On Almost-Uniform Generation of SAT Solutions: The power of 3-wise independent hashing

Remi Delannoy and Kuldeep S. Meel
School of Computing, National University of Singapore, Singapore

Abstract

Given a Boolean formula φ and a distribution parameter ε, the problem of almost-uniform generation seeks to design a randomized generator such that every solution of φ is output with probability within $(1+\varepsilon)$-factor of $\frac{1}{|sol(\varphi)|}$ where $sol(\varphi)$ is the set of all the solutions of φ. The prior state of the art scheme due to Jerrum, Valiant, and Vazirani, makes $O(n^2 \log n + n \log n \log \varepsilon^{-1})$ calls to a SAT oracle and employs 2−wise independent hash functions.

In this work, we design a new randomized algorithm that makes $O(\varepsilon^{-1} + \log n \log \varepsilon^{-1})$ calls to a SAT oracle and employs 3−wise independent hash functions. The widely used 2−wise independent hashing is tabulation hashing proposed by Carter and Wegman. Since this classical scheme is also 3−wise independent, we observe that practical implementation of our technique does not incur additional overhead. We demonstrate that theoretical improvements translate to practice; in particular, we conduct a comprehensive study over 562 benchmarks and demonstrate that while JVV would time out for 544 out of 562 instances, our proposed scheme can handle all the 562 instances. To the best of our knowledge, this is the **first almost-uniform generation scheme** that can handle practical instances from real-world applications. We also present a nuanced analysis focusing on the both the size of SAT queries as well as the number of queries.

ACM Reference Format:
Remi Delannoy and Kuldeep S. Meel. 2022. On Almost-Uniform Generation of SAT Solutions: The power of 3-wise independent hashing. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Let φ denote a Boolean formula in conjunctive normal form (CNF), and let X be the set of variables appearing in F. Let $n = |X|$. A truth assignment σ maps variables in X to 0 or 1. A *satisfying assignment* or *witness* of F is an assignment that makes φ evaluate to true. We denote the set of all witnesses of φ by $sol(\varphi)$. We use $\Pr[\mathcal{X}]$ to denote the probability of event $\mathcal{X}$. In this paper, we consider the following problem:

Almost-Uniform Generation for Boolean Formulas
Input Formula φ and parameter ε
Output $y \in sol(\varphi) \cup \{\perp\}$. Let Ret be the event that $\sigma \in sol(\varphi)$ is output. Then,

$$\frac{1}{(1+\varepsilon)|sol(\varphi)|} \leq \Pr[\sigma \text{ is output} \mid \text{Ret}] \leq \frac{1+\varepsilon}{|sol(\varphi)|}$$

For some constant c, $\Pr[\text{Ret}] > c$

The problem of almost-uniform generation was first considered by Jerrum, Valiant, and Vazirani [10] who showed that almost-uniform generation can be achieved in probabilistic polynomial time given access to an approximate counter, which in turn employs a SAT oracle. In particular, the JVV algorithm[1] makes $O(n^2 \log n + n \log n \log \varepsilon^{-1}))$ calls to a SAT oracle (defined formally in Section 2) and uses 2-wise independent hash functions. The work of Jerrum et al. also considered the stricter notion of uniform generation (wherein $\varepsilon = 0$) and showed that uniform generation can be accomplished in probabilistic polynomial time given access to Σ_2^P oracle. Subsequently, Bellare, Goldreich, and Petrank [2] showed that uniform generation can be achieved in probabilistic polynomial time given access to NP oracle. In particular, their proposed procedure, henceforth referred to as BGP, makes $O(n^2 \log n)$ calls to SAT oracle but requires n−wise independent hash functions. One can express t−wise independent hash functions with polynomial of degree t over $GF(2^n)$. The high degree of polynomials make computations over them expensive and therefore, one would ideally like to work with polynomials of as small degree as possible. Therefore, from theoretical perspective, one wonders *whether we can design algorithmic procedure for almost-uniform generation that makes fewer queries to* SAT *oracle and requires hash functions with smaller independence.*

The need for efficient algorithms is exacerbated by the wide ranging applications of uniform sampling in diverse domains such as pathogen transmission inference in bioinformatics [17], constrained-random simulation for bug discovery for software and hardware [15], probabilistic inference

[1]We use initials of the authors to refer to the procedure proposed in their work

Conference'17, July 2017, Washington, DC, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

for graphical models [5], and the like. In all such applications, the practical implementation of uniform generators replace SAT oracle with a state of the art SAT solver. Therefore, one would ideally like to make as fewer calls to a SAT oracle (a SAT solver in practice) as possible and use hash functions with as small independence as possible.

1.1 Our Results

The key contribution of this work is to establish the following two theorems:

Theorem 1.1. *Given access to* SAT *oracle, for all $\varepsilon > 0$, there exists a probabilistic polynomial time almost-uniform generation procedure for Boolean Formulas that makes $O(\varepsilon^{-1} + \log n \cdot \log \varepsilon^{-1})$ queries to* SAT *oracle and uses 3−wise independent hashing.*

If given access to only 2−wise independent hash family, we obtain the following:

Theorem 1.2. *Given access to* SAT *oracle, for all $\varepsilon > 1$[2], there exists a probabilistic polynomial time almost-uniform generation procedure for Boolean Formulas that makes $O(\log n \cdot \log(\frac{1}{\varepsilon - 1}))$ queries to* SAT *oracle and uses 2−wise independent hashing.*

To put our results in the context of existing work, we summarize our results along with existing results in Table 3.

In typical use cases of uniform sampling, ε is set to a small constant. Therefore, from practical perspective, for a fixed ε, we improve the complexity of almost-uniform generation from $O(n^2 \log n)$ to $O(\log n)$ while requiring 3−wise independence instead of 2−wise independence.

It is worth noting that the standard construction of t−wise independent hash function is a random $t - 1$ degree polynomial in a large enough prime field. But thankfully, for small constant independence, more efficient constructions are known. In particular, the tabulation-based hashing dating back to Carter and Wegman [4] is known to be 3−wise (and hence, 2-wise) independent [12]. Formally, a random hash function belonging to tabulation-based hashing $h : \{0,1\}^n \mapsto \{0,1\}^m$ can be constructed as $h(X) = \boldsymbol{A}X + \boldsymbol{b}$ where $\boldsymbol{A}$ is a $m \times n$ matrix while $\boldsymbol{b}$ is $m \times 1$ 0-1 vector and each entry of $\boldsymbol{A}$ and $\boldsymbol{b}$ is either 0 or 1 with equal probability (i.e., 1/2). Tabulation-based scheme is shown to be significantly faster than alternative construction using degree 2-polynomials [16]. The availability of hash functions that can be described using linear function over GF(2) plays a consequential role in practical implementation of the techniques.

It is worth emphasizing that we consider a salient strength of this paper is the simplicity of the algorithm including usage of simple data structures. Furthermore, the associated analysis takes a different route in comparison to prior work [2, 8, 10]. In particular, in our view, we offer a much simpler and intuitive analysis methodology in comparison to prior work. Another important contribution of our work is a nuanced complexity analysis, presented in Section 6, that seeks to capture both time and space complexity of SAT queries; such an analysis is inspired from the practical behavior of SAT solvers.

To illustrate the practical impact of improvement due to our results [3], we implemented the three samplers wherein we employed the state of the art SAT solver, CryptoMiniSAT, to perform SAT calls. Our empirical evaluation over 562 benchmarks show that while our algorithm can successfully sample within one hour for all such instances, the algorithms BGP timed out on all the instances and JVV timed out on 544 out of 562 instances.

The rest of the paper is organized as follows: In Section 2, we introduce the basic notations and preliminaries. In Section 3, we present background overview of inner workings of JVV and BGP to put our contribution in context. We present the primary technical contribution of this paper in Section 4. In Section 5, we discuss how our proposed scheme can be extended to handle arbitrary NP relations. In Section 7, we provide empirical evidence to demonstrate that the theoretical improvements lead to significant improvements in practice. We finally conclude in Section 8.

2 Preliminaries

In this section, we will state basic notations and preliminaries.

2.1 Boolean Formulas

Let φ denote a Boolean formula in conjunctive normal form (CNF), and let X be the set of variables appearing in φ. Let $n = |X|$. A *satisfying assignment* or *witness* of φ is an assignment that makes φ evaluate to true. We denote the set of all witnesses of φ by $sol(\varphi)$.

SAT **Oracle**

In this work, we assume access to a SAT oracle that takes in a formula φ and returns a satisfying assignment σ if φ is satisfiable and $\perp$ is φ is unsatisfiable. The model of SAT oracle captures the behavior of the modern SAT solvers. Since the problem of satisfiability is self-reducible, one can easily show that the our model of SAT oracle can be linearly (in n) simulated via the model of NP decision oracle, which returns Yes or No.

BSAT **Procedure**

In this work, we use the gadget of BSAT, which was introduced by Bellare et al in [2]. Formally, give a formula φ and

[2]It is perhaps worth remarking that $\varepsilon > 1$ is not a typo and the problem is not trivial for the case when $\varepsilon > 1$

[3]We are aware that a typical LICS paper may not expect experiments but wish to demonstrate that improvements in theory also translate to significant impact in practice.

	SAT Queries	Indep.	Notes
BGP [2]	$O(n^2 \log n)$	n	$\varepsilon = 0$
CMV [8]	$O(\frac{1}{(\varepsilon - 1.71)^2} + \log n)$	3	$\varepsilon > 1.71$
JVV [10]	$O(n^2 \log n + n \log n \log \varepsilon^{-1})$	2	$\varepsilon > 0$
Theorem 1.1	$O(\varepsilon^{-1} + \log n \cdot \log \varepsilon^{-1})$	3	$\varepsilon > 0$
Theorem 1.2	$O(\log n \cdot \log(\frac{1}{\varepsilon - 1}))$	2	$\varepsilon > 1$

Table 1. Almost-uniform generators along with the number of queries and the independence of hash functions where $n = |X|$

threshold t, BSAT returns a set of solutions W such that $|W| = min(t, |sol(\varphi)|)$. The following proposition characterizes the complexity of BSAT.

Proposition 1. BSAT *can be implemented in polynomial time given access to a* SAT *oracle and makes* $O(t)$ *calls to* SAT *oracle.*

Proof. The following procedure suffices:

```
1: counter ← 0
2: while counter ≤ $t$ do
3:     $\sigma \leftarrow$ SAT($\varphi$)
4:     if $\sigma \neq \perp$ then
5:         counter ← counter+1
6:         $\varphi \leftarrow \varphi \wedge (X \neq \sigma)$
7:     else
8:         break
9: return counter
```

□

2.2 Approximate Model Counting

A problem closely related to almost-uniform generation is that of approximate model counting. Given a CNF formula φ, a tolerance $\theta > 0$ and a confidence parameter $\delta \in (0, 1]$, a *probabilistic approximate model counter* ApproxCount$(\cdot, \cdot, \cdot)$ ensures that

$$\Pr[\frac{|sol(\varphi)|}{1+\theta} \leq \text{ApproxCount}(\varphi, \theta, \delta) \leq (1+\theta)|sol(\varphi)|] \geq 1 - \delta.$$

We will employ ApproxCount as a gadget and the following theorem establishes the runtime complexity.

Proposition 2. *[3, 21] Given a CNF formula φ over n variables, tolerance $\theta > 0$ and a confidence parameter $\delta \in (0, 1]$, there exists a probabilistic approximate model counter that runs in probabilistic polynomial time given access to a* SAT *oracle and makes $O(\log(n/\theta) \log(1/\delta))$ calls to* SAT *oracle.*

Proof. We will use the procedure ApproxMC as a subroutine; the following claim captures the complexity of ApproxMC as stated in Theorem 4 of [9]:

Claim 1. *Given a CNF formula φ over n variables, tolerance $\hat{\theta} > 0$ and a confidence parameter $\delta \in (0, 1]$,* ApproxMC *runs in probabilistic polynomial time given access to a* SAT *oracle and makes $O(\log(n) \cdot \frac{1}{\hat{\theta}^2} \cdot \log(1/\delta))$ calls to* SAT *oracle, and returns an estimate c such that*

$$\Pr\left[\frac{|sol(\varphi)|}{1+\hat{\theta}} \leq c \leq (1+\hat{\theta})|sol(\varphi)|\right] \geq 1 - \delta$$

Given a formula φ over n variables. We first construct another formula ψ such that ψ is essentially t copies of φ where $t > \log \frac{1}{1+\theta}$. Formally, let φ be defined on the set of variables $X^{(1)}$ such that $n = |X^{(1)}|$. We now create another $(t-1)$ sets of fresh variables, denoted by $X^{(2)}, \ldots X^{(t)}$, such that each of these sets have n variables. Formally, $|X^{(1)}| = |X^{(2)}| = \ldots |X^{(t)}| = n$. We can now define ψ as follows:

$$\psi := \varphi(X^{(1)}) \wedge \varphi(X^{(1)} \mapsto X^{(2)}) \cdots \wedge \varphi(X^{(1)} \mapsto X^{(t)})$$

Where $X^{(1)} \mapsto X^{(i)}$ means that we substitute the variables of the set $X^{(1)}$ by the variables of the set $X^{(i)}$ in the formula. Note that $|sol(\psi)| = |sol(\varphi)|^t$. Next, we compute an estimate initEst = ApproxMC$(\psi, 1, \delta)$. Now our estimate of the $|sol(\varphi)|$ is simply $(\text{initEst})^{1/t}$. Note that the number of calls to SAT oracle is $O(\log(n/\hat{\theta}) \log(1/\delta))$. It is probably worth emphasizing that the trick of creating multiple copies is fairly standard, and is indeed used in Figure 1 of [3] as well.

□

2.3 k-wise Independent Hashing

Let $n, m \in \mathbb{N}$ and $\mathcal{H}(n, m) \subseteq \{h : \{0, 1\}^n \rightarrow \{0, 1\}^m\}$ be a family of hash functions mapping $\{0, 1\}^n$ to $\{0, 1\}^m$. We use $h \xleftarrow{R} \mathcal{H}(n, m)$ to denote the probability space obtained by choosing a function h uniformly at random from $\mathcal{H}(n, m)$.

Definition 1. *A family of hash functions $\mathcal{H}(n, m)$ is $k-$wise independent if for all distinct $y_1, y_2, \cdots y_k \in \{0, 1\}^n$ and $\forall \alpha_1, \alpha_2, \cdots \alpha_k \in \{0, 1\}^m, h \xleftarrow{R} \mathcal{H}(n, m)$,*

$$\Pr[(h(x_1) = \alpha_1) \wedge \cdots \wedge (h(x_k) = \alpha_k)] = \left(\frac{1}{2^m}\right)^k$$

We are interested in the set of elements of $sol(\varphi)$ mapped to cell α by h, denoted $\mathsf{Cell}_{\langle \varphi, h, \alpha \rangle}$ and its cardinality. Formally, $\mathsf{Cell}_{\langle \varphi, h, \alpha \rangle} = \{y \mid y \in sol(\varphi) \text{ and } h(y) = \alpha\}$. In our work, we will be interested in a fixed α and for simplicity, we will fix $\alpha = \mathbf{0}$. We will use shorthand $\mathsf{Cnt}_{\langle \varphi, h \rangle}$ to denote $|\mathsf{Cell}_{\langle \varphi, h, \mathbf{0} \rangle}|$.

Typically, higher the value of k, stronger one can obtain concentration bounds on the random variable $\mathsf{Cnt}_{\langle \varphi, h \rangle}$. In particular, the concentration bounds of the following form are typically obtain:

$$\Pr\left[\left|\mathsf{Cnt}_{\langle\varphi,h\rangle} - \mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle}]\right| \geq \beta \mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle}]\right] \leq f(\mathsf{E}[|\mathsf{Cnt}_{\langle\varphi,h\rangle}], k, \beta)$$

The standard concentration bounds lead to $f(\mathsf{E}[|\mathsf{Cnt}_{\langle\varphi,h\rangle}|], k, \beta)$ such that f is monotonically decreasing in $k, \beta, \mathsf{E}[|\mathsf{Cnt}_{\langle\varphi,h\rangle}|]$.

The following proposition concerning 2-wise and 3-wise hash functions is important in our analysis

Proposition 3. *Given a 2-wise independent hash family $H(n,m)$, for $h \xleftarrow{R} H(n,m), \forall \sigma \in sol(\varphi), \forall \alpha \in \{0,1\}^m$*

1. $\mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle}] = \frac{|sol(\varphi)|}{2^m}$
2. $\mathsf{E}\left[\mathsf{Cnt}_{\langle\varphi,h\rangle} \mid h(\sigma) = \mathbf{0}\right] = 1 + \frac{|sol(\varphi)|-1}{2^m}$
3. $Var\left[\mathsf{Cnt}_{\langle\varphi,h\rangle}\right] = \frac{|sol(\varphi)|}{2^m} - \frac{|sol(\varphi)|}{2^{2m}} \leq \mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle}]$
4. *If $H(n,m)$ is also a 3-wise independent hash family, then* $Var[\mathsf{Cnt}_{\langle\varphi,h\rangle} \mid h(\sigma) = \alpha] = \frac{|sol(\varphi)|-1}{2^m} - \frac{|sol(\varphi)|-1}{2^{2m}} \leq \mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle} \mid h(\sigma) = \mathbf{0}]$

Proof. $\forall \omega \in sol(\varphi)$, let $\gamma_{\omega,\alpha}$ be the indicator variable of the event $h(\omega) = \alpha$.

$$\gamma_{\omega,\alpha} = \begin{cases} 1 & \text{if } h(\omega) = \alpha \\ 0 & \text{otherwise} \end{cases}$$

Since $\mathsf{Cnt}_{\langle\varphi,h\rangle} = \sum_{\omega \in sol(\varphi)} \gamma_{\omega,\alpha}$, therefore,

$$\mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle}] = \sum_{\omega \in sol(\varphi)} \mathsf{E}[\gamma_{\omega,\alpha}] = \sum_{\omega \in sol(\varphi)} \Pr[h(\omega) = \alpha] = \frac{|sol(\varphi)}{2^m}$$

Given $h(\sigma) = \alpha$, we have $\gamma_{\sigma,\alpha} = 1$. Therefore,

$$\mathsf{E}[[\mathsf{Cnt}_{\langle\varphi,h\rangle} \mid h(\sigma) = \alpha] = 1 + \frac{|sol(\varphi)| - 1}{2^m}$$

The variance is $Var[\mathsf{Cnt}_{\langle\varphi,h\rangle}] = \sum_{\omega \in sol(\varphi)} Var[\gamma_{\omega,\alpha}] + \sum_{\omega \neq \omega'} Cov[\gamma_{\omega,\alpha}, \gamma_{\omega',\alpha}]$. We have $Var[\gamma_{\omega,\alpha}] = \frac{1}{2^m}(1 - \frac{1}{2^m})$. Then, let us compute the covariance
$Cov[\gamma_{\omega,\alpha}, \gamma_{\omega',\alpha}] = \Pr[h(\omega) = \alpha \wedge h(\omega') = \alpha] - \Pr[h(\omega) = \alpha]\Pr[h(\omega') = \alpha]$
$= \frac{1}{2^{2m}} - \frac{1}{2^m}\frac{1}{2^m} = 0$. Thus, we have $Var[\mathsf{Cnt}_{\langle\varphi,h\rangle}] = \frac{|sol(\varphi)|}{2^m} - \frac{|sol(\varphi)|}{2^{2m}}$

Also, observe that if $H(n,m)$ is 3-wise independent hash family, then,
$Var[\mathsf{Cnt}_{\langle\varphi,h\rangle} \mid h(\sigma) = \alpha] = \sum_{\omega \in sol(\varphi)\setminus\{\sigma\}} Var[\gamma_{\omega,\alpha}] + \sum_{\omega \neq \omega'} Cov[\gamma_{\omega,\alpha}, \gamma_{\omega',\alpha} \mid \gamma_{\sigma,\alpha} = 1]$. The conditional covariance becomes
$Cov[\gamma_{\omega,\alpha}, \gamma_{\omega',\alpha} | \gamma_{\sigma,\alpha} = 1]$
$= \frac{\Pr[h(\omega)=\alpha, h(\omega')=\alpha, h(\sigma)=\alpha]}{\Pr[h(\sigma)=\alpha]} - \frac{\Pr[h(\omega)=\alpha, h(\sigma)=\alpha]}{\Pr[h(\sigma)=\alpha]}\frac{\Pr[h(\omega')=\alpha, h(\sigma)=\alpha]}{\Pr[h(\sigma)=\alpha]}$
$= \frac{2^m}{2^{3m}} - \frac{2^m}{2^{2m}}\frac{2^m}{2^{2m}} = \frac{1}{2^{2m}} - \frac{1}{2^{2m}} = 0$
Thus we have $Var[\mathsf{Cnt}_{\langle\varphi,h\rangle} | h(\sigma) = \alpha] = \frac{|sol(\varphi)|-1}{2^m} - \frac{|sol(\varphi)|-1}{2^{2m}}$ □

The standard construction of $k-$wise independent hash families consists of a degree $k-1$ polynomially over a large enough prime field or $GF(2^n)$. But thankfully, for small constant independence, efficient constructions are known. In particular, the tabulation-based hashing proposed by Carter and Wegman [4] is known to be 3−wise (and hence, 2-wise) independent [12]. Formally, the tabulation-based scheme can be defined as follows:

Proposition 4. *Let H be the set of all functions of the form $h(X) = \boldsymbol{A}X + \boldsymbol{b}$ where $\boldsymbol{A} \in GF(2)^{m \times n}$ and $b \in GF(2)^m$. Note that a uniformly random $h \in H$ can be constructed by setting each entry in $\boldsymbol{A}$ and $\boldsymbol{b}$ to 0 or 1 independently with probability $\frac{1}{2}$ each. Then H is 3-wise independent, i.e., all distinct $y_1, y_2, y_3 \in \{0,1\}^n$ and $\forall \alpha_1, \alpha_2, \alpha_3 \in \{0,1\}^m, h \xleftarrow{R} \mathcal{H}(n,m)$,*

$$\Pr[(h(x_1) = \alpha_1) \wedge (h(x_2) = \alpha_2) \wedge (h(x_3) = \alpha_3)] = \frac{1}{2^{3m}}$$

Tabulation-based scheme is shown to be significantly faster than alternative construction using degree 2-polynomials [16].

3 Background

To put our contributions in context, we review the prior approaches to (almost-)uniform generation due to Bellare, Goldreich, and Petrank (referred to as BGP approach) and Jerrum, Valiant, and Vazirani (referred to as JVV approach).

JVV approach is based on the observation that for a fixed ordering of variables, say $x_1 \succ x_2 .... \succ x_n$, we can set x_1 to 0 with probability $\frac{|sol(\varphi \wedge (x_1=0))|}{|sol(\varphi)|}$. Then, once we have set x_1 to v_1, we can set x_2 to 0 with probability $\frac{|sol(\varphi \wedge x_1 = v_1 \wedge (x_2=0))|}{|sol(\varphi \wedge (x_1 = v_1))|}$, and accordingly we can set rest of the variables. It is easy to see that if we have access to an exact counter, then the scheme would give rise to an exact counter, the above scheme would lead to a uniform generator. Interestingly, the careful analysis due to Jerrum et al's also demonstrated that given access to a probabilistic approximate model counter the above process would lead to almost uniform generator wherein each call to counter is called with $\theta = 1/n$ and $\delta = \frac{\varepsilon}{2^{n+2}(2n+1)}$. There are two shortcomings of the above approach: (1) the procedure invokes a counter $2n$ times, and (2) each of the invokations of approximate counter employ very small values of θ and δ.

The BGP approach seeks to circumvent multiple invokations of counter and instead seeks to *directly* employ the hashing paradigm. The key idea is to use a $k-$wise independent hash family $\mathcal{H}(n,m)$ such that for a randomly chosen $h \in \mathcal{H}$ and for an appropriately chosen m, it is the case that all the cells are *small*, wherein a cell α is small if it has less than $2n^2$ solutions, i.e., a cell α is small if $|\mathsf{Cell}_{\langle F,h,\alpha\rangle}| < \text{thresh}$ for some appropriately chosen thresh. The key underpinning observation is that one can use a NP oracle to check if all the cells are small. If it is the case that all the cells are small, then one can randomly choose one of the cells, enumerate all the solutions in the cell using an NP oracle. Now if we were to just randomly sample a solution in a cell, then we may not achieve uniformity as while all the cells are small but it is not necessarily the case that all cells have equal number of solutions. Bellare et al. observed that we could just randomly pick a number between 1 and thresh, if the chosen number is less than the size of the cell, we pick an element from the cell uniformly at random else output $\perp$. One possible threat to this idea is what if most of the cells have very few solutions (e.g., a small constant number of solutions)? In such a case, we would be outputting $\perp$ with probability close to 1. Thankfully, Bellare et al also observed that for the appropriately chosen value of m, it is also the case that with probability 0.9, all cells are non-trivial, i.e., $|\mathsf{Cell}_{\langle F,h,\alpha\rangle}| > \frac{\mathsf{thresh}}{4}$, and small, i.e., $|\mathsf{Cell}_{\langle F,h,\alpha\rangle}| < \mathsf{thresh}$. It is perhaps worth noting that thresh is set to $2n^2$.

The primary shortcoming of BGP approach is its reliance on $\frac{n}{\log n}$wise independent hashing. The BGP paper worked with n−wise independent hash functions but one can show that $\frac{n}{\log n}$-wise independence suffices thanks to the concentrations bounds for limited independence due to Schmidt, Siegel, and Srinivasan [18]. The crucial aspect of the analysis of BGP is to compute the probability that for a random choice of h, for all α, $\mathsf{thresh}/4 \leq |\mathsf{Cell}_{\langle F,h,\alpha\rangle}| < \mathsf{thresh}$. We use $\frac{n}{\log n}$-independence of hash functions to ensure that for a given cell α, the probability of $\mathsf{thresh}/4 \leq |\mathsf{Cell}_{\langle F,h,\alpha\rangle}| < \mathsf{thresh}$ is bounded by $1 - 2^{n+c}$ for some constant c. Applying union bounds allow us to lower bound the probability that all the cells are non-trivial and small. The theoretical analysis framework of BGP can not be extended to handle hash functions with constant independence. It is perhaps worth emphasizing that BGP guarantees uniformity while our objective is to design an almost-uniform generator.

To summarize, JVV approach reduces the problem of almost-uniform generation to linear number of invocations of a probabilistic approximate counter with $\delta = O(2^{-n}n^{-1}\varepsilon)$ and $\theta = 1/n$. As noted above, there exist approximate counters that employ 2-wise independent hashing. On the other hand, BGP approach directly attempts to employ hashing-based paradigm but requires n−wise independent hash functions. In this work, we seek to design an approach that seeks to achieve the best of both the worlds: we seek to call approximate counter only once and then seek to directly employ hashing-based paradigm with 3−wise independence instead of $\frac{n}{\log n}$-wise independence as needed in BGP.

In addition BGP and JVV approaches, another attempt in similar spirit owes to Chakraborty, Meel, and Vardi [6, 8] who sought to blend the usage of approximate counting and 3-wise hashing; their proposed scheme could only provide guarantees of uniformity for $\varepsilon > 1.71$.

4 Almost-Uniform Generation

In our desired algorithm, we seek to employ hash families with constant independence (in particular, 3−wise and 2−wise). As a first step, we need to determine the appropriate value of m. To this end, we seek to rely on close relationship of counting and sampling, and invoke a probabilistic approximate counter to compute an estimate of $|sol(\varphi)|$, and compute the m based on the returned estimate by the counter. We will invoke the counter with a constant θ and $\delta \leq \varepsilon/4$ (Recall, JVV employs $\theta = 1/n$ and $\delta = O(2^{-n}n^{-1}\varepsilon)$). Given our usage of hash functions with constant independence, we can not ensure that size of all the cells would be below a predefined threshold and therefore, we seek to randomly pick a cell α, and check if its size is below a threshold. If the size of the cell is indeed below the threshold, we appropriately pick a solution. We now first present the algorithm formally, then perform the theoretical analysis, and close this section with interesting observations.

4.1 Algorithm

Algorithm 1 UniSamp(φ, ε)

```
1: pivot ← $\max(200, \frac{2}{\varepsilon})$    ▷ desired expected size of a cell
2: thresh ← $2 + \lceil 4\mathsf{pivot} \rceil$
3: $W$ ← BSAT($\varphi$, thresh + 1)
4: if $|W| \leq$ thresh then
5:     Choose $k$ uniformly at random in $[\![1, |W|]\!]$
6:     return $W[k]$
7: $\delta \leftarrow \min(0.1, \frac{\varepsilon}{4})$
8: $C$ ← ApproxCount($\varphi, \sqrt{2} - 1, \delta$)
9: $m \leftarrow \lfloor \log_2(\frac{C}{\mathsf{pivot}}) + \frac{1}{2} \rfloor$
10: Choose $h$ at random from $H(n, m)$
11: $W$ ← BSAT($F \wedge h(\sigma) = 0$, thresh + 1)
12: if $|W| \leq$ thresh then
13:     Choose $k$ uniformly at random in $[\![1, \mathsf{thresh}]\!]$
14:     if $|W| \geq k$ then return $W[k]$
15: return $\perp$
```

We present the pseudo-code of the desired algorithm, called UniSamp, in Algorithm 1. We describe the algorithm using the hash family $H(n, m)$ whose independence properties will be spelled out in the following sections. UniSamp first invokes BSAT to check if φ has less than thresh solutions (line 3). In such a case the strategy is to simply enumerate the solutions and pick one of the solutions uniformly at random. Otherwise, we invoke an approximate counter to determine the value of m (lines 8− 9). Once we have determined the appropriate value of m, we pick a hash function randomly from $H(n, m)$ and simply enumerate the solutions of F that map to $h^{-1}(0)$.

4.2 Analysis when $H(n, m)$ is 3-wise independent

The primary objective of this section is to establish the following theorem:

Theorem 1.1. *Given access to* SAT *oracle, for all $\varepsilon > 0$, there exists a probabilistic polynomial time almost-uniform generation procedure for Boolean Formulas that makes $O(\varepsilon^{-1} + \log n \cdot$*

$\log \varepsilon^{-1})$ *queries to* SAT *oracle and uses* 3*−wise independent hashing.*

Proof. We prove the above theorem by demonstrating that UniSamp with 3-wise independent hash family $H(n, m)$ gives rise to almost-uniform sampling (Lemma 3) and succeeds i.e., does not return $\perp$, with probability at least 0.05 (Lemma 4). The Lemma 5 establishes the complexity of UniSamp with respect to the number of queries to SAT oracles. As noted in Introduction, our analysis takes a different route compared to that of JVV [10], BGP [2], and CMV [8]. □

We first begin with list of shorthands for clarity of exposition:

1. Ret : The algorithm UniSamp doesn't return $\perp$
2. AcCnt : The count C returned by ApproxCount satisfies $\frac{|sol(\varphi)|}{\sqrt{2}} \leq C \leq \sqrt{2}|sol(\varphi)|$
3. $\mathsf{Cnt}_{\langle\varphi,h\rangle} = |\mathsf{Cell}_{\langle\varphi,h,\mathbf{0}\rangle}|$
4. for $\sigma \in sol(\varphi)$, O_σ : The algorithm UniSamp outputs σ

We begin by observing that if $|sol(\varphi)| \leq \mathsf{thresh}$ then it is immediate that the output distribution is uniform.

Therefore, we focus on the case $|sol(\varphi)| > \mathsf{thresh}$. First observe from line 14, we have $\forall\sigma \in sol(\varphi)$,

$$\Pr[O_\sigma] = \frac{1}{\mathsf{thresh}} \times \Pr[h(\sigma) = 0 \wedge \mathsf{Cnt}_{\langle\varphi,h\rangle} \leq \mathsf{thresh}]$$

Lemma 1. *$\forall\sigma \in sol(\varphi)$,*
$\Pr[O_\sigma|\mathsf{Ret}] = \frac{\Pr[\mathsf{Cnt}_{\langle\varphi,h\rangle} \leq \mathsf{thresh}|h(\sigma)=0]}{\sum_{\omega\in sol(\varphi)} \Pr[\mathsf{Cnt}_{\langle\varphi,h\rangle} \leq \mathsf{thresh}|h(\omega)=0]}$

Proof. Let $\sigma \in sol(\varphi)$

$$\Pr[O_\sigma|\mathsf{Ret}] = \frac{\Pr[O_\sigma \wedge \mathsf{Ret}]}{\Pr[\mathsf{Ret}]} = \frac{\Pr[O_\sigma]}{\Pr[\mathsf{Ret}]}$$
$$= \frac{1/\mathsf{thresh} \times \Pr[h(\sigma) = 0 \wedge \mathsf{Cnt}_{\langle\varphi,h\rangle} \leq \mathsf{thresh}]}{1/\mathsf{thresh} \times \sum_{\omega\in sol(\varphi)} \Pr[h(\omega) = 0 \wedge \mathsf{Cnt}_{\langle\varphi,h\rangle} \leq \mathsf{thresh}]}$$
$$= \frac{\Pr[\mathsf{Cnt}_{\langle\varphi,h\rangle} \leq \mathsf{thresh} \mid h(\sigma) = 0]}{\sum_{\omega\in sol(\varphi)} \Pr[\mathsf{Cnt}_{\langle\varphi,h\rangle} \leq \mathsf{thresh} \mid h(\omega) = 0]}$$

□

The last equality comes from the fact that $\forall\sigma \in sol(\varphi)$, $\Pr[h(\sigma) = 0] = 1/2^m$.

Lemma 2. *$\forall\sigma \in sol(\varphi), \Pr[\mathsf{Cnt}_{\langle\varphi,h\rangle} \leq \mathsf{thresh}|h(\sigma) = 0 \wedge \mathsf{AcCnt}] \geq 1 - \frac{1}{2\mathsf{pivot}}$*

Proof. From Proposition 3, we have
$\mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle} \mid h(\sigma) = 0] = 1 + \frac{|sol(\varphi)|-1}{2^m}$ and $Var[\mathsf{Cnt}_{\langle\varphi,h\rangle} \mid h(\sigma) = 0] = \frac{|sol(\varphi)|-1}{2^m}$.

From the definition of m line 10, we have $\frac{C}{\sqrt{2}\mathsf{pivot}} \leq 2^m \leq \frac{\sqrt{2}C}{\mathsf{pivot}}$, and if we have an approximate count, then, we also have $\frac{|sol(\varphi)|}{\sqrt{2}} \leq C \leq \sqrt{2}|sol(\varphi)|$. Thus,

$$\mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle}|h(\sigma) = 0 \wedge \mathsf{AcCnt}] \leq 1 + 2 \cdot \mathsf{pivot}$$
$$Var[\mathsf{Cnt}_{\langle\varphi,h\rangle}|h(\sigma) = 0 \wedge \mathsf{AcCnt}] \leq 2 \cdot \mathsf{pivot}.$$

Remember that $\mathsf{thresh} \geq 2 + 4\mathsf{pivot}$. Therefore,

$$\Pr\left[\mathsf{Cnt}_{\langle\varphi,h\rangle} > \mathsf{thresh}|h(\sigma) = 0 \wedge \mathsf{AcCnt}\right]$$
$$\leq \Pr\left[|\mathsf{Cnt}_{\langle\varphi,\mathrm{h}\rangle} - \mathsf{E}[\mathsf{Cnt}_{\langle\varphi,\mathrm{h}\rangle}]| \geq 1 + 2\mathsf{pivot}|\mathrm{h}(\sigma) = 0 \wedge \mathsf{AcCnt}\right].$$

Applying Chebyshev inequality we get

$$\Pr\left[|\mathsf{Cnt}_{\langle\varphi,\mathrm{h}\rangle} - \mathsf{E}[\mathsf{Cnt}_{\langle\varphi,\mathrm{h}\rangle}]| \geq 1 + 2\mathsf{pivot}|\mathrm{h}(\sigma) = 0 \wedge \mathsf{AcCnt}\right]$$
$$\leq \frac{Var[\mathsf{Cnt}_{\langle\varphi,h\rangle}|h(\sigma) = 0 \wedge \mathsf{AcCnt}]}{(1 + 2\mathsf{pivot})^2} \leq \frac{1}{2\mathsf{pivot}}$$

□

Remark 4.1. *Note that application of Chebyshev inequality required bounding $Var[\mathsf{Cnt}_{\langle\varphi,h\rangle} \mid h(\sigma) = 0]$ from above. We obtained such a bound on $Var[\mathsf{Cnt}_{\langle\varphi,h\rangle} \mid h(\sigma) = 0]$ with the usage of* 3*−wise independence.*

Lemma 3. *$\forall\sigma \in sol(\varphi)$, $\frac{1}{(1+\varepsilon)|sol(\varphi)|} \leq \Pr[O_\sigma|\mathsf{Ret}] \leq \frac{1+\varepsilon}{|sol(\varphi)|}$*

Proof. The high-level overview is to employ Lemma 2 to obtain the lower bound and employ Lemma 1 to obtain the desired upper bound.

$$\Pr[\mathsf{Cnt}_{\langle\varphi,h\rangle} \leq \mathsf{thresh}|h(\sigma) = 0]$$
$$\geq \Pr[\mathsf{AcCnt}] \Pr[\mathsf{Cnt}_{\langle\varphi,h\rangle} \leq \mathsf{thresh}|h(\sigma) = 0 \wedge \mathsf{AcCnt}]$$

Using Lemma 2, we obtain

$$\Pr[\mathsf{Cnt}_{\langle\varphi,h\rangle} \leq \mathsf{thresh}|h(\sigma) = 0] \geq (1 - \delta)(1 - \frac{1}{2\mathsf{pivot}})$$

Thus, we have $\forall\sigma \in sol(\varphi)$, $(1-\delta)(1-\frac{1}{2\mathsf{pivot}}) \leq \Pr[\mathsf{Cnt}_{\langle\varphi,h\rangle} \leq \mathsf{thresh}|h(\sigma) = 0] \leq 1$

Then using Lemma 1,

$$\frac{(1 - \frac{1}{2\mathsf{pivot}})(1 - \delta)}{|sol(\varphi)|} \leq \Pr[O_\sigma|S] \leq \frac{1}{(1 - \frac{1}{2\mathsf{pivot}})(1 - \delta)|sol(\varphi)|}.$$

If $0 < \varepsilon \leq 1$, $1 - \frac{\varepsilon}{2} \geq \frac{1}{1+\varepsilon}$, and as $\mathsf{pivot} \geq \frac{2}{\varepsilon}$ and $\delta \leq \frac{\varepsilon}{4}$, $(1 - \delta)(1 - \frac{1}{2\mathsf{pivot}}) \geq 1 - \frac{\varepsilon}{2} \geq \frac{1}{1+\varepsilon}$

Else if $\varepsilon > 1$, $\delta = 0.1$ and $\mathsf{pivot} = 200$ thus $(1 - \delta)(1 - \frac{1}{2\mathsf{pivot}}) = 0.89775 \geq \frac{1}{2} \geq \frac{1}{1+\varepsilon}$

Hence in all cases $\frac{1}{(1+\varepsilon)|sol(\varphi)|} \leq \Pr[O_\sigma|\mathsf{Ret}] \leq \frac{1+\varepsilon}{|sol(\varphi)|}$ □

Lemma 4. *$\Pr[\mathsf{Ret}] \geq 0.05$*

Proof. Let $\mathsf{loTh} = \frac{3\mathsf{pivot}}{10}$, we have

$$\Pr[\mathsf{Ret}] \geq (1 - \delta) \Pr[\mathsf{loTh} \leq \mathsf{Cnt}_{\langle\varphi,h\rangle} \leq \mathsf{thresh}|\mathsf{AcCnt}] \frac{\mathsf{loTh}}{\mathsf{thresh}}$$

This inequality comes from the observation that if $\mathsf{Cnt}_{\langle\varphi,h\rangle} \geq \mathsf{loTh}$ then there is a probability of at least $\frac{\mathsf{loTh}}{\mathsf{thresh}}$ that we return an element of $sol(\varphi)$ (we choose k uniformly at random in $[\![1, \mathsf{thresh}]\!]$ and there are at least loTh values that yield a return).

Next, observe that $\frac{\mathsf{loTh}}{\mathsf{thresh}} \geq \frac{3\mathsf{pivot}}{10*(3+4\mathsf{pivot})} \geq \frac{2}{27}$ (as pivot $\geq$ 200)

Remember that thresh $\geq 2 + 4$pivot, and if we have an accurate count, then, $\frac{\mathsf{pivot}}{2} \leq \mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle}] \leq 2\mathsf{pivot}$.

Hence, we have $\mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle}] - \frac{\mathsf{pivot}}{5} \geq \mathsf{loTh}$ and $\mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle}] + \frac{\mathsf{pivot}}{5} \leq \mathsf{thresh}$

Thus,

$$\Pr[\mathsf{loTh} \leq \mathsf{Cnt}_{\langle\varphi,h\rangle} \leq \mathsf{thresh} \mid \mathsf{AcCnt}]$$
$$\geq \Pr\left[\left| \mathsf{Cnt}_{\langle\varphi,h\rangle} - \mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle}] \right| < \frac{\mathsf{pivot}}{5} \mid \mathsf{AcCnt}\right]$$

Applying Chebyshev inequality $\Pr\left[\left| \mathsf{Cnt}_{\langle\varphi,h\rangle} - \mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle}] \right| < \frac{\mathsf{pivot}}{5} \mid \mathsf{AcCnt}\right] \leq \frac{25\mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle}]}{\mathsf{pivot}^2} \leq \frac{50}{\mathsf{pivot}} \leq \frac{1}{4}$

Hence $\Pr[\mathsf{loTh} \leq \mathsf{Cnt}_{\langle\varphi,h\rangle} \leq \mathsf{thresh} \mid \mathsf{AcCnt}] \geq \frac{3}{4}$ and $\Pr[S] \geq \frac{9}{10}\frac{3}{4}\frac{2}{27} = 0.05$ □

Lemma 5. *The algorithm* UniSamp *makes* $O(\log n \log \varepsilon^{-1} + \varepsilon^{-1})$ *calls to* SAT *oracle.*

Proof. ApproxCount is invoked with $\delta \leq \varepsilon/4$, therefore, from Proposition 2, this leads to $O(\log n \log \varepsilon^{-1})$ calls. Furthermore, since thresh $\in O(\varepsilon^{-1})$, the calls to BSAT add another $O(\varepsilon^{-1})$. Therefore, in total UniSamp makes $O(\log n \log \varepsilon^{-1} + \varepsilon^{-1})$ calls to SAT oracle. □

Choice of Parameters

The complexity of the algorithm is tightly related to the values of δ and pivot. The parameter δ is directly linked to the number of calls to the SAT oracle, and pivot is the main parameter that influences the number of calls made by BSAT. It is worth noting that we only need to verify the equation $(1-\delta)(1-\frac{1}{2\mathsf{pivot}}) \leq \frac{1}{1+\varepsilon}$ for the Lemma 3 to be true. We purposefully take simple values for the sake of proof simplicity. One could search better values by trying to minimize the number of calls to the oracle $O(\log n \log \delta^{-1} + \mathsf{pivot})$ and/or the query size $O(|\varphi| + \mathsf{pivot} + n^2)$.

Nevertheless, we expected the probability to be close to $\frac{\mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle}]}{\mathsf{thresh}}$. Indeed, we only fail if k is greater than $\mathsf{Cnt}_{\langle\varphi,h\rangle}$, as we sample k uniformly at random, the expected probability of success is $\frac{\mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle}]}{\mathsf{thresh}}$. Unfortunately, proving that the probability of success is, indeed, $\frac{\mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle}]}{\mathsf{thresh}}$ is not that simple because of several technical details. Mainly, BSAT halts after thresh calls, so we only have a truncated expectation ($\Pr[\mathsf{Ret}] = \sum_{i=1}^{\mathsf{thresh}} \frac{i}{\mathsf{thresh}} \Pr[\mathsf{Cnt}_{\langle\varphi,h\rangle} = i]$). Because of this, to guarantee a probability of success, we were forced to do a less careful analysis, and we likely get a very loose bound. This is also the reason why we add upper and lower bounds on δ and pivot respectively.

4.3 From 3-wise to 2-wise independent hash function family

As mentioned in Remark 4.1, the proof of Lemma 2 crucially relies on the usage of 3-wise independence. In this context, one naturally wonders whether it is possible to obtain stronger guarantees with access to only 2-wise independent hash functions. The objective of this section is to establish the following theorem:

Theorem 1.2. *Given access to* SAT *oracle, for all* $\varepsilon > 1$[4]*, there exists a probabilistic polynomial time almost-uniform generation procedure for Boolean Formulas that makes* $O(\log n \cdot \log(\frac{1}{\varepsilon-1}))$ *queries to* SAT *oracle and uses* 2*−wise independent hashing.*

4.4 Modifications to UniSamp

We will consider the variant of *UniSamp* wherein line 7 is replaced by $\delta \leftarrow min(0.1, \frac{\varepsilon-1}{\varepsilon+1})$. It is worth noting that this change still implies $\log(\delta^{-1}) = O(\varepsilon^{-1})$. However, thresh is now a constant, so therefore, the *UniSamp* makes $O(\log n \log \varepsilon^{-1} + \mathsf{thresh}) \in O(\log n \log \varepsilon^{-1})$ calls to SAT oracle.

Lemma 6. *If* $H(n,m)$ *is 2-wise independent,* $\forall \sigma \in sol(\varphi)$ $\Pr(\mathsf{Cnt}_{\langle\varphi,h\rangle} \leq \mathsf{thresh} | h(\sigma) = 0 \wedge \mathsf{AcCnt}) \geq \frac{1}{2}$

Proof. Reusing the notations introduced in Lemma 2, note that computation of $\mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle} | h(\sigma) = 0 \wedge \mathsf{AcCnt}]$ only requires 2-wise independence. Therefore, $\mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle} | h(\sigma) = 0 \wedge \mathsf{AcCnt}] \leq 1 + 2\mathsf{pivot}$.

But the 2-wise independence does not suffice to bound the $Var[\mathsf{Cnt}_{\langle\varphi,h\rangle} | h(\sigma) = 0 \wedge \mathsf{AcCnt}]$ and therefore, we can not proceed with the usage of Chebyshev inequality. That said, we observe that we can instead rely on the weaker concentration bound offered by Markov inequality as follows:

$$\Pr[\mathsf{Cnt}_{\langle\varphi,h\rangle} \leq \mathsf{thresh} | \gamma_\sigma \wedge \mathsf{AcCnt}]$$
$$\geq 1 - \frac{\mathsf{E}[\mathsf{Cnt}_{\langle\varphi,h\rangle} | h(\sigma) = 0 \wedge \mathsf{AcCnt}]}{\mathsf{thresh}}$$

Recalling that thresh $\geq 2 + 4$pivot,

$$\Pr[\mathsf{Cnt}_{\langle\varphi,h\rangle} \leq \mathsf{thresh} | \gamma_\sigma \wedge \mathsf{AcCnt}] \geq 1 - \frac{1+2\mathsf{pivot}}{2+4\mathsf{pivot}} = \frac{1}{2}$$

□

It is perhaps worth noting that with 3−wise independence, we could lower bound the above probability by $1 - \frac{1}{2\mathsf{pivot}}$, and therefore, allowing it to be arbitrarily close to 1.

Theorem 4.2. *If* $H(n,m)$ *is* 2*−wise independent, then*

$$\forall \varepsilon > 1, \forall \sigma \in sol(\varphi),$$
$$\frac{1}{(1+\varepsilon)|sol(\varphi)|} \leq \Pr(O_\sigma | \mathsf{Ret}) \leq \frac{1+\varepsilon}{|sol(\varphi)|}$$

[4]It is perhaps worth remarking that $\varepsilon > 1$ is not a typo and the problem is not trivial for the case when $\varepsilon > 1$

Proof. The proof follows along the similar lines that of Lemma 3 with the usage of Lemma 2 substituted with that of Lemma 6.

$$\begin{aligned}
&\Pr(\mathsf{Cnt}_{\langle\varphi,h\rangle} \le \mathsf{thresh} \mid h(\sigma) = 0)\\
&= \Pr(\mathsf{AcCnt}) \Pr(\mathsf{Cnt}_{\langle\varphi,h\rangle} \le \mathsf{thresh} \mid h(\sigma) = 0 \land \mathsf{AcCnt})\\
&+ \Pr(\overline{\mathsf{AcCnt}}) \Pr(\mathsf{Cnt}_{\langle\varphi,h\rangle} \le \mathsf{thresh} \mid h(\sigma) = 0 \land \overline{\mathsf{AcCnt}})\\
&\ge \Pr(\mathsf{AcCnt}) \Pr(\mathsf{Cnt}_{\langle\varphi,h\rangle} \le \mathsf{thresh} \mid h(\sigma) = 0 \land \mathsf{AcCnt})
\end{aligned}$$

Using Lemma 6, $\Pr(\mathsf{Cnt}_{\langle\varphi,h\rangle} \le \mathsf{thresh} \mid h(\sigma) = 0) \ge (1 - \delta)(1 - \frac{1}{2})$. Therefore, we have $\forall \sigma \in sol(\varphi), (1 - \delta)(1 - \frac{1}{2}) \le \Pr(\mathsf{Cnt}_{\langle\varphi,h\rangle} \le \mathsf{thresh} \mid h(\sigma) = 0) \le 1$

Then using Lemma 1,

$$\frac{(1-\delta)}{2|sol(\varphi)|} \le \Pr(O_\sigma \mid \mathrm{Ret}) \le \frac{2}{(1-\delta)|sol(\varphi)|}$$

As $\delta \le \frac{\varepsilon-1}{\varepsilon+1}$, $1 - \delta \ge \frac{2}{\varepsilon+1}$ and we have

$$\frac{1}{(1+\varepsilon)|sol(\varphi)|} \le \Pr(O_\sigma \mid \mathrm{Ret}) \le \frac{1+\varepsilon}{|sol(\varphi)|}$$

□

5 Universal Generators

So far we have focused on one particular NP relation: SAT but the core framework can be applied to any arbitrary NP relation with minor modifications. To this end, we will first fix some notations: given an NP language L, let R be the NP-relation defining L, i.e. $L = \{y \mid \exists w \text{ such that } R(y, w) = 1\}$. Also, let $n = |y|$, and without loss of generality, we can assume $R_y \overset{def}{=} \{x \mid R(y, x) = 1\} \subseteq \{0, 1\}^n$.

Similar to [2], for $h : \{0, 1\}^n \mapsto \{0, 1\}^m$, $y \in L_R$, we can define

$$R_{y,h} = \{w \in R_y \mid h(w) = 0\} = h^{-1}(0) \cap R_y$$

The following proposition generalizes the Proposition 1

Proposition 5. *[2] For every* NP *relation R, there is a polynomial time oracle algorithm* $\mathcal{M}_1$ *that takes in* $y \in L_R$, $h \in H(n, m)$, *and* thresh *as input and outputs* $R_{y,h}$ *if* $|R_{y,h}| \le$ thresh *and* $\perp$ *otherwise.* $\mathcal{M}_1$ *makes* $O(n \cdot \mathsf{thresh})$ *calls to* NP *oracle.*

Proof. Let consider the set $S_{R,h} = \{(y, k) \mid \exists z_1, z_2...z_k$ such that $z_1...z_k$ are distinct and $\forall i \in [\![1, k]\!], R(y, z_k) = 1$ and $h(z_k) = 0\}$ and $S'_{R,h} = \{(y, k, i, j) \mid \exists z_1 \prec z_2 \prec ... \prec z_k$ such that $z_{i,j} = 1$ and $\forall l \in [\![1, k]\!], R(y, z_l) = 1$ and $h(z_l) = 0\}$

Here, $z_{i,j}$ is the j-th bit of z_i and $\prec$ denotes some ordering relation. As R is an NP relation and h can be computed in polynomial time, these sets are in NP. Then, given the access to a NP oracle, $\mathcal{M}_1$ can be implemented as follow.

It is immediate that we make at most $1+\mathsf{thresh}+\mathsf{thresh}\cdot n \in O(n \cdot \mathsf{thresh})$ calls to the oracle.

□

Note that when $m = 0$, h is just an identity function, and therefore, we will omit mentioning h for the special case when m = 0. We also have an analogous generalization for Proposition 2.

Algorithm 2 $\mathcal{M}_1(R, y, h, \mathsf{thresh})$

```
1: if $(y, \mathsf{thresh} + 1) \in S_{R,h}$ then                ▷ call the oracle
2:     return $\perp$;
3: $count \leftarrow 0$;
4: while $(x, count) \in S_{R,h}$ do                     ▷ call the oracle
5:     $count \leftarrow count + 1$;
6: for $1 \le i \le count$ do
7:     for $1 \le j \le n$ do
8:         if $(y, count, i, j) \in S'_{R,h}$ then          ▷ call the oracle
9:             $y_{i,j} \leftarrow 1$;
10:        else
11:            $y_{i,j} \leftarrow 0$;
12:    $y_i \leftarrow y_{i,1}...y_{i,n}$;
   return $\{y_1, y_2, ..., y_{count}\}$
```

Algorithm 3 UniversalSampler(R, y, ε)

```
1: $pivot \leftarrow \max(200, \frac{2}{\varepsilon})$
2: $\mathsf{thresh} \leftarrow 1 + \lceil 4pivot \rceil$
3: $W \leftarrow \mathcal{M}_1(R, y, \mathsf{thresh})$
4: if $W \ne \perp$ then
5:     Choose $k$ uniformly at random in $[\![1, |W|]\!]$
6:     return $w_k$
7: $\delta \leftarrow \min(0.1, \frac{\varepsilon}{4})$
8: $C \leftarrow \mathcal{M}_2(R, y, \sqrt{2} - 1, \delta)$
9: $m \leftarrow \lfloor \log_2(\frac{C}{pivot}) + \frac{1}{2} \rfloor$
10: Choose $h$ at random from $H(n, m)$
11: $W \leftarrow \mathcal{M}_1(R, y, h, \mathsf{thresh})$
12: if $W \ne \perp$ then
13:     Choose $k$ uniformly at random in $[\![1, \mathsf{thresh}]\!]$
14:     if $|W| \ge k$ then return $W[k]$
15: return $\perp$
```

Proposition 6. *[3] For every* NP *relation R, there is a probabilistic polynomial time algorithm* $\mathcal{M}_2$ *that takes in* $y \in L_R$, $\theta > 0$, *and* $\delta > 0$ *as input and returns an estimate* v *such that*

$$\Pr[\frac{|R_y|}{1+\theta} \le v \le (1+\theta)|R_y|] \ge 1 - \delta$$

Furthermore, $\mathcal{M}_2$ *makes* $O(\log(n/\theta) \log(1/\delta))$ *calls to* NP *oracle.*

Equipped with the two propositions, we can now finally state the theorem:

Theorem 5.1. *Let R be an* NP*-relation. Then there exists an almost-uniform generation procedure for R for all* $\varepsilon > 0$ *that makes* $O(n\varepsilon^{-1})$ *queries to* NP *oracle and uses* $3-$*wise independent hashing.*

Proof. Given proposition 5 and 6 we can adapt our algorithm by replacing BSAT by $\mathcal{M}_1$ and ApproxCount by $\mathcal{M}_2$.

We state the pseudocode of the algorithm in Algorithm 3 for completeness. Then, it is immediate that our algorithm makes $O(n\varepsilon^{-1} + \log n \log(\varepsilon^{-1})) \in O(n\varepsilon^{-1})$ queries to the NP oracle. □

6 A Nuanced Analysis: On Size of SAT Queries

The traditional analysis often focuses on the number of queries to the given oracle. Over the past two decades, SAT solving has witnessed an unprecedented progress that has allowed the modern SAT solvers to solve problems involving millions of variables [11]. The progress in SAT solving has led to rise in designing algorithms such that the formulas over which SAT solvers are invoked with are easy for SAT solvers to solve [13]. While understanding the behavior of SAT solvers is a major open research question, a parameter of interest is often the size of the queries. In this regard, the past few years have witnessed design of algorithm that may trade off the number of queries for smaller sized queries.

To put this in perspective, we will survey the complexity of two different algorithms for approximate counting algorithms:

Proposition 7. *Given a CNF formula φ over n variables, tolerance $\theta > 0$ and a confidence parameter $\delta \in (0, 1]$:*

1. *there exists probabilistic approximate model counter $\mathcal{A}$ that runs in probabilistic polynomial time given access to a* SAT *oracle, makes $O(\log(n/\theta)\log(1/\delta))$ calls to* SAT *oracle and each query is of the size $O(|\varphi| + \frac{n^2}{\theta^2})$ [3, 10, 21]*
2. *probabilistic approximate model counter $\mathcal{B}$ that runs in probabilistic polynomial time given access to a* SAT *oracle, makes $O(\log(n)\frac{1}{\theta^2}\log(1/\delta))$ calls to* SAT *oracle and each query is of the size $O(|\varphi| + \frac{n}{\theta^2})$ [7, 9]*

In practice, it turns out that variants of algorithm $\mathcal{B}$ work more efficiently than those of $\mathcal{A}$ [1, 14]. The current state of the art algorithm for approximate model counting (based on empirical performance) is ApproxMC (a variant of $\mathcal{B}$) [9, 20]. We present a more nuanced analysis of the complexity of JVV approach and UniSamp in the following table (Table 2) based on the approximate model counters invoked by them. In Table 2, JVV-$\mathcal{A}$ refers to implementation of JVV with probabilistic model counter $\mathcal{A}$ stated in Proposition 7; similarly, JVV-$\mathcal{B}$ refers to JVV with probabilistic model counter $\mathcal{B}$.

The following propositions prove the results stated in Table 2.

Proposition 8. JVV-$\mathcal{A}$ *makes $O(n^2 \log n + n \log n \log \varepsilon^{-1})$ SAT queries. The size of the queries is $O(|\varphi| + n^4)$.*

Proof. From the pseudocode of JVV, the algorithm makes 2n+1 calls to ApproxCount with $\theta = 1/n$ and $\delta = \frac{\varepsilon}{(2n+1)2^{n+2}}$. Here, their procedure ApproxCount is implemented by a probabilistic approximate counter which makes $O(\log(n/\theta)\log(1/\delta)))$ calls to a SAT oracle. Substituing θ and δ by their value, one call to ApproxCount in the JVV algorithm makes $O(\log(n^2)\log((2n+1)2^{n+2}\varepsilon^{-1})) = O(\log n(\log n + (n+2) + \log(\varepsilon^{-1})) = O(n \log n + \log n \log(\varepsilon^{-1}))$ calls to a SAT oracle. As JVV makes $O(n)$ calls to ApproxCount, the total number of queries is $O(n^2 \log n + n \log n \log(\varepsilon^{-1}))$. The size of the queries is $O(|\varphi| + \frac{n^2}{\theta^2})$, substituing with $\theta = 1/n$ immediatly gives $O(|\varphi| + n^4)$. □

Proposition 9. JVV-$\mathcal{B}$ *makes $O(n^4 \log n + n^3 \log n \log \varepsilon^{-1})$ SAT queries. The size of the queries is $O(|\varphi| + n^3)$.*

Proof. Likewise, we substitute θ and δ by their value. The only difference is that the probabilistic approximate counter now makes $O(\theta^{-2} \log n \log \delta^{-1})$ SAT queries. We get that one call to ApproxCount in the JVV algorithm makes $O(n^2 \log(n) \log((2n+1)2^{n+2}\varepsilon^{-1})) = O(n^2 \log n(\log n + (n + 2) + \log(\varepsilon^{-1})) = O(n^3 \log n + n^2 \log n \log(\varepsilon^{-1}))$ calls to a SAT oracle. As JVV makes $O(n)$ calls to ApproxCount, the total number of queries is $O(n^4 \log n + n^3 \log n \log(\varepsilon^{-1}))$. The size of the queries is $O(|\varphi| + \frac{n}{\theta^2})$, substituing with $\theta = 1/n$ immediatly gives $O(|\varphi| + n^3)$. □

Proposition 10. UniSamp-$\mathcal{A}$ *makes $O(\varepsilon^{-1} + \log n \log \varepsilon^{-1})$ SAT queries. The size of the queries is $O(|\varphi| + n^2 + n\varepsilon^{-1})$.*

Proof. UniSamp makes only one call to the probabilistic approximate counter with $\theta = \sqrt{2} - 1$ and $\delta \leq \varepsilon/4$. Substituing these values, we makes $O(\log n \log \varepsilon^{-1})$ calls to a SAT oracle to get an approximate count. Then, we need to add the calls made by BSAT. BSAT makes at most thresh calls by design. As thresh $= O(pivot)$ and $pivot = O(\varepsilon^{-1})$, BSAT makes $O(\varepsilon^{-1})$ additionnals calls to the SAT oracle. Adding the calls of BSAT to the ones of ApproxCount, we get $O(\varepsilon^{-1} + \log n \log \varepsilon^{-1})$ As θ is constant, the size of the queries made by ApproxCount is $O(|\varphi| + n^2)$. For BSAT, we add a blocking clause for each witness already found. As each clause contain at most n variables and that we can find at most thresh $= O(\varepsilon^{-1})$ witness, the size of the queries made by BSAT is $O(|\varphi| + n\varepsilon^{-1})$. All in all, the queries size is $O(|\varphi| + n^2 + n\varepsilon^{-1})$.

□

Proposition 11. UniSamp-$\mathcal{B}$ *makes $O(\varepsilon^{-1} + \log n \log \varepsilon^{-1})$ SAT queries. The size of the queries is $O(|\varphi| + n^2 + n\varepsilon^{-1})$.*

Proof. As θ is constant, the number of calls to the oracle made by ApproxCount and the size of the queries do not change. As the only difference between UniSamp-$\mathcal{A}$ and UniSamp-$\mathcal{B}$ is the implementation of ApproxCount, both algorihtms have the same complexity in term of number and size of queries. □

7 Experimental results

Since uniform sampling has widespread applications, we are interested in measuring the impact of our algorithmic

Table 2. Almost-uniform generators along with the number of queries, the size of SAT Queries, and the independence of hash functions

	SAT Queries	Size	Independence
JVV-$\mathcal{A}$ [8]	$O(n^2 \log n + n \log n \log \varepsilon^{-1})$	$O(\lvert\varphi\rvert + n^4)$	2
UniSamp-$\mathcal{A}$[10]	$O(\varepsilon^{-1} + \log n \cdot \log \varepsilon^{-1})$	$O(\lvert\varphi\rvert + n^2 + n\varepsilon^{-1})$	3
JVV-$\mathcal{B}$[9]	$O(n^4 \log n + n^3 \log n \log \varepsilon^{-1})$	$O(\lvert\varphi\rvert + n^3)$	2
UniSamp-$\mathcal{B}$[11]	$O(\varepsilon^{-1} + \log n \cdot \log \varepsilon^{-1})$	$O(\lvert\varphi\rvert + n^2 + n\varepsilon^{-1})$	3

Table 3. Experimental Results

Benchmark	Variables	Clauses	JVV ($\varepsilon = 0.3$)		UniSamp ($\varepsilon = 0.3$)	
			time (sec/sample)	success rate	time (sec/sample)	success rate
s27_new_3_2	17	31	0.38	0.372	0.02	1.00
s27_new_7_4	17	35	0.38	0.372	0.02	1.00
4step	165	418	timeout	-	1.03	0.21
s420_15_7	366	994	timeout	-	1.56	0.29
GuidanceService2.sk	715	2181	timeout	-	1.00	0.23
min-6s	839	2762	timeout	-	34.18	0.20
GuidanceService.sk	988	3088	timeout	-	1.06	0.29
prod-2s	1113	4974	timeout	-	6.91	0.19
90-16-2-q	1216	1920	timeout	-	3254.91	0.31
UserServiceImpl.sk	1509	5009	timeout	-	0.72	0.29
PhaseService.sk	1686	5655	timeout	-	1.39	0.21
ProjectService3.sk	3175	11019	timeout	-	6.90	0.19
blasted_case104	3666	11589	timeout	-	319.88	0.32
blasted_squaring41	4185	13599	timeout	-	3348.58	0.55
ProcessBean.sk	4768	14458	timeout	-	5.87	0.23
doublyLinkedList.sk	6890	26918	timeout	-	6.28	0.20
01B-1	9159	39959	timeout	-	345.59	0.29
107.sk_3_90	8948	40147	timeout	-	7.51	0.32
LoginService.sk	8200	26689	timeout	-	7.13	0.19
sort.sk_8_52	12125	49611	timeout	-	11.46	0.20
enqueueSeqSK.sk	16466	58515	timeout	-	29.71	0.21
compress.sk	44901	166948	timeout	-	123.19	0.19
hash-4	188361	755882	timeout	-	315.34	0.29

framework. Given the lack of scalability of JVV, the practical implementations have to rely on algorithms without formal guarantees. Therefore, we seek to understand if our scheme can lead to practically scalable tools. To this end, we conducted a comprehensive performance evaluation of counting algorithms involving 562 benchmarks arising from wide range of application areas including probabilistic reasoning, plan recognition, DQMR networks, ISCAS89 combinatorial circuits, quantified information flow, program synthesis, functional synthesis, logistics [20]. We employ CryptoMiniSAT [19] as the underlying SAT solver given its native support for CNF-XOR formulas, i.e., formulas expressed as conjunction of CNF and XOR clauses.

The experiments were conducted on a high performance computer cluster, with each node consisting of an E5-2690 v3 CPU with 24 cores and 96GB of RAM such that each core's access was restricted to 4GB. The computational effort for the evaluation consisted of over 20,000 hours. We used timeout of 3600 (i.e., 1 hour) seconds for each experiment, which consisted of running a tool on a particular benchmark.

Since the algorithms are randomized, for an accurate measurement, we run each algorithm 100 times for every benchmark and compute the average time per sample generation and the observed success probability. The success probability is computed as the number of runs where a sample was returned divided by the total number of runs (i.e., 100). The average time per sample generation is computed as the total time taken across 100 runs divided by the number of runs that generated samples while the observed success probability is computed as the fraction of runs that output a sample (i.e, the output is not $\perp$).

We present the results on a subset of benchmarks in Table 3. The first column states the name of the benchmark, while the second and third columns state the number of variables and clauses corresponding to the benchmark. The fourth and fifth columns state the time taken per sample generation and the observed success probability for JVV while the sixth and seventh columns present the corresponding data for UniSamp. For lack of space, we present results on a subset of the benchmarks.

Table 3 clearly shows that while JVV times out on all except two instances, UniSamp is able to generate samples within *reasonable* time. Furthermore, observe that in practice, the success rate is significantly higher than our theoretical guarantees, thereby suggesting the potential for tighter analysis. Across all the benchmarks, JVV timed out on 544 out of 562 instances while UniSamp did not time on any of the 562 instances.

8 Conclusion

Uniform sampling is a fundamental problem with wide range of applications. The prior state of the art techniques, however, required $O(n^2 \log n + \log n \log \varepsilon^{-1})$ calls to SAT oracle while using 2-wise independence. In this work, we propose a new algorithm that makes only $O(\varepsilon^{-1} + \log n \log \varepsilon^{-1})$ calls to SAT oracle. The improvement in theory also leads to the development of the first almost-uniform sampling tool with rigorous guarantees and our empirical comparisons clearly demonstrate that UniSamp is able to handle instances that were beyond the ability of JVV.

References

[1] Durgesh Agrawal, Bhavishya, and Kuldeep S. Meel. On the sparsity of xors in approximate model counting. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 7 2020.

[2] M. Bellare, O. Goldreich, and E. Petrank. Uniform generation of NP-witnesses using an NP-oracle. *Information and Computation*, 163(2):510–526, 2000.

[3] Mihir Bellare and Erez Petrank. Making zero-knowledge provers efficient. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 711–722, 1992.

[4] J Lawrence Carter and Mark N Wegman. Universal classes of hash functions. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112. ACM, 1977.

[5] S. Chakraborty, D. J. Fremont, K. S. Meel, S. A. Seshia, and M. Y. Vardi. Distribution-aware sampling and weighted model counting for SAT. In *Proc. of AAAI*, pages 1722–1730, 2014.

[6] S. Chakraborty, K. S. Meel, and M. Y. Vardi. A scalable and nearly uniform generator of SAT witnesses. In *Proc. of CAV*, pages 608–623, 2013.

[7] S. Chakraborty, K. S. Meel, and M. Y. Vardi. A scalable approximate model counter. In *Proc. of CP*, pages 200–216, 2013.

[8] S. Chakraborty, K. S. Meel, and M. Y. Vardi. Balancing scalability and uniformity in SAT witness generator. In *Proc. of DAC*, pages 1–6, 2014.

[9] S. Chakraborty, K. S. Meel, and M. Y. Vardi. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls. In *Proc. of IJCAI*, 2016.

[10] M.R. Jerrum, L.G. Valiant, and V.V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43(2-3):169–188, 1986.

[11] S. Malik and L. Zhang. Boolean satisfiability from theoretical hardness to practical success. *Commun. ACM*, 52(8):76–82, 2009.

[12] George Markowsky, J Lawrence Carter, and Mark N Wegman. Analysis of a universal class of hash functions. In *International Symposium on Mathematical Foundations of Computer Science*, pages 345–354. Springer, 1978.

[13] Joao Marques-Silva. Computing with sat oracles: Past, present and future. In *Conference on Computability in Europe*, pages 264–276. Springer, 2018.

[14] S. Meel, Kuldeep S. ⓡ Akshay. Sparse hashing for scalable approximate model counting: Theory and practice. In *Proceedings of Logic in Computer science (LICS)*, 7 2020.

[15] Y. Naveh, M. Rimon, I. Jaeger, Y. Katz, M. Vinov, E. Marcus, and G. Shurek. Constraint-based random stimuli generation for hardware verification. In *Proc of IAAI*, pages 1720–1727, 2006.

[16] Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation hashing. *Journal of the ACM (JACM)*, 59(3):1–50, 2012.

[17] Palash Sashittal and Mohammed El-Kebir. Titus: Sampling and summarizing transmission trees with muti-strain infections. *bioRxiv*, 2020.

[18] J. P. Schmidt, A. Siegel, and A. Srinivasan. Chernoff-Hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8:223–250, May 1995.

[19] M. Soos, K. Nohl, and C. Castelluccia. Extending SAT Solvers to Cryptographic Problems. In *Proc. of SAT*. Springer-Verlag, 2009.

[20] Mate Soos and Kuldeep S Meel. Bird: Engineering an efficient cnf-xor sat solver and its applications to approximate model counting. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)(1 2019)*, 2019.

[21] L. Stockmeyer. The complexity of approximate counting. In *Proc. of STOC*, pages 118–126, 1983.