On Quantitative Testing of Samplers

Presentation at Simons Institute, Satisfiability: Theory, Practice, and Beyond Reunion

Mate Soos, Priyanka Golia, Sourav Chakraborty, Kuldeep S. Meel

National University of Singapore; Indian Institute of Technology Kanpur; Indian Statistical Institute, Kolkata

15th of June 2022

Table of Contents

Uniform Sampling of Solutions of a CNF

CNF : Conjunctive Normal Form

- Formula consisting of boolean variables, and a set of constraints: a conjunction of disjunctions
- Ex: $(a \vee b) \wedge (a \vee \neg b) \wedge (a \vee b \vee \neg c)$
- All satisfying assignments: $a = \top$ any value for b, c.

Uniform sampling

- Provide samples uniformly at random from the solution space.
- Say, we need 1M samples from CNF above. We expect it to contain roughly 0.5M samples with $b = \top$.
- Chance of 0 samples with $b = \top$ is $2^{-500000}$, i.e. not very high. Possible, but not realistic.

Use-Cases, Previous work

Use-cases

- Configuration testing [1, 2], Constrained-random simulation [3]
- Bug synthesis [4], Function synthesis [5]

Uniform samplers

- **With** guarantees: SPUR [9], KUS [10], UniGen [6, 7, 8]
- **Without** guarantees: SearchTreeSampler (STS) [11], Quicksampler [12], CMSGen [13]

Sampler checker: Barbarik [14]

Takes SUT, a base uniform sampler (SPUR), tolerance param ϵ, intolerance param η, confidence param δ, and formula φ and returns Accept/Reject.

Accept/Reject depending on whether the SUT is ϵ-additive close to a uniform sampler or whether it is η-far from the uniform sampler. Correct answer with probability at least $(1 - \delta)$

Barbarik vs CMSGen and Other Uniform-Like Samplers

The paper [14] on Barbarik could clearly distinguish QuickSampler and STS from UniGen3. However, it could not distinguish UniGen3 from CMSGen.

Table: Analysis of different samplers with Barbarik over 50 benchmarks. Parameters $\epsilon : 0.3, \eta : 1.8, \delta : 0.1$. Same benchmark suite as used in [12] (QuickSampler paper)

	QuickSampler	STS	UniGen3	CMSGen_{100}
Accept	0	14	50	50
Reject	50	36	0	0

In other words, CMSGen could "fool" Barbarik. This showcases the power of CMSGen, however, it also highlights a weakness in Barbarik. In this paper, we sought to address this issue.

The Initial Idea

- Let's divide the solution space into two
- Make one part **super-easy** to find solutions. Say, in this part of the solution space, there are no constraints other than $a = \top$
- Make one part **tunably hard** to find solutions. All constraints are conditioned on $a = \bot$
- For hard problem generator, we decided to use the SHA-1 preimage attack by Nossum [15]. Tunable by constraining the input/output bits and the number of rounds to have more/less solutions and to be easier/harder to reverse.

Mini-experiment with non-uniform sampler

CMSGen$_{100}$: Preimage attack with 11 rounds

```
soos@tiresias:build$ ./cnf_gen  --rounds 11 --easy 11 > out
num hard solutions : 2048
num easy solutions : 2048
num total solutions: 4096
easy vs hard ratio : 0.5000 vs 0.5000
soos@tiresias:build$ ./sample.sh 100 1 out | grep -E -o "v -?1 " | sort | uniq -c
     53 v -1
     47 v 1
```

When the SHA-1 preimage problem is easy, we get approx 50-50.

CMSGen$_{100}$: Preimage attack with 18 rounds

```
soos@tiresias:build$ ./cnf_gen  --rounds 18 --easy 11 > out2
num hard solutions : 1925
num easy solutions : 2048
num total solutions: 3973
easy vs hard ratio : 0.5155 vs 0.4845
soos@tiresias:build$ ./sample.sh 100 1 out2 | grep -E -o "v -?1 " | sort | uniq -c
      1 v -1
     99 v 1
```

When the preimage problem is hard, we get 99-1.

Mini-experiment with uniform sampler

UniGen3: Preimage attack with 11 and 18 rounds

```
soos@tiresias:build$ ../../unigen/build/unigen out --multisample 0 --samples 100 --arjun 0 | gre
p -E -o "^-?1 "  | sort | uniq -c
     43 -1
     57 1
soos@tiresias:build$ ../../unigen/build/unigen out2 --multisample 0 --samples 100 --arjun 0 | gr
ep -E -o "^-?1 "  | sort | uniq -c
     52 -1
     48 1
```

Using an probabilistically approximate uniform sampler, UniGen3, we get approx 50-50 in both cases.

Barbarik – Main Idea

- Take a satisfying assignment σ_1 from the SUT, and a σ_2 from the base uniform sampler. $T = \{\sigma_1, \sigma_2\}$
- If the distribution D_φ from which SUT is sampling is close to uniform distribution, then the conditional distribution $D_{\varphi|T}$ is also close to uniform distribution.
- If the distribution D_φ is far from uniform distribution, then the conditional distribution $D_{\varphi|T}$ is also far from uniform distribution.

Barbarik – The Code

Algorithm 1: Barbarik($\mathcal{G}$, $\mathcal{U}$, ε, η, δ, φ)

$S \leftarrow Supp(\varphi)$
for $j \leftarrow 1$ *to* $\lceil \log(\frac{4}{2\varepsilon+\eta}) \rceil$ **do**
    $t_j \leftarrow f(\eta, \epsilon, \delta)$, $N_j \leftarrow g(\eta, \epsilon, \delta)$
    for $i \leftarrow 1$ *to* t_j **do**
        while $L_1 = L_2$ **do**
            $L_1 \leftarrow \mathcal{G}(\varphi, S, 1)$; $\sigma_1 \leftarrow L_1[0]$ /* $\mathcal{G}$ samples $\sigma_1 \in$ Sol(φ) */
            $L_2 \leftarrow \mathcal{U}(\varphi, S, 1)$; $\sigma_2 \leftarrow L_2[0]$ /* $\mathcal{U}$ samples $\sigma_2 \in$ Sol(φ) */
        end
        $\hat{\varphi} \leftarrow Kernel(\varphi, \sigma_1, \sigma_2, N_j)$
        $L_3 \leftarrow \mathcal{G}(\hat{\varphi}, S, N_j)$ /* $\mathcal{G}$ samples N_j solutions from Sol($\hat{\varphi}$) */
        $b \leftarrow Bias(\sigma_1, L_3, S)$
        if $b < \frac{1}{2}(1 - c_j)$ *or* $b > \frac{1}{2}(1 + c_j)$ **then**
            return REJECT
        end
    end
    return ACCEPT
end

Kernel

- To generate distribution $D_{\varphi|T}$, Barbarik constructs formula $\hat{\varphi}$ from φ using subroutine *Kernel*.
- *Kernel* takes $\varphi, \sigma_1, \sigma_2, N$, where N is number of assignments needed, and returns $\hat{\varphi}$. It restricts φ to these T, and extend each using *Chain Formulas* to required no. of solutions.

Algorithm 2: Kernel$(\varphi, \sigma_1, \sigma_2, N)$

lit $\leftarrow (\sigma_1 \setminus \sigma_2)[0]$ /* Choose first literal lit s.t. lit $\in \sigma_1$, and lit $\notin \sigma_2$ */

$\varphi' = \varphi \wedge (\sigma_1 \vee \sigma_2)$

$\hat{\varphi} \leftarrow \varphi' \wedge (\text{lit} \rightarrow \text{ConstructChain}(N, Supp(\psi)))$

$\hat{\varphi} \leftarrow \hat{\varphi} \wedge (\neg\text{lit} \rightarrow \text{ConstructChain}(N, Supp(\psi)))$

return $\hat{\varphi}$.

ScalBarbarik

ScalBarbarik: new $kernel$

Essentially, we replace $Kernel$ in Barbarik with a new $Kernel$ that generates an asymmetrical problem. We call this $Kernel$ Shakuni. This new $Kernel$ uses chain formulas as per Barbarik for the "easy" side of the problem, and the new, GenHard algorithm for the "hard" side of the problem.

The *GenHard* algorithm

- Takes κ as hardness parameter, and τ as number of solutions
- Uses SHA-1 preimage attack as hard problem. $\mathcal{H}_{\text{SHA-1}} := \{h : \{0,1\}^{512} \mapsto \{0,1\}^{160}\}$.
- Encodes the problem h^{-1} with varying number of rounds, and varying number of input/output bits set.
- To know the exact number of solutions, it uses a fast implementation of SHA-1.

Shakuni

Algorithm 3: Shakuni$(\varphi, S, \sigma_1, \sigma_2, \tau, \kappa)$

```
1  lit ← (σ1 \ σ2)[0]     /* Choose first literal lit s.t.  lit ∈ σ1,
     and lit ∉ σ2 */
2  φ' = φ ∧ (σ1 ∨ σ2)
3  (ψ, τ̂) ← GenHard(τ, κ)
4  φ̂ ← φ' ∧ (lit → ψ)
5  φ̂ ← φ̂ ∧ (¬lit → ConstructChain(τ̂, Supp(ψ)))
6  Ŝ ← S ∪ Supp(φ̂)
7  return (φ̂, Ŝ).
```

Analysis of Various Samplers by ScalBarbarik

Table: Analysis of different samplers with ScalBarbarik. Total of 50 benchmarks. Parameters used: $\epsilon = 0.2, \eta = 1.6, \delta = 0.1$

ScalBarbarik (κ)	QuickSampler		STS		CMSGen_{100}	
	Accept	Reject	Accept	Reject	Accept	Reject
10	0	50	0	50	50	0
11	0	50	0	50	41	9
12	0	50	0	50	19	31
13	0	50	0	50	0	50

ScalBarbarik (κ)	UniGen3	
	Accept	Reject
10	50	0
11	50	0
12	50	0
13	50	0

Analysis of CMSGen by ScalBarbarik

Table: Analysis of CMSGen_{100}, CMSGen_{300}, CMSGen_{500} by ScalBarbarik. Total of 50 benchmarks. Parameters used: $\epsilon = 0.2, \eta = 1.6, \delta = 0.1,$

ScalBarbarik (κ)	CMSGen_{100}		CMSGen_{300}		CMSGen_{500}	
	Accept	Reject	Accept	Reject	Accept	Reject
11	41	9	47	3	47	3
15	0	50	37	13	42	8
18	0	50	0	50	36	14
22	0	50	0	50	0	50

ScalBarbarik (κ)	UniGen3	
	Accept	Reject
11	50	0
15	50	0
18	50	0
22	50	0

Conclusions & Future Work

- ScalBarbarik is a much improved testing tool based on Barbarik, that can help spur a new generation of scalable uniform-like samplers.
- ScalBarbarik came about as a response to the CMSGen, a uniform-like sampler without guarantees, that Barbarik could not distinguish from a true uniform sampler.
- We envisage this cycle to continue: with better samplers come better testers and vice versa.
- Improved uniform-like samplers can help with the scalability of tools: e.g. the Manthan [5] function synthesis tool significantly benefits from CMSGen.

Code: **PDF**:

Thank you for your time

Any questions?

Priyanka, Sourav, and Kuldeep are on-site to answer questions if you have ideas/questions after the talk!

L. A. Clarke, “A program testing system,” in *Proc. of ACM*, 1976.

J. C. King, “Symbolic execution and program testing,” *Comm. ACM*, 1976.

Y. Naveh, M. Rimon, I. Jaeger, Y. Katz, M. Vinov, E. s Marcu, and G. Shurek, “Constraint-based random stimuli generation for hardware verification,” *Proc. of AI magazine*, 2007.

S. Roy, A. Pandey, B. Dolan-Gavitt, and Y. Hu, “Bug synthesis: Challenging bug-finding tools with deep faults,” in *Proc. of ESEC/FSE*, 2018.

P. Golia, S. Roy, and K. S. Meel, “Manthan: A data-driven approach for Boolean function synthesis,” in *Proc. of CAV*, 2020.

S. Chakraborty, K. S. Meel, and M. Y. Vardi, “Balancing scalability and uniformity in SAT witness generator,” in *Proc. of DAC*, 2014.

S. Chakraborty, K. S. Meel, and M. Y. Vardi, “A scalable and nearly uniform generator of sat witnesses,” in *Proc. of CAV*, 2013.

M. Soos, S. Gocht, and K. S. Meel, “Tinted, detached, and lazy CNF-XOR solving and its applications to counting and sampling,” in *Proc. of CAV*, 2020.

D. Achlioptas, Z. S. Hammoudeh, and P. Theodoropoulos, “Fast sampling of perfectly uniform satisfying assignments,” in *Proc. of SAT*, 2018.

S. Sharma, R. Gupta, S. Roy, and K. S. Meel, “Knowledge compilation meets uniform sampling.,” in *Proc. of LPAR*, 2018.

S. Ermon, C. P. Gomes, A. Sabharwal, and B. Selman, “Uniform solution sampling using a constraint solver as an oracle,” in *Proc. of UAI*, 2012.

R. Dutra, K. Laeufer, J. Bachrach, and K. Sen, “Efficient sampling of SAT solutions for testing,” in *Proc. of ICSE*, 2018.

P. Golia, M. Soos, S. Chakraborty, and K. S. Meel, “Designing samplers is easy: The boon of testers,” in *Proc. of FMCAD*, 2021.

S. Chakraborty and K. S. Meel, “On testing of uniform samplers,” in *Proc. of AAAI*, 2019.

V. Nossum, “SAT-based preimage attacks on SHA-1,” Master’s thesis, University of Oslo (2012), https://www.duo.uio.no/handle/10852/34912.