On Testing of Uniform Samplers*†

Sourav Chakraborty
Indian Statistical Institute
Kolkata

Kuldeep S. Meel
School of Computing
National University of Singapore

Abstract

Recent years have seen an unprecedented adoption of artificial intelligence in a wide variety of applications ranging from medical diagnosis, automobile industry, security to aircraft collision avoidance. Probabilistic reasoning is a key component of such modern artificial intelligence systems. Sampling techniques form the core of the state of the art probabilistic reasoning systems.

The divide between the existence of sampling techniques that have strong theoretical guarantees but fail to scale and scalable techniques with weak or no theoretical guarantees mirrors the gap in software engineering between poor scalability of classical program synthesis techniques and billions of programs that are routinely used by practitioners. One bridge connecting the two extremes in the context of software engineering has been *program testing*. In contrast to testing for deterministic programs, where one trace is sufficient to prove the existence of a bug, in case of samplers one sample is typically not sufficient to prove non-conformity of the sampler to the desired distribution. This makes one wonder *whether it is possible to design testing methodology to test whether a sampler under test generates samples close to a given distribution.*

The primary contribution of this paper is an affirmative answer to the above question when the given distribution is a uniform distribution: We design, to the best of our knowledge, the first algorithmic framework, Barbarik, to test whether the distribution generated is $\varepsilon-$close or $\eta-$far from the uniform distribution. In contrast to the sampling techniques that require an exponential or sub-exponential number of samples for sampler whose support can be represented by n bits, Barbarik requires only $\mathcal{O}(1/(\eta-\varepsilon)^4)$ samples. We present a prototype implementation of Barbarik and use it to test three state of the art uniform samplers over the support defined by combinatorial constraints. Barbarik can provide a certificate of uniformity to one sampler and demonstrate non-uniformity for the other two samplers.

*The author list has been sorted alphabetically by last name; this should not be used to determine the extent of authors' contributions.

†An extended version of the paper along with open source tool is available at `https://github.com/meelgroup/barbarik`

1 Introduction

Recent years have seen an unprecedented adoption of artificial intelligence (AI) in a wide variety of applications ranging from medical diagnosis, automobile industry, security to aircraft collision avoidance. Such an unprecedented proliferation owes to the ability of modern AI-based systems to almost match human or surpass human abilities for several complex tasks. Probabilistic reasoning is a key component of such modern artificial intelligence systems. The usage of AI-based systems in domains where the decisions can have significant consequential effects on our lives has highlighted the need for verification and testing of AI systems (Seshia, Sadigh, and Sastry 2016). The modern AI systems significantly differ from traditional systems in their ability to learn and usage of probabilistic reasoning. In traditional verification, a trace of execution of a program is sufficient to prove existence of a bug. Such is not the case for AI systems whose behavior is inherently probabilistic. This has led to the call of development of *randomized formal methods* for synthesis, verification, and testing of AI systems. In this paper, we focus on a core component employed in the construction of modern AI systems: sampling from a discrete distribution specified by a probabilistic model.

The sampling techniques based on Monte Carlo Markov Chain (MCMC) techniques are employed to sample from the posterior distribution represented by the probabilistic model under consideration (Jerrum and Sinclair 1996). The correctness of such sampling methods requires that the underlying Markov Chain *mixes properly* (Jerrum and Sinclair 1996). Since mixing times of the underlying Markov Chains are often exponential, several heuristics have been proposed over the years. While the heuristics would not provide theoretical guarantees of mixing in polynomial time, such techniques tend to behave well in practice. On the other end of the spectrum of lie a wide spectrum of approaches such as variational inference (Jordan et al. 1999), rejection sampling (Gilks and Wild 1992), importance sampling (Neal 2001), hashing-based techniques (Chakraborty, Meel, and Vardi 2013; Ermon et al. 2013b; Chakraborty et al. 2015a), and the like.

Given the computational intractability of sampling (Koller and Friedman 2009), the implementation of different paradigms often resort to the usage of heuristics (Koller and Friedman 2009). The widespread usage of

heuristics in different sampling-based techniques creates a gap between theory and practice: In theory, heuristics would nullify guarantees while in practice the heuristics seem to work well for problems arising from real-world instances (Koller and Friedman 2009). Often statistical tests are employed to argue for the quality of distributions, but such statistical tests are usually performed on a very small number of samples for which no theoretical guarantees exist for their accuracy (Dutra et al. 2018). In contrast to testing for deterministic programs, where one trace is sufficient to prove the existence of a bug; such is not the case for samplers as one sample is typically not sufficient to prove non-conformity of the sampler to the desired distribution. This makes one wonder *whether it is possible to design testing methodology to test whether a sampler under test generates samples close to a given distribution.*

The primary contribution of this paper is an affirmative answer to the above question when the given distribution is a uniform distribution: We design, to the best of our knowledge, the first algorithmic framework, Barbarik[1], to test whether the distribution generated is $\varepsilon-$close or $\eta-$far from the uniform distribution. In contrast to the sampling techniques that require the exponential or sub-exponential number of samples for sampler whose support can be represented by n bits, Barbarik requires only $\mathcal{O}(1/(\eta-\varepsilon)^4)$ samples.

The key technical ideas in the design of Barbarik sit at the intersection of *property testing* and *constrained counting*, in particular, from the works on conditional sampling (Chakraborty et al. 2016; Canonne, Ron, and Servedio 2015; Chakraborty et al. 2015b). The design of Barbarik, to the best of our knowledge, is among one of the first usage of property testing in the context of verification.

To demonstrate practical efficiency of Barbarik, we developed a prototype implementation in Python and sought out to test three state of the art uniform samplers. Given the ubiquity of applications of uniform sampling, the three samplers under test were drawn from different domains with different underlying techniques: (i) SearchTreeSampler was designed to sample from highly challenging domains such as energy barriers and highly asymmetric spaces and was shown to outperform traditional approaches based on stimulated annealing and Gibbs sampling (Ermon, Gomes, and Selman 2012), (ii) Quicksampler was designed in the context of software testing and employs usage of random perturbations for sampling (Dutra et al. 2018), and (iii)UniGen2 was designed in the context of constrained random simulation and employs universal hashing-based techniques (Chakraborty, Meel, and Vardi 2013; Chakraborty et al. 2015a). While SearchTreeSampler and Quicksampler do not have formal guarantees of uniformity, UniGen2 provides guarantees of *almost-uniformity*. Another motivation for choice of the above three samplers was recent comparison of their uniformity based on statistical tests, which failed to distinguish between the samplers (Dutra et al. 2018). In contrast, Barbarik can provide the certificate of uniformity to UniGen2 demonstrate non-uniformity for the other two samplers.

Given the ubiquity of sampling-based techniques, we believe that Barbarik will allow developers of heuristics to test the quality of their samplers and accordingly tune their samplers akin to the way testing framework supports code development in software engineering.

[1]In Indian mythology, Barbarik's head watched the entire Mahabharata war and was suggested as a judge by Krishna to settle arguments among Pandavas about their contribution to the victory.

2 Notations and Preliminaries

A literal is a Boolean variable or its negation. Let φ be a Boolean formula in conjunctive normal form (CNF), and let X be the set of variables appearing in φ. The set X is called the *support* of φ, denoted by $Supp(\varphi)$. Given an array $\mathbf{a}$, $\mathbf{a}[i:j]$ represents the sub-array consists of all the elements of $\mathbf{a}$ between indices i and j. A *satisfying assignment* or *witness*, denoted by σ, of is an assignment of truth values to variables in its support such that φ evaluates to true. A satisfying assignment is also represented as a set of literals. For $S \subseteq X$, we use $\sigma_{\downarrow S}$ to indicate the projection of σ over the set of variables S. We denote the set of all witnesses of φ as R_φ. For notational convenience, whenever the formula φ is clear from the context, we omit mentioning it.

2.1 Uniform Generators

We use $\mathsf{Pr}\left[X\right]$ to denote the probability of event X. Given a Boolean formula φ, a *probabilistic generator* $\mathcal{G}$ of witnesses of φ is a probabilistic algorithm that generates a random witness in R_φ. We use $p_{\mathcal{G}}(\varphi, x)$ to denote the probability that $\mathcal{G}(\varphi,\cdot)$ generates x. We use $\mathcal{D}_{\mathcal{G}(\varphi)}$ to denote the distribution induced by $\mathcal{G}$ over the solution set of φ. For a set $T \subseteq R_\varphi$, we use $\mathcal{D}_{\mathcal{G}(\varphi)} \mid T$ to denote the distribution $\mathcal{D}_{\mathcal{G}(\varphi)}$ conditioned on the set T.

Definition 1. *Given a Boolean formula φ, A* uniform generator $\mathcal{G}^u(\varphi)$ *is a probabilistic generator that guarantees*

$$\forall y \in R_\varphi, \mathsf{Pr}\left[\mathcal{G}^u(\varphi)=y\right] = 1/|R_\varphi|, \tag{1}$$

Definition 2. *Given a Boolean formula φ and tolerance parameter ε, $\mathcal{G}^{aau}(\varphi,\varepsilon)$ is an additive* almost-uniform generator *(AAU) if the following holds:*

$$\forall y \in R_\varphi, \frac{1-\varepsilon}{|R_\varphi|} \leq \mathsf{Pr}\left[\mathcal{G}^{au}(F,\varepsilon)=y\right] \leq \frac{1+\varepsilon}{|R_\varphi|} \tag{2}$$

Definition 3. *A multiplicative* almost-uniform generator *(MAU) $\mathcal{G}^{amu}(\cdot,\cdot)$ if the following holds:*

$$\forall y \in R_\varphi, \frac{1}{(1+\varepsilon)|R_\varphi|} \leq \mathsf{Pr}\left[\mathcal{G}^{au}(F,\varepsilon)=y\right] \leq \frac{1+\varepsilon}{|R_\varphi|} \tag{3}$$

where $\varepsilon > 0$ is the specified tolerance*. Note that every MAU is an AAU but not vice versa.*

Definition 4. *A* near-uniform generator $\mathcal{G}^{nu}(\cdot)$ *further relaxes the guarantee of uniformity, and ensures that $\mathsf{Pr}\left[\mathcal{G}^{nu}(\varphi)=y\right] \geq c/|R_\varphi|$ for a constant c, where $0 < c \leq 1$.*

Probabilistic generators are allowed to occasionally "fail" in the sense that no witness may be returned even if R_φ is non-empty. The failure probability for such generators must be bounded by a constant strictly less than 1.

Definition 5. *Given a Boolean formula φ and an intolerance parameter η an generator $\mathcal{G}(\varphi,.)$ is η-far from uniform generator if the ℓ_1-distance (or, twice the variation distance) $\mathcal{D}_{\mathcal{G}(\varphi)}$ from uniform is at least η. That is, $\sum_{x\in R_\varphi}\left|p_{\mathcal{G}(\varphi,x)}-\frac{1}{|R_\varphi|}\right|\geq\eta$.*

2.2 Sampler Verifier

Definition 6. *Given a Boolean formula φ, a sampler $\mathcal{G}$, tolerance parameter ε, an intolerance parameter η, a sampler verifier $\mathcal{T}(\cdot,\cdot,\cdot,\cdot)$ returns ACCEPT or REJECT (with a witness) with the following guarantees:*

1. *If the sampler $\mathcal{G}(\varphi,\varepsilon)$ is an additive almost-uniform generator (AAU), then $\mathcal{T}(G,\varphi,\varepsilon,\eta)$ returns ACCEPT with probability at least $1-\delta$*
2. *If the sampler $\mathcal{G}(\varphi,\varepsilon)$ is η-far from uniform generator, then $\mathcal{T}(G,\varphi,\varepsilon,\eta)$ returns REJECT with probability at least $1-\delta$*

The focus of this paper is design a sampler verifier $\mathcal{T}$.

2.3 Maximum-Likelihood Estimator

Maximum-Likelihood Estimation (MLE) is a standard technique for estimating parameters of a distribution. One particular case is using MLE for estimating the probability of a Bernoulli distribution, which is a well-studied area as it is same as estimating the bias of a biased coin. In this case the maximum-likelihood estimator simple tosses the coin n times and outputs n_1/n as the estimate of the bias, where, n_1 is the number of times it observes HEAD in the n coin tosses. The following guarantee follows easily from the concentration inequalities like Chernoff Bound.

Lemma 1. *If a biased coin, with probability p of falling HEADS, is tossed n times then with probability $e^{O(\gamma^2 n)}$ the estimate n_1/n is within an additve error of γ from p, where, n_1 is the number of heads observed. That is, $p-\gamma\leq n_1/n\leq p+\gamma$.*

Optimizing the constants, we have that if $n=1/9\gamma^2$ then with probability $1/2$ the estimate is within an additive error of γ. We use the following definition:

Definition 7. *Let $M(\gamma)$ be the number of times a biased coin has to be flipped to estimate the bias up to $\pm\gamma$ with probability $\geq 1/2$. Note that we have that $M(\gamma)$ is at most $1/9\gamma^2$.*

Note that if the number of tosses is $M(\gamma)\times m$ then the probability that the additive error is less than γ is at least $(1-(1/2)^m)$.

2.4 Chain Formulas

Our work uses a special class of Boolean formulas, called *chain formulas*, which were introduced in (Chakraborty et al. 2015b) and inductively defined as follows. Every literal (i.e., a variable or its complement) is a chain formula. Besides, if l is a literal and ψ is a chain formula in which neither l nor $\neg l$ appears, then $(l\vee\psi)$ and $(l\wedge\psi)$ are also chain formulas. Note that chain formulas can be represented as $l_1\,C_1\,(l_2\,C_2\,(\cdots(l_{n-1}\,C_{n-1}\,l_n)\cdots))$, where the l_i's are literals and every C_i is either the connector "$\vee$" or the connector "$\wedge$". It is easy to see from De Morgan's laws that if ψ is a chain formula, then so is $\neg\psi$. The following lemma show that every chain formula can be efficiently represented in both CNF and DNF.

Lemma 2. *(Chakraborty et al. 2015b) Every chain formula ψ on n variables is equivalent to a CNF (resp., DNF) formula ψ^{CNF} (resp., ψ^{DNF}) having at most n clauses. In addition, $|\psi^{\mathsf{CNF}}|$ (resp., $|\psi^{\mathsf{DNF}}|$) is in $O(n^2)$.*

Let $m>0$ be a natural number, and $k<2^m$ be a positive odd number. Let $c_1c_2\cdots c_m$ be the m-bit binary representation of k, where c_m is the least significant bit. We then construct a chain formula $\varphi_{k,m}(\cdot)$ on m variables $a_1,\ldots a_m$ as follows. For every j in $\{1,\ldots m-1\}$, let C_j be the connector "$\vee$" if $c_j=1$, and the connector "$\wedge$" if $c_j=0$. Define

$$\psi_{k,m}(a_1,\cdots a_m)=a_1\,C_1\,(a_2\,C_2(\cdots(a_{m-1}\,C_{m-1}\,a_m)\cdots))$$

For example, consider $k=5$ and $m=4$. The binary representation of 5 using 4 bits is 0101. Therefore, $\varphi_{5,4}(a_1,a_2,a_3,a_4)=a_1\wedge(a_2\vee(a_3\wedge a_4))$. The following lemma shows that $\psi_{k,m}(\cdot)$ has exactly k satisfying assignments.

Lemma 3. *(Chakraborty et al. 2015b) Let $m>0$ be a natural number, $k<2^m$, and $\psi_{k,m}$ as defined above. Then $|\psi_{k,m}|$ is linear in m and $\psi_{k,m}$ has exactly k satisfying assignments.*

3 Background

Since sampling techniques form the core of the state of the art inference techniques, researchers have investigated several practical approaches to designing samplers. With the proposal of every new sampling techniques, a significant effort is put for the evaluation of the proposed techniques. The literature bears testimony to the focus on runtime performance comparison of the proposed technique with the previous state of the art (See, for example (Chakraborty, Meel, and Vardi 2013; Ermon et al. 2013a; Meel 2014; Chakraborty et al. 2015a; Meel 2017; Dutra et al. 2018; Sharma et al. 2018; Achlioptas, Hammoudeh, and Theodoropoulos 2018)). On the other hand, there is a less rigorous evaluation of the quality of generated samples. The hardness of testing properties of distribution is a major contributor to such a trend.

In most of the published articles, whenever the quality of the of the output distribution $\mathcal{D}_{\mathcal{G}(.)}$ (for a given probabilistic generator $\mathcal{G}$ is tested), the standard techniques applied are various versions of χ^2-tests. That is, the tests involve studying various parameters about the frequency vector of the outputs of the algorithms when run on various benchmark data. For example, in (Kitchen and Kuehlmann 2007; Dutra et al. 2018) the authors look at the number of unique outputs from various samplers to argue about the quality of the distribution. All these standard tests run the generator on a benchmark test input φ and collect a number of outputs of the generator on that particular formula φ and then analyses the output. Or in other words, the tester draws a number of

samples according to the distribution $\mathcal{D}_{\mathcal{G}(\varphi)}$, for a particular φ, and then analyses the samples to come up with a decision on the quality of the distribution. The testers might run this test on multiple different φ. Unfortunately, (Batu et al. 2013) proved that any test that uses only random samples from the distribution and would with probability at least $(1-\delta)$, accept the distribution if it is ϵ-close to the uniform distribution and, with probability at most δ accept if the distribution is η-far from uniform would need $\Omega(\sqrt{|R_\varphi|}\log(\delta^{-1})/(\eta-\epsilon)^2)$ samples. Since, for most benchmark input φ the set R_φ is large, typically larger than 2^{50}, the number of samples that is used to prove the quality of the distribution is usually not sufficient to provide rigorous guarantees.

Since testing whether a distribution is a uniform by drawing samples from the distribution is very expensive (and hence impractical), a number of other sophisticated ways to access the distribution has been studied in the subject of property testing (Batu et al. 2005; Batu, Kumar, and Rubinfeld 2004; Chakraborty et al. 2010). Either the models were very theoretical (and not practically implementable) or the models did not help in significantly reducing the number of samples needed.

Recently, (Chakraborty et al. 2016) and (Canonne, Ron, and Servedio 2015) proposed a new model called *conditional sampling* They showed that in this new model if one has access to *conditional sampling* from the distribution then the number of samples needed to test if a distribution is ϵ-close to uniform or η-far from uniform is $O(\log(\delta^{-1})/(\eta-\epsilon)^2)$, that is, it is independent of the size of the domain on which the distribution is supported. Since then, the *conditional sampling* model has been used to show significant theoretical improvements for testing various properties of distributions, but, no one has used the model to implement a tester in real life. The primary challenge is to draw conditional samples from a distribution, which we address in the next Section.

4 Barbarik **Algorithm**

We now discuss the primary contribution of this paper: a novel algorithmic framework, Barbarik that verifies whether a sampler is an almost-uniform generator (AAU). The pseudocode of Barbarik is presented in Algorithm 1.

If $\mathcal{G}$ is a probabilistic generator then $\mathcal{A}_{\mathcal{G}}(.,.,.)$ is a subroutine that takes a formula φ, a set $S \subseteq Supp(\varphi)$ and a number τ and uses $\mathcal{G}$ to generate τ satisfying assignments of φ and outputs the τ assignments of the variables in S. For simplicity, in the rest of the section and in the pseudocodes we would use $\mathcal{A}(.,.,.)$ instead of $\mathcal{A}_{\mathcal{G}}(.,.,.)$. If $\mathcal{G}_u$ is a uniform generator then we rename the subrountine $\mathcal{A}_{\mathcal{G}}(.,.,.)$ as $\mathcal{U}(.,.,.)$.

Barbarik takes in a sampler $\mathcal{G}$, a uniform generator $\mathcal{U}$, a tolerance parameter $\varepsilon > 0$, an intolerance parmaeter $\eta > \varepsilon$, a guarantee parameter δ and a CNF formula φ and returns ACCEPT or REJECT with the following guarantees:

- if the generator $\mathcal{G}(\varphi,.)$ is an ε-additive almost-uniform generator then Barbarik ACCEPTS with probability at least $(1-\delta)$.
- if $\mathcal{G}(\varphi,.)$ is η-far from a uniform generator then Barbarik REJECTS with probability at least $1-\delta$.
- if Barbarik outputs REJECT it also outputs a witness in the form of a CNF formula $\hat{\varphi}$ such that there are exactly two assignments of variable in $S \subseteq Supp(\varphi)$ that can be extended to a satisfying assignment of $\hat{\varphi}$ and it can be easily tested, emperically, that the distribution $\mathcal{D}_{\mathcal{G}(\hat{\varphi})}$ is η-far from uniform.

As discussed in the previous section, the primary challenge The core idea of Barbarik is that for evaluating quality of the distribution $\mathcal{D}_{\mathcal{G}(\varphi)}$ we not only draw samples from $\mathcal{D}_{\mathcal{G}(\varphi)}$ but also from another distribution $\mathcal{D}_{\mathcal{G}(\hat{\varphi})}$ where $\hat{\varphi}$ is obtained from φ. Therefore, Barbarik is able to *beat* the lower bound on the sample complexity of the standard χ^2-based test,

The main idea of the algorithm Barbarik is that if we draw one sample σ_1 according to the distribution $\mathcal{D}_{\mathcal{G}(\varphi)}$ and one sample σ_2 according to the uniform distribution over R_φ and let $T = \{\sigma_1, \sigma_2\}$ then is the following happens:

- if $\mathcal{D}_{\mathcal{G}(\varphi)}$ is close to the uniform distribution over R_φ then the conditional distribution $\mathcal{D}_{\mathcal{G}(\varphi)}|T$ is also close to uniform over the set T.
- if $\mathcal{D}_{\mathcal{G}(\varphi)}$ is far from the uniform distribution over R_φ then the conditional distribution $\mathcal{D}_{\mathcal{G}(\varphi)}|T$ is also far from the uniform over the set T.

And since $\mathcal{D}_{\mathcal{G}(\varphi)}|T$ is a distribution over a two element set it is easy to estimate the distance of $\mathcal{D}_{\mathcal{G}(\varphi)}|T$ from uniform if we can draw random samples from $\mathcal{D}_{\mathcal{G}(\varphi)}|T$. The algorithm then repeats the subroutine appropriate number of times to ensure the completeness, soundness, and accuracy of the final algorithm.

Barbarik assumes access to two subroutines, Bias and Kernel. While the Bias is a simple subroutine to estimate the distance of $\mathcal{D}_{\mathcal{G}(\varphi)}|T$ from uniform, the subroutine Kernel ensures a way to access samples from the distribution $\mathcal{D}_{\mathcal{G}(\varphi)}|T$.

Bias takes in two lists L_1, L_2 of assignments and a sampling set S as input and returns the cardinality of the intersection of the L_1 and L_2, where elements are compared under projection over the sampling set S.

Kernel takes in a Boolean formula φ, two assignments σ_1 and σ_2, and desired number of solutions τ and returns a formula $\hat{\varphi}$ such that the following hold true:

1. $|R_{\hat{\varphi}}| = 2\tau$
2. $Supp(\varphi) \subseteq Supp(\hat{\varphi})$
3. $|\{x \in R_{\hat{\varphi}} \mid x_{\downarrow S} = \sigma_1\}| = |\{x \in R_{\hat{\varphi}} \mid x_{\downarrow S} = \sigma_2\}|$, where $S = Supp(\varphi)$.
4. φ and $\hat{\varphi}$ has similar structure

Barbarik has two for loops - one outer (line 2-14) and one inner (line 6 - 14). The outer for loop has $\lceil \log(2/\varepsilon + \eta) \rceil$ rounds. In the inner for loop, in each round, the algorithm draws one sample σ_1 according to the distribution $\mathcal{D}_{\mathcal{G}(\varphi)}$ (line 8) and one sample σ_2 according to the uniform distribution on R_φ (line 9). In line 10, the subroutine Kernel uses φ, the two samples σ_1 and σ_2, and a number N_j to output

a new formula $\hat{\varphi}$ such that $Supp(\varphi) \subseteq Supp(\hat{\varphi})$. In line 11, Barbarik draws a list, L_3, of N_j samples accoding to the distribution $\mathcal{D}_{\mathcal{G}(\hat{\varphi})}$. Kernel ensures that for all $\sigma \in L_3$, $\sigma_{\downarrow S}$ is either σ_1 or σ_2. In line 12 Barbarik uses Bias to compute the fraction of N_j samples that is equal to σ_1 (on the variable set $S = Supp(\varphi)$), and if the fraction is not between $(1 - c_j)/2$ and $(1 + c_j)/2$ then Barbarik REJECTS (in line 14).

Algorithm 1 Barbarik($\mathcal{A}, \mathcal{U}, S, \varepsilon, \eta, \delta, \varphi$)

1: $S \leftarrow Supp(\varphi)$
2: **for** $j = 1$ to $\lceil \log(\frac{4}{\varepsilon+\eta}) \rceil$ **do**
3: $\quad t_j \leftarrow \lceil 2^j \frac{(\eta+\varepsilon)}{(\eta-\varepsilon)^2} \log(4(\varepsilon+\eta)^{-1})(\frac{4e}{(e-1)}) \ln(\delta^{-1}) \rceil$
4: $\quad \beta_j \leftarrow \frac{(2^{j-1}+1)(\varepsilon+\eta)}{4+(\varepsilon+\eta)(2^{j-1}-1)}$; $c_j \leftarrow (\beta_j + \varepsilon)/2$
5: $\quad N_j \leftarrow \lceil M(\frac{\beta_j-\varepsilon}{4}) \log\left(\frac{2^4 e}{e-1} \frac{\delta^{-1}}{(\eta-\varepsilon)^2} \log(\frac{4}{\varepsilon+\eta}) \ln(\frac{1}{\delta})\right) \rceil$
6: $\quad$ **for** $i = 1$ to t_j **do**
7: $\quad\quad$ **while** $L_1 = L_2$ **do**
8: $\quad\quad\quad L_1 \leftarrow \mathcal{A}(\varphi, S, 1)$; $\sigma_1 \leftarrow L_1[0]$
9: $\quad\quad\quad L_2 \leftarrow \mathcal{U}(\varphi, S, 1)$; $\sigma_2 \leftarrow L_2[0]$
10: $\quad\quad \hat{\varphi} \leftarrow kernel(\varphi, \sigma_1, \sigma_2, N_j)$
11: $\quad\quad L_3 \leftarrow \mathcal{A}(\hat{\varphi}, S, N_j)$
12: $\quad\quad b \leftarrow Bias(\sigma_1, L_3, S)$
13: $\quad\quad$ **if** $b < \frac{1}{2}(1 - c_j)$ or $b > \frac{1}{2}(1 + c_j)$ **then**
14: $\quad\quad\quad$ **return** REJECT
15: **return** ACCEPT

Algorithm 2 Bias($\hat{\sigma}$, L, S)

1: $count = 0$
2: **for** $\sigma \in L$ **do**
3: $\quad$ **if** $\sigma_{\downarrow S} = \hat{\sigma}$ **then**
4: $\quad\quad$ count $\leftarrow$ count +1
5: $\quad$ **return** $\frac{count}{|L|}$

Algorithm 3 presents the pseudocode of subroutine Kernel. As stated above, Kernel takes in a Boolean formula φ, two assignments σ_1, σ_2, and a desired number of solutions τ. Kernel assumes access to subroutine $NewVars$ which takes in two parameters, a formula φ and a number M, and returns a set of M variables that do not appear in φ. Kernel first constructs two sets of literals, denoted by $Lits_1$ (resp. $Lits_2$), which appear in σ_1 (resp. σ_2) but not σ_2 (resp. σ_1). Then, we construct two lists of factors of τ, K_1 and K_2, such that $\tau = \prod_{i=1}^{|Lits_1|} K_1[i]$ and $\tau = \prod_{i=1}^{|Lits_2|} K_2[i]$ (lines 3 and 4).

We then construct the formula $\hat{\varphi}$ in the code block from line 7 to line 16. First, we conjunct φ with $\sigma_1 \vee \sigma_2$ where $\sigma_1 \vee \sigma_2$ represents the formula that has exactly two satisfying assignments σ_1 and σ_2. The loop in lines 8– 11 constructs $\hat{\varphi}$ by conjuncting φ with subformulas of the form $l \leftrightarrow \psi_{k,m}(\mathbf{a})$, where $\psi_{k,m}$ is a chain formula over variables $a_1, a_2, \cdots a_m$ which has exactly k satisfying assignments. Therefore, at the end of the second loop, i.e. line 16, φ has $K_2 \times K_1 \times 2$ solutions. The subroutine allows randomization in the subroutine ComputeFactor because (i) if the sampler under test is an almost-uniform generator, then the randomization would not affect the distribution generated by $\mathcal{A}$ over $\hat{\varphi}$, (ii) on the contrary, if the sampler under test is not an almost-uniform generator, then we would not want to construct a formula that can be easily guessed by $\mathcal{A}$ and the sampler be somehow optimized for the formula we construct and be able to *fool* Barbarik by behaving as almost-uniform sampler over φ but without being an almost uniform sampler over φ.

Algorithm 3 $kernel(\varphi, \sigma_1, \sigma_2, \tau)$

1: $Lits_1 \leftarrow (\sigma_1 \setminus \sigma_2)$
2: $Lits_2 \leftarrow (\sigma_2 \setminus \sigma_1)$
3: $K_1 \leftarrow ComputeFactors(Lits_1, \tau)$
4: $K_2 \leftarrow ComputeFactors(Lits_2, \tau)$
5: $M \leftarrow \sum_{i=1}^{|Lits_1|} \lceil \log K[i] \rceil$
6: $\mathbf{a} \leftarrow NewVars(\varphi, m)$; $index \leftarrow 0$
7: $\hat{\varphi} \leftarrow \varphi \wedge (\sigma_1 \vee \sigma_2)$
8: **for** $(l, k) \in (Lits_1, K_1)$ **do**
9: $\quad m \leftarrow \lceil \log k \rceil$
10: $\quad \hat{\varphi} \leftarrow (\hat{\varphi} \wedge (l \leftrightarrow \psi_{k,m}(\mathbf{a}[index : index + m]))$
11: $\quad index \leftarrow index + m$
12: $currIndex \leftarrow 0$
13: **for** $(l, k) \in (Lits_2, K_2)$ **do**
14: $\quad m \leftarrow \lceil \log k \rceil$
15: $\quad \hat{\varphi} \leftarrow (\hat{\varphi} \wedge (l \leftrightarrow \psi_{k,m}(\mathbf{a}[index : index + m]))$
16: $\quad index \leftarrow index + m$
17: **return** $\hat{\varphi}$

4.1 Theoretical Analysis

In this section, we present theoretical analysis of Barbarik. We first show completeness of Barbarik and then demonstrate soundness under certain assumption, which is shown to hold true in practice in our experimental analysis.

Completeness

Theorem 1. *If the generator $\mathcal{G}$ is an ε-additive almost-uniform generator -approximates any CNF formula φ then* Barbarik *ACCEPTS with probability at least $(1 - \delta)$.*

The proof Theorem 1 uses multiple applications of the Chernoff and Union Bound. For lack of space we defer the proof of Theorem 1 to extended version.

Soundness

Definition 8. *The **non-adversarial sampler assumption** states that if $\mathcal{A}(\varphi, S, 1)$ outputs a sample by drawing according to a distribution $\mathcal{D}$ then the $(\hat{\varphi}, \hat{S})$ obtained from $kernel(\varphi, L_1, L_2, N)$ has the property that:*

- $S \subset \hat{S}$,
- *There are only two set of assignments to variables in S that can be extended to a satisfying assignment for $\hat{\varphi}$*
- *The restriction of the set L_3 to the variable set S is obtained by drawing N samples from the distribution $\mathcal{D}\mid_R$, where $R = L_1 \cup L_2$.*

The following theorem states that if non-adversarial sampler assumption holds then Barbarik is sound. Note that completeness does not require non-adversarial sampler assumption to hold.

Theorem 2. *If non-adversarial sampler assumption holds and if the distribution $\mathcal{D}_{\mathcal{G}(\varepsilon)}$ is η-far from uniform on the sampling set S then* Barbarik *REJECTS with probability at least $1-\delta$.*

Proof. Let define the set H as follows (let $N = |R_\varphi|$):$H = \{x \in R_\varphi \ : \ p_{\mathcal{G}}(\varphi, x) \geq \frac{1}{N}\}$. Since $\mathcal{G}(\varphi, .)$ is η-far from uniform generator, so $\sum_{x\in H}\left(p_{\mathcal{G}}(\varphi, x) - \frac{1}{N}\right) \geq \frac{\eta}{2}$.

Let us further subdivide H into $H_0, H_1, \ldots, H_r$ (with $r = \lceil \log(4/(\eta+\varepsilon))\rceil$) as follows, $H_0 = \{x : p(x) < \left(1+\frac{\eta+\varepsilon}{4}\right)\frac{1}{N}\}$, and, $H_r = \{x \ : \ p_{\mathcal{G}}(\varphi, x) \geq \frac{2}{N}\}$ And for all $1 \leq j \leq (r-1)$, $H_j = \left\{x : \left(1+2^{j-1}\frac{\eta+\varepsilon}{4}\right) \leq N p_{\mathcal{G}}(\varphi, x) < \left(1+2^{j}\frac{\eta+\varepsilon}{4}\right)\right\}$

Thus $\sum_{x\in H_0}(p_{\mathcal{G}}(\varphi, x) - \frac{1}{N}) < (\eta+\varepsilon)/4$. So there exist $1 \leq j(H) \leq r$ such that $\sum_{x\in H_{j(H)}}\left(p_{\mathcal{G}}(\varphi, x) - \frac{1}{N}\right) \geq \frac{\eta-\varepsilon}{4r}$

Claim 1. *In the iteration $j = j(H)$ if we had picked at least $r\frac{2^j(\varepsilon+\eta)}{\eta-\varepsilon}$ pairs of samples (x, y) such that x is sampled according to $\mathcal{D}_{\mathcal{G}(\varphi)}$ and y is sampled uniformly from $\mathcal{U}$ then with probability at least $(1-\frac{1}{e})(\frac{\eta-\varepsilon}{4})$ we would have picked a pair (x_i, y_i) such that*

- $x_i \in H_{j(H)}$, *and*
- $p_{\mathcal{G}}(\varphi, y_i) \leq (1-\frac{\varepsilon+\eta}{4})\frac{1}{N}$

Proof. Since $\sum_{x\in H_{j(H)}}\left(p_{\mathcal{G}}(\varphi, x) - \frac{1}{N}\right) \geq \frac{\eta-\varepsilon}{4r}$ and for all x in $H_{j(H)}$, $p_{\mathcal{A}}(\varphi, x)$ is at most $(1+2^j\frac{\eta+\varepsilon}{4})\frac{1}{N}$ so we have $\sum_{x\in H_{j(H)}}\left(\left(1+2^j\frac{\eta+\varepsilon}{4}\right)\frac{1}{N} - \frac{1}{N}\right) \geq \frac{\eta-\varepsilon}{4r}$ and, hence $|H_{j(H)}|$ is at least $\frac{N}{2^j r}\left(\frac{\eta-\varepsilon}{\varepsilon+\eta}\right)$. So if we pick $2^j r\left(\frac{\varepsilon+\eta}{\eta-\varepsilon}\right)$ pairs from $\mathcal{D}_{\mathcal{G}(\varphi)}$ with probability at least $(1-1/e)$ one of the picked sample would be from $H_{j(H)}$.

On the other hand, note that since $\sum_{y\in R_\varphi\setminus H}\left(\frac{1}{N} - p_{\mathcal{G}}(\varphi, y)\right) = \frac{\eta}{2}$ so, by averaging argument, the number of $y \in R_\varphi$ such that $p_{\mathcal{G}}(\varphi, y_i) \leq (1-\frac{\varepsilon+\eta}{4})\frac{1}{N}$ is at least $N(\eta-\epsilon)/4$. So when y is drawn uniformly from the set R_φ then, the probability that y is such that $p_{\mathcal{G}}(\varphi, y) \leq (1-\frac{\varepsilon+\eta}{4})\frac{1}{N}$, is at least $(\eta-\varepsilon)/4$.

Since x and y are drawn independently so probability that (x, y) satisfies the condition of the claim is the product of the probability that $x \in H_{j(H)}$ and the probability that $p_{\mathcal{G}}(\varphi, y_i) \leq (1-\frac{\varepsilon+\eta}{4})\frac{1}{N}$. And hence our claim. □

Once we have a pair of samples (x_i, y_i) such that $x_i \in H_{j(H)}$ and $p(y_i) \leq (1-\frac{\varepsilon+\eta}{2})\frac{1}{N}$, if we draw samples from $\mathcal{D}_{T_i}$ (where $T_i = \{x_i, y_i\}$) then, assuming the non-adversarial sampler assumption, the probability of getting x_i is $p_{\mathcal{G}}(\varphi, x_i)/\left(p_{\mathcal{G}}(\varphi, x_i) + p_{\mathcal{G}}(\varphi, y_i)\right)$, by simple calculus it is easy to see that the probability of getting x_i is at least

$$(1+2^{j-1}\frac{\eta+\varepsilon}{2})/((1+2^{j-1}\frac{\eta+\varepsilon}{2}) + (1-\frac{\varepsilon+\eta}{2}))$$

which is $\frac{1}{2}(1+\beta_j)$.

So if we can measure the probabilities upto an additive error of $\frac{\beta_j-\varepsilon}{4}$ with confidence at least $1/2$ then we can reject if x_i appears less than $\frac{1}{2}(1-\frac{\beta_j+\eta}{2})$ fraction of times or more than $\frac{1}{2}(1+\frac{\beta_j+\eta}{2})$ fraction of times. And the algorithm is REJECTED (during the run of (x_i, y_i)) with probability at least $\frac{1}{2}(1-\frac{1}{e})(\frac{\eta-\varepsilon}{2})$. The number t_j is set such that the success probability is at least $(1-\delta)$.

□

Query Complexity

Theorem 3. *Given ε, η and δ,* Barbarik *need at most $\widetilde{O}\left(\frac{1}{(\eta-\varepsilon)^4}\right)$ samples for any input formula φ, where the tilde hides a poly logarithmic factor of $1/\delta$ and $1/(\eta-\varepsilon)$.*

Proof. The proof is deferred to extended version. □

5 Evaluation

To evaluate the runtime performance of Barbarik and test the uniformity of the state of the art samplers, we implemented a prototype of Barbarik and employed SPUR (Achlioptas, Hammoudeh, and Theodoropoulos 2018) as the ideal uniform sampler, i.e., $\mathcal{U}$ in the Algorithm 1. Note that Barbarik allows choice of any other available samplers such as KUS (Sharma et al. 2018). The experiments were conducted on a high-performance computer cluster, where each node consists of E5-2690 v3 CPU with 24 cores and 96GB of RAM.

We set tolerance parameter ε, intolerance parameter η, and confidence δ for Barbarik to be 0.6, 0.9, and 0.1 respectively. The chosen setting of parameters implies that for a given Boolean formula φ, if the sampler under test $\mathcal{G}(\varphi, \varepsilon)$ is an additive almost-uniform generator (AAU), then Barbarik returns ACCEPT with probability at least 0.9, otherwise Barbarik returns REJECT with probability at least 0.9. Note that, the number of samples required for ACCEPT depends only on ε and η, and for our chosen values of ε and η, the number of samples is 1.72975×10^6.

We choose the three state of the art samplers, Quicksampler (Dutra et al. 2018), UniGen2 (Chakraborty et al. 2015a), and SearchTreeSampler (Ermon, Gomes, and Selman 2012). We use the default parameters for Quicksampler, UniGen2, and SearchTreeSampler, which were also employed in previous case studies for uniformity. Quicksampler and SearchTreeSampler provides theoretical guarantees that are weaker than that of near-uniform generator. In contrast, UniGen2 is multiplicative almost-uniform generator (MAU), which is still weaker than additive almost-uniform generator (AAU) (Chakraborty et al. 2015a; Meel et al. 2016). Therefore, apriori we expect Barbarik to return REJECT for Quicksampler and SearchTreeSampler for most of the benchmarks within reasonable samples while return ACCEPT for UniGen2 for some of the benchmarks.

We employed publicly available benchmark suite used in the evaluation of Quicksampler (Dutra et al. 2018) and UniGen2, which included bit-blasted versions of constraints arising in bounded model checking of circuits, bit-blasted

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

Table 1: The output and analysis of the number of samples consumed by Barbarik for different samplers. "A" and "R" in the Output columns indicate ACCEPT and REJECT respectively.

versions of SMTLib benchmarks, constraints arising from automated program synthesis, and constraints arising from ISCAS89 circuits with parity conditions on randomly chosen subsets of outputs and next-state variables. In total, our benchmark suite consisted of 101 instances with the count of the solutions for instances ranging to 2^{90}. It is worth noting that the benchmark suite was also used in an empirical evaluation of the uniformity of SearchTreeSampler, Quicksampler, and UniGen2 in (Dutra et al. 2018). The primary objective of our experimental evaluation was to seek an answer to the following questions:

1. Does Barbarik return ACCEPT for the distributions generated by the various state of the art samplers?

2. How does the required number of samples vary with different benchmarks?

In summary, we observe that Barbarik returns REJECT for SearchTreeSampler and Quicksampler for all the 101 instances requiring only less than 250 samples for all the instances. In contrast, Barbarik returns ACCEPT for UniGen2 on all the 101 instances. These results offer strong support for our *non-adversarial sampler* assumption over the formulas generated by Kernel and offer support for the claim that the theoretical analysis of UniGen2 may indeed be conservative. Furthermore, these results demonstrate that Barbarik requires only a few samples to catch a non-uniform sampler in practice.The low requirement of the number of samples should be viewed in the context of observations of Duatra et al. (2018), who employed Pearson's chi-squared test with more than tens of thousands of samples generated by each sampler. Despite using more samples for SearchTreeSampler and Quicksampler than Barbarik, Duatra et al. concluded the confidence in their conclusions with ". However, this result should be taken with care, since the uniformity test is not very reliable on benchmarks....when the number of produced samples is too low".

5.1 Analysis of Benchmarks

To analyze whether it is possible to have standard sampling techniques to test the uniformity of samplers reliably, we computed the distribution of the number of solutions for our benchmark suite. Figure 1 presents the scatter plot of the number of solutions vis-a-vis different benchmarks. The y-axis represents the count of the number of solutions on log scale while the x-axis represents the ID of the benchmark. For ease of presentation, assign a unique ID to our benchmarks from 1 to 101. Since our counts vary to 2^{90}, the requirement of statistical techniques in terms of the number of samples is infeasible.

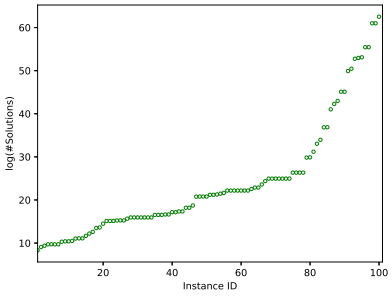

Figure 1: Scatter plot of the count of benchmarks

5.2 Sample requirement of Barbarik

Table 1 presents the number of samples consumed by Barbarik to return ACCEPT or REJECT for a subset of benchmarks. The column 1 and 2 represents the id and the number of solutions of the benchmark. The columns 2 and 3 present the output of Barbarik and the number of samples generated by UniGen2 for the corresponding benchmark while column 4 and 5 represent the output and the

number of samples generated by SearchTreeSampler. Finally, columns 6 and 7 present the output and the number of samples generated by Quicksampler. First, Barbarik returns the same answers, i.e. ACCEPT for UniGen2 and REJECT for SearchTreeSampler and Quicksampler for all the 101 benchmarks. Furthermore, the number of samples generated by SearchTreeSampler and Quicksampler is 250 for all the benchmarks. It is worth noting that that guarantees provided by UniGen2 are weaker than that of an almost-uniform sampler with $\varepsilon = 0.6$. These results may be viewed as another support for the claim of authors of UniGen2 that the theoretical analysis of UniGen2 is conservative and could perhaps be improved. On the other hand, while SearchTreeSampler and Quicksampler do not claim to have theoretical guarantees of almost-uniformity, the low number of samples required by Barbarik to return REJECT may indicate SearchTreeSampler and Quicksampler have significant weaknesses in their algorithmic approach. These results also demonstrate strong support for *non-adversarial sampler* assumption of underlying samplers for the formulas generated by Kernel.

6 Conclusion

Sampling techniques form the core of the state of the art inference engines. The computational intractability of sampling forces the usage of heuristics which may nullify the theoretical gurantees of the underlying algorithms. In this context, there is strong need for techniques that can test whether a sampler under test generates samples close to the desired distribution for a given formula. In this work, we design, to the best of our knoweldge, the first algorithmic framework, Barbarik, to test wehther the distribution generated is $\varepsilon-$ close or $\eta-$ far from the uniform distribution. We demonstrated that Barbarik is able to REJECT samplers, which do not have theoretical guarantees while accepts the sampler UniGen2, which have theoretical guarantees of uniformity.

Acknowledgments We are grateful to Mate Soos for his insights on workings of modern SAT solver and Alexis de Colnet for several insightful suggestions on the earlier draft. This work was supported in part by NUS ODPRT Grant R-252-000-685-133 and AI Singapore Grant R-252-000-A16-490. The computational work for this article was performed on resources of the National Supercomputing Centre, Singapore `https://www.nscc.sg`.

References

Achlioptas, D.; Hammoudeh, Z.; and Theodoropoulos, P. 2018. Fast sampling of perfectly uniform satisfying assignments. In *Proc. of SAT*.

Batu, T.; Dasgupta, S.; Kumar, R.; and Rubinfeld, R. 2005. The complexity of approximating the entropy. *SIAM J. Comput.* 35(1):132–150.

Batu, T.; Fortnow, L.; Rubinfeld, R.; Smith, W. D.; and White, P. 2013. Testing closeness of discrete distributions. *J. ACM* 60(1):4:1–4:25.

Batu, T.; Kumar, R.; and Rubinfeld, R. 2004. Sublinear algorithms for testing monotone and unimodal distributions. In *Proc. of STOC*, 381–390.

Canonne, C. L.; Ron, D.; and Servedio, R. A. 2015. Testing probability distributions using conditional samples. *SIAM J. Comput.* 44(3):540–616.

Chakraborty, S.; Fischer, E.; Matsliah, A.; and de Wolf, R. 2010. New results on quantum property testing. In *Proc. of FSTTCS*, 145–156.

Chakraborty, S.; Fremont, D. J.; Meel, K. S.; Seshia, S. A.; and Vardi, M. Y. 2015a. On parallel scalable uniform sat witness generation. In *Proc. of TACAS*, 304–319.

Chakraborty, S.; Fried, D.; Meel, K. S.; and Vardi, M. Y. 2015b. From weighted to unweighted model counting. In *Proc. of IJCAI*, 689–695.

Chakraborty, S.; Fischer, E.; Goldhirsh, Y.; and Matsliah, A. 2016. On the power of conditional samples in distribution testing. *SIAM J. Comput.* 45(4):1261–1296.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2013. A Scalable and Nearly Uniform Generator of SAT Witnesses. In *Proc. of CAV*, 608–623.

Dutra, R.; Laeufer, K.; Bachrach, J.; and Sen, K. 2018. Efficient sampling of sat solutions for testing. In *Proc. of ICSE*.

Ermon, S.; Gomes, C. P.; Sabharwal, A.; and Selman, B. 2013a. Embed and project: Discrete sampling with universal hashing. In *Proc. of NIPS*, 2085–2093.

Ermon, S.; Gomes, C. P.; Sabharwal, A.; and Selman, B. 2013b. Taming the curse of dimensionality: Discrete integration by hashing and optimization. In *Proc. of ICML*, 334–342.

Ermon, S.; Gomes, C. P.; and Selman, B. 2012. Uniform solution sampling using a constraint solver as an oracle. In *Proc. of UAI*.

Gilks, W. R., and Wild, P. 1992. Adaptive rejection sampling for gibbs sampling. *Applied Statistics* 337–348.

Jerrum, M. R., and Sinclair, A. 1996. The Markov Chain Monte Carlo method: an approach to approximate counting and integration. *Approximation algorithms for NP-hard problems* 482–520.

Jordan, M. I.; Ghahramani, Z.; Jaakkola, T. S.; and Saul, L. K. 1999. An introduction to variational methods for graphical models. *Machine learning* 37(2):183–233.

Kitchen, N., and Kuehlmann, A. 2007. Stimulus generation for constrained random simulation. In *Proc. of ICCAD*, 258–265.

Koller, D., and Friedman, N. 2009. *Probabilistic graphical models: principles and techniques*. MIT press.

Meel, K. S.; Vardi, M. Y.; Chakraborty, S.; Fremont, D. J.; Seshia, S. A.; Fried, D.; Ivrii, A.; and Malik, S. 2016. Constrained sampling and counting: Universal hashing meets sat solving. In *Proc. of Beyond NP Workshop*.

Meel, K. S. 2014. *Sampling Techniques for Boolean Satisfiability*. Rice University. M.S. Thesis.

Meel, K. S. 2017. *Constrained Counting and Sampling: Bridging the Gap between Theory and Practice*. Ph.D. Dissertation, Rice University.

Neal, R. M. 2001. Annealed importance sampling. *Statistics and computing* 11(2):125–139.

Seshia, S. A.; Sadigh, D.; and Sastry, S. S. 2016. Towards Verified Artificial Intelligence. *ArXiv e-prints*.

Sharma, S.; Gupta, R.; Roy, S.; and Meel, K. S. 2018. Knowledge compilation meets uniform sampling. In *Proc. of LPAR-22*, 620–636.