

On Testing of Samplers and Related Problems



Shayak Chakraborty

Indian Statistical Institute

Supervisor

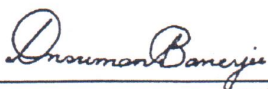
Dr. Sourav Chakraborty Dr. Ansuman Banerjee
Advanced Computing and Microelectronics Unit
Indian Statistical Institute

In partial fulfillment of the requirements for the degree of
Master of Technology in Computer Science

July 2019

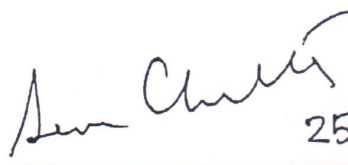
Certificate

This is to certify that the dissertation titled "On Testing of Samplers and Related Problems" submitted by **Shayak Chakraborty**, Roll No. **CS1715** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** embodies the work done by him under our supervision. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in our opinion, has reached the standard needed for submission.



25/06/2019

Dr. Ansuman Banerjee
Associate Professor,
Advanced Computing and Microelectronics
Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA



25/06/2019

Dr. Sourav Chakraborty
Associate Professor,
Advanced Computing and Microelectronics
Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA

Acknowledgements

First of all, I would like to express my sincere gratitude towards my supervisors, Dr. Sourav Chakraborty and Dr. Ansuman Banerjee, for constantly supporting me and having faith in me so that I could gain the courage to go along with this work. It is because of their patience, guidance and encouragement that I have been able to present this work.

I would also like to thank Dr. Kuldeep Meel, for his guidance, for providing me access to benchmarks and helping with the access to NSCC Singapore for the computational requirements. Further, I would like to express my sincere thanks to Dr. Rahul Sharma and Dr. Shuvendu Lahiri from Microsoft Research for their early inputs on this research direction when I started this work.

Finally, I would also want to thank my friends and most importantly my parents. It is because of their encouragement and belief in me that I have been able to come this far.

Abstract

The last decade has seen an unprecedented adoption of Artificial Intelligence and Machine Learning Algorithms in various engineering applications such as medical imaging, autonomous vehicles, web services, etc. Probabilistic Reasoning lies at the heart of such algorithms. Probabilistic reasoning techniques rely highly on sampling techniques and thus samplers that are able to generate quality samples from a discrete distribution uniformly at random, are in heavy demand. This has created a need for efficient samplers that come with some correctness and performance guarantees, and indeed, a number of proposals to this effect have been developed in literature. This has also created a need to find an efficient and scalable way for testing the reliability of a sampler with respect to the quality of the samples it generates. Unlike in the field of program testing, in which finding a single trace of execution is sufficient to prove the existence of a bug, the number of samples needed for sampler testing is not one, in fact, most of the testing mechanisms existing in literature, use exponential or sub-exponential number of samples to test for uniformity of a given sampler. The objective of this work is to propose techniques that can cut down on this complexity for certain classes of samplers. To overcome the high sample complexity for property testing of distributions, a framework for conditional sampling was proposed for checking properties of distributions, with polynomial complexity in the number of dimensions of the sample space. A recent framework named Barbarik that works on the guidelines of conditional sampling has suggested a way to test uniformity of samplers sampling from Boolean domains, using a constant number of samples. Barbarik has been proposed in the context of uniformity testing of samplers which claim to provide uniform witnesses for arbitrary Boolean formulae. The objective of this thesis is to examine ways to develop similar algorithms for testing other kind of samplers.

In the first result presented in this thesis we propose a framework to harness Barbarik to test uniformity of state-of-art samplers which are used in the

context of Horn formulae. Horn formulae is a subclass of general Boolean formulae and have been quite popular in a variety of domains, due to their simple yet powerful expressiveness, while being able to entail polynomial procedures for solving. In this thesis, we propose an efficient procedure to test samplers for Horn clause, and we demonstrate the efficacy of our testing framework with experiments on large scale benchmarks.

In the second part of this thesis, using a different form of the conditional sampling model, we investigate the problem of testing samplers in the context of uniform Perfect Matching in a Graph. The problem of finding uniform perfect matching in a graph is of importance in computer science and has a large volume of research associated with it. Modern complex networks often need to find a uniform perfect matching. The problem of finding a uniform perfect matching finds its root in physical sciences like statistical mechanics and chemistry. This has inspired research on methods for testing whether the algorithm in use generates a perfect matching uniformly at random. Our contribution in this thesis is a randomized algorithm using the framework of sub-cube conditional sampling to test samplers for uniform perfect matching.

Contents

1	Introduction	1
1.1	Importance of Samplers	1
1.2	Verification of Samplers	3
1.3	Contributions of this Thesis	4
2	Definitions and Preliminaries	6
2.1	The problem of Sampling	6
2.1.1	Uniform Sampling	7
2.1.2	Chernoff Bound	9
2.1.3	Maximum Likelihood Estimator	10
2.1.4	Conditional Sampling	10
2.1.4.1	Sub-cube Conditioning	11
2.2	Samplers for Boolean Formulae	13
2.2.1	State of the Art CNF Samplers	13
2.2.2	Horn Samplers	15
2.2.3	Chain Formula	16
2.3	Sampler Verification	17
2.4	Perfect Matching in Graphs	17
3	Testing samplers for Horn Clause	19
3.1	Introduction	19
3.1.1	Horn Sampler Verifier	20
3.1.2	Contributions of this work	21
3.1.3	Organisation of the Chapter	22
3.2	Pure Horn Chain Formula	23
3.3	Methodology of FLASH	25
3.3.1	<i>Kernel</i> Routine	28
3.3.2	<i>Encode</i> Routine	30

3.4	Analysis of FLASH	35
3.5	Evaluation and Results	36
3.5.1	Evaluation	36
3.5.2	Analysis of Horn formula Instances	37
3.5.3	Results	38
3.6	Conclusion	38
4	Testing Samplers for Perfect Matching	40
4.1	Introduction	40
4.1.1	Our Contributions	41
4.2	Preliminaries	42
4.2.1	A Uniform Perfect Matching Sampler	42
4.2.2	Sub-cube conditioning in Graphs	42
4.3	Methodology of UPMST	43
4.3.1	<i>Restrict</i> Routine	46
4.3.2	<i>BasicIDTester</i> Routine	47
4.4	Theoretical Analysis of UPMST	48
4.5	Conclusion	49
5	Conclusion	50
	References	54

List of Figures

3.1	<i>kernel</i> takes two witnesses from R_φ then creates formula φ' , such that each witness in $R_{\varphi'}$ can be mapped to some element in C , this mapping partitions $R_{\varphi'}$ into $ C $ parts of equal size	29
-----	--	----

List of Tables

3.1	Analysis of Benchmarks used for FLASH	37
3.2	The output and analysis of the number of samples consumed by FLASH .	38

List of Algorithms

1	$\text{FLASH}(A, U, \epsilon, \eta, \delta, \varphi)$	26
2	$\text{Bias}(\sigma, L, S)$	28
3	$\text{Kernel}(\varphi, \sigma_1, \sigma_2, \tau)$	29
4	$\text{Encode}(\sigma_1, \sigma_2)$	31
5	$\text{UPMST}(A, U, G, \eta, \delta)$	45
6	$\text{Restrict}(G, m, i, S)$	46
7	$\text{BasicIDTester}(A, U, G, e, \eta, \delta)$	47

Chapter 1

Introduction

Sampling techniques form the core backbone of probabilistic reasoning and state of art inferencing techniques. Sampling has widespread usage not only in computer science and statistics, but it also has wide applications in statistical mechanics, chemistry, particle physics, etc. A sampler is an implementation of an ensemble of algorithms that generates samples from a sample space using some underlying distribution. Samplers with strong theoretical guarantees often do not scale well to suit practical needs. Most of the modern samplers in use, therefore, rely on heuristic approaches to scale to practical needs. The main motivation of this thesis is to look for approaches for possible testing methodologies to test whether a sampler under test generates samples close to a given distribution. In particular, our main objective is to develop novel testing methodologies for different kinds of samplers with particular focus on testing for uniformity of samplers.

1.1 Importance of Samplers

Sampling techniques find widespread usage in computer science, statistics, statistical mechanics, chemistry, etc. We outline a few of them in this section.

Sampling and AI: The last decade has seen adoption of Artificial Intelligence and Machine Learning based algorithms in safety critical systems like aircraft controllers, etc. and various other engineering applications such as medical imaging, autonomous vehicles, web services, etc. Such unprecedented proliferation owes its credit to the ability of such algorithms to match and at times surpass human abilities for complex tasks. At the core of such applications lie probabilistic reasoning. Sampling forms a key component of probabilistic reasoning, in particular, uniform sample generation has found varied usage

in probabilistic reasoning [1] and a variety of other disciplines.

Sampling and Statistical Inference: Statistical Inference is about making propositions about population using the data drawn from the population using sampling. Given a hypothesis about a given population, statistical inference consists of drawing data from the population using sampling and then analyzing the samples to draw conclusions. In order to prove or disprove a hypothesis made about the population, some confidence about the quality of the samples is essential. An unbiased uniform sample from the population is often regarded the best for drawing conclusions. This use of sampling thus requires efficient and quality samples from a distribution over the population.

Sampling and estimation: Sampling is used for various parameter estimations in the field of geostatistics, biostatistics, forecasting, signal processing etc. Here a small available sample is analyzed to estimate some parameter for the entire population. If the sample is biased or not of expected quality, the estimations become inappropriate. In some cases poor quality of samples could distort the estimation which could in turn affect the values significantly. Thus estimation requires very high quality of samples.

In addition to the above, sampling has also found widespread usage in network reliability[2], constrained random verification[3], program analysis and many other engineering and non-engineering fields.

Despite such widespread usage, it is worth noting that sampling is computationally intensive problem. Probabilistic reasoning has been shown to be $\#P$ -complete for variables with finite domains [1]. Also model counting is another problem that has been shown to be $\#P$ -complete by Valiant in [4]. Sampling is equivalent to approximate counting, since both the problems self reduce to each other. Most of the techniques used for sampling are approximations over real-world instances. Most of these techniques are based on Monte Carlo Markov Chains (MCMC) and variational approximation. But these techniques fail to scale to large real-world instances and fail to provide rigorous approximation guarantees in practice. To cope with these problems, most of the deployed samplers use heuristic based approaches. Unfortunately, heuristic based approaches do not usually lend themselves to easy theoretical guarantees. This motivates the need to test samplers for uniformity. Unlike in the field of property testing where finding a single bug is sufficient to prove the existence of a bug in a program under test, for a sampler this is not the case where the number of samples needed to test for uniformity is generally exponential

in the sample space. Before putting any sampler for any class of problem to use, it is essential to test its performance in order to ensure reliability. In particular, the samplers need to be verified with respect to their ability to generate uniform samples. This is the central motivation of this thesis, as discussed in the various chapters. We discuss more about this below.

1.2 Verification of Samplers

Since sampling techniques are of such utmost importance, researchers have investigated several practical and scalable approaches to design samplers. With such large volume of research, significant effort has also been put in for the evaluation of the proposed techniques. In most of the published articles, to test the quality of the distribution generated by a sampler, various versions of the classical χ^2 -test are applied. The tests involve studying various parameters of the frequency vector of the output of the sampler when run on benchmark data. Some of the published work look into the number of unique samples that are generated from various samplers to argue about the quality of the distribution [5], [6]. They use a variation of the standard χ^2 -test in their approach. Unfortunately, it has been proved in [7] that any test that uses only random samples from the distribution and would with probability at least $(1 - \delta)$ accept the distribution if it is ϵ -close to uniform and with probability at most δ reject if the distribution is η -far from uniform would need $\Omega(\sqrt{R_\varphi} \log(\delta^{-1}) / (\eta - \epsilon)^2)$ samples, where $0 < \delta < 1$. Recently, a new model called *conditional sampling* has been proposed in [8], [9]. The authors showed with access to *conditional sampling* from a distribution, the number of samples required to check for ϵ -close or η -far from uniform is $O(\log(\delta^{-1}) / (\eta - \epsilon)^2)$ which is independent of the dimension or support of the distribution.

In a recent work, authors of [10] came up with an algorithmic framework named Barbarik using the guidelines of conditional sampling to test whether the underlying distribution of the sampler is ϵ -close or η -far from the uniform distribution. Barbarik requires only $\tilde{O}(1/(\eta - \epsilon)^4)^1$ samples, which is a constant with respect to the size of the sample space.

On similar grounds of conditional sampling, authors of [11] introduced a generic property testing framework for joint distributions. They used the sub-cube conditioning model in order to test whether a sampler on a product domain is indeed uniform or η -far

¹ \tilde{O} is an extension of the standard big- O notation, ignoring all poly logarithmic factors

from uniform using $\tilde{O}(n^2/\eta^2)$ samples, where n is the dimension of the space over which the distribution is defined. This framework presents an useful adaptive mechanism for testing samplers for uniformity in various contexts like perfect matching, cryptography, data analytics, etc.

1.3 Contributions of this Thesis

The contributions of this thesis are summarized as follows.

- As our first contribution, we present a probabilistic algorithm for testing whether a Horn clause samplers is ϵ -close to uniform or η -far from uniform ¹. We call our testing mechanism, **Flash**. The framework for **Flash** adapts it's methodology from the Barbarik framework introduced in [10]. Since Barbarik was designed for unrestricted Conjunctive Normal Form (CNF) formulae, adapting it to suit Horn formulae is not straight forward. Horn being a restricted sub-class of CNF doesn't lend itself to the general model that Barbarik deals with. Even a trivial task of encoding in Barbarik needs to be remodeled to make it work for Horn formulae. This is the novelty of our first contribution, as outlined in Chapter 3. The testing methodology that we propose uses only $\tilde{O}(1/(\eta - \epsilon)^4)$ samples, which is constant with respect to the size of the sample space. In this chapter, we discuss the working methodology of FLASH, the theoretical guarantees that it presents and the run-time efficacy of our approach on state-of-the-art samplers on standard Horn benchmarks.
- Further, we present a probabilistic algorithm for testing whether a sampler for perfect matching in graphs samples according to the uniform distribution or whether the underlying distribution is η -far from uniform. We call it the **Uniform Perfect Matching Sampler Test** (UPMST). It may be noted that the witness based conditional sampling method suggested in Barbarik does not readily fit in the context of graphs. We therefore design a different novel strategy for conditional sampling. We use the model of sub-cube conditional sampling and adapt the test for identity of joint distributions introduced in [11] to derive an algorithm that tests uniform samplers for perfect matching. The testing methodology that we propose uses $\tilde{O}(|E|^2/\eta^2)$ samples, where E is the edge set of a given graph $G = (V, E)$. We also derive the theoretical guarantees that this methodology provides.

¹where the ϵ -close and η -far refers to the ℓ_1 or variation distance of the underlying distribution of the sampler from uniform

The rest of this thesis is organized into 4 chapters. In the following chapter, we introduce some necessary background concepts and discuss some relevant literature. Chapter 3 outlines our proposal on sampler verifier for Horn formulae, while the following discusses about techniques for testing uniform perfect matching. Chapter 5 concludes this discussion.

Chapter 2

Definitions and Preliminaries

In this chapter, we present an overview of some background concepts that are relevant to the contributions of this thesis. We begin with a discussion on the general concept of sampling, and move on to describe relevant work on samplers and their verifiers. Further, we present an overview of the conditioning method that forms the backbone of Chapter 4. We also discuss briefly the concept of matching in graphs.

2.1 The problem of Sampling

In probability theory, we define a sample space as the set of values a random variable can take. Intuitively, a sample space consists of all possible outcomes of any experiment. A sampler is responsible for generating samples from the sample space over the distribution the sampler follows. For us, a sampler is a generator of samples from a sample space using some underlying distribution. In this thesis, we use the terms sampler and generator interchangeably.

In many problem domains, it may so happen that the sample space is not initially defined to the sampler, and we need to indicate the sample space before the sampler is ready to generate samples. For example for a SAT sampler, the sample space, denoted by R_φ , is different for every other Boolean formula φ . We first specify the formula to the sampler before requesting for satisfying assignments to φ . Another example is sampling for perfect matching of a given graph $G = (V, E)$. A perfect matching sampler on receiving as input a graph G , outputs a perfect matching M of G using some distribution from the set of all perfect matchings of G , denoted by Γ_G . Formally a sampler is defined as follows.

Definition 2.1.1. *Given a problem instance I and an underlying distribution D_{Ω_I} , where*

Ω_I is the sample space associated with I , a **sampler** S is an algorithm that generates elements from Ω_I using the distribution D_{Ω_I} , i.e.

$$\forall x \in \Omega_I, Pr[S(I) = x] = Pr_{D(\Omega_I)}[x],$$

In general, as mentioned in the previous chapter, since sampling itself is a difficult problem, we focus mainly on uniform random sampling. To us, a sampler is like a blackbox access, which on request generates a given number of samples from the sample space. Since most of the times the sample space is too large for enumeration, samplers use varied heuristics to generate samples, quality of which do not often have theoretical guarantees. Thus it is necessary to verify the quality of the samples generated with respect to the underlying distribution which these samplers follow.

2.1.1 Uniform Sampling

Given an instance I , a *probabilistic sampler* G is an algorithm that outputs a random sample from Ω_I . Let $D_{G(I)}$ denote the underlying probability distribution induced by G over the set Ω_I , i.e. $D_{G(I)}$ assigns a probability to each element of Ω_I , that denotes the probability with which $G(I)$ returns $x \in \Omega_I$. Let $Q \subseteq \Omega_I$, $D_{G(I)}|Q$ be the probability distribution $D_{G(I)}$ conditioned over the set Q . When conditioned on the subset Q , $G(I)$ returns only samples from Q and $D_{G(I)}|Q$ dictates the probability with which $G(I)$ returns each element in Q .

Definition 2.1.2. *Given an instance I , a uniform generator $G^u(I)$ is a probabilistic generator that guarantees*

$$\forall y \in \Omega_I, Pr[G^u(I) = y] = \frac{1}{|\Omega_I|}$$

The following example explains this in greater detail.

Example 2.1.1. Uniform Sampling of witnesses for Boolean formula: *Consider the Boolean formula $\varphi = (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3)$, in Conjunctive Normal Form (CNF), defined over the variables x_1, x_2 and x_3 . Witnesses of φ are 000, 001, 010, 011, 101, where 0 refers to false and 1 refers to true, 000 assignment means x_1 is assigned false, x_2 is assigned false, x_3 is assigned false, similarly, 001 assignment means x_1 is assigned false, x_2 is assigned false, x_3 is assigned true and so on. If we call a uniform sampler A with the formula φ , it must return one of the five witnesses each with probability $1/5$. In other words, for a uniform sampler, we should have:*

$$Pr[A(\varphi) = 000] = Pr[A(\varphi) = 001] = Pr[A(\varphi) = 010] = Pr[A(\varphi) = 011] = Pr[A(\varphi) = 101] = \frac{1}{5}$$

From this example, we can conclude that if we ask a uniform sampler A for 100 random samples for φ , A should return 20 samples corresponding to each witness of φ .

Since it is almost practically impossible to generate uniform samples, some variants of uniform sampling have been proposed. These variants relax the exact uniform sampling guarantee by a constant factor. We now define a few more important concepts.

Definition 2.1.3. Given a problem instance I and a tolerance parameter ϵ , $G_{\text{aaU}}(I, \epsilon)$ is an ϵ -Additive Almost Uniform generator if:

$$\forall y \in \Omega_I, \frac{1-\epsilon}{|\Omega_I|} \leq \Pr[G^u(I, \epsilon) = y] \leq \frac{1+\epsilon}{|\Omega_I|}$$

Intuitively an additive almost-uniform generator is a relaxation on the uniform generator by a factor of $\frac{\epsilon}{|\Omega_I|}$.

Example 2.1.2. Let $\varphi = (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3)$ be a CNF formula, G be an additive almost generator and ϵ be 0.2. $R_\varphi = \{000, 001, 010, 011, 101\}$, $\frac{1-\epsilon}{|R_\varphi|} = 0.16$ and $\frac{1+\epsilon}{|R_\varphi|} = 0.24$. Then for each element in $y \in R_\varphi$, the probability with which $G(\varphi, \epsilon)$ generates y is in the range $[0.16, 0.24]$. \square

From this example, we can conclude that if we request for 100 samples to G for the formula φ , the expected number of times each element in R_φ appears is within 16 to 24.

Definition 2.1.4. A generator $G^{\text{maU}}(.,.)$ is a Multiplicative Almost Uniform generator (MAU) if the following holds:

$$\forall y \in \Omega_I, \frac{1}{(1+\epsilon)|\Omega_I|} \leq \Pr[G^{\text{maU}}(I, \epsilon) = y] \leq \frac{1+\epsilon}{|\Omega_I|}$$

where $\epsilon > 0$ is the tolerance parameter.

Lemma 2.1.1. Every Multiplicative Almost Uniform (MAU) generator is an Additive Almost Uniform generator (AAU).

Proof. Let M be a multiplicative almost-uniform generator. So, for any given instance I , the following holds,

$$\forall y \in \Omega_I, \frac{1}{(1+\epsilon)|\Omega_I|} \leq \Pr[M(I, \epsilon) = y] \leq \frac{1+\epsilon}{|\Omega_I|}$$

Suppose M is not an additive almost-uniform generator. The upper bound on the probability is same as that of any AAU. Thus M must violate the lower bound, i.e. $\exists y \in \Omega_I, \Pr[M(I, \epsilon) = y] < \frac{1-\epsilon}{|\Omega_I|}$. But $\forall y \in \Omega_I, \Pr[M(I, \epsilon) = y] > \frac{1}{(1+\epsilon)|\Omega_I|}$ and

$\frac{1}{(1+\epsilon)|\Omega_I|} > \frac{1-\epsilon}{|\Omega_I|}$, which is a contradiction. Therefore our assumption that M is not an AAU is false. This proves that every multiplicative almost-uniform generator is an additive almost-uniform generator. ■

It may be noted that every AAU need not be a MAU.

Definition 2.1.5. A near-uniform generator $G^{nu}(\cdot)$ is one that further relaxes the guarantee of uniformity and ensures that

$$Pr[G^{nu}(I) = y] \geq \frac{c}{|\Omega_I|}$$

for a constant c , where $0 \leq c \leq 1$.

In practice, we are rarely provided with the exact probabilities with which a sampler samples. In situations where we need to estimate the probabilities with a certain guarantee, the *Maximum Likelihood function* is a good way. In order to determine the number of samples needed to estimate the unknown probability with a certain probabilistic guarantee, we use *Chernoff bounds*. In the next subsection we explain the notion of *Chernoff bounds* and in the following subsection we explain how parameter estimation is done using *Maximum Likelihood estimation*.

2.1.2 Chernoff Bound

Throughout the years there have been different variations of the Chernoff Bound, with slightly different assumptions. We will start with a simple case of the sum of independent Bernoulli's trials, i.e. where each random variable takes 0 with probability p and 1 with probability $(1 - p)$. For example, this corresponds to the case of tossing unfair coins with each having it's own probability of head and counting the total number of heads appearing.

Theorem 2.1.2. (*Chernoff Bounds*) Let $X = \sum_{i=1}^n X_i$, where $X_i = 1$ with probability p_i and $X_i = 0$ with probability $(1-p_i)$, and all X_i s are independent. Let $\mu = E[X] = \sum_{i=1}^n p_i$. Then

- (i) **Upper bound** : $P[X \geq (1 + \delta)\mu] \leq e^{-\frac{\delta^2}{2+\delta}\mu}$ for all $\delta > 0$
- (ii) **Lower bound** : $P[X \leq (1 - \delta)\mu] \leq e^{-\frac{\delta\mu}{2}}$ for all $0 < \delta < 1$

For $\delta \in (0, 1)$, we can combine the upper and lower bounds in the above theorem and obtain the following simple two sided bound:

Corollary 2.1.2.1. Given $X_1, X_2, X_3, \dots, X_n$, where $X_i = 1$ with probability p_i and $X_i = 0$ with probability $(1 - p_i)$, $X = \sum_{i=1}^n X_i$ and $\mu = E[X]$,
 $P[|X - \mu| > \delta\mu] \leq 2e^{-\frac{\mu\delta^2}{3}}$, for all $0 < \delta < 1$.

2.1.3 Maximum Likelihood Estimator

A maximum likelihood estimator (MLE) is a value of a parameter a such that the *likelihood function* for a is maximum. A likelihood function $L(a)$ is the probability or probability density for the occurrence of a sample configuration x_1, \dots, x_n given that the probability density $f(x, a)$ with parameter a is known, $L(a) = f(x_1; a) \dots f(x_n; a)$ [12]. It is often used for determining the bias of a biased coin. In order to determine the bias of a biased coin using MLE, we simply toss the coin n times and output n_1/n as the bias or probability of getting a HEAD, where n_1 is the number of times HEAD appeared in the n tosses. A simple concentration bound like Chernoff bound gives the number of tosses needed to bound the probability of error to an additive constant, typically if the bias probability is p , and the coin is tossed n times, then with probability $e^{O(\gamma^2 n)}$, the estimate $\frac{n_1}{n}$ is within an additive error of γ , i.e. $P[|p - \frac{n_1}{n}| \leq \gamma] \geq e^{-\gamma^2 n/2}$. Let $M(\gamma)$ be the number of times a biased coin has to be tossed to estimate the bias to upto $\pm\gamma$ with probability $\geq 1/2$. Note that if the number of tosses is $M(\gamma) \times m$, then the probability that the additive error is less than equal to γ is greater than equal to $(1 - (1/2)^m)$.

In modern application domains, the general sample complexity for most problems is exponential in the dimension of the sample space. To address the issue of scalability, [8] and [9] proposed a different model called conditional sampling. We now discuss the notion of conditional sampling.

2.1.4 Conditional Sampling

Conditional sampling model suggests that the tester be allowed to sample from a conditioned set over the original domain using the same distribution. This means is that if the distribution μ is over the domain Σ , the tester could use any subset $S \subseteq \Sigma$ and sample any $i \in S$ with probability $\frac{\mu(i)}{\sum_{j \in S} \mu(j)}$, where $\mu(i)$ is the probability of i occurring in the sample. The sample complexity depends on the nature of the conditioned set. When there is no restriction on the conditioned set, a tester could take any conditioned set over the domain. The sampling complexity is as low as $\tilde{O}(\frac{1}{\eta^2})$ when the conditioning is over arbitrary subsets of size 2. When the conditioned set is structured and restricted to intervals, the lower bound for sample complexity was proved by authors of [13] to be

$\Omega(\frac{\log n}{\log \log n})^1$. One interesting case and the one important to us is when the domain is a Cartesian product of a set and we are allowed to condition on the Cartesian product of subsets rather than arbitrary subsets of the domain. This is called sub-cube conditioning [11], as explained in the following.

2.1.4.1 Sub-cube Conditioning

Informally, the sub-cube model can be explained in the following manner: Let the distribution μ be defined over the domain Σ^n . A sub-cube conditioning Oracle takes as input $A_1, A_2, \dots, A_n \subseteq \Sigma$ and constructs the conditioning sub-set as $S = A_1 \times A_2 \times A_3 \dots \times A_n$, where $S \subseteq \Sigma^n$. The Oracle returns a vector $x = (x_1, x_2, \dots, x_n)$ where each $x_i \in A_i$, with probability $\frac{\mu(x)}{\sum_{w \in S} \mu(w)}$. These kinds of samples are referred to as sub-cube conditioned samples and the corresponding sample complexity is referred to as the sub-cube sample complexity. It is worth noting that there is no restriction on the way we choose the individual A_i 's. The simplest case is when μ is a product of n independent marginal distributions and we wish to test uniformity. Using any tester over Σ , to check if μ_i is far from uniform needs only $\text{poly}(n)$ samples. If μ is a product distribution over $\{0, 1\}^n$, it has been shown in [13] that uniformity and identity testing can be done using $O(\sqrt{n}/\eta^2)$ unconditional samples, since marginals for independent sub-cube conditional sampling is same as unconditional sampling followed by projection. Thus we do not get any added advantage using sub-cube conditioning. But if the marginals μ_i are not independent, then it may be very well be possible that despite the marginals being uniform, the product distribution μ is still η -far. As shown in [13], any algorithm in this case requires $O(e^n)$ samples. To bypass this barrier, authors of [11] define the notion of *conditional distance*, that shows that there exists at least one $i \in [n]^2$ such that the *conditional distance* of the i^{th} marginal from uniform is more than $\eta/\text{poly}(n)$. In the discussion below, we define the notations and the notion of sub-cube conditioning along with the notion of conditional distance.

Let x be a vector of length n . The i^{th} element of the vector x can be denoted by x^i . $x^{(i)}$ denotes the sub-string of the first i elements of x , i.e. $x^{(i)} = (x_1, x_2, \dots, x_i)$. The n^{th} harmonic number is denoted by $H(n)$. Consider μ as a distribution over Σ^n with marginals $\mu_1, \mu_2, \dots, \mu_n$. In case the marginals are independent (i.e. μ is a product distribution), then we would write $\mu = \mu_1 \otimes \mu_2 \otimes \mu_3 \otimes \dots \otimes \mu_n$.

¹Here n is the number of dimensions over which the set Σ is defined.

²Here $[n]$ refers to the set $\{1, 2, 3, \dots, n\}$.

Definition 2.1.6. Total Variation Distance: Let μ, μ' be two distributions over Ω . The variation distance between μ and μ' denoted by $d(\mu, \mu')$ is defined as

$$d(\mu, \mu') = \frac{1}{2} \sum_{x \in \Omega} |Pr_{\mu}[x] - Pr_{\mu'}[x]|$$

We say μ is η -far if $d(\mu, \mu') \geq \eta$.

If μ is a distribution over Ω and $S \subseteq \Omega$, then $\mu|S$ denotes the distribution over S . For any $x \in S$, the probability that x is in a random sample drawn from S according to the distribution $\mu|S$ is given by

$$Pr_{\mu|S}[x] = \frac{Pr_{\mu}[x]}{\sum_{y \in S} Pr_{\mu}[y]}.$$

Definition 2.1.7. Conditional Distance : Let μ, μ' be two distributions over Ω . Let $S \subseteq \Omega$. The conditional distance between μ and μ' conditioned on S denoted by $d(\mu, \mu'|S)$ is defined as

$$d(\mu, \mu'|S) = \frac{1}{2} \sum_{x \in \Omega} |Pr_{\mu|S}[x] - Pr_{\mu'|S}[x]|$$

We say μ is η -far from μ' conditioned on the set S if $d(\mu, \mu'|S) > \eta$.

Sub-cube Conditional Model : Let μ be a distribution over Σ^n . A sub-cube conditional oracle, denoted by $SUBCOND_{\mu}$ accepts as input a collection of sets $\{A_i\}_{i \in [n]}, \forall A_i \subseteq \Sigma$. For $A = A_1 \times A_2 \times \dots \times A_n$, the oracle returns $x \in A$ with probability $\frac{Pr_{\mu}[x]}{\sum_{y \in A} Pr_{\mu}[y]}$ independent of all previous calls to the oracle.

An (η, δ) - $SUBCOND$ tester for any property P with conditional sample complexity τ is a randomized algorithm operating in the following fashion.

- It receives $0 < \eta, \delta < 1, n \in \mathbb{N}$ and makes an access to $SUBCOND_{\mu}$.
- In every iteration, it generates $A = A_1 \times A_2 \times \dots \times A_n$ randomly and draws an x using $SUBCOND_{\mu}$ with conditioning on A .
- Based on the sample received, it either *ACCEPTs* or *REJECTs* the distribution μ .

The algorithm makes at most τ calls to $SUBCOND_{\mu}$ where τ depends on η, δ, n and Σ . If μ satisfies the property P , then the tester must accept with probability at least $1 - \delta$ and similarly if μ is η -far, then the tester must reject with probability at least $1 - \delta$.

2.2 Samplers for Boolean Formulae

A Boolean formula is an expression that evaluates to either *True* or *False*. Every variable in a Boolean Formula is restricted to take only two values *True* and *False*. We restrict our focus to Boolean formulae expressed in CNF (Conjunctive Normal Form), where a CNF formula is a conjunction of Clauses, where a clause is a disjunction of literals. By a literal we mean either a variable or it's negation, where the positive literal is the variable itself and a negative literal is a negation of the variable. For the class of CNF formula it has been proved that the decision problem of whether there exists a satisfiable assignment to a given formula is *NP*-Complete[14]. In contrast to satisfiability of a CNF formula where we output one satisfying assignment of variables, the counting problem corresponds to counting the number of witnesses of a Boolean formula. Consider the following.

$$\varphi = (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3)$$

Witnesses of φ (minterm representation) are 000, 001, 010, 011, 101, thus the number of witnesses is 5. An algorithm that counts the number of witnesses should output 5 when it receives input φ . Counting the number of witnesses for a CNF formula is $\#P$ -complete [4]. Informally $\#P$ is the class of problems where we compute a function $f(x)$, where f represents the number of accepting paths of a non-deterministic Turing machine running in polynomial time. Counting of the number of witnesses for CNF is $\#P$ -complete. Sampling of witnesses uniformly at random is of very high complexity due to the use of Markov Chain Monte Carlo (MCMC) based techniques. So most of the samplers available for CNF formulae rely on heuristics in order to sample. Throughout this thesis for a given Boolean formula φ , we denote the set of all witnesses by R_φ .

2.2.1 State of the Art CNF Samplers

We now describe some of the state-of-art CNF samplers. We also explain in brief the methodologies adapted by them to generate samples, and the associated theoretical guarantees provided by them.

- Satisfying Perfectly Uniform Random Sampler (SPUR) [15] is built on top of the SharpSAT model counter [16]. SPUR is almost hundred times faster than any other uniform sampler and primarily uses Caching of satisfying assignments along with reservoir sampling to generate uniform assignments to CNF formulas. SharpSAT is an exact model counter, it uses caching of residual formulas, which is analogous to the CDCL algorithm for SAT solving. The caching of residual formula significantly

prunes the recursion tree searching for a satisfying assignment. Reservoir sampling [17] is then performed on the set of leaves encountered by SharpSAT during its normal course of execution. Since probability of a leaf being selected is proportional to its model count, SPUR generates uniform samples.

- UniGen2 [18] is built on the guidelines of UniGen [19] but with parallel scalability in order to adapt to industrial requirements. It has two primary phases, the first phase is a one-time pre-processing phase, *Estimate Parameter* for parameter estimation, while the second phase is the sample generation step, named *Generate Samples*. Both the routines employ a family of random hash functions. UniGen2 primarily employs the random family of hash functions to partition the solution space into uniform equivalence classes. In order to generate samples, a sat solver is invoked to generate samples from each equivalence class. If more samples are required, multiple calls to the *Generate Samples* routine is made. UniGen2 is a probabilistic algorithm and thus is sometimes allowed to fail with some probability as is the case with most of the probabilistic algorithms. UniGen2 is a multiplicative almost-uniform generator(MAU) as defined earlier, and thus provides strong theoretical guarantees. UniGen2 was built on top of [20], because of its ability to handle XOR-CNF clauses efficiently.
- QuickSampler [6] is probably the fastest SAT sampler which generates varied samples from the witness space. It is built on top of the SMT solver Z3 for solving the MAX-SAT query strategy it employs. Not all samples generated by QuickSampler are guaranteed to be valid although experimentally it shows almost 75% of the samples generated are valid. In order to validate a generated sample, a Z3 interface is provided which returns the set of valid samples. The key idea behind QuickSampler is to make minimum number of solver calls to generate a large number of potential witnesses to a given CNF formula. The core algorithm assumes access to a random solution and computes atomic mutations on the variables in order to generate potential solutions. It tries to generate a different solution corresponding to each variable in the support of the formula such that it is different from the initial random sample it was provided. Sometimes such a solution may not exist and thus all samples generated by QuickSampler are not valid. QuickSampler is a near uniform generator as defined earlier.
- Search Tree Sampler (STS): [21] Like all the other state-of-art samplers, STS also uses a SAT solver at the back end but this solver is used as a blackbox sampler to generate samples. It uses a simple recursive strategy on the recursion tree to gener-

ate samples. The recursive method is based on a notion defined as pseudosolutions, which is a partial assignment to the variables at the i^{th} level of the search tree. The idea is to divide the search tree into some L number of levels, where L is a parameter of the sampler and recursively sample pseudosolutions using previously generated pseudosolutions. The algorithm assumes access to generated pseudosolutions at level i to generate pseudosolutions at level $i + l$ using a blackbox SAT solver. Generating samples from pseudosolutions of level n where the support of the formula is n in size corresponds to the problem of sampling from all witnesses of the formula which is intractable. STS provides an uniformity guarantee for 2-CNF formulae, although for the general CNF setting the guarantees are weaker than a near uniform generator.

2.2.2 Horn Samplers

A CNF clause is a Horn Clause if it contains no more than one positive literal. A CNF formula where all clauses are Horn Clauses is referred to as a Horn Formula.

The following is an example of a Horn formula.

$$\varphi = (x_1 \vee \neg x_2) \wedge (\neg x_3 \vee \neg x_4)$$

In the clause $(x_1 \vee \neg x_2)$, there is only one positive literal i.e. x_1 , while the clause $(\neg x_3 \vee \neg x_4)$ has no positive literals. Thus, both the clauses are Horn Clauses and the corresponding formula φ is a Horn formula.

Horn formulae constitute a restricted class of CNF. While Boolean satisfiability is NP-complete in general, satisfiability for Horn is in P. However, the problem of counting the number of witnesses of a given Horn formula is still $\#P$ -complete [22], making the problem of counting witnesses computationally intractable for Horn formulae as well. Approximate counting techniques are thus applied for model counting for Horn formula as well. Although there has not been much progress in Horn specific counting or sampling, tools developed for CNF can be applied to Horn formulae, since these are restricted versions of CNF. Some of the popular tools that are used to sample witnesses for CNF formulae and therefore, can be used for Horn formulae as well include CryptoMiniSat5[23], UniGen2[18], QuickSampler[6]. However, none of these tools, to the best of our knowledge, use the special structure of Horn Clauses to sample witnesses. Horn clauses appear in a lot of domains in the formal methods community such as program analysis, network reliability analysis, business rules equivalence checking, etc. This motivates a need for

efficient samplers for Horn Clauses.

For most known classes of Boolean formulae, counting or estimating the number of solutions is difficult. Intuitively, given a general CNF formula, it is not straight forward to count the number of witnesses, in most of the cases we need to enumerate all assignments to find the number of witnesses. In order to overcome this issue, a special class of Boolean formula was introduced where counting the number of witnesses is straight forward. [24] introduced the notion of a *Chain Formula* for which it is easy to count the number of witnesses, as explained in the following subsection.

2.2.3 Chain Formula

A chain formula is a special class of Boolean formula defined inductively as follows.

- A literal l itself is a chain formula.
- If l is a literal and φ a chain formula such that l or $\neg l$ do not appear in φ , then $(l \vee \varphi)$ and $(l \wedge \varphi)$ are both chain formulae.
- Let $m > 0$ be a natural number and $k < 2^m$ be a positive odd number. Let $c_1c_2\dots c_m$ be the m -bit representation of k , where c_m is the LSB. For every $j \in \{1, \dots, m-1\}$, if $c_j = 1$ then C_j is " \vee " else if $c_j = 0$ then C_j is " \wedge ". The chain formula $\psi_{k,m}$ is defined as follows:

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

where a_1, a_2, \dots, a_m are variables.

Example 2.2.1. Let $m = 3$ and $k = 5$, then the 3-bit binary representation of k is 101 and thus $\psi_{5,3}(a_1, a_2, a_3) = a_1 \vee (a_2 \wedge a_3)$. \square

Lemma 2.2.1. Let $m > 0$ be a natural number, $k < 2^m$, and $\psi_{k,m}$ as defined above. $|\psi_{k,m}|$ is linear and has exactly k satisfying assignments [24].

Lemma 2.2.2. [24] Let $m > 0$ be a natural number, $k < 2^m$, and $\psi_{k,m}$ as defined above. Then $\psi_{k,m}$ can be converted to an equivalent CNF formula or DNF formula with m variables and at most m clauses.

2.3 Sampler Verification

Definition 2.3.1. Given an instance I and an intolerance parameter η a generator $G(I, \cdot)$ is η -far from the uniform generator if the distance of $D_{G(I)}$ from uniform is at least η . That is,

$$\sum_{x \in \Omega_I} |p_{G(I,x)} - \frac{1}{|\Omega_I|}| \geq \eta$$

A sampler verifier is an algorithm which tests the quality of the underlying distribution generated by a sampler. The notion of a sampler verifier can vary with the type of distribution that is been tested for. The following definition formally defines the notion of a sampler verifier for an Additive Almost Uniform distribution[10].

Definition 2.3.2. [10] Given an instance I , a sampler G , tolerance parameter ϵ , intolerance parameter η , guarantee parameter δ , a **sampler verifier** $T(\cdot, \cdot, \cdot, \cdot)$ returns *ACCEPT* or *REJECT* (with a witness) with the following guarantees:

- If the sampler $G(I, \epsilon)$ is an Additive Almost Uniform generator (AAU), then $T(G, I, \epsilon, \eta)$ returns *ACCEPT* with probability at least $1 - \delta$.
- If the sampler $G(I, \epsilon)$ is η -far from a uniform generator, $T(G, I, \epsilon, \eta)$ returns *REJECT* with probability at least $1 - \delta$.

Barbarik as a sampler verifier [10] is a framework that can test a sampler for the AAU property. Barbarik uses conditional sampling as the backbone of it's working method.

2.4 Perfect Matching in Graphs

Given a graph $G = (V, E)$, a matching M in G is a set of edges, such that no two edges share the same vertex. In other words, matching of a graph is a subgraph where each node of the subgraph has either zero or one edge incident to it.

Definition 2.4.1. A matching M of a graph G is called a **perfect matching** of G if every vertex of G is incident on exactly one edge in M .

A perfect matching is therefore a matching with exactly $n/2$ edges. This means only graphs with even number of vertices are candidates where perfect matching could exist. Every perfect matching is also a maximum matching in a graph. Perfect matching is also referred to as 1-factor or a complete matching. There are algorithms to find perfect matching in graphs in polynomial time. Thus the decision problem of checking if a perfect

matching exists in a given graph is in P .

The number of matchings in a graph is known as the Hosoya index of the graph. It is $\#P$ -complete to compute this quantity, even for bipartite graphs. It is also $\#P$ -complete to count perfect matchings, even in bipartite graphs, because computing the permanent of an arbitrary 01 matrix (another $\#P$ -complete problem) is the same as computing the number of perfect matchings in the bipartite graph having the given matrix as its biadjacency matrix. Evaluation of permanents has attracted researchers for long time, but without much progress. This lack of progress was explained in [4], where the author proved that computation of the permanent of a matrix is $\#P$ -complete and hence not possible in polynomial time. Thus, computation of the number of perfect matchings is also $\#P$ -complete and is intractable. The best that could be done is with a *fully polynomial randomized approximation scheme* (FPRAS), which provides an arbitrarily close approximation where the complexity depends polynomially on the input size and the desired error. Out of the several approaches proposed over the years, the one proposed in [25] stands out. This approach is based on the Markov Chain Monte Carlo (MCMC) technique. This gives an algorithm for sampling of perfect matching from bipartite graphs almost uniformly at random.

Perfect matchings are relevant in several areas like computer networks, job scheduling, statistical mechanics, chemistry, etc. Since most methods are based on MCMC, heuristic based approaches are deployed for almost all practical purposes where sampling of perfect matching is needed.

Chapter 3

Testing samplers for Horn Clause

3.1 Introduction

Horn clauses have played the basic role in constructive logic and computational logic. Because of their simple yet powerful structure of Horn clause, they become a very natural choice for such widespread usage. The use of Horn clause become advantageous from the perspective of Computational Complexity. The decision problem of HORNSAT, that is, whether a given Horn formula is satisfiable or not is P -complete and solvable in linear time. Horn clause is one of the Schaefer's classes in Computational Complexity.

Horn clauses have been used to model various problems. For example, Horn clauses have been used for automated theorem proving by first order resolution [26]. Horn clause forms the basis of logic programming and thus forms the root of almost all inference engines for logic programming languages like Prolog. Horn formulae have been used in the field of program analysis for intermediate representation and transformations [27], for program verification [28] and various other domains of formal methods.

For most of the topological networks that exist in real-life, like power transmission lines, telecommunications, water and gas supply lines, the structure of the network can be modelled using Horn clause. This modelling is often used to solve various real life problems. For example, for checking network reliability of power transmission lines one needs to compute the probability of two points being reachable under conditions of natural disaster. This is done using weighted model counting for the Horn formulas representing the network graph.

Since weighted model counting is similar to sampling, there is a strong demand for de-

signing efficient samplers for Horn clauses. Sampling of Horn clause may also find usage in probabilistic reasoning. Also, in program analysis when representation is done using Horn clause, sampling helps in producing test cases for such programs.

Despite HORNSAT being polynomial time solvable, the problem of counting the number of witnesses is $\#P$ -complete [22]. While sampling can be done using various techniques like MCMC methods, the runtime efficiency of these techniques makes them impractical. Thus the samplers would, naturally, rely on heuristic based techniques for sampling. Although there has not been much progress in designing samplers that use the special structure of Horn formulas, samplers developed for CNF apply to Horn formulae as well, since these are restricted versions of CNF. Some of the popular tools that are used to sample witnesses for CNF formulae and therefore, can be used for Horn formulae as well include CryptoMiniSat5[23], UniGen2[18], QuickSampler[6].

Most of these CNF-samplers are based on heuristic techniques and hence either provide very weak or no guarantees at all. Even when the input is restricted to Horn formulas, in most of the cases we don't have any guarantee of the correctness of the samplers. This suggests the need for testing uniformity of samplers, when the input is restricted to a Horn clause. Also, it is just a matter of time before one develops a sampler specific for Horn formulas. In such a case one need to design a testing framework for such samplers.

As discussed earlier, Barbarik, introduced in [10], is a framework that could be used to test for uniformity of CNF samplers. Most of the previous distribution testing mechanisms required exponential or sub-exponential samples¹. Barbarik overcomes this shortcoming using techniques from conditional sampling, introduce in [9],[13] and uses only constant number of samples. But since Barbarik has been designed to test samplers for CNF formulae, it is not straightforward retrofit to work for a restricted class like Horn. So in order to test for samplers that could sample for Horn formula we need to redesign the framework in order to suit our needs.

3.1.1 Horn Sampler Verifier

Most of the provable techniques for generating uniform witnesses for Horn clauses do not scale to real world instances. In remedy to this, samplers rely on heuristics to generate samples for Horn clause. We are not aware of any samplers which specifically work for

¹exponential with respect to the number of dimensions of the sample space

Horn clause but since it is a sub-class of CNF, available CNF-samplers are often used to generate samples for Horn formulae. As a result none of the known samplers actually consider the special structure of Horn clause in order to generate samples. Almost none of the samplers guarantee uniformity of samples that are generated. So designing a verifier for sampler for Horn clauses is the need of the hour.

Let us start by formally defining the notion of a verifier for samplers[10] of Horn clause. If G is a probabilistic generator/sampler then $A_G(., ., .)$ is a subroutine that takes input a Horn formula φ , a set $S \subset \text{Supp}(\varphi)$ ¹ and t . A_G returns t many satisfying assignments of φ using the sampler G . Since the underlying generator G is obvious, for the rest of the chapter we would use $A(., ., .)$ instead of $A_G(., ., .)$. If G is a uniform generator then we use $U(., ., .)$ as alias to $A_U(., ., .)$.

Definition 3.1.1. *A Horn Sampler Verifier takes as input a sampler G , a uniform generator U , a tolerance parameter $\epsilon > 0$, an intolerance parameter $\eta > \epsilon$, a guarantee parameter δ , and a Horn formula φ . The Horn Sampler Verifier returns *ACCEPT* or *REJECT* with the following guarantees:*

- *if the generator is an ϵ -additive almost-uniform generator over the sample space of valid witnesses to φ then we *ACCEPT* with probability at least $1 - \delta$.*
- *if $G(\varphi, .)$ is η -far from a uniform generator over the sample space of valid witnesses to φ then we *REJECT* with probability at least $1 - \delta$.*

Naturally the goal is to design a Horn Sampler verifier that is efficient in terms of scalability. Just like in the case of CNF-samplers the standard verifiers that use only black-box access requires exponential number of samples and hence cannot be scaled up.

3.1.2 Contributions of this work

In this chapter we design a fast testing framework for uniformity testing of Horn clause samplers. We develop our algorithm on similar guidelines as that of Barbarik [10] which uses techniques from conditional sampling.

We call our Horn Sample Verifier, FLASH. In order to test uniformity of a sampler, FLASH uses only $\tilde{O}(1/(\eta - \epsilon)^4)$ samples, where ϵ is the tolerance parameter and η is the intolerance parameter as discussed in the previous section. We also prove the following theoretical guarantees about FLASH:

¹for a given Boolean formula φ , $\text{Supp}(\varphi)$ denotes the set of independent variables in φ

1. **Completeness** of FLASH states that if a sampler is an ϵ -additive almost uniform generator and approximates any Horn formula φ , then FLASH ACCEPTs with probability at least $1 - \delta$.
2. **Soundness** of FLASH states that if the underlying distribution of the sampler that approximates any Horn formula φ is η -far from the uniform distribution, then FLASH REJECTs with probability at least $1 - \delta$.

Just like in the case of Barbarik, we need to design a subroutine that blows-up the conditioned witness space. In Barbarik, the blow-up is achieved by using the chain formula design in [24], discussed in Section 2.2.3. Conversion of Chain Formula to CNF and DNF have been shown in [24]. But for our case of Horn Formula we need to convert the Chain Formula to an equivalent Horn formula while preserving all other properties of Chain Formula. In quest of this we propose a variant called **Pure Horn Chain Formula** and prove the following lemma in Section 3.1.

Theorem. *Given a natural number $m > 0$ and $k < 2^m$, any constructed chain formula $\psi_{k,m}$ can be transformed into a pure Horn chain formula $\psi'_{k,m}$, where the following properties hold :*

1. *support and structure of $\psi_{k,m}$ is preserved in $\psi'_{k,m}$*
2. *$|\psi_{k,m}|$ is linear in m and $\psi'_{k,m}$ has exactly k satisfying assignments*
3. *$\psi'_{k,m}$ can be converted to an equivalent Horn formula with m variables and at most m clauses*

3.1.3 Organisation of the Chapter

The organisation of the chapter is as follows. In Section 3.2 we discuss a variation of Chain formula discussed in Section 2.2.3. Section 3.3 describes the working methodology of our Horn Sampler Verifier FLASH. In Section 3.4 we provide the theoretical analysis for FLASH where we comment on the correctness of our algorithm proving completeness, soundness and sample complexity for our algorithm. In Section 3.5 we show the run-time efficacy of FLASH on testing uniformity of some of the state-of-art samplers when run on some real life data.

3.2 Pure Horn Chain Formula

We define a variation of the chain formula designed in [24], we call it pure Horn chain formula. Given the chain formula $\psi_{k,m}$ we transform it to obtain our pure Horn chain formula $\psi'_{k,m}$. We replace every a_i in $\psi_{k,m}$ by \tilde{a}_i in $\psi'_{k,m}$, where \tilde{a}_i represent the complement of a_i . Intuitively a pure Horn chain formula is a chain formula where every literal is a negative literal.

The following lemma proves that the pure Horn chain formula preserves the structure, and other properties of the corresponding chain formula. We also prove that a pure Horn chain formula could be converted to an equivalent Horn formula.

Lemma 3.2.1. *Given a natural number $m > 0$ and $k < 2^m$, any chain formula $\psi_{k,m}$ can be transformed into a pure Horn chain formula $\psi'_{k,m}$, where the following properties hold :*

1. *support and structure of $\psi_{k,m}$ is preserved in $\psi'_{k,m}$*
2. *$|\psi_{k,m}|$ is linear on m and $\psi'_{k,m}$ has exactly k satisfying assignments*
3. *$\psi'_{k,m}$ can be converted to an equivalent Horn formula with m variables and at most m clauses*

Proof. Let $m > 0$ be a natural number and $k < 2^m$ and $\psi_{k,m}$ be the corresponding chain formula.

Proof of part 1 of the lemma : Let $\psi'_{k,m} = \psi_{k,m}$, i.e. $\psi'_{k,m}$ is assigned $\psi_{k,m}$. Now, $\forall j \in \{1, \dots, m\}$ we replace a_j by \tilde{a}_j in $\psi'_{k,m}$. Since none of the variables have changed, $Supp(\psi) = Supp(\psi')$. Also we do not change any of the connectors, i.e. $\forall j, C_j$ is unchanged. Thus, the structure of $\psi_{k,m}$ is preserved in $\psi'_{k,m}$. This proves the first part of the lemma.

Proof of part 2 of the lemma : We use induction over m . Let the statement in part 2 of the theorem be true for all pure Horn chain formula upto a length of $m - bits$ length. Now we consider the following two cases.

- **Case 1:** Let $m' = m + 1$. we consider a m' -bit positive odd integer, if the MSB is 0 then we could write the above formula as $\tilde{a}_1 \wedge \psi'_{k_1, m-1}$. The only way to satisfy this is by satisfying both \tilde{a}_1 and $\psi'_{k_1, m-1}$. Thus the number of solutions is k_1 . Since $k = k_1$ because the MSB is 0, the statement holds true for this case.

- **Case 2:** Let us suppose that the MSB is 1 then we could write the above formula as $\tilde{a}_1 \vee \psi'_{k_1, m-1}$. To satisfy this, we could either satisfy \tilde{a}_1 , which can be done in 2^{m-1} ways by setting a_1 as *false* and rest of the variables can be assigned any value, or by setting a_1 as *true* and satisfying $\psi'_{k_1, m-1}$ which has k_1 solutions. Thus the total number of solutions for this case is $2^{m-1} + k_1$, also $k = 2^{m-1} + k_1$. The statement holds true for this case.

Since the statement holds for both cases our inductive hypothesis is true.

A similar inductive argument on m proves the linearity of space for pure Horn chain formula, where two lists are sufficient to store the pure Horn chain formulae, one list stores the m -bit binary representation of k and another list stores the m literals of $\psi_{k,m}$. This proves the second part of the lemma.

Proof of part 3 of the lemma : We use induction over the number of variables n to prove the statement. Let the statement in part 3 of the theorem be true for all pure Horn chain formula upto $m - 1$ many variables. So, keeping aside the variable associated with the MSB, we can write the formula as $\psi'_{k,m} = \tilde{a}_1 C_1(\psi'_{k_1, m-1})$. So $\psi'_{k_1, m-1} = \psi'_1 \wedge \psi'_2 \dots \wedge \psi'_j$, where $\psi'_i, \forall i \in \{1, \dots, j\}$ is a Horn clause with at most $m - 1$ variables and $j \leq m - 1$. If C_1 is \wedge then \tilde{a}_1 becomes a unit negative clause which is Horn, and $\psi'_{k,m}$ is a Horn formula with m variables and at most m clauses. If C_1 is \vee then $\psi'_{k,m} = \tilde{a}_1 \vee (\psi'_1 \wedge \dots \wedge \psi'_j)$, by the distributive property we can expand it into $\psi'_{k,m} = (\tilde{a}_1 \vee \psi'_1) \wedge (\tilde{a}_1 \vee \psi'_2) \dots \wedge (\tilde{a}_1 \vee \psi'_j)$, since addition of a negative literal does not affect the nature of the Horn formula $\psi'_{k,m}$ now has $m - 1$ clauses each with at most m variables, so our hypothesis is correct for this case too. Therefore, $\psi'_{k,m}$ can be converted to an equivalent Horn formula. This proves the third part of the lemma. ■

The above lemma shows that with pure Horn chain formula we still preserve all properties of chain formula. The following example runs through all properties of a pure Horn chain formula proved in the above lemma.

Example 3.2.1. Let $k = 5$ and $m = 4$, the binary representation of 5 in 4-bits is 0101. The original chain formula would then be $\psi_{5,4}(a_1, a_2, a_3, a_4) = a_1 \wedge (a_2 \vee (a_3 \wedge a_4))$. We can then point to the following via Lemma 3.2.1 :

1. The transformed pure Horn chain formula $\psi'_{5,4}$ would be $\psi'_{5,4} = \tilde{a}_1 \wedge (\tilde{a}_2 \vee (\tilde{a}_3 \wedge \tilde{a}_4))$, where \tilde{a}_i represents $\neg a_i$ i.e. the negative literal corresponding to variable a_i .

2. $\psi'_{5,4} = \tilde{a}_1 \wedge (\tilde{a}_2 \vee (\tilde{a}_3 \wedge \tilde{a}_4))$ has exactly 5 satisfying solutions, we have to set a_1 as false, then $(\tilde{a}_2 \vee (\tilde{a}_3 \wedge \tilde{a}_4))$ can be satisfied in exactly 5 ways. The solutions to $\psi'_{5,4}$ are 0000, 0001, 0010, 0011 and 0100.
3. We can expand $\psi'_{5,4}$ as $\tilde{a}_1 \wedge (\tilde{a}_2 \vee \tilde{a}_3) \wedge (\tilde{a}_2 \vee \tilde{a}_4)$. It is easy to see that $\psi'_{5,4} = \tilde{a}_1 \wedge (\tilde{a}_2 \vee \tilde{a}_3) \wedge (\tilde{a}_2 \vee \tilde{a}_4)$, where each clause is a Horn clause. Each clause in the expanded $\psi'_{5,4}$ is of length less than or equal to 4 and the number of clauses is 3 i.e. both are bounded by the number of variables in $\psi'_{5,4}$.

3.3 Methodology of FLASH

FLASH starts with drawing one sample σ_1 according to the distribution $D_{G(\varphi)}$ and another sample σ_2 from the uniform distribution over R_φ . Using σ_1 and σ_2 the subroutine *Encode* produces a Horn formula T that has exactly three witnesses. The subroutine *kernel* then conjuncts φ and T to generate the a Horn formula $\tilde{\varphi}$, which has either two or three witnesses¹. FLASH then uses the pure Horn chain formula (via the subroutine *kernel*) to produce a new Horn formula φ' , which blows up the witness space of the Horn formula $\tilde{\varphi}$. FLASH achieves this such that the set of all witnesses of φ' , i.e. $R_{\varphi'}$ can be partitioned into two (or three) equal sized sets, where each partition corresponds to some witness of $\tilde{\varphi}$. Finally FLASH concludes on uniformity based on the number of times the witnesses corresponding to σ_1 occurs when N_j number of samples are drawn from the sampler with input φ' .

FLASH assumes access to procedures *Bias* and *Kernel*. *Bias* takes an assignment σ , a list L of assignments and a Sampling set S as input to compute the ratio of the number of times σ occurs in the list L to the total number of witnesses in the list L . Let $\sigma'_{\downarrow S}$ be an assignment belonging to L projected on the set S of variables.

The subroutine *kernel* help us to obtain the conditional samples. The subroutine *kernel* takes as input a Horn formula, two assignments σ_1 and σ_2 and the desired number of solutions τ , it returns another Horn formula φ' . The Horn formula φ' is of similar structure as that of φ . *kernel* assumes access to subroutine *Encode* which takes input σ_1 and σ_2 to generate a Horn formula T which has exactly three witnesses.

¹Witnesses of T are σ_1 , σ_2 and $\tilde{0}$, where $\tilde{0}$ is the assignment where all common positive literals of σ_1 and σ_2 are assigned *True* and all other variables are set to *False*. If $\tilde{0}$ is a witness of φ , then $\tilde{\varphi}$ has exactly three witnesses, else $\tilde{\varphi}$ has two witnesses

Algorithm 1: FLASH($A, U, \epsilon, \eta, \delta, \varphi$)

```
1  $S \leftarrow \text{Supp}(\varphi)$  ;
2 for  $j \leftarrow 1$  to  $\lceil \log(\frac{2}{\epsilon + \eta}) \rceil$  do
3    $t_j \leftarrow \lceil 2^j \frac{\eta + \epsilon}{(\eta + \epsilon)^2} \log(2(\eta + \epsilon)^{-1}) (\frac{4\epsilon}{e-1}) \ln(\delta^{-1}) \rceil$  ;
4    $\beta_j \leftarrow \frac{(2^{j-1} + 1)(\eta + \epsilon)}{4 + (2^{j-1} - 1)(\eta + \epsilon)}$  ;
5   for  $i \leftarrow 1$  to  $t_j$  do
6     while  $L_1 == L_2$  do
7        $L_1 \leftarrow A(\varphi, S, 1)$ ;
8        $\sigma_1 \leftarrow L_1[0]$ ;
9        $L_2 \leftarrow U(\varphi, S, 1)$ ;
10       $\sigma_2 \leftarrow L_2[0]$ ;
11       $\text{mod}T \leftarrow 2$  ;
12      if  $(\varphi \wedge \tilde{0}) \neq \text{UNSAT}$  then
13         $\text{mod}T \leftarrow 3$  ;
14       $N_j \leftarrow \lceil M(\frac{\beta_j - \epsilon}{2^{\text{mod}T}}) \log(\frac{2^4 e}{e-1} \frac{\delta^{-1}}{(\eta - \epsilon)^2} \log(\frac{2}{\eta + \epsilon}) \ln(\delta^{-1})) \rceil$  ;
15       $c_j \leftarrow (\beta_j + \epsilon)/2$  ;
16       $\tilde{\varphi} \leftarrow \text{kernel}(\varphi, \sigma_1, \sigma_2, N_j)$  ;
17       $L_3 \leftarrow A(\tilde{\varphi}, S, N_j)$  ;
18       $b \leftarrow \text{Bias}(\sigma_1, L_3, S)$  ;
19      if  $b < \frac{1}{\text{mod}T}(1 - c_j)$  or  $b > \frac{1}{\text{mod}T}(1 + c_j)$  then
20        return REJECT
21 return ACCEPT
```

Algorithm 1 represents the working of FLASH. In line 1, the variable S keeps track of the $Supp(\varphi)$. The outer loop (lines 2-20) runs for $\log \lceil \frac{2}{\epsilon+\eta} \rceil$ rounds, in each round we determine the bias we allow in the probability estimate computed by the routine $Bias$. Since any estimate is not absolutely exact we need to allow some bias and denote by the variable β_j . The variable N_j in line 14, denotes the minimum number of samples we need to draw to estimate the probability with which the sampler under test generates samples upto a certain guarantee with β_j as the allowed bias in the estimate.

Variable t_j denotes the number of times we need to run the inner loop (lines 5-20) in order to maximize the probability with which we *ACCEPT* any sampler under test. With each round we reduce that probability of wrongly accepting the sampler under test A . This simple argument suffices to show that with each round we maximize the probability. Suppose that the probability with which we wrongly accept is x , then probability of correctly accepting is $1 - x$. Now the probability of wrongly accepting for continuous n rounds would be x^n , thus the probability of correctly accepting at the end of n rounds would be $1 - x^n$, which greater than $1 - x$. Thus the inner loop is run t_j times in order to maximize the probability of correctly accepting the sampler.

Variable c_j denotes the feasible distance we need to allow from the uniform distribution in order to test for additive almost uniformity with ϵ -closeness. In every round in the inner loop we sample two different witnesses, σ_2 from the known uniform Sampler U and σ_1 from the sampler under test A .

From lines(11-13) we deal with whether $\tilde{0}$, which denotes the assignment of variables where all common positive literals in σ_1 and σ_2 are assigned 1 and rest all variables in $Supp(\varphi)$ are assigned 0. If $\tilde{0}$ is a witness we assign the variable $modT$ with value 3. The variable $modT$ denotes the size of the conditioning set C we generate. If $\tilde{0}$ is a witness the conditioning set C is of size 3, otherwise the size of the set is C is 2.

Using σ_1 and σ_2 , the *kernel* creates another Horn formula φ' . It is essential since the witness space of φ' is known and we can easily estimate the probabilities with which the sampler A generates witnesses.

On receiving φ' , A generates N_j samples and returns them in a list L_3 (line 17). In line 18, $Bias$ routine computes the ratio of number of times σ_1 occurs in L_3 when projected on the $Supp(\varphi)$ to the total number of witnesses in the list L_3 . If A is an additive almost uniform

sampler with ϵ -closeness then the estimate will be within $[\frac{1}{\text{mod}T}(1 - c_j), \frac{1}{\text{mod}T}(1 + c_j)]$.

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

```

1  $count \leftarrow 0$  ;
2 for  $\sigma' \in L$  do
3   if  $\sigma'_{\downarrow S} == \sigma$  then
4      $count \leftarrow count + 1$ 
5 return  $\frac{count}{|L|}$ 

```

Algorithm 2 presents the routine *Bias* which is responsible for estimating the probabilities by which the sampler under test A samples. *Bias* uses *Maximum Likelihood* estimating technique to estimate the probability with which σ is sampled. In order to compute the estimate, *Bias* computes the ratio of the number of times σ projected on S occurs in the list of samples L . Here samples refer to witnesses or solutions to φ .

3.3.1 Kernel Routine

Algorithm 3 presents the subroutine *kernel*. As stated before it takes a Horn formula φ , assignments σ_1 and σ_2 and τ as input and outputs another Horn Formula φ' .

In line 1 we extract the set of all common literals to the two assignments σ_1 and σ_2 . In line 2 we compute $|Lits|$ number of factors such that their product is greater than equal to τ . In Lines (3-4) we compute the number of extra variables needed M and generate M extra variables.

In line 6, the routine *Encode* returns a Horn formula assigned to T . *Encode* does this using the assignments σ_1 and σ_2 . The routine *Encode* enforces that T has only 3 satisfying assignments σ_1 , σ_2 and $\tilde{0}$. We explain the exact working of *Encode* in the next subsection, for now we assume that the set of satisfying witnesses of T is $\{\sigma_1, \sigma_2, \tilde{0}\}$.

In line 7, we conjunct φ and T . This step enforces the conditioning on R_φ . Since, R_φ contains σ_1 and σ_2 , we just need to check if $\tilde{0}$ is in R_φ . Line 12 in Algorithm 1 does this checking, where a *UNSAT* means $\tilde{0} \notin R_\varphi$. In that case the formula φ' has only two solutions σ_1 and σ_2 , since $R_{\varphi'} \cap R_T = \{\sigma_1, \sigma_2\}$.

Let us denote the conditioning set by C . The size of the conditioning set C is 2 if $\tilde{0} \notin R_\varphi$, i.e. $C = \{\sigma_1, \sigma_2\}$. If $\tilde{0} \in R_\varphi$ then the formula φ' in line 6, has 3 witnesses σ_1 , σ_2 , $\tilde{0}$. In

this case the conditioning set $C = \{\sigma_1, \sigma_2, \tilde{0}\}$ and it is of size 3.

Thereafter we need to project the conditioning set C into a larger witness space. Here project means that we blowup the witnesses space of φ' , such that the each solution can be tracked back to the conditioning set C . From lines 8-11 we perform this blow up such that the number of solutions that can be tracked back for each witness in the conditioning set is same. This ensures that we do not make the witness space biased towards any witness in the conditioning set.

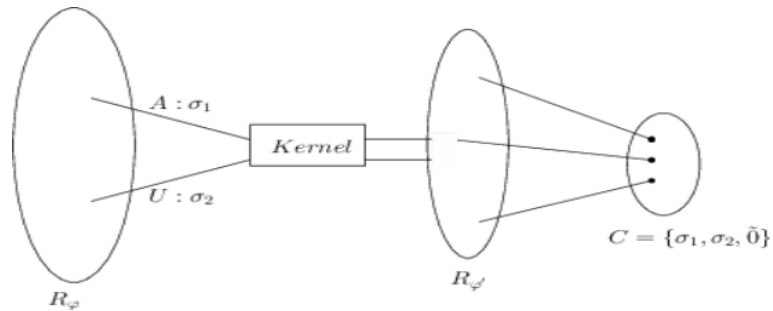
Algorithm 3: $Kernel(\varphi, \sigma_1, \sigma_2, \tau)$

```

1  $Lits \leftarrow (\sigma_1 \cap \sigma_2)$  ;
2  $K \leftarrow ComputeFactors(Lits, \tau)$  ;
3  $M \leftarrow \sum_{i=1}^{|Lits_1|} \lceil \log K[i] \rceil$  ;
4  $a \leftarrow NewVars(\varphi, M)$  ;
5  $index \leftarrow 0$  ;
6  $T \leftarrow Encode(\sigma_1, \sigma_2)$  ;
7  $\tilde{\varphi} \leftarrow \varphi \wedge T$  ;
8  $\varphi' \leftarrow \tilde{\varphi}$  ;
9 for  $(l, k)$  in  $(Lits, K)$  do
10    $m \leftarrow \lceil \log k \rceil$  ;
11    $\varphi_l \leftarrow \varphi' \wedge (l \Rightarrow \psi'_{k,m}(a[index : index + m]))$  ;
12    $index \leftarrow index + m$  ;
13 return  $\varphi_l$  ;
```

In the figure below we abstract out the working of the *kernel* in a diagrammatic fashion.

Figure 3.1: *kernel* takes two witnesses from R_φ then creates formula φ' , such that each witness in $R_{\varphi'}$ can be mapped to some element in C , this mapping partitions $R_{\varphi'}$ into $|C|$ parts of equal size



We present a working example of the *kernel* routine at the end of this section. The underlying lemma proves our claim of generating an unbiased witness space.

Lemma 3.3.1. *Given the conditioning set C , the number of solutions to φ' returned by kernel is $|C| \times \prod_{i=1}^{|K|} K[i]$, such that for each element $\sigma \in C$, σ is projected to $\prod_{i=1}^{|K|} K[i]$ solutions.*

Proof. To prove this lemma we assume that $\varphi \wedge T$ has exactly $|C|$ solutions¹. In each iteration, we add a clause of the form $(l \Rightarrow \psi'_{k,m})$. Since the literal l is common to both σ_1 and σ_2 , l evaluates to *true* in each solution of φ' . Since l is *true*, $\psi_{k,m}$ has to be satisfied and the number of ways to satisfy $\psi_{k,m}$ is exactly k , thus for each clause $(l \Rightarrow \psi'_{k,m})$ we increase the number of solutions k times. Since at the end of line 6, $\tilde{\varphi}$ has $|C|$ solutions, after all iterations are over the number of clauses added is $|K|$, the number of solutions at the end is $|C| \times \prod_{i=1}^{|K|} K[i]$.

For the next part of the Lemma, we can write φ' as follows : $\varphi' = \varphi \wedge T \wedge \psi$, where ψ is the conjunction of clauses added in lines 8-11. ψ has $\prod_{i=1}^{|K|} K[i]$ solutions and $\varphi \wedge T$ has $|C|$ solutions. Irrespective of any solution in C , ψ would have $\prod_{i=1}^{|K|} K[i]$ solutions. Thus, we can say for each element $\sigma \in C$, σ can be projected to $\prod_{i=1}^{|K|} K[i]$ solutions. ■

If the size of the conditioning set C is 2, then for the uniform distribution probability of sampling any element in C would be $1/2$. Otherwise if the size of C is 3, then for the uniform distribution probability of sampling any element from C would be $1/3$. Since we are testing for additive almost uniformity with ϵ -closeness, the probability of sampling would be within the range of $[\frac{1-\epsilon}{|C|}, \frac{1+\epsilon}{|C|}]$. The variable *modT* in Algorithm 1 keeps track of the size of the conditioning set C .

3.3.2 Encode Routine

In this subsection we describe the working of Algorithm 4 that represents the *Encode* subroutine. As suggested before, *Encode* takes as input two assignments σ_1 and σ_2 and generates a Horn formula Γ that has exactly three solutions σ_1 , σ_2 and $\tilde{0}$, i.e. $R_\Gamma = \{\sigma_1, \sigma_2, \tilde{0}\}$. It is essential to do this since this routine helps create the conditioning set C , which is a subset of R_Γ generated by *Encode*. We now describe how *Encode* does this.

In line 1, *findDifferVar* routine returns a tuple where *diffVar* is the index of the first variable where σ_1 and σ_2 differ, the second entry in the tuple σ is the assignment corresponding to which *diffVar* has been assigned 1.

¹The proof of this comes in the next subsection

Algorithm 4: *Encode*(σ_1, σ_2)

```
1 (diffVar,  $\sigma$ )  $\leftarrow$  findDiffVar( $\sigma_1, \sigma_2$ ) ;
2 cmmTrueLits  $\leftarrow$  findCommonTrueLiterals( $\sigma_1, \sigma_2$ ) ;
3  $\Sigma \leftarrow [\sigma_1, \sigma_2]$  ;
4  $\Gamma \leftarrow \text{True}$  ;
5 for each  $\sigma'$  in  $\Sigma$  do
6   if  $\sigma' == \sigma$  then
7     for each  $x_i$  in  $\sigma'$  and  $x_i \neq x_{diffVar}$  do
8        $\Gamma \leftarrow \Gamma \wedge (x_{diffVar} \Rightarrow \text{val}(x_i))$  ;
9   else
10    uncmmTrueLits  $\leftarrow$  differentTrueLiterals( $\sigma', \sigma$ ) ;
11    for each  $x_i$  in  $\sigma'$  and  $\text{val}(x_i) \neq x_i$  and  $x_i \neq x_{diffVar}$  do
12       $\Gamma \leftarrow (\neg x_{diffVar} \Rightarrow \text{val}(x_i))$  ;
13    fLit  $\leftarrow$  uncmmTrueLits[0] ;
14    for each  $l$  in uncmmTrueLits and  $l \neq fLit$  do
15       $\Gamma \leftarrow \Gamma \wedge (fLit \Leftrightarrow l)$  ;
16    cmmFLit  $\leftarrow$  cmmTrueLits[0] ;
17    for each  $l$  in cmmTrueLits and  $l \neq cmmFLit$  do
18       $\Gamma \leftarrow \Gamma \wedge (cmmFLit \Leftrightarrow l)$  ;
19     $\Gamma \leftarrow \Gamma \wedge cmmFLit$  ;
20 return  $\Gamma$  ;
```

Example 3.3.1. Suppose σ_1 is 110011 and σ_2 is 111100, then the first differing variable is x_3 and x_3 is assigned 1 in σ_2 . In this case **findDifferVar** would return $(3, \sigma_2)$.

In line 2, *findCommonTrueLiterals* returns a list of the positive literals common in both σ_1 and σ_2 .

Example 3.3.2. We continue with example 3.3.1, here x_1 and x_2 are both positive literals in σ_1 and σ_2 . Thus **cmmTrueLits** = $[x_1, x_2]$.

In line 3, we make a list Σ where the first element is σ_1 and second element is σ_2 . In lines 5-19, we construct the formula Γ iterating over Σ . In lines 6-8, if the assignment corresponds to σ returned by *FindDifferVar*, then we add clauses of the type $(x_{diffVar} \Rightarrow val(x_i))$, where $val(x_i)$ returns a positive literal x_i if it is assigned 1 in σ , else it returns a negative literal $\neg x_i$ if it is assigned 0 in σ .

Example 3.3.3. Using the example 3.3.1, σ_2 is 111100, $\sigma = \sigma_2$ and $diffVar = 3$. In lines 6-8, we add the following clauses:

$$(x_3 \Rightarrow x_1) \wedge (x_3 \Rightarrow x_2) \wedge (x_3 \Rightarrow x_4) \wedge (x_3 \Rightarrow \neg x_5) \wedge (x_3 \Rightarrow \neg x_6)$$

By adding these clauses we ensure that if $x_{diffVar}$ is assigned 1, then the only possible witness to Γ is necessarily σ .

In lines 9-19, if $\sigma' \neq \sigma$ then we add a different set of clause, such that if $x_{diffVar}$ is assigned 0, then the only possible witnesses are σ' and $\tilde{0}$.

Firstly in line 10 we compute the set of literals assigned 1 in σ' and 0 in σ . Then in lines 11-12, for all variables assigned 0 in σ' , we add the clause of type $(\neg x_{diffVar} \Rightarrow val(x_i))$. This enforces that if $x_{diffVar}$ is assigned 0 then x_i would necessarily be assigned 0.

Example 3.3.4. Considering the assignments in example 3.3.1, $\sigma' = \sigma_1$. We add the clause $(\neg x_3 \Rightarrow \neg x_4)$ which ensures that if x_3 is assigned 0 variable x_4 is also assigned 0.

fLit is assigned the first literal in the list *uncmmTrueLits*. In line 14-15, we add a clause of type $(fLit \Leftrightarrow l)$, for all l in *uncmmTrueLits*. This ensures that these literals are either all assigned 0 or all assigned 1.

Example 3.3.5. As in example 3.3.1, we add the clause $(\neg x_5 \Leftrightarrow \neg x_6)$. If x_3 is assigned 1, then we already have added the clauses $(x_3 \Rightarrow \neg x_5) \wedge (x_3 \Rightarrow \neg x_6)$, thus enforcing x_5 and x_6 to take 00 as the only possible assignment needed for getting to assignment σ_2 . If x_3 is assigned 0, then x_5 and x_6 could take values 00 or 11, where 00 assigned would correspond to $\tilde{0}$ and 11 assignment would correspond to σ_1 .

Now in lines 16-19, we add clauses to ensure that the common positive literals in σ_1 and σ_2 remain positive literals for any satisfying assignment of Γ . $cmmFLit$ is assigned the first literal in the list $cmmTrueLits$ that contains the set of common positive literals. There after in lines 17-18 we add clauses of the form $(cmmLit \Leftrightarrow l)$, where l is in $cmmTrueLits$. In line 19, we add the unit positive clause $cmmFLit$ that contains only one literal. Since, $cmmFLit$ has to be satisfied to satisfy Γ , all literals in $cmmTrueLits$ also have to be assigned 1 to satisfy Γ , so that clauses added by lines 17-18 are all satisfied.

Example 3.3.6. Again we use the example 3.3.1, $cmmTrueLits = [x_1, x_2]$. $cmmFLit = x_1$, the following clauses are added to Γ in lines 16-19 : $(x_1 \Leftrightarrow x_2) \wedge x_1$. The only satisfying assignment of x_3 and x_4 is 11. Thus, for any satisfying assignment of Γ , x_3 and x_4 are always assigned 11.

Example 3.3.7. Examining at all the examples in this subsection, for $\sigma_1 = 110011$ and $\sigma_2 = 111100$, we form the following Γ .

$$\Gamma = (x_3 \Rightarrow x_1) \wedge (x_3 \Rightarrow x_2) \wedge (x_3 \Rightarrow x_4) \wedge (x_3 \Rightarrow \neg x_5) \wedge (x_3 \Rightarrow \neg x_6) \wedge (\neg x_3 \Rightarrow \neg x_4) \wedge (\neg x_5 \Leftrightarrow \neg x_6) \wedge (x_1 \Leftrightarrow x_2) \wedge x_1.$$

Now if we compute the satisfying assignments of Γ , they would be σ_1 , σ_2 and $\tilde{0}$, where $\sigma_1 = 110011$, $\sigma_2 = 111100$ and $\tilde{0} = 110000$.

The following lemma formalizes the claim that *Encode* routine generates a Horn formula Γ , such that the only witnesses of Γ are σ_1 , σ_2 and $\tilde{0}$.

Lemma 3.3.2. Γ generated by *Encode* is a Horn formula and $R_\Gamma = \{\sigma_1, \sigma_2, \tilde{0}\}$, where R_Γ is the witness set of Γ .

Proof. It is self evident that every clause we add is of length less than equal to 2. All clauses of length two have at least one negative literal. Therefore they are all Horn clauses by nature. We add only one clause of length 1, which trivially satisfies the property for Horn clause. Therefore Γ is a Horn formula.

For the other part of the lemma we will use induction on the number of variables n in Γ . Let us suppose that $R_\Gamma = \sigma_1, \sigma_2, \tilde{0}$ for m variables. Suppose σ_1 and σ_2 , use $m + 1$ variables. Let x_{m+1} represent the $m + 1^{th}$ variable.

- Let x_{m+1} be assigned 0 in σ_1 and 0 in σ_2 . There would we two clauses of the form $(x_i \Rightarrow \neg x_{m+1})$ and $(\neg x_i \Rightarrow \neg x_{m+1})$, so for any value of x_i , x_{m+1} has to be assigned 0. So, $R_\Gamma = \sigma_1, \sigma_2, \tilde{0}$.

- Let x_{m+1} be assigned 1 in both σ_1 and σ_2 . Then x_{m+1} is included in two clauses $(x_i \Rightarrow), (x_j \Leftrightarrow x_{m+1}) \wedge x_j$. To satisfy them x_{m+1} has to be assigned 1 again. So, $R_\Gamma = \sigma_1, \sigma_2, \tilde{0}$.
- Let x_{m+1} be assigned 1 in σ_1 , 0 in σ_2 and σ be σ_1 . Then x_{m+1} , is associated with clauses $(x_i \Rightarrow x_{m+1}), (\neg x_i \Rightarrow \neg x_{m+1})$. If x_i is 1 then x_{m+1} as 1 corresponds to σ_1 . If x_i is 0 then x_{m+1} is 0 which corresponds to σ_2 or $\tilde{0}$. So, still R_Γ remains the same.
- Let x_{m+1} be assigned 0 in σ_1 , 1 in σ_2 and σ be σ_1 . Then x_{m+1} is associated with clauses $(x_i \Rightarrow x_{m+1})$ and $(x_j \Leftrightarrow \neg x_{m+1})$. If x_i is 1, x_{m+1} as 0 corresponds to σ_1 . If x_i is 0, then x_{m+1} if assigned 1 corresponds to σ_2 , else if assigned 0 it would correspond to $\tilde{0}$. Therefore R_Γ is still unchanged.

Therefore, our hypothesis is still valid for $m + 1$ variables. So, R_Γ is $\{\sigma_1, \sigma_2, \tilde{0}\}$. ■

The following example demonstrates a single run of the subroutine *kernel*.

Example 3.3.8. Let the **kernel** routine be called with the following arguments $\varphi = (x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5)$, $\sigma_1 = 10011$, $\sigma_2 = 11001$ and $\tau = 10$.

In line 1 we compute $Lits = [x_1, x_5]$. In line 2, suppose *ComputeFactors* returns $[7, 7]$ such that $\prod_{i=1}^{|K|} K[i] \geq \tau$. In line 3, M is assigned $\lceil \log K[0] \rceil + \lceil \log K[1] \rceil = \lceil \log 7 \rceil + \lceil \log 7 \rceil = 3+3 = 6$. In line 4, *NewVars* returns $[x_6, x_7, \dots, x_{11}]$ and is assigned to variable a .

Encode routine is called with arguments σ_1 and σ_2 . As described earlier it would return the following formula:

$$T = (x_2 \Rightarrow x_1) \wedge (x_2 \Rightarrow \neg x_3) \wedge (x_2 \Rightarrow \neg x_4) \wedge (x_2 \Rightarrow \neg x_5) \wedge (\neg x_2 \Rightarrow \neg x_3) \wedge (x_1 \Leftrightarrow x_5) \wedge x_1$$

In line 8, we have $\varphi' = \varphi \wedge T$. The loop in line 9 runs for 2 iterations, adding the following clauses:

$$(x_1 \Rightarrow (\neg x_6 \vee \neg x_7 \vee \neg x_8)) \text{ and } (x_5 \Rightarrow (\neg x_9 \vee \neg x_{10} \vee \neg x_{11})).$$

So at the end of the loop, φ' is as follows :

$$\begin{aligned} \varphi' = & (x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5) \wedge \\ & (x_2 \Rightarrow x_1) \wedge (x_2 \Rightarrow \neg x_3) \wedge \\ & (x_2 \Rightarrow \neg x_4) \wedge (x_2 \Rightarrow \neg x_5) \wedge \\ & (\neg x_2 \Rightarrow \neg x_3) \wedge (x_1 \Leftrightarrow x_5) \wedge \\ & x_1 \wedge (x_1 \Rightarrow (\neg x_6 \vee \neg x_7 \vee \neg x_8)) \wedge \\ & (x_5 \Rightarrow (\neg x_9 \vee \neg x_{10} \vee \neg x_{11})) \end{aligned}$$

The witness space of φ' , $R_{\varphi'} = (R_\varphi \cap R_T) \times (R_{\psi_{7,3}}) \times (R_{\psi_{7,3}})$. So each witness in $R_{\varphi'}$ can be tracked back to the conditioning set formed by $R_\varphi \cap R_T$.

In the next section we would look into the theoretical analysis of FLASH.

3.4 Analysis of FLASH

In this section we present the theoretical analysis for FLASH. We first discuss the completeness of our framework. We further discuss about the soundness of our approach under certain assumptions. Lastly we discuss the sampling complexity for our framework.

Theorem 3.4.1. Completeness : *If a generator(sampler) G is an ϵ -additive almost-uniform generator and approximates any Horn Formula φ , then FLASH ACCEPTS with probability at least $(1 - \delta)$.*

Proof. If the generator(sampler) G is an ϵ -additive almost-uniform generator then for any Horn formula φ , S and N_j , the sampler outputs L_3 which is a collection of N_j samples drawn independently according to the ϵ -additive almost-uniform distribution. And the bias we allow is in the range of $1/\text{mod}T(1 - \epsilon)$ to $1/\text{mod}T(1 + \epsilon)$, where $\text{mod}T$ is the size of the conditioning set and can have values 2 or 3.

FLASH REJECTS in the $(i, j)^{th}$ loop, where i is the inner loop and j is the outer loop, if the probability estimate of the bias b is not in the range $(\frac{1}{\text{mod}T}(1 - \frac{\beta_j + \epsilon}{2}), \frac{1}{\text{mod}T}(1 + \frac{\beta_j + \epsilon}{2}))$, that is the estimate is off by more than $(\frac{\beta_j - \epsilon}{2\text{mod}T})$. By definition of $M(\gamma)$, the probability that this happens is at most $(1/2)^{N_j/M(\beta_j - \epsilon/2\text{mod}T)}$. i.e. at most $(\frac{2^4 e}{e-1} \frac{\delta^{-1}}{(\eta - \epsilon)^2} \log(\frac{2}{\eta + \epsilon}) \ln(\delta^{-1}))^{-1}$.

Since, the number of calls to the *Bias* routine is at most $\sum_{j=1}^{\lceil \log 2^{2/(\epsilon + \eta)} \rceil} t_j$, which is at most $(\frac{2^4 e}{e-1} \frac{1}{(\eta - \epsilon)^2} \log(\frac{2}{\eta + \epsilon}) \ln(\delta^{-1}))$, the probability that we REJECT is (using union bound) $\leq (\sum_{j=1}^{\lceil \log 2^{2/(\epsilon + \eta)} \rceil} t_j) \times ((1/2)^{N_j/M(\beta_j - \epsilon/2\text{mod}T)}) = \delta$. Thus, the probability that we accept the sampler if it is an ϵ -additive almost uniform generator is at least $(1 - \delta)$. ■

For Soundness we need to define the concept of a non-adversarial sampler[10].

Definition 3.4.1. [10] Non-adversarial sampler assumption states that if $A(\varphi, S, 1)$ outputs a sample by drawing according to a distribution D then the (φ', S') output by the Kernel routine has the property that:

- $S \subset S'$.
- there are only three sets of assignments to variables in S that could be extended to satisfying assignments for φ'
- The restriction of the set L_3 to the variable set S is obtained by drawing N samples from the distribution $D|_R$, where $R = L_1 \cup L_2 \cup \{\tilde{0}\}$.

Since the assumptions to the original algorithm are preserved, we adapt the same proof for **Soundness** for FLASH using the soundness proof for the Barbarik framework [10].

Theorem 3.4.2. [10] ***Soundness*** : If the non-adversarial assumption holds and if the distribution $D_{G(\varphi)}$ is η -far from uniform on the sampling set S , then FLASH REJECTS with probability at least $(1 - \delta)$.

Theorem 3.4.3. Sample Complexity : Given a tolerance parameter ϵ , an intolerance parameter η and probability bound δ , FLASH needs at most $\tilde{O}(\frac{1}{(\eta-\epsilon)^4})$ samples for any input Horn formula φ , where \tilde{O} hides poly logarithmic factor of $1/\delta$ and $1/(\eta - \epsilon)$.

Proof. for any j , $t_j = \lceil 2^j \frac{\eta+\epsilon}{(\eta+\epsilon)^2} \log(2(\eta+\epsilon)^{-1}) (\frac{4e}{e-1}) \ln(\delta^{-1}) \rceil$ and in every i in the inner loop we sample $2 + N_j$ samples, where

$$N_j \text{ is at most } \lceil M(\frac{\beta_j-\epsilon}{6}) \log(\frac{2^4 e}{e-1} \frac{\delta^{-1}}{(\eta-\epsilon)^2} \log(\frac{2}{\eta+\epsilon}) \ln(\delta^{-1})) \rceil.$$

For every j , we sample a total of $2t_j + t_j N_j$ samples.

Thus the total number of samples we need to draw is at most $\sum_{j=1}^{\lceil \log(\frac{2}{\epsilon+\eta}) \rceil} (2 + N_j)t_j$.

Note that , $\sum_{j=1}^{\lceil \log(\frac{2}{\epsilon+\eta}) \rceil} 2t_j = (\frac{2^5 e}{e-1} \frac{1}{(\eta-\epsilon)^2} \log(\frac{2}{\eta+\epsilon}) \ln(\delta^{-1}))$.

Now taking $M(\gamma)$ as $1/6\gamma^2$, we have $\sum_{j=1}^{\lceil \log(\frac{2}{\epsilon+\eta}) \rceil} t_j N_j$ as $\tilde{O}(\frac{1}{(\eta-\epsilon)^4})$, where \tilde{O} hides poly logarithmic factors of $1/\delta$ and $1/(\eta - \epsilon)$. ■

3.5 Evaluation and Results

3.5.1 Evaluation

In order to evaluate the run-time performance of FLASH and test uniformity of some of the state-of-art samplers for Horn clauses, we implement a prototype of FLASH using Python 2.7 as the language of choice. The prototype implementation of FLASH is built on top of the prototype implementation of Barbarik [10]. We employ SPUR [15] as the known uniform sampler, i.e. U in Algorithm 1. We use Cryptominisat5 [23] as the sat solver used to determine if $\tilde{0}$ is a solution of φ in line 12 of Algorithm 1. The experiments are conducted partially on a high-performance computer cluster¹, where each node consists of E5-2690 v3 CPU with 24 cores and 96GB of RAM.

We use the tolerance parameter ϵ as 0.6, the intolerance parameter η as 0.9 and the

¹The computational work for this article was partially performed on resources of the National Super-computing Centre, Singapore (<https://www.nsc.sg>)

guarantee parameter δ as 0.1. The chosen set of parameter implies that given a Boolean formula φ and an additive almost-uniform generator G , for $G(\varphi, \epsilon)$, FLASH returns ACCEPT with at least 0.9 probability (under the non-adversarial sampler assumption) and REJECT with at least 0.9 probability.

UniGen2 is a multiplicative almost-uniform sampler. As discussed previously this implies that it is also an additive almost-uniform sampler, Whereas, Quicksampler and STS do not provide such strong guarantees. We apriori expect FLASH to ACCEPT UniGen2 and REJECT Quicksampler and STS for most of the instances of the Horn clause using reasonable number of samples¹.

3.5.2 Analysis of Horn formula Instances

Instances	#Variables	#Clauses
Net6_count_91	153	184
Net8_count_96	162	194
Net12_count_106	191	214
Net22_count_116	219	234
Net27_count_118	230	238
Net29_count_164	280	330
Net39_count_240	403	482
Net43_count_243	437	488
Net46_count_322	564	646
Net53_count_362	640	726
Net53_count_339	619	680
Net54_count_706	1224	1414

Table 3.1: Analysis of Benchmarks used for FLASH

All instances used in the evaluation are Horn Formulae which have been used to model power transmission lines. We use the data listed in Table 3.1 for the run-time evaluation of FLASH. The columns represent the instances, number of variables and number of clauses respectively.

The witness space for the listed instances of Horn formula vary from a size of order of 10^{30} to order of 10^{144} making it infeasible for standard statistical techniques to test uniformity of any sampler.

¹We use the default parameters for Quicksampler, STS and UniGen2, which were employed in the previous studies for uniformity in [6].

3.5.3 Results

Instances	UniGen2		Quicksampler		STS	
	#samples	Output	#samples	Output	#samples	Output
Net6_count_91	2296910	A	771	R	771	R
Net8_count_96	2296910	A	771	R	771	R
Net12_count_106	2296910	A	771	R	771	R
Net22_count_116	2296910	A	771	R	771	R
Net27_count_118	2296910	A	771	R	771	R
Net29_count_164	2296910	A	771	R	771	R
Net39_count_240	2296910	A	771	R	771	R
Net43_count_243	2296910	A	771	R	771	R
Net46_count_322	2296910	A	771	R	771	R
Net53_count_362	2296910	A	771	R	771	R
Net53_count_339	2296910	A	771	R	771	R
Net54_count_706	2296910	A	771	R	771	R

Table 3.2: The output and analysis of the number of samples consumed by FLASH

Table 3.2 represents the number of samples sampled by FLASH in order to ACCEPT¹ or REJECT² for the samplers under test. Column 1 represents instances of Horn formula listed in Table 3.1 supplied as input to FLASH. Columns 2 and 3 represent the statistics for UniGen2. Columns 4 and 5 represent the statistics for Quicksampler and columns 6 and 7 represent statistics for STS.

We observe that FLASH returns REJECT for Quicksampler and STS for 771 samples for all 12 instances, whereas, it returns ACCEPT for UniGen2 for all 12 instances. Our algorithm is probabilistic in nature and it could very well be the case that for some other run or instance of Horn formula we fail to generate the expected result. FLASH uses significantly less number of samples to REJECT Quicksampler and STS.

3.6 Conclusion

The results demonstrate the support for the non-adversarial sampler assumption. As expected, samplers with weak guarantees fail to pass the test while samplers with stronger theoretical guarantees pass our test. Overall, FLASH could be used as an inexpensive tool to test uniformity, especially when the distribution followed by the sampler is far

¹ACCEPT in Table 3.2 is represented by A

²REJECT in Table 3.2 is represented by R

from uniform. To the best of our knowledge, FLASH is the first algorithmic test to test uniformity of a Horn sampler. Our results demonstrate that our test rejects samplers with weak guarantees and accepts samplers with strong guarantees.

Chapter 4

Testing Samplers for Perfect Matching

4.1 Introduction

The problem of finding perfect matching in graphs is of significant importance in computer science and various other fields like statistical mechanics, chemistry, etc. For example, perfect information games are often modelled using perfect matching in graphs. Such a model helps in the analysis of the games. Also perfect matching is used in combinatorial optimization to model the problem of alldifferent constraints [29]. Another field where perfect matching finds usage is quantum computing, where topological codes are decoded using minimum-weighted perfect matching [30]. Experiments are often designed by modelling of the system as a graph and finding a perfect matching in such a graph. For example, job scheduling algorithms often view the system and constrains as an implication graph, where a perfect matching corresponds to a schedule for jobs.

Perfect matching finds it's roots in physical sciences as well. For example, in the field of chemistry, perfect matching is used for storing information, finding bond length, estimating resonance energy, etc. [31]. Such widespread usage acts as strong evidence about the importance of perfect matchings.

The problem of uniform perfect matching has widespread usage in computer science, statistical mechanics, chemistry, particle physics, etc. The problem has been studied for the last few decades and some standard algorithms have been proposed. The problem of counting perfect matching closely relates to that of finding the 0-1 permanent of an

matrix. Valiant in [4] showed that exact computation of permanent is $\#P$ -complete. The problem of finding the 0-1 permanent of a matrix can be reduced to counting the number of perfect matchings in a bipartite graph. This shows that counting of total number of perfect matchings is $\#P$ -complete even for bipartite graphs. Algorithms that have been proposed for uniform random sampling of perfect matching of graphs are mostly *fully polynomial randomized approximation schemes* (FPRAS). Few of the most promising have been based on Markov Chain Monte Carlo (MCMC) based approaches [[25], [32]]. These algorithms are polynomial time algorithms and sample perfect matchings from a bipartite graph uniformly at random. This widespread usage of perfect matchings suggests a demand for perfect matching samplers. It is probably not very far that researchers would develop samplers for perfect matchings.

Since most of the promising approaches are based on MCMC methods which have very high complexity making them an infeasible choice for practical purposes, we expect samplers for perfect matchings to rely on heuristics for scalable generation of samples. This motivates a need for testing mechanisms for uniformity of perfect matching samplers. In this work we present an algorithmic framework that could be used to test uniformity of samplers for perfect matchings.

4.1.1 Our Contributions

In this work we present an algorithm for testing if a perfect matching sampler on a given graph outputs according to the uniform distribution over all perfect matchings in the graph or whether the underlying distribution is η -far from uniform. We refer to our proposed testing algorithm as ***Uniform Perfect Matching Sampler Test*** (UPMST).

UPMST works on the guidelines of conditional sampling but unlike FLASH, it adopts to the sub-cube conditioning model of conditional sampling. This is because the methodology of witness based conditioning is not appropriate in the case of perfect matching. So, in order to use conditional sampling we resort to the sub-cube conditional model shown in [11]. Sub-cube model of conditional sampling becomes a natural choice for perfect matching since, every perfect matching can be viewed an element in $\Sigma^{|E|}$, where $\Sigma = \{0, 1\}$ and E represents the number of edges in a graph. We discuss a routine *Restrict* which dictates the sub-cube conditioning strategy for our framework. We adapt our framework from the testing algorithm for identity testing of joint distributions using conditional sampling presented in [11]. The algorithm in [11] assumes access to *BasicIDTester* pointed in [33]. We also adapt this algorithm to work for our framework.

Our approach uses only $\tilde{O}(|E|^2/\eta^2)$ samples where, E is the edge set of a given graph and η is the target distance from uniform distribution. We show the correctness of our algorithm in terms of completeness and soundness along with the sample complexity.

4.2 Preliminaries

4.2.1 A Uniform Perfect Matching Sampler

Given a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. Any subgraph $H = (V', E')$ can be represented as a 0-1 vector m_H of dimension $|E| \times 1$, such that if the i^{th} entry of m_H is 0 then the i^{th} edge of G is not in E' and if the i^{th} entry is 1 then the i^{th} edge of G is in E' .

Every perfect matching M can be viewed as a sub-graph of G , such that $M = (V, E'')$. So, a given perfect matching can be viewed as a 0-1 vector such that $M \in \Sigma^{|E|}$, where $\Sigma = \{0, 1\}$.

Let the set of all perfect matchings of a given graph be denoted by \mathcal{P}_G , such that $\mathcal{P}_G \subset \Sigma^{|E|}$. A Uniform perfect matching sampler on receiving a graph G as input outputs a perfect matching $M \in \mathcal{P}_G$ uniformly at random. Formally we can define it as follows :

Definition 4.2.1. *Given a graph G , a Uniform Perfect Matching Sampler $U(\cdot)$ returns any element M in \mathcal{P}_G with probability $1/|\mathcal{P}_G|$, i.e.*

$$\forall M \in \mathcal{P}_G, Pr[U(G) = M] = \frac{1}{|\mathcal{P}_G|}$$

4.2.2 Sub-cube conditioning in Graphs

Intuitively, the sub-cube model of conditional sampling suggests the following. That given a distribution μ defined over Σ^n , fix a set $A = A_1 \times A_2 \times \dots \times A_n$, where $A_i \subseteq \Sigma$ for all i and then sample from the set A which is a subset of Σ^n .

In a different form of this model, we randomly select a number $j < n$ and fix a subset $A^{(j)} = A_1 \times A_2 \times \dots \times A_j$, where $A^{(j)} \subseteq \Sigma^j$. In this case we sample from Σ^n conditioned on the set $A^{(j)}$.

Now coming to the context of graphs, we present a way of using sub-cube conditioning for a given graph instance. Given a graph $G = (V, E)$ and any sampler $A(\cdot)$ for perfect

matching, A generates a perfect matching M from the set of all perfect matchings \mathcal{P}_G . $P_G \subset \Sigma^{|E|}$, where $\Sigma = \{0, 1\}$. Since every perfect matching can be seen as a 0-1 vector with dimension $|E| \times 1$, where a 0-entry would mean the corresponding edge is not in the matching and a 1-entry would mean the corresponding edge belongs to the perfect matching. Formally, for a perfect matching M of a graph G , the 0-1 vector representation of M can be written as $m_M = (m_1, m_2, \dots, m_{|E|})$, where if m_i is 0 then edge $e_i \notin M$, else if m_i is 1 then edge $e_i \in M$.

We define the notion of partial assignment for perfect matching as follows :

Definition 4.2.2. Consider a graph $G = (V, E)$, an integer $j < |E|$ and 0-1 vector $m = (m_1, m_2, \dots, m_j)$. The vector m is called a **partial assignment for perfect matching** M , if the first j entries of m_M are equal to m .

Intuitively, we decide for the first j edges of G and then form a perfect matching M . This strategy can be used to apply sub-cube conditioning on graphs in the following manner.

1. Generate a valid partial assignment $m = (m_1, m_2, \dots, m_j)$ of length $j < |E|$
2. Then we transform the graph G in the following manner:
 - (a) If the i^{th} entry in m and m_i is 0, we remove the edge e_i from G .
 - (b) Else if the i^{th} entry in m and m_i is 1, we remove both incident vertices of e_i from G .

Let us call the transformed graph as G' . Now every perfect matching in G' has the partial assignment m . This strategy gives us a way of conditioning using partial assignments for perfect matching. We adhere to this strategy in order to perform sub-cube conditional sampling in our proposed framework UPMST.

4.3 Methodology of UPMST

UPMST is an adaptation of the algorithm to test identity of joint distributions to some known distribution presented in [11]. UPMST uses techniques from conditional sampling where it adheres to the sub-cube conditional model. The idea behind UPMST is of viewing any perfect matching M as an element in $\Sigma^{|E|}$, where $\Sigma = \{0, 1\}$ and E is the set of edges in a graph $G = (V, E)$. UPMST assumes access to two subroutines *Restrict*

and *BasicIDTester*.

UPMST starts by drawing a perfect matching m_G for a given graph $G = (V, E)$ from the known uniform sampler. It then randomly selects a set of edges denoted by S_j .

Then UPMST uses the subroutine *Restrict* to facilitate sub-cube conditional sampling. The subroutine *Restrict* using the set of edges S_j and the matching m_G uses the concept of partial assignment defined in the Section 4.2.2 to generate a new graph G' . The subroutine *Restrict* uses the transformation mentioned in Section 4.2.2 to construct a new graph G' . So the subroutine *Restrict* helps us obtain conditional samples.

After this transformation, UPMST calls the subroutine *BasicIDTester* to determine whether sampler is to be rejected or not using sub-cube conditioned samples. *BasicIDTester* has been adapted from [33].

Intuition behind UPMST

UPMST is similar to a sampler verifier discussed in section 2.3. UPMST takes as input the sampler to be tested A , a known uniform sampler for perfect matching U , an instance of a graph G , a target distance $0 < \eta < 1$, a guarantee parameter $0 < \delta < 1$. We assume that the graph G has a set of valid perfect matchings. UPMST returns ACCEPT or REJECT with the following guarantees:

- for any $0 < \eta < 1$, if $d(D_A, D_U) = 0$ we accept with probability at least $(1 - \delta)$.
- for any $0 < \eta < 1$, if $d(D_A, D_U) > \eta$ we reject with probability at least δ .

Here D_A denotes the distribution generated by the perfect matching sampler A , D_U (uniform distribution) is the distribution generated by the known uniform sampler U and $d(D_A, D_B)$ denotes the total variance distance. The algorithm was initially developed to test distributions. For our case we are provided with the samplers which still suffices to perform the test in a different manner.

Algorithm 5 represents UPMST, the main idea is to generate the sub-cube conditioning set C in every iteration, then restrict out the sampling set to a projection of C and use the sampler to sample from the conditioned set C . Then based on the samples we decide whether to accept or reject the sampler A .

Since algorithm 5 has been adapted from [11], the fundamental step is the sub-cube conditioning while using a blackbox sampler. It is essential to do this, since Bhattacharyya

Algorithm 5: $UPMST(A, U, G, \eta, \delta)$

```
1  $n \leftarrow numEdges(G)$  ;
2  $\delta' \leftarrow \delta\eta/64n(\log(n)^2)$  ;
3 for  $j \leftarrow 1$  to  $\log n + 1$  do
4    $\eta_j = \eta/2^j H(n)$  ;
5    $l_j \leftarrow \log \frac{2^{j+1}H(n)}{\eta}$  ;
6    $S_j \leftarrow random([n], min(n, 4n/2^j))$  ;
7   for all  $i \in S_j$  do
8     for  $k \leftarrow 0$  to  $l_j$  do
9        $\eta_{j,k} \leftarrow 2^{k-1}\eta_j$  ;
10       $\delta_k \leftarrow \delta'/(k+3)^2$  ;
11      for  $t \leftarrow 1$  to  $2^{k+2}(k+3)^2$  do
12         $m_G \leftarrow U(G, 1)$  ;
13         $G' \leftarrow Restrict(G, m_G, i, S_j)$  ;
14        if  $BasicIDTester(A, U, G', i, E, \eta_{j,k}, \delta_k) == REJECT$  then
15          return REJECT ;
16 return ACCEPT ;
```

et. al in [11] showed that there must exist at least one sub-cube conditioned marginal distribution for which the conditional distance from uniform is more than $\eta/poly(n)$.

In line 1, we assign variable n with the number of edges in G . The outer loop from lines 3-15 runs for $\log n + 1$ times. In each iteration of the outer loop we determine the variables η_j , l_j and determine the set S_j which contains the edges we later condition on. We iterate over every element in S_j and in each iteration determine the allowed target conditional distance $\eta_{j,k}$ and the guarantee parameter δ_k .

In each iteration we run another loop (lines 11-15) for $2^{k+2}(k+3)^3$ times which amplifies the probability with which we output ACCEPT or REJECT correctly. In this loop (lines 11-15) we do the sub-cube conditioning. Firstly, we draw a perfect matching m_G from the known uniform sampler U using the graph G . Then we construct a sub-cube conditioned graph G' which dictates the sub-cube conditioning in the algorithm. The *Restrict* routine takes as input the original graph G , the sample m_G , the current edge to be tested i and the set of edges for conditioning S_j . Now *Restrict* routine returns a graph G' , where G' is the sub-cube conditioned graph on the the set S_j and matching m_G . In line 14, we call the routine *BasicIDTester* represented by Algorithm 6.

If the *BasicIDTester* routine returns REJECT it indicates we have arrived at the conditional marginal distribution or the adversarial edge for which sampler A cannot sample close to uniformly. If the *BasicIDTester* routine returns ACCEPT we keep on iterating further. The test guarantees that if any sampler samples far off from the uniform distribution it cannot pass the test. All uniform samplers should be able to pass the test.

4.3.1 *Restrict Routine*

Algorithm 6: *Restrict*(G, m, i, S)

```

1  $n \leftarrow \text{numVertices}(G)$  ;
2  $G' \leftarrow G$  ;
3 for each  $e \in S$  and  $e < i$  do
4   if  $m[i] == 1$  then
5      $v \leftarrow \text{vertices}(e)$  ;
6      $G' \leftarrow G' \setminus v$  ;
7   else
8      $G' \leftarrow G' \setminus e$  ;
9 if  $\text{numVertices}(G') < n/2$  then
10   $\tilde{G} \leftarrow \text{constructClique}(n)$  ;
11   $G' \leftarrow \text{append}(G', \tilde{G})$  ;
12 return  $G'$  ;
```

The restrict routine dictates the sub-conditioning in our framework. As explained in section 4.2.2, we use the restrict routine to transform a given graph using a form of partial assignment. Also if at the end of this transformation the size of the graph reduces significantly then we generate a blown-up graph such that the matchings in the reduced graph are still preserved. Such a blow-up can be done simply by attaching a large clique to the reduced graph by a single edge.

Algorithm 6 represents the *Restrict* routine. It takes as input a graph G , a 0-1 vector representation of a matching m in G , an edge i of G and the set of edges S on which the conditioning is to be done. It returns another graph G' which is sub-cube conditioned on a form of partial assignment.

In line 1 we assign variable n with the number of vertices in the original graph G . From lines 2-8 we do the sub-cube conditioning on the graph G to form a graph G' . In the loop

(lines 3-8) for each edge e in S and if e is numbered less than i , we check if e is in the matching m or not. If e is in the matching m we remove both vertices from the graph G' , else if e is not in the matching m we remove the edge e from the graph G' .

At the end of this transformation, G' is the sub-cube conditioned graph we want but it may so happen that it is reduced to very small sized graph. In that case we construct a complete graph \tilde{G} with n vertices in line 10. In line 11, we attach the graphs G' and \tilde{G} with an edge. This blow-up mechanism still preserves the sub-cube conditioning. In line 12, we return the sub-cube conditioned graph G' .

4.3.2 *BasicIDTester* Routine

Algorithm 7: *BasicIDTester*(A, U, G, e, η, δ)

```

1  $n \leftarrow \lceil 3/\eta^2 \rceil$  ;
2  $count \leftarrow 0$  ;
3 for  $i = 1$  to  $\lceil 18 \log \frac{1}{\delta} \rceil$  do
4    $n_1 \leftarrow pois(n)$  ;
5    $n_2 \leftarrow pois(n)$  ;
6    $\sigma_1 \leftarrow A(G, n_1)$  ;
7    $\sigma_2 \leftarrow U(G, n_2)$  ;
8    $r_{n_1} \leftarrow 0$  ;
9   for all  $m \in \sigma_1$  do
10    if  $e \in m$  then
11       $r_{n_1} \leftarrow r_{n_1} + \frac{1}{n_1}$ 
12   $r_{n_2} \leftarrow 0$  ;
13  for all  $m \in \sigma_2$  do
14    if  $e \in m$  then
15       $r_{n_2} \leftarrow r_{n_2} + \frac{1}{n_2}$ 
16  if  $|r_{n_1} - r_{n_2}| \leq \eta$  then
17     $count \leftarrow count + 1$  ;
18 if  $count < \lceil 9 \log \frac{1}{\delta} \rceil$  then
19   return REJECT ;
20 return ACCEPT ;

```

Algorithm 6 represents an adaptation of the *BasicIDTester*. It takes as input the sampler under test A , the known uniform sampler U , an instance of a sub-cube conditioned graph G , the edge or the marginal currently under consideration e , the allowed target

distance η and the guarantee parameter δ .

In line 1, the variable n denotes the order of number of samples we need to draw in each round of the algorithm. Variable *count* in line 2 is assigned 0 and keeps track of the number of times the estimate we compute is far from η .

In lines 3-17 we run a loop for $\lceil 18 \log 1/\delta \rceil$ times. In each round we take two random values n_1 and n_2 using the Poisson distribution with parameter n . Then in line 6 we sample n_1 perfect matchings from the sampler A . Similarly in line 7 we sample n_2 perfect matchings from the known uniform sampler U . In lines 8-11, we compute the number of times the edge e appears in the list of perfect matching σ_1 , and denote it by r_{n_1} . Again in lines 12-15, we compute the number of times the edge e appears in the list of perfect matching σ_2 , and denote it by r_{n_2} . Then in lines 16-17, we check the difference of the two estimates. If the difference is $\leq \eta$ we then increment *count* by one. In line 18-19, *count* indicates the majority vote of whether the distributions are equal or not. Algorithm 6 outputs the majority of the outcomes arrived in all rounds. So, if *count* is less than $\lceil 9 \log 1/\delta \rceil$, Algorithm 6 returns REJECT, otherwise it returns ACCEPT.

4.4 Theoretical Analysis of UPMST

In this section we present the theoretical guarantees that our proposed method provides. Firstly we provide the guarantees and sampling complexity of Algorithm 6.

Algorithm 6 is an adaptation of the *BasicIDTester* algorithm in [33]. Since the assumptions to the algorithm remain unchanged we state and adopt the same proof as in [33].

Theorem 4.4.1. [33] *Let μ be a known distribution over Σ . Given $0 < \eta < 1$ and $0 < \delta < 1$ and a distribution μ' over Σ then the (η, δ) -SUBCOND Identity tester has a conditional sampling complexity $\tilde{O}(\frac{1}{\eta^2} \log \frac{1}{\delta})$. In other words, the tester draws $\tilde{O}(\frac{1}{\eta^2} \log \frac{1}{\delta})$ samples and*

- *if $\mu = \mu'$, then the tester accepts with probability $(1 - \delta)$ and*
- *if $d(\mu, \mu') > \eta$ then the tester rejects with probability $(1 - \delta)$.*

Algorithm 5 is an adoption of the algorithm for identity testing with known distributions presented in [11]. The assumptions and constants remain the same and so we adhere to

the same proof for the sampling complexity and correctness of the algorithm presented in [11].

Theorem 4.4.2. [11] (**Sample complexity**) *Given any $0 < \eta < 1$, any $0 < \delta < 1$ and a graph $G = (V, E)$, UPMST is an (η, δ) -SUBCOND Identity tester for uniformity with conditional sampling complexity of $\tilde{O}(|E|^2/\eta^2)$, where \tilde{O} hides polynomial log factors of n , $1/\eta$ and $1/\delta$.*

Theorem 4.4.3. [11] (**Completeness**) *Consider any $0 < \eta < 1$, any $0 < \delta < 1$ and a graph $G = (V, E)$, μ, μ' are distributions over $\Sigma^{|E|}$, where $\Sigma = \{0, 1\}$. If $d(\mu, \mu') = 0$, UPMST rejects with probability at most δ .*

Theorem 4.4.4. [11] (**Soundness**) *Consider any $0 < \eta < 1$, any $0 < \delta < 1$ and a graph $G = (V, E)$, μ, μ' are distributions over $\Sigma^{|E|}$, where $\Sigma = \{0, 1\}$. If $d(\mu, \mu') > \eta$, UPMST rejects with probability at least δ .*

4.5 Conclusion

The theoretical guarantees showcase strong evidence that UPMST would be able to REJECT perfect matching samplers that generate samples far off from the uniform distribution with low number of samples. Also the sub-cube conditioning model is generic to fit to our needs. We believe that the same approach can be adapted to be used for various other problems like equivalence checking of neural networks over finite input spaces. Since neural networks are by default not generators, they need to be converted to a generative model using the test dataset available before actually applying the test for identity of joint distributions. We leave this as future work.

Chapter 5

Conclusion

In this thesis we have proposed two frameworks for testing uniformity of samplers. Firstly, we presented a uniformity testing framework FLASH which can be used to test uniformity of samplers for Horn formulae. Secondly, we proposed an algorithmic framework Uniform Perfect Matching Sampler Test (UPMST) to test uniformity of samplers for perfect matchings in graphs.

Experimental results for FLASH show that it can be used as an inexpensive tool for testing uniformity. Also we were able to REJECT non-uniform samplers when run on standard benchmarks. Overall, the results suggest strong evidence on the theoretical guarantees that we provide for FLASH and showcases that it can capture samplers generating samples far from uniform distribution with very few number of samples. To the best of our knowledge, FLASH is the first testing framework proposed to check uniformity of Horn Clause specific samplers. It would be of tremendous interest to design uniformity testing frameworks for other classes of CNF like 2-SAT, Dual-Horn and some non-CNF classes like XOR-CNF. These classes are of keen interest in various fields and are used in context of sampling. Thus coming up with testing frameworks exclusively for such classes could give a new direction to this research.

The theoretical guarantees provided for UPMST suggest that it can be used inexpensively for testing uniformity of perfect matching samplers. To the best of our knowledge UPMST is the first testing framework proposed targeted towards samplers for perfect matching. The sub-cube conditioning model is very dynamic and opens up another direction of research where this setting can be adapted to solve problems like equivalence checking of neural networks used in hybrid systems, side channel cryptanalysis and many other problems where the sample space is a product set.

References

- [1] D. Roth and W. T. Yih, “Probabilistic reasoning for entity & relation recognition,” in *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002. 2
- [2] R. Paredes, L. Duenas-Osorio, K. S. Meel, and M. Y. Vardi, “A weighted model counting approach for critical infrastructure reliability,” in *Proceedings of International Conference on Applications of Statistics and Probability in Civil Engineering, (ICASP13)*, 7 2019. 2
- [3] Y. Naveh, M. Rimon, I. Jaeger, Y. Katz, M. Vinov, E. s Marcu, and G. Shurek, “Constraint-based random stimuli generation for hardware verification.,” *AI Magazine*, vol. 28, pp. 13–30, 01 2007. 2
- [4] L. Valiant, “The complexity of computing the permanent,” *Theoretical Computer Science*, vol. 8, no. 2, pp. 189 – 201, 1979. 2, 13, 18, 41
- [5] N. Kitchen and A. Kuehlmann, “Stimulus generation for constrained random simulation,” in *2007 IEEE/ACM International Conference on Computer-Aided Design*, pp. 258–265, Nov 2007. 3
- [6] R. Dutra, K. Laeuffer, J. Bachrach, and K. Sen, “Efficient sampling of SAT solutions for testing,” in *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pp. 549–559, 2018. 3, 14, 15, 20, 37
- [7] T. Batu, L. Fortnow, R. Rubinfeld, W. D. Smith, and P. White, “Testing closeness of discrete distributions,” *CoRR*, vol. abs/1009.5397, 2010. 3
- [8] C. L. Canonne, D. Ron, and R. A. Servedio, “Testing probability distributions using conditional samples,” *CoRR*, vol. abs/1211.2664, 2012. 3, 10

- [9] S. Chakraborty, E. Fischer, Y. Goldhirsh, and A. Matsliah, “On the power of conditional samples in distribution testing,” *CoRR*, vol. abs/1210.8338, 2012. 3, 10, 20
- [10] S. Chakraborty and K. S. Meel, “On testing of uniform samplers,” in *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 1 2019. 3, 4, 17, 20, 21, 35, 36
- [11] R. Bhattacharyya and S. Chakraborty, “Property testing of joint distributions using conditional samples,” *CoRR*, vol. abs/1702.01454, 2017. 3, 4, 11, 41, 43, 44, 45, 48, 49
- [12] J. W. Harris and H. Stocker, *The Handbook of Mathematics and Computational Science*. Berlin, Heidelberg: Springer-Verlag, 1st ed., 1997. 10
- [13] C. L. Canonne, I. Diakonikolas, D. M. Kane, and A. Stewart, “Testing bayesian networks,” *CoRR*, vol. abs/1612.03156, 2016. 10, 11, 20
- [14] S. A. Cook, “The complexity of theorem-proving procedures,” in *Proceedings of the Third Annual ACM Symposium on Theory of Computing, STOC '71*, (New York, NY, USA), pp. 151–158, ACM, 1971. 13
- [15] D. Achlioptas, Z. S. Hammoudeh, and P. Theodoropoulos, “Fast sampling of perfectly uniform satisfying assignments,” in *Theory and Applications of Satisfiability Testing – SAT 2018* (O. Beyersdorff and C. M. Wintersteiger, eds.), (Cham), pp. 135–147, Springer International Publishing, 2018. 13, 36
- [16] M. Thurley, “sharpsat – counting models with advanced component caching and implicit bcp,” in *Theory and Applications of Satisfiability Testing - SAT 2006* (A. Biere and C. P. Gomes, eds.), (Berlin, Heidelberg), pp. 424–429, Springer Berlin Heidelberg, 2006. 13
- [17] J. S. Vitter, “Random sampling with a reservoir,” *ACM Trans. Math. Softw.*, vol. 11, pp. 37–57, Mar. 1985. 14
- [18] S. Chakraborty, D. J. Fremont, K. S. Meel, S. A. Seshia, and M. Y. Vardi, “On parallel scalable uniform sat witness generation,” in *Tools and Algorithms for the Construction and Analysis of Systems* (C. Baier and C. Tinelli, eds.), (Berlin, Heidelberg), pp. 304–319, Springer Berlin Heidelberg, 2015. 14, 15, 20
- [19] S. Chakraborty, K. S. Meel, and M. Y. Vardi, “Balancing scalability and uniformity in sat witness generator,” in *Proceedings of the 51st Annual Design Automation Conference, DAC '14*, (New York, NY, USA), pp. 60:1–60:6, ACM, 2014. 14

- [20] M. Soos, K. Nohl, and C. Castelluccia, “Extending SAT solvers to cryptographic problems,” in *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, pp. 244–257, 2009. 14
- [21] S. Ermon, C. P. Gomes, A. Sabharwal, and B. Selman, “Embed and project: Discrete sampling with universal hashing,” in *Advances in Neural Information Processing Systems 26* (C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, eds.), pp. 2085–2093, Curran Associates, Inc., 2013. 14
- [22] N. Creignou and M. Hermann, “Complexity of generalized satisfiability counting problems,” *Information and Computation*, vol. 125, no. 1, pp. 1 – 12, 1996. 15, 20
- [23] M. Soos, “Cryptominisat5, <https://www.msoos.org/cryptominisat5/>,” 15, 20, 36
- [24] S. Chakraborty, D. Fried, K. S. Meel, and M. Y. Vardi, “From weighted to unweighted model counting,” in *Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI’15*, pp. 689–695, AAAI Press, 2015. 16, 22, 23
- [25] M. Jerrum and A. Sinclair, “The markov chain monte carlo method: An approach to approximate counting and integration,” 09 1996. 18, 41
- [26] J. Makowsky, “Why horn formulas matter in computer science: Initial structures and generic examples,” *Journal of Computer and System Sciences*, vol. 34, no. 2, pp. 266 – 292, 1987. 19
- [27] G. Gange, J. A. Navas, P. Schachte, H. Sondergaard, and P. J. Stuckey, “Horn clauses as an intermediate representation for program analysis and transformation,” *Theory and Practice of Logic Programming*, vol. 15, no. 4-5, p. 526542, 2015. 19
- [28] E. De Angelis, F. Fioravanti, A. Pettorossi, and M. Proietti, “Verifying relational program properties by transforming constrained horn clauses,” in *Proceedings of the 31st Italian Conference on Computational Logic, Milano, Italy, June 20-22, 2016.*, pp. 69–85, 2016. 19
- [29] B. Anastasatos, “Propagation algorithms for the alldifferent constraint,” 06 2019. 40
- [30] A. G. Fowler, “Minimum weight perfect matching of fault-tolerant topological quantum error correction in average $o(1)$ parallel time,” *Quantum Information Computation*, vol. 15, pp. 145–158, 2015. 40

- [31] D. Vukičević, *Applications of Perfect Matchings in Chemistry*, pp. 463–482. Boston: Birkhäuser Boston, 2011. 40
- [32] M. Jerrum and A. Sinclair, “Approximating the permanent,” *SIAM Journal on Computing*, vol. 18, no. 6, pp. 1149–1178, 1989. 41
- [33] M. Falahatgar, A. Jafarpour, A. Orlitsky, V. Pichapati, and A. T. Suresh, “Faster algorithms for testing under conditional sampling,” in *Proceedings of The 28th Conference on Learning Theory (P. Grnwald, E. Hazan, and S. Kale, eds.)*, vol. 40 of *Proceedings of Machine Learning Research*, (Paris, France), pp. 607–636, PMLR, 03–06 Jul 2015. 41, 44, 48
- [34] M. Jerrum, A. Sinclair, and E. Vigoda, “A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries,” *J. ACM*, vol. 51, pp. 671–697, July 2004.
- [35] O. Goldreich, S. Goldwasser, and D. Ron, “Property testing and its connection to learning and approximation,” *J. ACM*, vol. 45, pp. 653–750, July 1998.
- [36] R. Rubinfeld and M. Sudan, “Robust characterizations of polynomials with applications to program testing,” *SIAM Journal on Computing*, vol. 25, no. 2, pp. 252–271, 1996.
- [37] T. Batu, E. Fischer, L. Fortnow, R. Kumar, R. Rubinfeld, and P. White, “Testing random variables for independence and identity,” in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pp. 442–451, Oct 2001.
- [38] S. Chan, I. Diakonikolas, G. Valiant, and P. Valiant, “Optimal algorithms for testing closeness of discrete distributions,” *CoRR*, vol. abs/1308.3946, 2013.
- [39] O. Goldreich and D. Ron, *On Testing Expansion in Bounded-Degree Graphs*, pp. 68–75. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.
- [40] R. Levi, D. Ron, and R. Rubinfeld, “Testing properties of collections of distributions,” *Theory of Computing*, vol. 9, no. 8, pp. 295–347, 2013.
- [41] T. Batu, S. Dasgupta, R. Kumar, and R. Rubinfeld, “The complexity of approximating the entropy,” in *Proceedings 17th IEEE Annual Conference on Computational Complexity*, pp. 17–, May 2002.