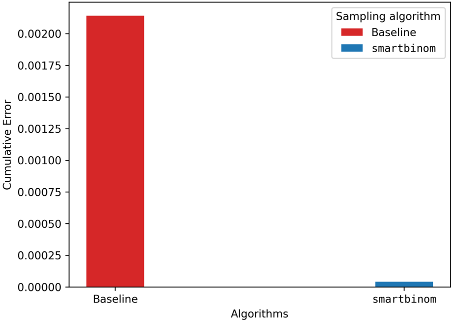


Figure: Upper bound estimation of the cumulative error reported by `smartBinom` and Baseline on 350,000 calls to (n, p) instances.

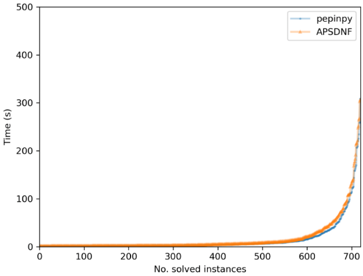
Figure: Performance comparison of `smartBinom` against the Baseline sampler.

Union of Sets

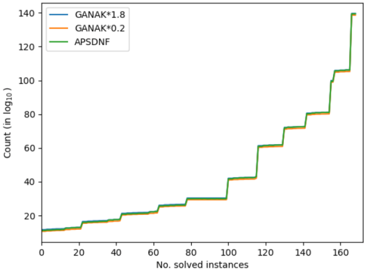
Algorithm 7: APS-Estimator

```
Initialize Bucket threshold T;
Initialize probability p;
Initialize empty Buckets X;
for i = 1 to m do
    for all σ ∈ X do
        if σ ⊨ F_i then
            remove σ from X;
    Pick a number N_i from the binomial distribution B_{|F_i|,p};
    Add N_i distinct random solutions of F_i to X;
    while |X| is more than bucket threshold T do
        p = p/2;
        Throw away each element of X with probability 1/2;
Output |X|/p;
```

Experimental Results I: Runtime



Experimental Results I: Quality



Conclusion

Q1 What do distributions look like in the real world?

Ans Probability distributions are first-class objects in modern computing

Q2 What properties matter to the practitioners?

Ans Equivalence

Q3 How to develop practical scalable testers for distributions?

Ans Greybox access, which can be modeled via Conditional Sampling

Q4 Can distribution testing influence the design of systems ?

Ans Yes. It can allow us to design state of the art samplers via a different approach. And such samplers dramatically improve downstream applications.

Where do we go from here?

We have just started!

- Scalable testers for distributions beyond uniform
- Scalable samplers for SMT/CSP via Test-Driven Development
- Developing the notion of counterexample for testing distributions
- How do we certify the correctness of distribution testers?

CMSGen (MIT License): `https://github.com/meelgroup/cmsgen`

Barbarik (MIT License): `https://github.com/meelgroup/barbarik`

These slides are available at `https://www.cs.toronto.edu/~meel/talks.html`