On Testing of Samplers *

Kuldeep S. Meel ⓡ
School of Computing
National University of Singapore

Yash Pote ⓡ
School of Computing
National University of Singapore

Sourav Chakraborty
Indian Statistical Institute
Kolkata

Abstract

Given a set of items $\mathcal{F}$ and a weight function $\mathtt{wt} : \mathcal{F} \mapsto (0, 1)$, the problem of sampling seeks to sample an item proportional to its weight. Sampling is a fundamental problem in machine learning. The daunting computational complexity of sampling with formal guarantees leads designers to propose heuristics-based techniques for which no rigorous theoretical analysis exists to quantify the quality of generated distributions. This poses a challenge in designing a testing methodology to test whether a sampler under test generates samples according to a given distribution. Only recently, Chakraborty and Meel (2019) designed the first scalable verifier, called Barbarik, for samplers in the special case when the weight function $\mathtt{wt}$ is constant, that is, when the sampler is supposed to sample uniformly from $\mathcal{F}$. The techniques in Barbarik, however, fail to handle general weight functions.

The primary contribution of this paper is an affirmative answer to the above challenge: motivated by Barbarik, but using different techniques and analysis, we design Barbarik2, an algorithm to test whether the distribution generated by a sampler is ε-close or η-far from any target distribution. In contrast to black-box sampling techniques that require a number of samples proportional to $|\mathcal{F}|$, Barbarik2 requires only $\tilde{O}(tilt(\mathtt{wt}, \varphi)^2 / \eta(\eta - 6\varepsilon)^3)$ samples, where the $tilt$ is the maximum ratio of weights of two satisfying assignments. Barbarik2 can handle any arbitrary weight function. We present a prototype implementation of Barbarik2 and use it to test three state-of-the-art samplers.

1   Introduction

Motivated by the success of statistical techniques, automated decision-making systems are increasingly employed in critical domains such as medical [19], aeronautics [33], criminal sentencing [20], and military [2]. The potential long-term impact of the ensuing decisions has led to research in the correct-by-construction design of AI-based decision systems. There has been a call for the design of randomized and quantitative formal methods [35] to verify the basic building blocks of the modern AI systems. In this work, we focus on one such core building block: *constrained sampling*.

Given a set of constraints φ over a set of variables X and a weight function $\mathtt{wt}$ over assignments to X, the problem of constrained sampling is to sample a satisfying assignment σ of φ with probability proportional to $\mathtt{wt}(\sigma)$. Constrained sampling is a fundamental problem that encapsulates a wide range of sampling formulations [24, 23, 12, 30, 14]. For example, $\mathtt{wt}$ can be used to capture a given prior distribution often represented implicitly through probabilistic models, and φ can be used to

---

capture the evidence arising from the observed data, then the problem of constrained sampling models the problem of sampling from the resulting posterior distribution.

The problem of constrained sampling is computationally hard and has witnessed a sustained interest from theoreticians and practitioners, resulting in the proposal of several approximation techniques. Of these, Monte Carlo Markov Chain(MCMC)-based methods form the backbone of modern sampling techniques [3, 7]. The runtime of these techniques depends on the length of the random walk, and the Markov chains that require polynomial walks are called rapidly mixing Markov chains. Unfortunately, for most distributions of practical interest, it is infeasible to design rapidly mixing Markov chains [26], and the practical implementations of such techniques have to resort to the usage of heuristics that violate theoretical guarantees. The developers of such techniques, often and rightly so, strive to demonstrate their effectiveness via empirical behavior in practice [6].

The need for the usage of heuristics to achieve scalability is not restricted to just MCMC methods but is widely observed for other methods such as simulated annealing [29], variational methods [18], and hashing-based techniques [12, 23, 13, 32]. Consequently, a fundamental problem for the designers of sampling techniques is: *how can one efficiently test whether a given technique samples from the desired distribution?* Most of the existing approaches rely on the computations of statistical metrics such as variation distance and KL-divergence by drawing samples and perform hypothesis testing with a preset p-value. Sound computations of statistical metrics require a large number of samples that is proportional to the support of the posterior distribution [4, 36], which is prohibitively large; it is not uncommon for the distribution support to be significantly larger than 2^{70}. Consequently, the existing approaches tend to estimate the desired quantities using a fraction of the required samples, and such estimates are often without the required confidence. The usage of unsound metrics may lead to unsound conclusions, as demonstrated by a recent study where the usage of unsound metric would lead one to conclude that two samplers were indistinguishable (it is worth mentioning that the authors of the study clearly warn the reader about the unsoundness of the underlying metrics) [21].

The researchers in the sub-field of property testing within theoretical computer science have analyzed the sample complexity of testing under different models of samplers and computation. The resulting frameworks have not witnessed widespread adoption to practice due to a lack of samplers that can precisely fit the models under which results are obtained. In recent work, Chakraborty and Meel [10], building on the concepts developed in the condition sampling model (rf. [1]), designed the first practical algorithmic procedure, called Barbarik, that can rigorously test whether a given sampler samples from the uniform distribution using a constant number of samples, assuming that the given sampler is *subquery-consistent* (see Definition 9). Empirically, Barbarik was shown to be able to distinguish samplers that were indistinguishable in prior studies based on unsound metrics. While Barbarik made significant progress, it is marred by its ability to handle only the uniform distribution. Therefore, one wonders: *Can we design an algorithmic framework to test whether the distribution generated by a given sampler is close to a desired (but arbitrary) posterior distribution of interest?*

This paper's primary contribution is the first efficient algorithmic framework, Barbarik2, to test whether the distribution generated by a sampler is ε-close or η-far from the desired distribution specified by the set of constraints φ and a weight function wt. In contrast to the statistical techniques that require an exponential or sub-exponential number of samples for samplers whose support can be represented by n bits, the number of samples required by Barbarik2 depends on the *tilt* of the distribution, where *tilt* is defined as the maximum ratio of non-zero weights of two solutions of φ. Like Barbarik, the key technical idea of Barbarik2 sits at the intersection of *property testing* and *formal methods* and uses ideas from conditional sampling and employs chain formulas. However, the key algorithmic framework of Barbarik2 differs significantly from Barbarik, and, as demonstrated, the proof of its correctness and sample complexity requires an entirely new set of technical arguments.

Given access to an ideal sampler $\mathcal{A}$, Barbarik2 accepts every sampler that is ε-close to $\mathcal{A}$ while its ability to reject a sampler that is η-far from $\mathcal{A}$ assumes that the sampler under test is *subquery consistent*. Since Barbarik2 assumes access to an ideal sampler, one might wonder if a tester such as Barbarik2 is needed when we already have access to an ideal sampler. Since sampling is computationally intractable, it is almost always the case that an ideal sampler $\mathcal{A}$ is quite slow and one would prefer to use some other efficient sampler $\mathcal{G}$ instead of $\mathcal{A}$, if $\mathcal{G}$ can be certified to be close to $\mathcal{A}$.

To demonstrate the practical efficiency of Barbarik2, we developed a prototype implementation in Python and performed an experimental evaluation with several samplers. While our framework does not put a restriction on the representation of wt, we perform empirical validation with weight

distributions corresponding to log-linear models, a widely used class of distributions. Our empirical evaluation shows that Barbarik2 returns ACCEPT for the samplers with formal guarantees but returns REJECT for other samplers that are without formal guarantees. Our ability to reject samplers provides evidence in support of our assumption of subquery consistency of samplers. We believe our formalization of testing of samplers and the design of the algorithmic procedure, Barbarik2, contributes to the design of *randomized formal methods* for verified AI, a principle argued by Seshia et al [35].

2 Notations and Preliminaries

A Boolean variable is denoted by a lowercase letter. For a Boolean formula φ, the set of variables appearing in φ, called the *support* of φ, is denoted by $Supp(\varphi)$. An assignment $\sigma \in \{0,1\}^{|Supp(\varphi)|}$ to the variables of φ is a *satisfying assignment* or *witness* if it makes φ evaluate to 1. We denote the set of all satisfying assignments of φ as R_φ. For $S \subseteq Supp(\varphi)$, we use $\sigma_{\downarrow S}$ to indicate the projection of σ over the set of variables in S. And we denote by $R_{\varphi \downarrow S}$ the set $\{\sigma_{\downarrow S} \mid \sigma \in R_\varphi\}$.

Definition 1 (Weight Function). *For a set S of Boolean variables, a weight function $\mathtt{wt} : \{0,1\}^{|S|} \to (0,1)$ maps each assignment to some weight.*

Definition 2 (Sampler). *A sampler $\mathcal{G}(\varphi, S, \mathtt{wt}, \tau)$ is a randomized algorithm that takes in a Boolean formula φ, a weight function $\mathtt{wt}$, a set $S \subseteq Supp(\varphi)$ and a positive integer τ and outputs τ independent samples from $R_{\varphi \downarrow S}$. For brevity of notation we may sometimes refer to a sampler as $\mathcal{G}(\varphi)$ or simply, $\mathcal{G}$.*

For any $\sigma \in \{0,1\}^{|S|}$ the probability of the sampler outputting σ is denoted by $p_{\mathcal{G}}(\varphi, S, \sigma)$ (or $p_{\mathcal{G}}(\varphi, \sigma)$ when the set S in question is clear from the context).

We use $D_{\mathcal{G}(\varphi,S)}$ to represent the distribution induced by $\mathcal{G}(\varphi, S)$ on $R_{\varphi \downarrow S}$. When the set S is understood from the context we will denote $D_{\mathcal{G}(\varphi,S)}$ by $D_{\mathcal{G}(\varphi)}$.

Definition 3 (Ideal Sampler). *For a weight function $\mathtt{wt}$, a sampler $\mathcal{A}(\varphi, S, \tau)$ is called an ideal sampler w.r.t. weight function $\mathtt{wt}$ if for all $\sigma \in R_{\varphi \downarrow S}$: $p_{\mathcal{A}}(\varphi, S, \mathtt{wt}, \sigma) = \frac{\mathtt{wt}(\sigma)}{\sum_{\sigma' \in R_{\varphi \downarrow S}} \mathtt{wt}(\sigma')}$. In the rest of the paper, $\mathcal{A}(\cdot, \cdot, \cdot, \cdot)$ denotes the ideal sampler. When $\mathtt{wt}(\sigma) = \frac{1}{|R_\varphi|}$ then the ideal sampler is called a uniform sampler.*

Definition 4 (Tilt). *For a Boolean formula φ and weight function $\mathtt{wt}$, we define $tilt(\mathtt{wt}, \varphi) = \max\limits_{\sigma_1, \sigma_2 \in R_\varphi} \frac{\mathtt{wt}(\sigma_1)}{\mathtt{wt}(\sigma_2)}$.*

Our goal is to design a program that can test the quality of a sampler with respect to an ideal sampler. We use two different notions of distance of the sampler from the ideal sampler.

Definition 5 (ε-closeness and η-farness). *A sampler $\mathcal{G}$ is ε-multiplicative-close (or simply ε-close) to an ideal sampler $\mathcal{A}$, if for all φ and all $\sigma \in R_\varphi$, we have $(1-\varepsilon)p_{\mathcal{A}}(\varphi, \sigma) \leq p_{\mathcal{G}}(\varphi, \sigma) \leq (1+\varepsilon)p_{\mathcal{A}}(\varphi, \sigma)$. For a formula φ, a sampler $\mathcal{G}(\varphi)$ is η-ℓ_1-far (or simply η-far) from the ideal sampler $\mathcal{A}(\varphi)$, if $\sum_{\sigma \in R_\varphi} |p_{\mathcal{A}}(\varphi, \sigma) - p_{\mathcal{G}}(\varphi, \sigma)| \geq \eta$*

It is worth emphasising that the asymmetry in the notions of ε-close and η-far stems from the availability of practical samplers. Since the available off-the-shelf solvers with theoretical guarantees provide the guarantee of ε-closeness, we are interested in accepting a sampler that is ε-close [24, 23, 12, 14]. On the other hand, we would like to be more forgiving to the samplers without guarantees and would like to reject only if they are η-far in ℓ_1 distance, a notion more relaxed than multiplicative closeness.

Definition 6 (($\varepsilon, \eta, \delta$)-tester for samplers). *A ($\varepsilon, \eta, \delta$)-tester for samplers is a randomized algorithm that takes a sampler $\mathcal{G}$, an ideal sampler $\mathcal{A}$, a tolerance parameter ε, an intolerance parameter η, a guarantee parameter δ and a CNF formula φ such that (1) If $\mathcal{G}(\varphi)$ is ε-close to $\mathcal{A}(\varphi)$, then the tester returns ACCEPT with probability at least $(1-\delta)$, and (2) If $\mathcal{G}(\varphi)$ is η-far from $\mathcal{A}(\varphi)$ then the tester returns REJECT with probability at least $(1-\delta)$.*

2.1 Chain Formula

A crucial component in our algorithm is the chain formula. Chain formulas, introduced in [15], are a special class of Boolean formulas. Given a positive integer k and m, chain formulas provide

an efficient construction of a Boolean formula $\psi_{k,m}$ with exactly k satisfying assignments with $\lceil log(k) \rceil \leq m$ variables. We employ chain formulas for inverse transform sampling and in the subroutine Barbarik2Kernel

Definition 7. *[15] Let $c_1 c_2 \cdots c_m$ 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*

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

For example, consider $k = 11$ and $m = 4$. The binary representation of 11 using 4 bits is 1011. Therefore, $\varphi_{5,4}(a_1, a_2, a_3, a_4) = a_1 \vee (a_2 \wedge (a_3 \vee a_4))$.

Lemma 1. *[15] Let $m > 0$ be a natural number, $k < 2^m$, and $\varphi_{k,m}$ as defined above. Then $|\varphi_{k,m}|$ is linear in m and $\varphi_{k,m}$ has exactly k satisfying assignments. Every chain formula ψ on n variables is equivalent to a CNF formula ψ^{CNF} having at most n clauses. In addition, $|\psi^{CNF}|$ is in $O(n^2)$.*

2.2 Barbarik2Kernel and the Subquery Consistency Assumption

Barbarik2Kernel is a crucial subroutine that we use in our algorithm to help us draw *conditional samples* from $R_{\varphi \downarrow S}$. This is similar to the subroutine Kernel used by the Barbarik in [10]. We will now define a collection of functions KernelFamily.

Definition 8. KernelFamily *is family of functions that take a Boolean formula φ, a set of variables $S \subseteq Supp(\varphi)$, and two assignments $\sigma_1, \sigma_2 \in R_{\varphi \downarrow S}$, and return $\hat{\varphi}$ such that $R_{\hat{\varphi} \downarrow S} = \{\sigma_1, \sigma_2\}$.*

[10] introduced a concept called the "non-adversarial assumption" and it was crucial to their proof. In this paper, we will refer to it as "subquery consistency". A more formal description of subquery consistency is given using the following definition:

Definition 9. *Let Barbarik2Kernel $\in$ KernelFamily. A sampler $\mathcal{G}$ is subquery consistent w.r.t. a particular Barbarik2Kernel for φ if for all $S \subseteq Supp(\varphi)$, $\sigma_1, \sigma_2 \in R_{\varphi \downarrow S}$, let $\hat{\varphi} \leftarrow$ Barbarik2Kernel$(\varphi, S, \sigma_1, \sigma_2)$ then the output of $\mathcal{G}(\hat{\varphi}, \mathtt{wt}, S, \tau)$ is τ independent samples from the conditional distribution $\mathcal{D}_{\mathcal{G}(\varphi)|T}$, where $T = \{\sigma_1, \sigma_2\}$.*

For proving the correctness of our tester we will be using the above definition, similar to its usage in [10]. While it is not known how to test whether a sampler is subquery consistent w.r.t a particular Barbarik2Kernel, it is reasonable to believe that Barbarik2Kernel with enough randomization should allow a reasonably designed sampler to behave consistently on subqueries. We present the design of one such Barbarik2Kernel in Section 4. Our empirical results provide evidence to our belief of most samplers behaving consistently on subqueries.

3 Related Work

Distribution testing involves testing whether an unknown probability distribution is identical or close to a given distribution. This problem has been studied extensively in the property testing literature [11, 8, 36, 37] . The sample space is exponential, and for many fundamental distributions, including uniform, it is prohibitively expensive in terms of samples to verify closeness. This led to the development of the conditional sampling model [11, 8], which can provide sub-linear or even *constant* sample complexities for the testing of the above-given properties[1, 28, 5, 9, 17]. A detailed discussion on prior work in property testing and their relationship to Barbarik2 is given in Appendix A.

The first practically efficient algorithm for verification of samplers with a formal proof of correctness was presented by Chakraborty and Meel in form of Barbarik [10]. The central idea of Barbarik, building on the work of Chakraborty et al. [11] and Canonne et al. [8], was that if one can have conditional samples from the distribution, then one can test properties of the distribution using only a constant number of conditional samples.

Barbarik constructs a two-element set $T \subset R_\varphi$, with one element drawn according to the distribution $D_{\mathcal{G}(\varphi)}$ and one element drawn uniformly at random from the set R_φ. Using a subroutine Kernel Chakraborty et al. argued that one can draw samples from the conditional distribution $D_{\mathcal{G}(\varphi)|T}$. Their

sample complexity was $\tilde{O}(1/(\eta - 2\varepsilon)^4)$. They proved that if a sampler $\mathcal{G}$ is ε-close to a uniform sampler then Barbarik will accept with probability at least $(1 - \delta)$, while if $\mathcal{G}(\varphi)$ is η-far from the uniform sampler and if $\mathcal{G}$ is subquery consistent w.r.t Kernel for φ, then Barbarik rejects with probability at least $(1-\delta)$. Their underlying assumption was that many samplers that are in use would in fact be *subquery consistent* and the success of Barbarik in rejecting several samplers provides evidence in support of the aforementioned assumption. They used Barbarik to test the correctness of samplers like STS, Quicksampler, and UniGen.

Note that Barbarik can only distinguish a uniform sampler from a far-from uniform sampler, and the techniques used cannot be generalized easily to the case where the ideal sampler is not necessarily uniform. While Barbarik2, that we present in this paper, does borrow several techniques from Barbarik, including drawing inspiration from the concept of conditional sampling for their design; Barbarik2 is very different from Barbarik both in terms of the algorithmic design and its implementation.

4   An overview of the Barbarik2 Algorithm

In this section, we present the algorithmic framework of Barbarik2, the pseudocode, presented as Algorithm 1, and then the theoretical justification for the algorithm. Barbarik2 takes as input a blackbox sampler $\mathcal{G}$, a Boolean formula φ with the associated weight function wt and three parameters $(\varepsilon, \eta, \delta)$. It also has access to an ideal sampler $\mathcal{A}$. Barbarik2 is an $(\varepsilon, \eta, \delta)$-tester for samplers. Also if Barbarik2 returns REJECT (that is, when $\mathcal{G}$ is η-far from $\mathcal{A}$), it provides as witness a new formula $\hat{\varphi}$ which is similar to φ, except that $\hat{\varphi}$ has only two assignments to the variables in S (namely σ_1 and σ_2) that can be extended to satisfying assignments of $\hat{\varphi}$ and the relative probability masses of σ_1 and σ_2 in $\mathcal{D}_{\mathcal{G}}$ are significantly different from that in $\mathcal{D}_{\mathcal{A}}$.

The core idea of Barbarik2 is that for verifying the quality of the sampler $\mathcal{G}(\varphi)$, we can proceed in two stages. In the first stage, if the sampler is far from the ideal sampler $\mathcal{A}$, we hope to find a witness (in the form of two satisfying assignments) for farness with good probability. This can be guaranteed by drawing one sample each from $\mathcal{D}_{\mathcal{G}(\varphi)}$ and $\mathcal{D}_{\mathcal{A}(\varphi)}$. In the second stage, we confirm whether the witness is indeed far. That is, if the witness is the (σ_1, σ_2) pair, we check that the probability of σ_1 and σ_2 in $\mathcal{D}_{\mathcal{G}(\varphi)}$ and $\mathcal{D}_{\mathcal{A}(\varphi)}$ are similar or not.

Here Barbarik2 differs from Barbarik in a significant way. Barbarik employs a bucketing strategy. But, Barbarik2 chooses a simpler yet equally effective method to check the similarity between σ_1 and σ_2. This is also the most difficult stage of the tester as one may have to draw a exponential number of samples to confirm the similarity. We manage this by drawing samples from the conditional distribution $\mathcal{D}_{\mathcal{G}(\varphi)|\{\sigma_1,\sigma_2\}}$ instead of $\mathcal{D}_{\mathcal{G}(\varphi)}$. Since $\mathcal{D}_{\mathcal{G}(\varphi)|\{\sigma_1,\sigma_2\}}$ is supported on a set of size only two estimating the distance of $\mathcal{D}_{\mathcal{G}(\varphi)|\{\sigma_1,\sigma_2\}}$ from $\mathcal{D}_{\mathcal{A}(\varphi)|\{\sigma_1,\sigma_2\}}$ can be done with constant number of samples.

Now since we do not have direct access to the distribution $\mathcal{D}_{\mathcal{G}(\varphi)|\{\sigma_1,\sigma_2\}}$ we circumvent the problem by drawing samples from a new distribution $\mathcal{D}_{\mathcal{G}(\hat{\varphi})}$ where $\hat{\varphi}$ is obtained from φ and has similar structure as φ (with $Supp(\varphi) \subseteq Supp(\hat{\varphi})$) and there are only two assignments (namely σ_1 and σ_2) to the variables in $Supp(\varphi)$ that can be extended to the satisfying assignments of $\hat{\varphi}$. The subroutine Barbarik2Kernel helps us simulate the drawing of samples from $\mathcal{D}_{\mathcal{G}(\varphi)|\{\sigma_1,\sigma_2\}}$ by drawing samples from $\mathcal{D}_{\mathcal{G}(\hat{\varphi})}$. The subroutine Bias helps to estimate the distance of $\mathcal{D}_{\mathcal{G}(\hat{\varphi})}$ from $\mathcal{D}_{\mathcal{A}(\hat{\varphi})}$.

Finally, we repeat the whole process for a certain number of rounds, and we argue that if the sampler is indeed far then, with high probability, in at least one round, we will find a witness of farness and confirm that the witness is indeed far. On the other hand, if the sampler is close to ideal, then there does not exist any such witness of farness.

Barbarik2 accesses two subroutines, Bias and Barbarik2Kernel: Bias$(\hat{\sigma}, \Gamma, S)$ takes as input an assignment $\hat{\sigma}$, a list Γ of assignments and a sampling set S. It returns the fraction of assignments of Γ whose projections on S is equal to $\hat{\sigma}$. Barbarik2Kernel$(\varphi, \sigma_1, \sigma_2)$ is a Barbarik2Kernel subroutine (Definition 8). Its aim is to create a $\hat{\varphi}$ such the behaviour of the sampler on $\hat{\varphi}$ is similar to it's behaviour on φ, i.e. $\mathcal{D}_{\mathcal{G}(\varphi)|\{\sigma_1,\sigma_2\}} \approx \mathcal{D}_{\mathcal{G}(\hat{\varphi})}$.

In Barbarik2, in the for loop (in lines $7-20$), in each round, the algorithm draws one sample σ_1 according to the distribution $\mathcal{D}_{\mathcal{G}(\varphi)}$ and one sample σ_2 according to the ideal distribution on R_{φ} (line 8). In the case that $\sigma_1 = \sigma_2$ it moves the to next iteration (in line 9-10). In line 16, the subroutine Barbarik2Kernel uses φ, the two samples σ_1 and σ_2, to output a new formula $\hat{\varphi}$ such

Algorithm 1 Barbarik2($\mathcal{G}, \mathcal{A}, \varepsilon, \eta, \delta, \varphi, S, \mathtt{wt}$)

1: $t \leftarrow ln(1/\delta)ln\left(\frac{10}{10-\eta(\eta-6\varepsilon)}\right)^{-1}$
2: $n \leftarrow 8ln\left(t/\delta\right)$
3: $lo = (1+\varepsilon)/(1-\varepsilon)$
4: $hi = 1 + (\eta + 6\varepsilon)/4$
5: $\Gamma_1 \leftarrow \mathcal{G}(\varphi, S, t);$
6: $\Gamma_2 \leftarrow \mathcal{A}(\varphi, S, t);$
7: **for** $i = 1$ to t **do**
8:     $\sigma_1 \leftarrow \Gamma_1[i]; \sigma_2 \leftarrow \Gamma_2[i];$
9:     **if** $\sigma_1 = \sigma_2$ **then**
10:         **continue**
11:     $\alpha \leftarrow \mathtt{wt}(\sigma_1)/\mathtt{wt}(\sigma_2)$
12:     $L \leftarrow (\alpha \cdot lo)/(1 + \alpha \cdot lo)$
13:     $H \leftarrow (\alpha \cdot hi)/(1 + \alpha \cdot hi)$
14:     $T = (H+L)/2$
15:     $N \leftarrow n \cdot H/(H-L)^2$
16:     $\hat{\varphi} \leftarrow$ Barbarik2Kernel($\varphi, \sigma_1, \sigma_2$)
17:     $\Gamma_3 \leftarrow \mathcal{G}(\hat{\varphi}, S, N)$
18:     $Bias \leftarrow$ Bias(σ_1, Γ_3, S)
19:     **if** $Bias > T$ **then**
20:         **return** REJECT
21: **return** ACCEPT

Algorithm 2 Barbarik2Kernel($\varphi, \sigma_1, \sigma_2$)

1: $m \leftarrow 12, k \leftarrow 2^m - 1$
2: $\mathbf{Lits_1} \leftarrow (\sigma_1 \setminus \sigma_2)$
3: $\mathbf{Lits_2} \leftarrow (\sigma_2 \setminus \sigma_1)$
4: $\mathbf{V} \leftarrow NewVars(\varphi, m);$
5: $\hat{\varphi} \leftarrow \varphi \wedge (\sigma_1 \vee \sigma_2)$
6: $l \sim \mathbf{Lits_1} \cup \mathbf{Lits_2}$
7: $\hat{\varphi} \leftarrow \hat{\varphi} \wedge (\neg l \rightarrow \psi_{k,m}(\mathbf{V}))$
8: $\hat{\varphi} \leftarrow \hat{\varphi} \wedge (l \rightarrow \psi_{k,m}(\mathbf{V}))$
9: **return** $\hat{\varphi}$

Algorithm 3 Bias($\hat{\sigma}, \Gamma, S$)

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

that $Supp(\varphi) \subseteq Supp(\hat{\varphi})$. On line 17, Barbarik2 draws a list, Γ_3, of N samples according to the distribution $\mathcal{D}_{\mathcal{G}(\hat{\varphi})}$. Barbarik2Kernel ensures that for all $\sigma \in \Gamma_3$, $\sigma_{\downarrow S}$ is either σ_1 or σ_2. In line 18 Barbarik2 uses Bias to compute the fraction of samples that are equal to σ_1 (on the variable set S), and if the fraction is greater than the threshold then Barbarik2 returns REJECT (in line 20).

Algorithm 2 presents the pseudocode of subroutine Barbarik2Kernel. As stated above, Barbarik2Kernel takes in a Boolean formula φ, a set $S \subseteq Supp(\varphi)$ and two partial assignments $\sigma_1, \sigma_2 \in R_{\varphi \downarrow S}$. Since the set S is implicit from σ_1 and σ_2 it may not be explicitly given as an input. Barbarik2Kernel assumes access to subroutine $NewVars$ which takes in two parameters, a formula φ and a number m, and returns a set of m fresh variables that do not appear in φ. Barbarik2Kernel first constructs two sets of literals, denoted by $\mathbf{Lits_1}$ (resp. $\mathbf{Lits_2}$), which appear in σ_1 (resp. σ_2) but not σ_2 (resp. σ_1). The algorithm then constructs the formula $\hat{\varphi}$. First it generates $\varphi \wedge (\sigma_1 \vee \sigma_2)$ on Line 5, a formula with exactly two solutions. Next, it randomly chooses a literal l from $\mathbf{Lits_1} \cup \mathbf{Lits_2}$ and constructs a chain formula $(l \rightarrow \psi_{k,m})$ over the fresh Boolean variables $\mathbf{V}[1], \mathbf{V}[2] \cdots, \mathbf{V}[m]$ where k is the number of satisfying assignments the formula has. Conjuncting the two generated formulas, we get $\hat{\varphi} \equiv \varphi \wedge (\sigma_1 \vee \sigma_2)$. Therefore, at the end of Barbarik2Kernel, i.e. line 8, $\hat{\varphi}$ has $2k$ solutions. We choose the value of k such that it is odd (see [15]). The chain formula is linked to a random Boolean literal from the given set of literals for two reasons,

1. An ideal or ε-close to ideal sampler would not be affected by the randomization and would generate the same distribution over $\hat{\varphi}$ as it does over $\varphi \wedge (\sigma_1 \vee \sigma_2)$.

2. If the sampler under test $\mathcal{G}$ is η-far from ideal, then we want to construct a formula which *cannot* be easily guessed by $\mathcal{A}$. We wish to avoid the scenario where $\mathcal{A}$, an η-far sampler on φ, somehow behaves as an almost-ideal sampler over $\hat{\varphi}$ and hence manages to fool Barbarik2.

4.1 Theoretical Analysis

The following theorem gives the mathematical guarantee about the correctness of Barbarik2.

Theorem 1. *Given sampler $\mathcal{G}$, ideal sampler $\mathcal{A}$, $\varepsilon < \frac{1}{3}$, $\eta > 6\varepsilon$, δ, φ and weight function $\mathtt{wt}$, Barbarik2 needs at most $\widetilde{O}\left(\frac{tilt(\mathtt{wt}, \varphi)^2}{\eta(\eta - 6\varepsilon)^3}\right)$ samples, where $\widetilde{O}$ hides a poly logarithmic factor of $1/\delta$.*

- *If $\mathcal{G}$ is an ε-close to $\mathcal{A}$ then* Barbarik2 *returns* ACCEPT *with probability at least* $(1-\delta)$.

6

- *If $\mathcal{G}$ is subquery consistent w.r.t Barbarik2Kernel and if the distribution $\mathcal{D}_{\mathcal{G}(\varphi)}$ is η-far from the ideal sampler then Barbarik2 returns* REJECT *with probability at least* $(1 - \delta)$.

Note that if $\mathcal{G}$ is ε-close to $\mathcal{A}$ then Barbarik2 accepts (with high probability) even if the sampler $\mathcal{G}$ isn't subquery consistent w.r.t Barbarik2Kernel. It is also worth noting that Barbarik2 terminates with REJECT as soon as the check in line 19 succeeds. Therefore, we expect Barbarik2 to require significantly less number of samples when it returns REJECT. Furthermore, in the case of ACCEPT, the bound on N, as calculated on line 15 in terms of $tilt$, is pessimistic as the probability of observing σ_1 and σ_2 such that $\alpha \approx tilt$ for a sampler close to ideal is very small when the tilt is large. The proof of Theorem 1 is presented in Appendix B.

5 Evaluation

The objective of our evaluation was to answer the following questions:

RQ1. Is Barbarik2 able to distinguish between off-the-shelf samplers by returning ACCEPT for samplers ε-close to the ideal distribution and REJECT for the η-far samplers?

RQ2. What improvements do we observe over the baseline?

RQ3. How does the required number of samples scale with the $tilt(\mathtt{wt}, \varphi)$ of the distribution?

To evaluate the runtime performance of Barbarik2 and test the quality of some state of the art samplers, we implemented a prototype of Barbarik2 in Python. Our algorithm utilizes an ideal sampler, for which we use the state of the art sampler WAPS [25]. All experiments were conducted on a high performance computing cluster with 600 E5-2690 v3 @2.60GHz CPU cores. For each benchmark, we use a single core with a timeout of 24 hours.

We focus on the log-linear distributions given their ubiquity of usage in machine learning; a formal description is provided in Appendix C for completeness. Observe that Barbarik2 does not put any restrictions on the representation of the weight distribution. We conducted our experiments on 72 publicly available benchmarks, which have been employed in the evaluation of samplers proposed in the past [13, 21]. The $tilt$ of the benchmarks spans many orders of magnitude, between 1 and 10^{11}.

Samplers Tested The past few years have witnessed a multitude of sampling techniques ranging from variational methods [38], MCMC-based techniques [27, 31], mutation-based sampling [21], importance sampling-based methods [22], knowledge-compilation techniques [25] and the like. The conceptual simplicity of uniform samplers encourages designers to tune their algorithms for uniform sampling, and the standard technique for weighted sampling employs the well-known method of the inverse transform. For the sake of completeness, we provide a detailed discussion of the transformation technique in Appendix C. We perform empirical evaluation with the three state of the art samplers wUniGen, wQuicksampler, and wSTS constructed by augmenting inverse sampling with underlying samplers UniGen [13], Quicksampler [21] and STS(SearchTreeSampler) [22] respectively.

While wUniGen is known to have theoretical guarantees of $\varepsilon-$closeness, there is no theoretical analysis of the distributions generated by wQuicksampler and wSTS. Of the 72 instances, wUniGen can handle only 35 instances while wQuicksampler and wSTS can handle all the 72 instances. The variation in the number of instances that are amenable to sampling for a particular sampler highlights the trade-off between the runtime performance and theoretical guarantees. It is perhaps worth emphasizing that wQuicksampler and wSTS are significantly more efficient in runtime performance than the ideal sampler WAPS.

Test Parameters We set tolerance parameter ε, intolerance parameter η, and confidence δ for Barbarik2 to be and 0.1, 1.6 and 0.2 respectively. The chosen setting of parameters implies that for a given Boolean formula φ, if the sampler under test $\mathcal{G}(\varphi)$ is ε-close to the ideal sampler, then Barbarik2 returns ACCEPT with probability at least 0.8, otherwise if the sampler is η-far from ideal sampler then Barbarik2 returns REJECT with probability at least 0.8. Note that, the number of samples required for ACCEPT depends only on the parameters $(\varepsilon, \eta, \delta)$ and $tilt(\mathtt{wt}, \varphi)$. We instantiate Barbarik2Kernel with the values $m = 12$ and $k = 2^m - 1$. Observe that Theorem 1 does not put restrictions on k and m.

| | | Barbarik2 | | |
Benchmark	$tilt$ (maxSamp)	wUniGen (samples)	wSTS (samples)	wQuicksampler (samples)
s349_3_2	28 (3e+07)	A (1e+05)	A (1e+05)	R (22854)
s820a_3_2	37 (5e+07)	A (96212)	R (87997)	A (2e+05)
UserServiceImpl.sk	140 (6e+08)	A (1e+05)	R (1e+05)	R (4393)
LoginService2.sk	232 (2e+09)	A (1e+05)	R (38044)	R (13350)
s349_7_4	603 (1e+10)	A (75555)	R (4284)	R (5150)
s344_3_2	3300 (3e+11)	A (1e+05)	R (59952)	R (5150)
s420_new_7_4	3549 (4e+11)	A (82312)	A (96659)	R (49955)
54.sk_12_97	4e+11 (6e+27)	DNS	R (14012)	R (4627)
s641_7_4	9e+07 (3e+20)	DNS	R (8747)	A (1e+06)
s838_3_2	2e+08 (1e+21)	DNS	R (9504)	R (4627)

Table 1: "A"(resp. "R") represents Barbarik2 returning `ACCEPT`(resp. `REJECT`). maxSamp represents the upper bound on the number of samples required by Barbarik2 to return `ACCEPT/REJECT`.

Description of the table We present the experimental results in Table 1. Due to lack of space, we present results for a subset of benchmarks while the extended table is presented in the full version of this paper []. The first column indicates the name of the benchmark, the second the $tilt$, and the following columns indicate the outcome of the experiments with wUniGen, wSTS and wQuicksampler in that order. Every cell in the table has two entries. In the second column, the first entry shows the value of $tilt$ for the corresponding benchmark, while in the other columns, it contains "A" and "R" to indicate the output of Barbarik2 for the corresponding sampler. The second entry for the cells in the column corresponding to $tilt$ indicates the theoretical upper bound on the samples required for Barbarik2 to terminate, while for rest of the columns, the second entry indicates the number of samples consumed by Barbarik2 for the corresponding instance and the sampler.

RQ1 Our experiments demonstrate that Barbarik2 returns `REJECT` for wQuicksampler on 68 benchmarks and `ACCEPT` on the remaining four benchmarks. For wSTS we found Barbarik2 returned `REJECT` on 62 of the benchmarks and `ACCEPT` on 7 while it times out on the remaining 3. Since wSTS and wQuicksampler are samplers with no formal guarantees and therefore one may expect them to generation distributions away from the ideal distributions. In this context, the results in Table 1 provide strong evidence for the reasonableness of the *subquery consistency* assumption in practice.

In contrast, Barbarik2 returned `ACCEPT` for wUniGen on all the 35 benchmarks for which wUniGen could sample. Recall, wUniGen formally guarantees ε-closeness of the samples to the required distribution, hence Barbarik2 returning `ACCEPT` on all the benchmarks provides evidence in support of soundness of Barbarik2.

RQ2 We also computed the number of samples required by the baseline approach owing to [4]. Since the number of samples is so large that exhaustive experimentation is infeasible, we had to resort to estimating the average time taken by a sampler for a given instance. Based on the estimated time, we can estimate the time taken by the baseline for our benchmark set. We observe that the time taken by the baseline would be over 10^6 seconds for 43, 42 and 16 benchmarks for wQuicksampler,

wSTS and wUniGen respectively. In this context, it is worth highlighting that Barbarik2 terminates within 24 hours for all the instances for all the samplers. We observe that the geometric means of the speedups over the baseline approach are $10^{5.0}, 10^{20.2}$ and 58 for wSTS, wQuicksampler and, wUniGen respectively. The lower speedup in the case of wUniGen owes to its ability to handle only small benchmarks, for which the number of models was not very large. The extended results are available in Appendix D.

RQ3   The number of trials required (indicated by the the variable t as on Line 7 of Algorithm 1) depends only on $(\varepsilon, \eta, \delta)$, so for the values we use, $(0.1, 1.6, 0.2)$, we find that we require $t = 14$ trials. The analysis of the algorithm reveals an upper bound on the sample complexity of the tester (See Section 4, Theorem 1) which is quadratic in terms of the $tilt(\mathtt{wt}, \varphi)$. We now return to Table 1 and observe that the number of samples required by Barbarik2 before returning ACCEPT were significantly lower than the theoretical bound provided in the second column. Furthermore, as noted earlier, the number of samples required before Barbarik2 returns REJECT is typically significantly less than the worst case – a trend demonstrated in Table 1.

6   Conclusion

In this paper, we study the problem of verifying whether a probabilistic sampler samples from a given discrete distribution. Existing approaches require samples linear in the size of the sampling set, which is commonly exponentially large. We present a conditional sampling technique that can verify the sampler in sample complexity constant in terms of the sampling set. We also test a prototype implementation of our algorithm against three state-of-the-art samplers.

We noticed that the analytical upper bound on the sample complexity is extraordinarily weak compared to our observed values; this suggests that the bounds could be further tightened. Our algorithm does not deal with all possible discrete distributions, thus extending the approach to other distributions would enable the testing of a broader set of samplers.

Broader Impact

The recent advances in machine learning techniques have led to increased adoption of the said techniques in safety-critical domains. The usage of a technique in a safety-critical domain necessitates appropriate verification methodology. This paper seeks to take a step in this direction and focused on one core component. Our analysis is probabilistic, and therefore, practical adoption of such techniques requires careful design of frameworks to handle failures.

Acknowledgements

References

[1] Jayadev Acharya, Clément L. Canonne, and Gautam Kamath. A chasm between identity and equivalence testing with conditional queries. *Theory of Computing*, 2018.

[2] Jürgen Altmann and Frank Sauer. Autonomous weapon systems and strategic stability. *Survival*, 2017.

[3] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to MCMC for machine learning. *Machine learning*, 2003.

[4] Tugkan Batu, Lance Fortnow, Ronitt Rubinfeld, Warren D. Smith, and Patrick White. Testing closeness of discrete distributions. *J. ACM*, 2013.

[5] Rishiraj Bhattacharyya and Sourav Chakraborty. Property testing of joint distributions using conditional samples. *TOCT*, 2018.

[6] Stephen P Brooks and Andrew Gelman. General methods for monitoring convergence of iterative simulations. *Journal of computational and graphical statistics*, 1998.

[7] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of Markov Chain Monte Carlo*. CRC press, 2011.

[8] Clément L. Canonne, Dana Ron, and Rocco A. Servedio. Testing probability distributions using conditional samples. *SIAM J. Comput.*, 2015.

[9] Clément L. Canonne, Ilias Diakonikolas, Daniel M. Kane, and Alistair Stewart. Testing conditional independence of discrete distributions. *CoRR*, 2017.

[10] Sourav Chakraborty and Kuldeep S. Meel. On testing of uniform samplers. In *Proc. of AAAI*, 2019.

[11] Sourav Chakraborty, Eldar Fischer, Yonatan Goldhirsh, and Arie Matsliah. On the power of conditional samples in distribution testing. *SIAM J. Comput.*, 2016.

[12] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. A Scalable and Nearly Uniform Generator of SAT Witnesses. In *Proc. of CAV*, 2013.

[13] Supratik Chakraborty, Daniel J. Fremont, Kuldeep S. Meel, Sanjit A. Seshia, and Moshe Y. Vardi. Distribution-aware sampling and weighted model counting for SAT. In *Proc. of AAAI*, 2014.

[14] Supratik Chakraborty, Daniel J. Fremont, Kuldeep S. Meel, Sanjit A. Seshia, and Moshe Y. Vardi. On parallel scalable uniform SAT witness generation. In *Proc. of TACAS*, 2015.

[15] Supratik Chakraborty, Dror Fried, Kuldeep S Meel, and Moshe Y Vardi. From weighted to unweighted model counting. In *Proc. of IJCAI*, 2015.

[16] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 2008.

[17] Xi Chen, Rajesh Jayaram, Amit Levi, and Erik Waingarten. Learning and testing junta distributions with subcube conditioning. *CoRR*, 2020.

[18] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. In *ACM SIGGRAPH Papers*. 2004.

[19] Elliott Crigger and Christopher Khoury. Making policy on augmented intelligence in health care. *AMA Journal of Ethics*, 2019.

[20] Michael E Donohue. A replacement for Justitia's scales?: Machine learning's role in sentencing. *Harvard Journal of Law & Technology*, 2019.

[21] Rafael Dutra, Kevin Laeufer, Jonathan Bachrach, and Koushik Sen. Efficient sampling of SAT solutions for testing. In *Proc. of ICSE*, 2018.

[22] Stefano Ermon, Carla P. Gomes, and Bart Selman. Uniform solution sampling using a constraint solver as an oracle. In *Proc. of UAI*, 2012.

[23] Stefano Ermon, Carla P Gomes, Ashish Sabharwal, and Bart Selman. Embed and project: Discrete sampling with universal hashing. In *Proc. of NIPS*, 2013.

[24] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Near-uniform sampling of combinatorial spaces using XOR constraints. In *Proc. of NIPS*, 2007.

[25] Rahul Gupta, Shubham Sharma, Subhajit Roy, and Kuldeep S. Meel. WAPS: Weighted and Projected Sampling. In *Proc. of TACAS*, 2019.

[26] Mark Jerrum. Mathematical foundations of the markov chain monte carlo method. In *Probabilistic methods for algorithmic discrete mathematics*. 1998.

[27] Mark R. Jerrum and Alistair Sinclair. The Markov Chain Monte Carlo method: an approach to approximate counting and integration. *Approximation algorithms for NP-hard problems*, 1996.

[28] Gautam Kamath and Christos Tzamos. Anaconda: A non-adaptive conditional sampling algorithm for distribution testing. SIAM, 2019.

[29] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 1983.

[30] Chris J Maddison, Daniel Tarlow, and Tom Minka. A* sampling. In *Proc. of NIPS*, 2014.

[31] Neal Madras. Lectures on Monte Carlo Methods, Fields Institute Monographs 16. *American Mathematical Society*, 2002.

[32] Kuldeep S. Meel, Moshe Y. Vardi, Supratik Chakraborty, Daniel J Fremont, Sanjit A Seshia, Dror Fried, Alexander Ivrii, and Sharad Malik. Constrained sampling and counting: Universal hashing meets sat solving. In *AAAI Workshop: Beyond NP*, 2016.

[33] Kathleen L Mosier and Linda J Skitka. Human decision makers and automated decision aids: Made for each other? In *Automation and human performance*. 2018.

[34] K.P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[35] Sanjit A Seshia, Dorsa Sadigh, and S Shankar Sastry. Towards verified artificial intelligence. *arXiv preprint arXiv:1606.08514*, 2016.

[36] Gregory Valiant and Paul Valiant. The Power of Linear Estimators. In *Proc of FOCS*, 2011.

[37] Paul Valiant. Testing symmetric properties of distributions. *SIAM J. Comput.*, 2011.

[38] Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Found. Trends Machine Learning*, 2008.

Appendix

A   Relationship of Barbarik2 with Property Testing

Testing of samplers is basically testing if two distributions $\mathcal{D}_{\mathcal{G}(\varphi)}$ and $\mathcal{A}_{\mathcal{G}(\varphi)}$ are similar, where $\mathcal{G}$ is the sampler under test and $\mathcal{A}$ is the ideal sampler. As stated in the Introduction and the Related Work section, the sub-field of property testing in theoretical computer science has been studying this problem for over two decades and our tester Barbarik2 draws ideas from some of the latest research in this area.

In understanding the closeness between two distributions one may consider a variety of different distance measures. The variation distance (also called the ℓ_1 distance) is possibly most commonly used. In property testing the problem is to distinguish between the case where the two distributions are ε-close in ℓ_1 distance from the case where the distributions are η-far from each other in ℓ_1 distance. An easier question, called the "equivalence testing of distributions" considers the problem of distinguish identical distributions from distributions that are η-far from each other in ℓ_1 distance. The former question, often referred to as the tolerant version of equivalence testing of distributions or estimation of variation distance, is more suitable for various applications. The goal in all the settings is to minimize the sample complexity. The time complexity or other complexity measures are usually not considered in property testing literature.

The problem of equivalence testing of distributions was first considered by [4] and they (along with [37]) showed that the sample complexity was $\Theta(N^{2/3})$, where N is the size of the support of the distributions. Note that, in the setting of samplers, N is exponential in the input size and hence the number is prohibitively large. The tolerant version of the problem was proved to have even higher sample complexity of $\Theta(N)$ ([37, 36]). This was a significant bottleneck is practicality of these property testing algorithms and the tight lower bounds implied that no improvement was possible for algorithms that has only blackbox access to the distributions. Even the much simpler problem of testing if a distribution is uniform requires $\Omega(\sqrt{N})$ samples.

In [11, 8] a new model for sampling was introduced called the conditional sampling. This model allowed access to the distributions that the standard sampling method (or the blackbox access to the distributions) could not give. It allowed a kind of grey-box access to the distributions. It was shown that in this model only $O(1/\varepsilon^2)$ conditional samples were needed to test if a distribution is uniform or ε-far from uniform. In fact similar conditional sample complexity is sufficient for the non-tolerant version of the equivalence testing of distributions. For the tolerant version of equivalence testing of distributions it was shown that polynomial in $\log(N)$ number of conditional samples suffice. Although this brings down the sample complexity drastically but still it was quite high for practical implementations. On top of that a major obstacle was whether the conditional samples were at all practical and were they implementable.

In [10] they successfully used the idea from the conditional sampling testing to test if samplers are uniform. They crucially used a special kind of conditional sampling. In [8] a concept of pair-conditioning (they called PCOND) was introduced to define a restricted version of the conditional sampling model. A normal conditional sample is obtained by specifying a subset S of the domain of the distribution $\mathcal{D}$ and then drawing a random sample from the conditional distribution $\mathcal{D}|_S$. A PCOND-sample is a normal conditional sample where the subset S is of size 2. In [10] it was shown how this kind of restricted samples can be successfully implemented using a clever use of chain-formulas.

When it come to the more general problems of non-tolerant version of equivalence testing of distributions it can be shown that the sample complexity in the PCOND-model is at least polynomial in $\log N$. The tolerant version has even higher PCOND-sample complexity. Since our primary objective was to have a tester that can be practical and implementable we had to circumvent the problem of high sample complexity and also of implementational issues of conditional sampling. In our tester Barbarik2 we addressed these problems by using another trick from [10], that of, using two different notions of distance - ℓ_∞ for closeness and ℓ_1 for farness. In Barbarik2 we re-designed the sampler and give a proof of correctness in this paradigm using very different techniques as compare to that used in [10].

It is worth noting here that recently conditional sampling and its various variants has been used to design efficient testing and learning algorithms for various other properties of distributions ([1, 28, 5, 9, 17]). Many of these have the potential to be used more efficient and sophisticated testing of samplers and related questions. But the major question is the practicality of the models and the implementability of the algorithms.

B Proof of Correctness of Barbarik2

In this section, we present the theoretical analysis of Barbarik2, and the proof of Theorem 1. The proof clearly follows from the the following three lemmas.

Lemma 2. *If a sampler $\mathcal{G}$ is ε-close [2] to the ideal sampler $\mathcal{A}$ then* Barbarik2 *returns* ACCEPT *with probability at least $(1 - \delta)$.*

Lemma 3. *If $\mathcal{G}$ is subquery consistent w.r.t* Barbarik2Kernel *and if the distribution $\mathcal{D}_{\mathcal{G}(\varphi)}$ is η-far from the ideal sampler then* Barbarik2 *returns* REJECT *with probability at least $(1 - \delta)$.*

Lemma 4. *Given ε, η and δ,* Barbarik2 *needs at most $\widetilde{O}\left(\frac{tilt(\mathtt{wt}, \varphi)^2}{\eta(\eta - 6\varepsilon)^3}\right)$ samples for any input formula φ and weight function $\mathtt{wt}$, where the tilde hides a poly logarithmic factor of $1/\delta, 1/\eta$ and $1/(\eta - 6\varepsilon)$.*

We will present the proofs of Lemma 2, Lemma 3 and Lemma 4 in Section B.1, Section B.2 and Section B.3 respectively.

In the rest of this section we will use the following notations:

- We use $\mathbb{1}(E)$ to represent the indicator variable for the event E.
- We use R_i to denote the event that Barbarik2 returns REJECT in iteration i.

For the proof of correctness of our algorithm, we need some standard concentration inequalities. The following versions of the Chernoff Bound will be used.

Lemma 5. *Let $Y_1, Y_2, \ldots, Y_n$ be i.i.d 0-1 random variables.*

1. *If $\mathrm{E}[Y_i] \geq \theta \geq 0$, then for any $t \leq \theta$,*

$$\Pr\left[\sum_{j \in [n]} \frac{Y_j}{n} \leq t\right] < exp\left(-\frac{(\theta - t)^2 n}{2\theta}\right)$$

2. *If $\mathrm{E}[Y_i] \leq \theta$, then for any $t \geq \theta$,*

$$\Pr\left[\sum_{j \in [n]} \frac{Y_j}{n} \geq t\right] < exp\left(-\frac{(t - \theta)^2 n}{2t}\right)$$

We are now ready to present the proofs of Lemma 2, Lemma 3 and Lemma 4.

B.1 Proof of Lemma 2

Lemma 2. *If a sampler $\mathcal{G}$ is ε-close [3] to the ideal sampler $\mathcal{A}$ then* Barbarik2 *returns* ACCEPT *with probability at least $(1 - \delta)$.*

For the proof of Lemma 2 we will firstly show (in Lemma 6) that in each iteration of the loop, the probability that Barbarik2 returns REJECT is less than δ/t and then the proof of Lemma 2 follows by the application of the Chernoff Bound. Recall that R_i denotes the event that Barbarik2 returns REJECT in iteration i.

---

[2] for any $\varepsilon < \frac{1}{3}$ and $\eta > 6\varepsilon$
[3] for any $\varepsilon < \frac{1}{3}$ and $\eta > 6\varepsilon$

Lemma 6. *If sampler $\mathcal{G}$ is ε-close to an ideal sampler $\mathcal{A}$ then the probability that* Barbarik2 *returns* REJECT *in any particular iteration of the loop, is atmost δ/t. Then*

$$\Pr\left[\overline{R_i} \mid \bigwedge_{j\in[i-1]} \overline{R_j}\right] \geq \left(1 - \frac{\delta}{t}\right)$$

Proof. (of Lemma 6) Barbarik2 returns REJECT in the ith iteration if the $Bias$ (in the ith iteration) is more than T, where $T = \frac{L+H}{2}$ with

$$L = \frac{(1+\varepsilon)p_{\mathcal{A}}(\varphi, S, \sigma_1)}{(1+\varepsilon)p_{\mathcal{A}}(\varphi, S, \sigma_1) + (1-\varepsilon)p_{\mathcal{A}}(\varphi, S, \sigma_2)}$$

And since, by definition, all the elements in Γ_1, Γ_2 and Γ_3 are obtained by drawing independent samples from $\mathcal{D}_{\mathcal{G}(\varphi)}$, $\mathcal{D}_{\mathcal{A}(\varphi)}$ and $\mathcal{D}_{\mathcal{G}(\hat{\varphi})}$ respectively so

$$\Pr\left[\overline{R_i} \mid \bigwedge_{j\in[i-1]} \overline{R_j}\right] = \Pr\left[\,Bias \leq T \text{ in the } i\text{th iteration}\right]$$

$$= 1 - \Pr\left[\,Bias > T \text{ in the } i\text{th iteration}\right]$$

$$= 1 - \Pr\left[\sum_{j\in[N]} \frac{\mathbb{1}(\Gamma_3[j]_{\downarrow S} = \sigma_1)}{N} > T\right]$$

Note that the random variables $\mathbb{1}(\Gamma_3[j]_{\downarrow S} = \sigma_1)$ are an i.i.d 0-1 random variable. And since the sampler $\mathcal{G}$ is assumed to be ε-close to the ideal sampler so we have

$$(1-\varepsilon)p_{\mathcal{A}}(\hat{\varphi}, \Gamma_3[j]) \leq p_{\mathcal{G}}(\hat{\varphi}, \Gamma_3[j]) \leq (1+\varepsilon)p_{\mathcal{A}}(\hat{\varphi}, \Gamma_3[j]).$$

Thus we have,

$$\mathrm{E}[\mathbb{1}(\Gamma_3[j]_{\downarrow S} = \sigma_1)] = p_{\mathcal{G}}(\hat{\varphi}, S, \sigma_1) \leq (1+\varepsilon)p_{\mathcal{A}}(\hat{\varphi}, S, \sigma_1)$$

Now, since $p_{\mathcal{A}}(\hat{\varphi}, S, \sigma_1) = \frac{p_{\mathcal{A}}(\varphi, S, \sigma_1)}{p_{\mathcal{A}}(\varphi, S, \sigma_1) + p_{\mathcal{A}}(\varphi, S, \sigma_2)}$ we have

$$\mathrm{E}[\mathbb{1}(\Gamma_3[j]_{\downarrow S} = \sigma_1)] = p_{\mathcal{G}}(\hat{\varphi}, S, \sigma_1) \leq \frac{(1+\varepsilon)p_{\mathcal{A}}(\varphi, S, \sigma_1)}{p_{\mathcal{A}}(\varphi, S, \sigma_1) + p_{\mathcal{A}}(\varphi, S, \sigma_2)} \tag{1}$$

Similarly, we have that

$$\mathrm{E}[\mathbb{1}(\Gamma_3[j]_{\downarrow S} = \sigma_2)] = p_{\mathcal{G}}(\hat{\varphi}, S, \sigma_2) \geq \frac{(1-\varepsilon)p_{\mathcal{A}}(\varphi, S, \sigma_2)}{p_{\mathcal{A}}(\varphi, S, \sigma_1) + p_{\mathcal{A}}(\varphi, S, \sigma_2)} \tag{2}$$

Now we consider two cases depending on whether $p_{\mathcal{A}}(\varphi, S, \sigma_1)$ is greater or lesser than $p_{\mathcal{A}}(\varphi, S, \sigma_2)$. If $p_{\mathcal{A}}(\varphi, S, \sigma_1) \leq p_{\mathcal{A}}(\varphi, S, \sigma_2)$ then from Equation 1 we have

$$\mathrm{E}[\mathbb{1}(\Gamma_3[j]_{\downarrow S} = \sigma_1)] = p_{\mathcal{A}}(\hat{\varphi}, S, \sigma_1)$$

$$\leq \frac{(1+\varepsilon)p_{\mathcal{A}}(\varphi, S, \sigma_1)}{p_{\mathcal{A}}(\varphi, S, \sigma_1) + p_{\mathcal{A}}(\varphi, S, \sigma_2)}$$

$$\leq \frac{(1+\varepsilon)p_{\mathcal{A}}(\varphi, S, \sigma_1)}{(1+\varepsilon)p_{\mathcal{A}}(\varphi, S, \sigma_1) + (1-\varepsilon)p_{\mathcal{A}}(\varphi, S, \sigma_2)} = L \tag{3}$$

But if $p_{\mathcal{A}}(\varphi, S, \sigma_1) \geq p_{\mathcal{A}}(\varphi, S, \sigma_2)$ then from Equation 1 we have

$$\mathrm{E}[\mathbb{1}(\Gamma_3[j]_{\downarrow S} = \sigma_2)] = p_{\mathcal{A}}(\hat{\varphi}, S, \sigma_2)$$

$$\geq \frac{(1-\varepsilon)p_{\mathcal{A}}(\varphi, S, \sigma_2)}{p_{\mathcal{A}}(\varphi, S, \sigma_1) + p_{\mathcal{A}}(\varphi, S, \sigma_2)}$$

$$\geq \frac{(1-\varepsilon)p_{\mathcal{A}}(\varphi, S, \sigma_2)}{(1+\varepsilon)p_{\mathcal{A}}(\varphi, S, \sigma_1) + (1-\varepsilon)p_{\mathcal{A}}(\varphi, S, \sigma_2)}$$

And in that case since $p_{\mathcal{A}}(\hat{\varphi}, S, \sigma_1) + p_{\mathcal{A}}(\hat{\varphi}, S, \sigma_2) = 1$ we have

$$
\begin{aligned}
\mathrm{E}[\mathbb{1}(\Gamma_3[j]_{\downarrow S} = \sigma_1)] &= p_{\mathcal{A}}(\hat{\varphi}, S, \sigma_1) \\
&= 1 - p_{\mathcal{A}}(\hat{\varphi}, S, \sigma_2) \\
&\le 1 - \left(\frac{(1-\varepsilon)p_{\mathcal{A}}(\varphi, S, \sigma_2)}{(1+\varepsilon)p_{\mathcal{A}}(\varphi, S, \sigma_1) + (1-\varepsilon)p_{\mathcal{A}}(\varphi, S, \sigma_2)} \right) \\
&\le \frac{(1+\varepsilon)p_{\mathcal{A}}(\varphi, S, \sigma_1)}{(1+\varepsilon)p_{\mathcal{A}}(\varphi, S, \sigma_1) + (1-\varepsilon)p_{\mathcal{A}}(\varphi, S, \sigma_2)} = L
\end{aligned}
\tag{4}
$$

Thus in either case, from Equation (3) and Equation (4) we have $\mathrm{E}[\mathbb{1}(\Gamma_3[j]_{\downarrow S} = \sigma_1)] \le L$. Now applying the Chernoff bound from Lemma 5 we have

$$
\begin{aligned}
\Pr[Bias \ge T] = \Pr\left[\sum_{j \in [N]} \frac{\mathbb{1}(\Gamma_3[j]_{\downarrow S} = \sigma_1)}{N} > T \right] \\
= exp\left(-\frac{(T-L)^2 N}{2L} \right) = exp\left(-\frac{(H-L)^2 N}{8L} \right) \\
\le exp\left(-\frac{(H-L)^2 N}{8H} \right) \quad \text{Because } [H \ge L] \\
\le \frac{\delta}{t},
\end{aligned}
\tag{5}
$$
$$
\tag{6}
$$

where the inequality in line (5) follows because $H \ge L$ when $\varepsilon \le 1/3$ and $\eta \ge 6\varepsilon$[4] and last inequality follows because $N = n.H/(H-L)^2$ where $n = 8\log(t/\delta)$. □

Proof. (of Lemma 2) Let R_i denote the event that Barbarik2 returns REJECT in iteration i and $\overline{R}$ denote the event that Barbarik2 returns ACCEPT. Thus $\overline{R} = \cap_i \overline{R_i}$.

In the ith iteration if the bias is less than the threshold, Barbarik2 fails to REJECT. Thus from Lemma 6 if the sampler $\mathcal{G}$ is ε-close to the ideal sampler $\mathcal{A}$ then

$$
\Pr\left[\overline{R_i} \mid \bigwedge_{j \in [i-1]} \overline{R_j} \right] \ge 1 - \frac{\delta}{t}
$$

If Barbarik2 has not returned REJECT in any of the iteration then after the last iteration Barbarik2 returns ACCEPT. The probability of Barbarik2 returning ACCEPT (event $\overline{R}$) is

$$
\Pr\left[\overline{R} \right] \ge \prod_{i \in [t]} \Pr\left[\overline{R_i} \mid \bigwedge_{j \in [i-1]} \overline{R_j} \right] \ge \left(1 - \frac{\delta}{t} \right)^t \ge 1 - \delta
$$

□

B.2 Proof of Lemma 3

Lemma 3. *If $\mathcal{G}$ is subquery consistent w.r.t Barbarik2Kernel and if the distribution $\mathcal{D}_{\mathcal{G}(\varphi)}$ is η-far from the ideal sampler then* Barbarik2 *returns* REJECT *with probability at least $(1 - \delta)$.*

Proof. To prove the Lemma, we will start by splitting the set R_φ into disjoint subsets depending on the distribution $D_{\mathcal{G}(\varphi)}$.

Definition 10. *We define the following sets for use in the soundness proof:*

- $D = \{x \in R_\varphi : p_{\mathcal{G}}(\varphi, x) \le p_{\mathcal{A}}(\varphi, x)\}$

---
[4]$H \ge L$ if $hi \ge lo$ that is $(6\varepsilon + \eta)/4 \ge (2\varepsilon)/(1-\varepsilon)$

- $U = R_\varphi \setminus D$

- $U_0 = \{x \in R_\varphi :\ p_\mathcal{A}(\varphi, x) < p_\mathcal{G}(\varphi, x) \le \left(1 + \frac{\eta + 6\varepsilon}{4}\right) p_\mathcal{A}(\varphi, x)\}$.

- $U_1 = \{x \in R_\varphi :\ \left(1 + \frac{\eta + 6\varepsilon}{4}\right) p_\mathcal{A}(\varphi, x) < p_\mathcal{G}(\varphi, x)\}$

Recall, R_i is the event that Barbarik2 returns REJECT in the ith iteration of the for loop. Then the following lemmas helps us to lower bound the probability of $\Gamma_1[i] \in U_1 \wedge \Gamma_2[i] \in D$ and the probability of R_i under the condition that $\Gamma_1[i] \in U_1 \wedge \Gamma_2[i] \in D$.

Lemma 7. *If the sampler $\mathcal{G}$ is η-far from the ideal sampler then*

$$\Pr\left[R_i | (\bigwedge_{j \in [i-1]} \overline{R_j}) \wedge (\Gamma_1[i] \in U_1 \wedge \Gamma_2[i] \in D)\right] \ge \frac{4}{5}.$$

Lemma 8. *If the sampler $\mathcal{G}$ is η-far from the ideal sampler on input φ then*

$$\Pr\left[\Gamma_1[i] \in U_1 \wedge \Gamma_2[i] \in D\right] \ge \frac{\eta(\eta - 6\varepsilon)}{8}.$$

And now using Lemmas 8 and 7 we can complete the proof of soundness. The probability that Barbarik2 returns REJECT in the ith iteration of the for loop is

$$
\begin{aligned}
&\Pr\left[R_i \mid \bigwedge_{j \in [i-1]} \overline{R_j}\right] \\
&= \Pr\left[R_i \mid (\bigwedge_{j \in [i-1]} \overline{R_j}) \wedge (\Gamma_1[i] \in U_1 \wedge \Gamma_2[i] \in D)\right] \cdot \Pr[\Gamma_1[i] \in U_1 \wedge \Gamma_2[i] \in D] \\
&\ge \left(\frac{4}{5}\right) \frac{\eta(\eta - 6\varepsilon)}{8} \quad \text{[From Lemma 8 and Lemma 7]}
\end{aligned}
\tag{7}
$$

The probability of Barbarik2 returning REJECT in any iteration (event R) is given by

$$
\begin{aligned}
\Pr\left[\cup_i R_i\right] &= 1 - \prod_{i \in [t]} \Pr\left[\overline{R_i} \mid \bigwedge_{j \in [i-1]} \overline{R_j}\right] \\
&\ge 1 - \prod_{i \in [t]} \left(1 - \frac{\eta(\eta - 6\varepsilon)}{10}\right) \quad \text{[Using Equation (7)]} \\
&\ge 1 - \left(1 - \frac{\eta(\eta - 6\varepsilon)}{10}\right)^t \\
\text{Substituting } t, \quad &\ge 1 - \delta
\end{aligned}
$$

$\square$

Now to complete the proof of Lemma 3 we have to prove the Lemma 7 and Lemma 8. They are presented next.

Lemma 7. *If the sampler $\mathcal{G}$ is η-far from the ideal sampler then*

$$\Pr\left[R_i | (\bigwedge_{j \in [i-1]} \overline{R_j}) \wedge (\Gamma_1[i] \in U_1 \wedge \Gamma_2[i] \in D)\right] \ge \frac{4}{5}.$$

Proof. (of Lemma 7) Let us assume $\Gamma_1[i] \in U_1$ and $\Gamma_2[i] \in D$. That is, we have $p_{\mathcal{G}}(\varphi, S, \Gamma_2[i]) \leq p_{\mathcal{A}}(\varphi, S, \Gamma_2[i])$ and $p_{\mathcal{G}}(\varphi, S, \Gamma_1[i]) > \left(1 + \frac{\eta + 6\varepsilon}{4}\right) p_{\mathcal{A}}(\varphi, S, \Gamma_1[i])$. It follows that

$$\frac{p_{\mathcal{G}}(\varphi, S, \Gamma_1[i])}{p_{\mathcal{G}}(\varphi, S, \Gamma_2[i])} \geq \left(1 + \frac{6\varepsilon + \eta}{4}\right) \cdot \frac{p_{\mathcal{A}}(\varphi, S, \Gamma_1[i])}{p_{\mathcal{A}}(\varphi, S, \Gamma_2[i])} \tag{8}$$

Since $\forall x > 0, a/b > x \implies a/(a+b) > x/(x+1)$, we have from Equation 8

$$\frac{p_{\mathcal{G}}(\varphi, S, \Gamma_1[i])}{p_{\mathcal{G}}(\varphi, S, \Gamma_2[i]) + p_{\mathcal{G}}(\varphi, S, \Gamma_1[i])}$$

$$\geq \left(1 + \frac{6\varepsilon + \eta}{4}\right) \cdot \frac{p_{\mathcal{A}}(\varphi, S, \Gamma_1[i])}{p_{\mathcal{A}}(\varphi, S, \Gamma_2[i])} \cdot \left(1 + \left(1 + \frac{6\varepsilon + \eta}{4}\right) \cdot \frac{p_{\mathcal{A}}(\varphi, S, \Gamma_1[i])}{p_{\mathcal{A}}(\varphi, S, \Gamma_2[i])}\right)^{-1}$$

Thus we have

$$\mathrm{E}[\mathbb{1}(\Gamma_3[j]_{\downarrow S} = \sigma_1)] = p_{\mathcal{G}}(\hat{\varphi}, S, \Gamma_1[i])$$

$$= \frac{p_{\mathcal{G}}(\varphi, S, \Gamma_1[i])}{p_{\mathcal{G}}(\varphi, S, \Gamma_2[i]) + p_{\mathcal{G}}(\varphi, S, \Gamma_1[i])} \quad [\text{ by the subquery consistent sampler assumption}]$$

$$\geq \left(1 + \frac{6\varepsilon + \eta}{4}\right) \cdot \frac{p_{\mathcal{A}}(\varphi, S, \Gamma_1[i])}{p_{\mathcal{A}}(\varphi, S, \Gamma_2[i])} \cdot \left(1 + \left(1 + \frac{6\varepsilon + \eta}{4}\right) \cdot \frac{p_{\mathcal{A}}(\varphi, S, \Gamma_1[i])}{p_{\mathcal{A}}(\varphi, S, \Gamma_2[i])}\right)^{-1}$$

$$= H \quad [\text{By definition of } H] \tag{9}$$

Barbarik2 returns REJECT in the ith iteration if the $Bias$ (in the ith iteration) is more than T, where $T = \frac{L+H}{2}$ with

$$H = \frac{(1 + \frac{6\varepsilon + \eta}{4})p_{\mathcal{A}}(\varphi, S, \sigma_1)}{(1 + \frac{6\varepsilon + \eta}{4})p_{\mathcal{A}}(\varphi, S, \sigma_1) + p_{\mathcal{A}}(\varphi, S, \sigma_2)}$$

And since, by definition, all the elements in Γ_1, Γ_2 and Γ_3 are obtained by drawing independent samples from $\mathcal{D}_{\mathcal{G}(\varphi)}$, $\mathcal{D}_{\mathcal{A}(\varphi)}$ and $\mathcal{D}_{\mathcal{G}(\hat{\varphi})}$ respectively so

$$\Pr\left[R_i \mid (\bigwedge_{j \in [i-1]} \overline{R_j}) \bigwedge (\Gamma_1[i] \in U_1 \wedge \Gamma_2[i] \in D)\right]$$

$$= \Pr\left[\, Bias > T \text{ in the } i\text{th iteration} \mid (\Gamma_1[i] \in U_1 \wedge \Gamma_2[i] \in D)\right]$$

$$= \Pr\left[\sum_{j \in [N]} \frac{\mathbb{1}(\Gamma_3[j]_{\downarrow S} = \sigma_1)}{N} \geq T \mid (\Gamma_1[i] \in U_1 \wedge \Gamma_2[i] \in D)\right]$$

Now since $\mathbb{1}(\Gamma_3[j]_{\downarrow S} = \sigma_1)$ are i.i.d 0-1 random variables and since $\Gamma_1[i] \in U_1$ and $\Gamma_2[i] \in D$ implies $\mathrm{E}[\mathbb{1}(\Gamma_3[j]_{\downarrow S} = \sigma_1)] \geq H$ (from Equation 9) by applying Chernoff bound from Lemma 5 we have:

$$\Pr\left[\frac{1}{N} \sum_{j \in [N]} \mathbb{1}(\Gamma_3[j]_{\downarrow S} = \sigma_1) \geq T\right] \leq exp\left(-\frac{(H - T)^2 N}{8H}\right)$$

$$\text{by the choice of N} \quad \leq \frac{\delta}{t}$$

$$\text{since } \delta < 0.5 \text{ and } t \geq 3 \quad \leq 1/5$$

$\square$

Lemma 8. *If the sampler $\mathcal{G}$ is η-far from the ideal sampler on input φ then*

$$\Pr\left[\Gamma_1[i] \in U_1 \wedge \Gamma_2[i] \in D\right] \geq \frac{\eta(\eta - 6\varepsilon)}{8}.$$

Proof. of Lemma 8) Since the sampler $\mathcal{G}$ is ε-far from the ideal sampler on input φ so, the ℓ_1 distance between $\mathcal{D}_{\mathcal{G}}(\varphi)$ and $\mathcal{D}_{\mathcal{A}}(\varphi)$ is at least η. By the definition of sets U and D we have,

$$\sum_{x \in U}(p_{\mathcal{G}}(\varphi, x) - p_{\mathcal{A}}(\varphi, x)) = \sum_{x \in D}(p_{\mathcal{A}}(\varphi, x) - p_{\mathcal{G}}(\varphi, x)) \geq \frac{\eta}{2} \tag{10}$$

Now by definition of U_0, we have

$$\sum_{x \in U_0} (p_{\mathcal{G}}(\varphi, x) - p_{\mathcal{A}}(\varphi, x)) < \frac{\eta + 6\varepsilon}{4} \quad \sum_{x \in U_0} p_{\mathcal{A}}(\varphi, x) < \frac{\eta + 6\varepsilon}{4} \tag{11}$$

As $U = U_0 \cup U_1$,

$$\sum_{x \in U_1} (p_{\mathcal{G}}(\varphi, x) - p_{\mathcal{A}}(\varphi, x))$$
$$= \sum_{x \in U} (p_{\mathcal{G}}(\varphi, x) - p_{\mathcal{A}}(\varphi, x)) - \sum_{x \in U_0} (p_{\mathcal{G}}(\varphi, x) - p_{\mathcal{A}}(\varphi, x)) \tag{12}$$

Substituting Equation (11) and Equation (10) in Equation (12) we get:-

$$\sum_{x \in U_1} (p_{\mathcal{G}}(\varphi, x) - p_{\mathcal{A}}(\varphi, x)) \geq \frac{\eta}{2} - \frac{\eta + 6\varepsilon}{4} = \frac{\eta - 6\varepsilon}{4}$$

$$\text{Therefore, } \sum_{x \in U_1} p_{\mathcal{G}}(\varphi, x) \geq \frac{\eta - 6\varepsilon}{4}$$

Thus we have,

$$\Pr[\Gamma_1[i] \in U_1] = \sum_{x \in U_1} p_{\mathcal{G}}(\varphi, x) \geq \frac{\eta - 6\varepsilon}{4} \tag{13}$$

From Equation (10) we know that,

$$\Pr[\Gamma_2[i] \in D] = \sum_{x \in D} p_{\mathcal{A}}(\varphi, x) \geq \frac{\eta}{2} \tag{14}$$

Since $\Gamma_1[i] \in U_1$ and $\Gamma_2[i] \in D$ are independent events, putting together Equation (13) and Equation (14), we see that

$$\Pr[\Gamma_1[i] \in U_1 \wedge \Gamma_2[i] \in D] \geq \frac{\eta(\eta - 6\varepsilon)}{8}$$

$\square$

B.3 Proof of Lemma 4

Lemma 4. *Given ε, η and δ,* Barbarik2 *needs at most $\widetilde{O}\left(\frac{tilt(\mathtt{wt}, \varphi)^2}{\eta(\eta - 6\varepsilon)^3}\right)$ samples for any input formula φ and weight function* wt*, where the tilde hides a poly logarithmic factor of $1/\delta, 1/\eta$ and $1/(\eta - 6\varepsilon)$.*

Proof. From Algorithm 1, line 1, we see that the number of trials is:

$$t = \frac{ln(1/\delta)}{ln\left(\frac{10}{10 - \eta(\eta - 6\varepsilon)}\right)}$$

$$(ln(x) \leq x - 1) \quad t \leq ln(1/\delta)\frac{10}{(\eta(\eta - 6\varepsilon))}$$

In every iteration we calculate a value N according to the expression:

$$N = 8ln\left(\frac{t}{\delta}\right) \cdot \frac{\alpha \cdot hi}{1 + \alpha \cdot hi} \cdot \left(\frac{\alpha \cdot hi}{1 + \alpha \cdot hi} - \frac{\alpha \cdot lo}{1 + \alpha \cdot lo}\right)^{-2}$$

$$= 8ln\left(\frac{t}{\delta}\right) \cdot \left(\frac{1}{hi - lo}\right)^2 \cdot hi \cdot \frac{1 + \alpha \cdot hi}{\alpha} \cdot (1 + \alpha \cdot lo)^2$$

$$(1 < lo < hi < 2) \quad < 8ln\left(\frac{t}{\delta}\right) \cdot \left(\frac{1}{hi - lo}\right)^2 \cdot 2 \cdot \frac{1 + \alpha \cdot 2}{\alpha} \cdot (1 + \alpha \cdot 2)^2$$

On Line (11) in Algorithm 1 we define:

$$\alpha = \frac{\texttt{wt}(\sigma_1)}{\texttt{wt}(\sigma_2)}$$

$$(\text{Definition } 4) \quad tilt(\texttt{wt}, \varphi) = \max_{\sigma_1, \sigma_2 \in R_\varphi} \frac{\texttt{wt}(\sigma_1)}{\texttt{wt}(\sigma_2)}$$

Thus, $\alpha \leq tilt(\texttt{wt}, \varphi)$. Substituting the values of α, lo and hi, we get:

$$N < 8ln\left(\frac{t}{\delta}\right) \cdot \left(\frac{tilt(\texttt{wt}, \varphi)}{\eta - 6\varepsilon}\right)^2$$

The maximum number of samples drawn after t trials is:

$$2t + tN < 2tN$$

$$(\text{Substituting for t,N}) \quad < 8ln\left(\frac{1}{\delta} \cdot \frac{10 \cdot ln(1/\delta)}{\eta(\eta - 6\varepsilon)}\right) \times \frac{10 \cdot ln(1/\delta)}{\eta(\eta - 6\varepsilon)} \times \frac{tilt(\texttt{wt}, \varphi)^2}{(\eta - 6\varepsilon)^2}$$

$$= \tilde{O}\left(\frac{tilt(\texttt{wt}, \varphi)^2}{\eta(\eta - 6\varepsilon)^3}\right)$$

$\square$

C Log-Linear Distributions and Inverse Transform Sampling

Log-linear models capture wide class of distributions of interest including those arising from graphical models, conditional random fields, skip-gram models [34]. Formally, for $\sigma \in \{0,1\}^n$, we define

$$\Pr[\sigma|\theta] \propto e^{\theta \cdot \sigma}$$

Following Chavira and Darwiche [16], we describe the following equivalent representation, called literal-weighted functions, of log-linear models.

Definition 11 (Literal-Weighted Functions). *For a CNF formula φ and set $S \subseteq Supp(\varphi)$, a weight function $\texttt{wt} : \{0,1\}^{|S|} \to (0,1)$ is called a literal-weighted function if there is a map $\texttt{W} : S \to (0,1)$ such that for any assignment $\sigma \in R_{\varphi \downarrow S}$*

$$\texttt{wt}(\sigma) = \prod_{x \in \sigma} \begin{cases} \texttt{W}(x) & if \quad x = 1 \\ 1 - \texttt{W}(x) & if \quad x = 0 \end{cases}$$

In this case we call $\texttt{wt}$ a literal-weighted function w.r.t. $\texttt{W}$. And note that we have $\Pr[\sigma] \propto \texttt{wt}(\sigma)$.

We now discuss the standard technique of inverse transform sampling for completeness. For completeness, we follow the description due to Chakrborty et al [15].

Lemma 9. *For any ε-close uniform sampler $\mathcal{V}$, any CNF formula φ with support S and a literal-weighted function $\texttt{wt} : \{0,1\}^{|S|} \to (0,1)$, we can construct a $\hat{\varphi}$ s.t.*

$$\forall_{\sigma \in R_\varphi}, \frac{(1 - \varepsilon)\texttt{wt}(\sigma)}{\sum_{\sigma' \in R_\varphi} \texttt{wt}(\sigma')} \leq p_\mathcal{V}(\hat{\varphi}, S, \sigma) \leq \frac{(1 + \varepsilon)\texttt{wt}(\sigma)}{\sum_{\sigma' \in R_\varphi} \texttt{wt}(\sigma')}$$

Proof. Let $S_i = \{x_{i,1}, \cdots, x_{i,m_i}\}$ be a set of m_i "fresh" variables (i.e. variables that were not used before) for each $x_i \in S$. Given any integer $m_i > 0$ and a positive odd number $k_i < 2^{m_i}$, we construct $\varphi_{k_i, m_i}(x_{i,1}, \cdots x_{i,m_i})$ using the chain formula construction in [15] such that $|R_{\varphi_{k_i, m_i}}| = k$. For notational clarity, we simply write φ_{k_i, m_i} when the arguments of the chain formula are clear from context. For each variable $x_i \in S$, such that $\texttt{W}(x_i^1) = \frac{k_i}{2^{m_i}}$, and $\texttt{W}(x_i^0) = 1 - \texttt{W}(x_i)$, let $(x_i \leftrightarrow \varphi_{k_i, m_i})$ be the representative clause. Thus let $\varphi^{CNF} = \bigwedge_{i \in S}(x_i \leftrightarrow \varphi_{k_i, m_i})$. We then define the formula $\hat{\varphi}$ as follows:

$$\hat{\varphi} = \varphi \wedge \varphi^{CNF}$$

We can see that model count of the formula $|R_{\hat{\varphi}}|$ can be given by:

$$|R_{\hat{\varphi}}| = \sum_{\hat{\sigma} \in R_{\hat{\varphi}}} 1 = \sum_{\sigma \in R_{\varphi}} \sum_{(\hat{\sigma} \in R_{\hat{\varphi}}:\hat{\sigma}_{\downarrow S}=\sigma)} 1 \tag{15}$$

Since the representative formula of every variable uses a fresh set of variables, we have from the structure of $\hat{\varphi}$ that if σ is a witness of φ then:

$$\sum_{(\hat{\sigma} \in R_{\hat{\varphi}}:\hat{\sigma}_{\downarrow S}=\sigma)} 1 = \prod_{i \in \sigma^0} (2^{m_i} - k_i) \prod_{i \in \sigma^1} k_i \tag{16}$$

For any $\sigma \in R_{\varphi}$:

$$
\begin{aligned}
p_{\mathcal{U}}(\hat{\varphi}, S, \sigma) &= \sum_{(\hat{\sigma} \in R_{\hat{\varphi}}:\hat{\sigma}_{\downarrow S}=\sigma)} p_{\mathcal{U}}(\hat{\varphi}, \hat{S}, \hat{\sigma}) \\
&= \sum_{(\hat{\sigma} \in R_{\hat{\varphi}}:\hat{\sigma}_{\downarrow S}=\sigma)} \frac{1}{|R_{\hat{\varphi}}|} \\
&= \frac{\sum_{(\hat{\sigma} \in R_{\hat{\varphi}}:\hat{\sigma}_{\downarrow S}=\sigma)} 1}{\sum_{\sigma' \in R_{\varphi}} \sum_{(\hat{\sigma} \in R_{\hat{\varphi}}:\hat{\sigma}_{\downarrow S}=\sigma')} 1} \quad \text{Using (15)} \\
&= \frac{\prod_{i \in \sigma^0}(2^{m_i} - k_i) \prod_{i \in \sigma^1} k_i}{\sum_{\sigma' \in R_{\varphi}} \prod_{i \in \sigma'^0}(2^{m_i} - k_i) \prod_{i \in \sigma'^1} k_i} \quad \text{Using (16)} \\
&= \frac{\prod_{i \in \sigma^0}(2^{m_i} - k_i) \prod_{i \in \sigma^1} k_i}{\prod_{i \in S} 2^{m_i}} \cdot \frac{\prod_{i \in S} 2^{m_i}}{\sum_{\sigma' \in R_{\varphi}} \prod_{i \in \sigma'^0}(2^{m_i} - k_i) \prod_{i \in \sigma'^1} k_i} \\
&= \frac{\prod_{i \in S} \mathtt{W}(\sigma_{\downarrow x_i})}{\sum_{\sigma' \in R_{\varphi}} \prod_{i \in S} \mathtt{W}(\sigma'_{\downarrow x_i})} \\
&= \frac{\mathtt{wt}(\sigma)}{\sum_{\sigma' \in R_{\varphi}} \mathtt{wt}(\sigma')} \tag{17}
\end{aligned}
$$

From the definition of ε-additive closeness (Def. 5) we have:

$$(1 - \varepsilon)p_{\mathcal{U}}(\varphi, S, \sigma) \le p_{\mathcal{V}}(\varphi, S, \sigma) \le (1 + \varepsilon)p_{\mathcal{U}}(\varphi, S, \sigma)$$

Substituing into 17, we get:

$$\forall_{\sigma \in R_{\varphi}}, \frac{(1 - \varepsilon)\mathtt{wt}(\sigma)}{\sum_{\sigma' \in R_{\varphi}} \mathtt{wt}(\sigma')} \le p_{\mathcal{V}}(\hat{\varphi}, S, \sigma) \le \frac{(1 + \varepsilon)\mathtt{wt}(\sigma)}{\sum_{\sigma' \in R_{\varphi}} \mathtt{wt}(\sigma')}$$

$\square$

Remark 1. It is worth noting that Lemma 9 implies that if $\mathcal{V}$ is ε-close uniform sampler $\mathcal{V}$ then it can be used as a blackbox to obtain a ε-close to an ideal sampler w.r.t any literal-weighted function wt. It should also be noted that Lemma 9 does not imply that if $\mathcal{V}$ is η-far from a uniform sampler, then the new sampler (obtained using the above transformation) is also far from the ideal sampler w.r.t wt. Therefore, to test whether $p_{\mathcal{V}}(\hat{\varphi}, S, \sigma)$ is close to ideal sampler, one can not rely on merely testing uniformity of $\mathcal{V}$.

D   Extended Tables of Results

D.1   Comparing sample complexity.

"A"("R") represent Barbarik2 returning `ACCEPT`(`REJECT`). "DNS" is used against those instances on which the indicated sampler Did Not Sample. "-" indicates that Barbarik2 timed out on that particular instance on the indicated sampler. Note that "DNS" is different from "-" as "DNS" indicates the failure of the underlying sampler to sample the initial set of samples, while "-" indicates the failure of Barbarik2 to finish within the timeout period. The timeout was set to 50,000 seconds for wSTS and wQuicksampler, while for wUniGen it was 24 hours.

Table 2: The Extended Table

Benchmark	$tilt$ (maxSamp)	Barbarik2		
		wUniGen (samples)	wSTS (samples)	wQuicksampler (samples)
107.sk_3_90	1 (2e+05)	DNS	R (5146)	R (6009)
tableBasedAddition.sk	1 (2e+05)	DNS	R (6009)	R (24534)
55.sk_3_46	1 (2e+05)	DNS	R (8911)	R (4354)
111.sk_2_36	1 (2e+05)	DNS	R (23543)	R (5150)
17.sk_3_45	1 (2e+05)	DNS	R (1e+05)	R (4677)
80.sk_2_48	1 (2e+05)	DNS	R (4284)	R (4627)
27.sk_3_32	1 (2e+05)	A (1e+05)	R (25329)	R (6009)
70.sk_3_40	1 (2e+05)	DNS	R (10402)	R (17704)
32.sk_4_38	1 (2e+05)	A (1e+05)	R (18081)	R (14682)
84.sk_4_77	1 (2e+05)	DNS	R (5146)	R (4354)
53.sk_4_32	1 (2e+05)	A (1e+05)	R (35618)	R (6009)
s35932_3_2	3 (6e+05)	DNS	TO	R (11756)
s35932_7_4	3 (6e+05)	DNS	TO	R (11756)
s832a_3_2	3 (6e+05)	A (1e+05)	R (8708)	R (54138)
109.sk_4_36	8 (3e+06)	DNS	R (26218)	R (6009)
77.sk_3_44	11 (5e+06)	DNS	R (47582)	R (47907)
s35932_15_7	12 (6e+06)	DNS	TO	R (4354)
s832a_7_4	15 (8e+06)	A (1e+05)	R (4393)	R (13350)
51.sk_4_38	18 (1e+07)	A (78661)	R (4284)	R (4627)
29.sk_3_45	26 (2e+07)	DNS	R (4284)	R (55989)
81.sk_5_51	27 (3e+07)	DNS	R (28409)	A (2e+05)
s349_3_2	28 (3e+07)	A (1e+05)	A (1e+05)	R (22854)

Benchmark	*tilt* (maxSamp)	Barbarik2		
		wUniGen (samples)	wSTS (samples)	wQuicksampler (samples)
s298_3_2	32 (3e+07)	A (1e+05)	R (80883)	R (26491)
s820a_3_2	37 (5e+07)	A (96212)	R (87997)	A (2e+05)
s298_15_7	44 (6e+07)	A (1e+05)	R (42520)	R (53107)
63.sk_3_64	58 (1e+08)	DNS	R (4393)	R (4677)
s820a_15_7	79 (2e+08)	A (84310)	R (2e+05)	R (16714)
s1488_15_7	110 (4e+08)	A (86152)	R (17168)	R (7341)
s1488_3_2	132 (6e+08)	A (89686)	A (89236)	R (7341)
s382_15_7	138 (6e+08)	A (92159)	R (2e+05)	R (6009)
UserServiceImpl.sk_8_32	140 (6e+08)	A (1e+05)	R (1e+05)	R (4393)
20.sk_1_51	144 (7e+08)	DNS	R (30895)	R (5146)
s820a_7_4	167 (9e+08)	A (95566)	A (1e+05)	R (6009)
s832a_15_7	194 (1e+09)	A (96984)	R (9434)	R (13350)
s1488_7_4	206 (1e+09)	A (1e+05)	R (4677)	R (4627)
s344_15_7	218 (2e+09)	A (90183)	R (94481)	R (4354)
LoginService2.sk_23_36	232 (2e+09)	A (1e+05)	R (38044)	R (13350)
s420_new1_15_7	265 (2e+09)	DNS	R (19224)	A (3e+05)
s349_15_7	412 (5e+09)	A (99215)	R (28400)	R (14682)
s444_15_7	501 (8e+09)	A (1e+05)	A (1e+05)	R (26627)
s349_7_4	603 (1e+10)	A (75555)	R (4284)	R (5150)
s444_7_4	644 (1e+10)	DNS	R (4393)	R (4354)
s420_new1_7_4	982 (3e+10)	A (1e+05)	R (4354)	R (18473)
s298_7_4	986 (3e+10)	A (83681)	R (8638)	R (6009)
s420_new1_3_2	1226 (5e+10)	DNS	A (1e+05)	R (5150)

Benchmark	$tilt$ (maxSamp)	Barbarik2		
		wUniGen (samples)	wSTS (samples)	wQuicksampler (samples)
s382_7_4	1283 (5e+10)	A (92307)	R (26491)	R (7341)
s420_3_2	1552 (8e+10)	A (1e+05)	R (14756)	R (48983)
s1238a_7_4	1856 (1e+11)	A (95095)	R (5150)	R (7341)
s1238a_3_2	1965 (1e+11)	A (1e+05)	R (28848)	R (4627)
s444_3_2	2028 (1e+11)	A (1e+05)	R (2e+05)	R (9500)
s1238a_15_7	2317 (2e+11)	DNS	R (9020)	R (88233)
s420_new_15_7	2317 (2e+11)	A (99198)	R (1e+05)	R (4393)
30.sk_5_76	2453 (2e+11)	DNS	R (5216)	R (4677)
s344_7_4	2607 (2e+11)	A (1e+05)	R (14170)	R (16818)
s344_3_2	3300 (3e+11)	A (1e+05)	R (59952)	R (5150)
s420_new_7_4	3549 (4e+11)	A (82312)	A (96659)	R (49955)
s953a_7_4	8984 (3e+12)	DNS	A (2e+05)	R (4627)
s953a_15_7	10596 (4e+12)	DNS	R (11734)	R (59735)
10.sk_1_46	15268 (7e+12)	DNS	R (35179)	R (1e+05)
s420_new_3_2	17449 (1e+13)	A (1e+05)	R (44937)	R (5150)
19.sk_3_48	18253 (1e+13)	DNS	R (59014)	R (4627)
s953a_3_2	20860 (1e+13)	DNS	R (51161)	R (1e+05)
s641_3_2	1e+06 (5e+16)	DNS	R (14454)	R (4627)
ProjectService3.sk_12_55	5e+06 (7e+17)	DNS	R (9020)	R (4393)
71.sk_3_65	1e+07 (3e+18)	DNS	R (1e+05)	R (4284)
s838_7_4	1e+07 (5e+18)	DNS	R (4393)	R (4284)
s838_15_7	3e+07 (3e+19)	DNS	R (5150)	R (4393)
s713_3_2	6e+07 (1e+20)	DNS	R (56386)	R (5827)

Benchmark	*tilt* (maxSamp)	Barbarik2		
		wUniGen (samples)	wSTS (samples)	wQuicksampler (samples)
s713_7_4	6e+07 (1e+20)	DNS	R (5827)	R (37419)
s641_7_4	9e+07 (3e+20)	DNS	R (8747)	A (1e+06)
s838_3_2	2e+08 (1e+21)	DNS	R (9504)	R (4627)
54.sk_12_97	4e+11 (6e+27)	DNS	R (14012)	R (4627)

D.2 Comparing the runtime performance of Barbarik2 against the baseline approach

In each of the following tables we compare the runtime of Barbarik2 against the runtime of the baseline approach. The runtime of Barbarik2 on REJECT instances depends on which iteration the tester terminated on. The runtime of the baseline is extrapolated from the expected number of samples and the average sampling rate of the sampler. To do this we use the ℓ_1-testing algorithm given in [4]. In the context of this paper, the algorithm assumes black box sample access to a uniform sampler over the models of a Boolean formula φ, and the sampler under test, and requires $O(\#\varphi^{2/3}(\eta - \varepsilon)^{-8/3}\log(\#\varphi/\delta))$ samples, where $\#\varphi$ is the model count, (ε, η) are the closenes and farness parameters, and δ is the confidence parameter.

D.2.1 Comparision with baseline for wSTS

Table 3: Extended table comparing the baseline tester for wSTS with Barbarik2

Benchmark	Baseline	Barbarik2(s)	Speedup
s349_7_4	16457.21	5	3428.58
s420_new1_7_4	5.4E+6	6	8.6E+5
s298_7_4	705.13	8	94.02
s444_7_4	1.1E+7	8	1.3E+6
s832a_7_4	3725.35	10	372.53
s1488_7_4	184.99	12	15.16
s344_7_4	24751.45	15	1683.77
s420_3_2	2.2E+6	17	1.3E+5
s1238a_7_4	1.4E+6	20	66538.64
s832a_3_2	2149.58	22	98.60
s832a_15_7	15121.66	24	622.29
s838_15_7	2.9E+13	27	1.1E+12
s349_15_7	16457.21	28	587.76
s838_7_4	3.7E+13	29	1.3E+12
s382_7_4	14915.27	32	469.03
s298_15_7	384.62	32	12.09
s420_new1_15_7	4.1E+6	33	1.3E+5
27.sk_3_32	79531.43	34	2346.06
s1238a_15_7	1.8E+6	37	49906.05
111.sk_2_36	2.9E+8	42	6.8E+6
51.sk_4_38	2.0E+6	44	45904.52
80.sk_2_48	6.0E+7	46	1.3E+6
s1488_15_7	128.69	48	2.67
s953a_15_7	1.1E+9	49	2.2E+7
s344_3_2	15750.92	51	309.45
s298_3_2	229.07	52	4.42
s838_3_2	2.7E+13	57	4.8E+11
s420_new_3_2	2.9E+6	65	44288.35
84.sk_4_77	3.4E+13	68	5.0E+11
s641_3_2	4.1E+10	70	5.9E+8
55.sk_3_46	2.0E+7	70	2.9E+5
s349_3_2	30563.39	73	416.96
107.sk_3_90	1.7E+15	86	1.9E+13
s1238a_3_2	2.2E+6	87	25824.41
s344_15_7	24751.45	91	271.10
32.sk_4_38	5.8E+5	94	6228.23
10.sk_1_46	6.5E+7	112	5.8E+5
29.sk_3_45	2.2E+8	150	1.5E+6
s420_new_7_4	4.1E+6	152	27272.30
s1488_3_2	52.52	163	0.32
s953a_3_2	6.4E+8	165	3.9E+6

Table 3: Extended table comparing the baseline tester for wSTS with Barbarik2

Benchmark	Baseline	Barbarik2(s)	Speedup
s420_new_15_7	4.5E+6	186	24014.34
70.sk_3_40	2.9E+6	201	14544.89
s444_15_7	13470.05	202	66.82
s420_new1_3_2	2.6E+6	211	12084.36
s820a_3_2	2189.81	221	9.91
s444_3_2	11186.45	247	45.22
s713_3_2	8.8E+10	255	3.5E+8
109.sk_4_36	6.6E+5	269	2459.36
s820a_7_4	4240.22	277	15.33
63.sk_3_64	5.8E+11	282	2.1E+9
s641_7_4	8.2E+10	311	2.6E+8
53.sk_4_32	55060.22	313	176.08
s382_15_7	33182.79	343	96.86
s820a_15_7	4154.77	370	11.23
ProjectService3.sk_12_55	1.3E+10	458	2.9E+7
s35932_3_2	3.6E+2	TO	-
s35932_7_4	3.6E+2	TO	-
s35932_15_7	3.6E+2	TO	-
s953a_7_4	5.7E+8	689	8.3E+5
UserServiceImpl.sk_8_32	479.33	720	0.67
30.sk_5_76	7.0E+14	1116	6.2E+11
77.sk_3_44	5.3E+6	1687	3156.66
tableBasedAddition.sk_240_1024	3.8E+14	1832	2.1E+11
81.sk_5_51	5.0E+9	2099	2.4E+6
LoginService2.sk_23_36	12951.33	2368	5.47
20.sk_1_51	1.1E+10	2568	4.1E+6
19.sk_3_48	3.1E+8	2760	1.1E+5
17.sk_3_45	4.5E+7	3016	14948.13
71.sk_3_65	4.7E+12	4365	1.1E+9
54.sk_12_97	2.7E+18	4688	5.8E+14

D.3 wQuicksampler

Table 4: Extended table comparing the baseline tester for wQuicksampler with Barbarik2

Benchmark	Baseline	Barbarik2(s)	Speedup
s344_3_2	24751.45	3	8534.98
s344_15_7	24751.45	4	7071.84
s349_7_4	28212.36	4	7624.96
s298_7_4	512.82	4	119.26
s420_new1_3_2	5.1E+6	4	1.2E+6
s420_new_3_2	2.2E+6	4	5.1E+5
s420_new_15_7	3.5E+6	4	7.8E+5
s382_7_4	12429.39	5	2589.46
s444_7_4	51980.83	5	10192.32
s820a_7_4	2283.19	5	430.79
s1488_7_4	128.07	6	20.99
s444_3_2	8700.57	6	1359.46
s838_7_4	1.3E+13	7	1.8E+12
27.sk_3_32	48942.42	7	6797.56
s1238a_3_2	1.6E+6	7	2.2E+5
s953a_7_4	6.6E+8	8	8.8E+7
s1488_3_2	65.65	8	8.31
s838_3_2	1.9E+13	8	2.4E+12
s1488_15_7	60.56	9	6.80
s349_15_7	35265.44	9	3833.20
s344_7_4	22501.32	9	2393.76
s349_3_2	14106.18	10	1424.87
55.sk_3_46	4.5E+7	10	4.3E+6
s1238a_7_4	1.1E+6	11	97431.59
s298_3_2	534.49	11	46.89
s832a_7_4	4139.28	12	344.94
111.sk_2_36	5.2E+5	12	41613.34
s838_15_7	2.6E+13	12	2.1E+12
s420_new1_7_4	2.2E+6	13	1.7E+5
s832a_15_7	13861.52	14	1011.79
UserServiceImpl.sk_8_32	326.81	14	23.68
s382_15_7	27149.56	15	1859.56
53.sk_4_32	91767.04	16	5595.55
s820a_15_7	5665.59	17	335.24
84.sk_4_77	2.1E+13	18	1.2E+12
51.sk_4_38	1.8E+6	19	91363.08
s444_15_7	14817.06	19	763.77
109.sk_4_36	6.6E+5	20	33425.00
107.sk_3_90	1.6E+15	21	7.4E+13
71.sk_3_65	1.3E+12	27	5.0E+10
s641_3_2	2.8E+10	28	1.0E+9
s298_15_7	1153.85	30	38.98
32.sk_4_38	1.2E+6	34	36689.91
s420_3_2	4.5E+6	34	1.3E+5
s420_new_7_4	3.5E+6	36	96896.05
80.sk_2_48	2.1E+8	37	5.7E+6
s832a_3_2	2149.58	45	47.66
19.sk_3_48	4.5E+8	50	9.0E+6
63.sk_3_64	2.1E+11	51	4.0E+9
17.sk_3_45	8.3E+7	55	1.5E+6
s713_3_2	9.4E+10	56	1.7E+9
s953a_15_7	6.7E+8	79	8.5E+6

Table 4: Extended table comparing the baseline tester for wQuicksampler with Barbarik2

Benchmark	Baseline	Barbarik2(s)	Speedup
20.sk_1_51	4.0E+9	82	4.8E+7
70.sk_3_40	4.3E+6	101	42475.10
s1238a_15_7	1.0E+6	107	9614.31
10.sk_1_46	7.1E+7	128	5.5E+5
s953a_3_2	3.4E+8	132	2.6E+6
s820a_3_2	1167.90	137	8.54
30.sk_5_76	3.0E+14	210	1.4E+12
ProjectService3.sk_12_55	6.4E+9	219	2.9E+7
LoginService2.sk_23_36	12692.30	229	55.52
s420_new1_15_7	3.2E+6	232	13726.91
77.sk_3_44	1.2E+7	409	30125.88
29.sk_3_45	1.3E+8	658	2.0E+5
54.sk_12_97	4.0E+17	690	5.8E+14
s641_7_4	6.8E+10	1117	6.1E+7
s35932_15_7	1.4E+356	1182	1.2E+353
tableBasedAddition.sk_240_1024	3.0E+13	1430	2.1E+10
s35932_7_4	1.2E+356	2227	5.5E+352
s35932_3_2	1.1E+356	2346	4.5E+352
81.sk_5_51	2.0E+9	2461	8.3E+5

D.4 wUniGen

Table 5: Extended table comparing the baseline tester for wUniGen with Barbarik2

Benchmark	Baseline	Barbarik2(s)	Speedup
s1488_3_2	229.78	6648	0.03
s298_7_4	7564.11	10758	0.70
s1488_15_7	643.45	11493	0.06
s298_15_7	2948.72	12325	0.24
s349_7_4	1.8E+06	12858	136.40
s820a_15_7	48724.11	14070	3.46
s344_15_7	3.8E+05	14074	27.18
s1488_7_4	853.78	15049	0.06
s820a_7_4	42728.33	16124	2.65
s349_15_7	3.9E+05	17690	21.80
s382_7_4	9.7E+05	21785	44.45
s349_3_2	3.0E+05	22395	13.54
s832a_15_7	5.6E+05	23036	24.45
s420_new_7_4	4.0E+09	24092	1.7E+5
s344_7_4	1.7E+06	26423	64.55
51.sk_4_38	2.7E+09	26612	1.0E+5
s820a_3_2	2.3E+05	27408	8.47
s298_3_2	2061.62	30262	0.07
s344_3_2	5.0E+05	32378	15.29
s1238a_7_4	1.5E+09	33689	45408.69
s832a_7_4	76990.55	34315	2.24
s382_15_7	1.0E+07	39024	263.98
s1238a_3_2	7.1E+08	40406	17575.38
s420_new_15_7	4.9E+09	40725	1.2E+5
27.sk_3_32	7.4E+06	41997	176.26
s832a_3_2	74844.43	42696	1.75
UserServiceImpl.sk_8_32	21547.88	45090	0.48
32.sk_4_38	4.9E+08	45126	10872.88
s420_new1_7_4	2.8E+08	48911	5639.38
s444_3_2	1.9E+06	55017	34.61
LoginService2.sk_23_36	1.3E+06	56229	22.38
s420_3_2	2.3E+09	68048	33247.50
53.sk_4_32	2.2E+07	70590	312.87
s420_new_3_2	1.2E+10	75284	1.6E+5

D.5 Number of samples required for baseline approach

Table 6: Number of samples required for baseline tester

Benchmark	Number of samples
s344_3_2	2E+6
s344_15_7	2E+6
s349_7_4	2E+6
s298_7_4	6E+4
s420_new1_3_2	3E+8
s420_new_3_2	3E+8
s420_new_15_7	3E+8
s382_7_4	1E+6
s444_7_4	4E+6
s820a_7_4	3E+5
s1488_7_4	1E+4
s444_3_2	1E+6
s838_7_4	2E+15
27.sk_3_32	6E+6
s1238a_3_2	1E+8
s953a_7_4	4E+10
s1488_3_2	7E+3
s838_3_2	2E+15
s1488_15_7	8E+3
s349_15_7	2E+6
s344_7_4	2E+6
s349_3_2	2E+6
55.sk_3_46	2E+9
s1238a_7_4	1E+8
s298_3_2	4E+4
s832a_7_4	4E+5
111.sk_2_36	3E+7
s838_15_7	2E+15
s420_new1_7_4	3E+8
s832a_15_7	1E+6
UserServiceImpl.sk_8_32	2E+4
s382_15_7	3E+6
53.sk_4_32	6E+6
s820a_15_7	4E+5
84.sk_4_77	1E+15
51.sk_4_38	8E+7
s444_15_7	1E+6
109.sk_4_36	4E+7
107.sk_3_90	7E+16
71.sk_3_65	5E+13
s641_3_2	3E+12
s298_15_7	6E+4
32.sk_4_38	7E+7
s420_3_2	3E+8
s420_new_7_4	3E+8
80.sk_2_48	6E+9
s832a_3_2	2E+5
19.sk_3_48	1E+10
63.sk_3_64	5E+12
17.sk_3_45	2E+9
s713_3_2	6E+12

Table 6: Number of samples required for baseline tester

Benchmark	Number of samples
s953a_15_7	5E+10
20.sk_1_51	7E+10
70.sk_3_40	2E+8
s1238a_15_7	1E+8
10.sk_1_46	6E+9
s953a_3_2	4E+10
s820a_3_2	1E+5
30.sk_5_76	2E+15
ProjectService3.sk_12_55	2E+11
LoginService2.sk_23_36	1E+5
s420_new1_15_7	3E+8
77.sk_3_44	3E+8
29.sk_3_45	3E+9
54.sk_12_97	4E+18
s641_7_4	5E+12
s35932_15_7	1E+357
tableBasedAddition.sk_240_1024	1E+15
s35932_7_4	1E+357
s35932_3_2	1E+357
81.sk_5_51	4E+10