Designing Samplers is Easy: The Boon of Testers

Mate Soos\textsuperscript{1}

(Substitute Presenter: Kuldeep S. Meel\textsuperscript{1})

Joint work with Sourav Chakraborty\textsuperscript{2} and Priyanka Golia\textsuperscript{1}

\textsuperscript{1} National University of Singapore
\textsuperscript{2} Indian Statistical Institute, Kolkata

(Relevant Publications: AAAI-19, FMCAD-21, CP-22)
Input: A CNF Formula $F$ and tolerance parameter $\varepsilon$
Output: $\sigma \in Sol(F)$ such that

$$\frac{1}{(1 + \varepsilon)|Sol(F)|} \leq Pr[A(F) = \sigma] \leq \frac{1 + \varepsilon}{|Sol(F)|}$$

Motivation: Fundamental problem in CS (theory) and applications in hardware and software testing (practice)

Snapshot from early 2010's

**Scalability**  WES04, NRJK+06, KK07

**Guarantees**  JVV86, BGP00, YAPA04
UniGen: Almost-uniform Sampler

- Core Idea: Use 3-wise independence (random XORs) to partition the solution space
- Makes $O(\log n)$ calls to SAT oracle
- Theoretical guarantees
  \[
  \frac{1}{(1 + \varepsilon) |\text{Sol}(F)|} \leq \Pr[A(F) = y] \leq \frac{1 + \varepsilon}{|\text{Sol}(F)|}
  \]
- Scalability: CryptoMiniSat (A specialized solver for CNF+XOR)
How do you test a sampler is uniform?

Input: A reference sampler $\mathcal{U}$, a test sampler $\mathcal{A}$, and a formula $F$

Approach: Run both samplers and plot their distributions

- Eyeball the distributions
- Run statistical tests (KL divergence, chi-square)

Caveat: Requires number of samples $\gg$ number of solutions
How do you test a sampler is uniform?

**Input:** A reference sampler $\mathcal{U}$, a test sampler $\mathcal{A}$, and a formula $F$

**Approach:** Run both samplers and plot their distributions

- Eyeball the distributions
- Run statistical tests (KL divergence, chi-square)

**Caveat** Requires number of samples $>>$ number of solutions

What if you try to draw conclusions based on fewer samples?

DLBS18: Efficient Sampling of SAT Solutions for Testing

“We can see that SearchTreeSampler and UniGen2 are more uniform, but QuickSampler is still close to uniform on most benchmarks. However, this result should be taken with care, since the uniformity test is not very reliable on benchmarks where QuickSampler completed a small number of epochs or when the number of produced samples is too low.”
In search of principled approach

**Input:** A reference sampler \( \mathcal{U} \), a test sampler \( \mathcal{A} \), and a formula \( F \)

**Problem:** Return **Yes** if the distribution of \( \mathcal{U}(F) \) (known to be uniform) and \( \mathcal{A}(F) \) are close, else return **No**

**Approach II:** Just keep sampling and stop the first time you see a collision

---

**Figure:** \( \mathcal{U} \): Reference Distribution

**Figure:** \( \mathcal{A} \): far from uniform

No collisions until you have generated at least \( \sqrt{|\text{Sol}(F)|} \) solutions!

**BFRSW98 \implies** The above technique is *optimal* (i.e., if we are only allowed to look at samples)
Definition (Conditional Sampling)

Given a distribution $D$ on $S$; allow one to specify a set $T \subseteq S$ and draw samples from $A$ conditioned on $T$

$$
\Pr[\sigma \text{ is generated}] = \begin{cases} 
0 & \text{if } \sigma \notin T \\
\frac{D(\sigma)}{\sum_{\sigma \in T} D(\sigma)} & \text{otherwise}
\end{cases}
$$

Conditional sampling is at least as powerful as drawing normal samples but is it more powerful?
• Draw $\sigma_1$ uniformly at random from the domain and draw $\sigma_2$ according to the distribution $\mathcal{A}$. Let $T = \{\sigma_1, \sigma_2\}$.

• In the case of the “far” distribution, with constant probability, $\sigma_1$ will have “low” probability and $\sigma_2$ will have “high” probability.

• We will be able to distinguish the far distribution from the uniform distribution using constant number of samples from $\mathcal{A}|_T$.

• The constant depend on the farness parameter.

The above algorithm works for all cases.
• Input formula: $F$ over variables $X$
• **Challenge:** Conditional Sampling over $T = \{\sigma_1, \sigma_2\}$.
• Construct $G = F \land (X = \sigma_1 \lor X = \sigma_2)$
• Most of the samplers will just enumerate all the solutions when the number of solutions is very small
• Need way to construct formulas whose solution space is large but every solution can be mapped to either $\sigma_1$ or $\sigma_2$. 
Input: A Boolean formula \( \varphi \), two assignments \( \sigma_1 \) and \( \sigma_2 \), and desired number of solutions \( \tau \)

Output: Formula \( \hat{\varphi} \)

- \( \tau = |Sol(\hat{\varphi})| \)
- \( z \in Sol(\hat{\varphi}) \implies z_{\downarrow Supp(\varphi)} \in \{\sigma_1, \sigma_2\} \)
- \( |\{z \in Sol(\hat{\varphi}) | z_{\downarrow Supp(\varphi)} = \sigma_1\}| = |\{z \in Sol(\hat{\varphi}) | z_{\downarrow Supp(\varphi)} = \sigma_2\}| \)
- \( \varphi \) and \( \hat{\varphi} \) has “similar” structure
Input: A Boolean formula $\varphi$, two assignments $\sigma_1$ and $\sigma_2$, and desired number of solutions $\tau$

Output: Formula $\hat{\varphi}$

- $\tau = |\text{Sol}(\hat{\varphi})|$
- $z \in \text{Sol}(\hat{\varphi}) \implies z\downarrow_{\text{Supp}(\varphi)} \in \{\sigma_1, \sigma_2\}$
- $|\{z \in \text{Sol}(\hat{\varphi}) \mid z\downarrow_{\text{Supp}(\varphi)} = \sigma_1\}| = |\{z \in \text{Sol}(\hat{\varphi}) \mid z\downarrow_{\text{Supp}(\varphi)} = \sigma_2\}|$
- $\varphi$ and $\hat{\varphi}$ has “similar” structure

**Definition**

The **non-adversarial sampler assumption** states that the distribution of the projection of samples obtained from $A(\hat{\varphi})$ to variables of $\varphi$ is same as the conditional distribution of $A(\varphi)$ restricted to either $\sigma_1$ or $\sigma_2$

- If $A$ is a uniform sampler for all the input formulas, it satisfies non-adversarial sampler assumption
- If $A$ is not a uniform sampler for all the input formulas, it may not necessarily satisfy non-adversarial sampler assumption
**Input:** A sampler under test $A$, a reference uniform sampler $U$, a tolerance parameter $\varepsilon > 0$, an intolerance parameter $\eta > \varepsilon$, a guarantee parameter $\delta$ and a CNF formula $\varphi$

**Output:** ACCEPT or REJECT with the following guarantees:

- if the generator $A$ is an $\varepsilon$-additive almost-uniform generator then Barbarik ACCEPTS with probability at least $(1 - \delta)$.
- if $A(\varphi, .)$ is $\eta$-far from a uniform generator and if non-adversarial sampler assumption holds then Barbarik REJECTS with probability at least $1 - \delta$.
- Barbarik needs at most $K = \tilde{O}\left(\frac{1}{(\eta - \varepsilon)^4}\right)$ samples.
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

<table>
<thead>
<tr>
<th></th>
<th>QuickSampler</th>
<th>STS</th>
<th>UniGen3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPTs</td>
<td>0</td>
<td>14</td>
<td>50</td>
</tr>
<tr>
<td>REJECTs</td>
<td>50</td>
<td>36</td>
<td>0</td>
</tr>
</tbody>
</table>

To ACCEPT, we needed $10^6$ samples but we could reject with just 250 samples.
How can we use the availability of Barbarik to design a good sampler? Is it even possible?

Wishlist

- Sampler should be at least as fast as STS and QuickSampler.
- Sampler should pass the Barbarik test.
- Sampler should perform well on real world applications.
• Exploits the flexibility of 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.

```plaintext
./cryptominisat5 --maxsol $1 --nobansol --restart fixed --maple 0 --verb 0 --scc 1 --n 1
--presimp 0 --polar rnd --freq 0.9999 --fixedconfl $2 --random $3 --dumpresult $4 [CNFFILE]
```
CMSGen

- Exploits the flexibility of 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.

```
./cryptominisat5 -maxsol $1 -nobansol -restart fixed -maple 0 -verb 0 -scc 1 -n 1 -presimp 0 -polar rnd -freq 0.9999 -fixedconfl $2 -random $3 -dumpresult $4 [CNFFILE]
```

- Parameters of CMSGen are decided iteratively with the help of Barbarik

- Lack of theoretical analysis.
CMSGen vs. Other State-of-the-Art Samplers (I)

![Graph comparing CMSGen, STS, and QuickSampler runtime over benchmarks.]

<table>
<thead>
<tr>
<th>QuickSampler</th>
<th>STS</th>
<th>CMSGen</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>37</td>
<td>52</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th></th>
<th>QuickSampler</th>
<th>STS</th>
<th>UniGen3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPTs</td>
<td>0</td>
<td>14</td>
<td>50</td>
</tr>
<tr>
<td>REJECTs</td>
<td>50</td>
<td>36</td>
<td>0</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th></th>
<th>QuickSampler</th>
<th>STS</th>
<th>UniGen3</th>
<th>CMSGen</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPTs</td>
<td>0</td>
<td>14</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>REJECTs</td>
<td>50</td>
<td>36</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
• Sampler should be at least as fast as STS and QuickSampler. ✓

• Sampler should pass the Barbarik test. ✓

• Sampler should perform well on real world applications.
A powerful paradigm for testing configurable systems.

Challenge: To generate test suites that maximizes $t$-wise coverage.

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

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 $t = 2$. 
Higher is better

<table>
<thead>
<tr>
<th></th>
<th>QuickSampler</th>
<th>STS</th>
<th>CMSGen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. Coverage</td>
<td>51.5%</td>
<td>80.15%</td>
<td>~100%</td>
</tr>
</tbody>
</table>

Remark: UniGen3 could sample for only 6 benchmarks
State of the art approach (Manthan): Sampling + Machine Learning + Counter-example guided repair
Where do we go from here?

Summary  Design of a practically efficient sampler via test-driven development that works well in real-world applications

Practice  A Virtuous cycle: Improve Barbarik so that it can reject CMSGen and then improve CMSGen
- Trade-off between runtime performance and quality
- Frequent restarts degrade solution quality

Theory  Explain why CMSGen works well
- Perhaps CDCL with randomization is all you need in practice?
- Perhaps, you don’t really need uniformity in most cases. What do we really need?

Theory and Practice  And a testing methodology independent of non-adversarial assumption