Inverse problems in approximate uniform generation

Anindya De[*] University of California, Berkeley
Ilias Diakonikolas[†] University of Edinburgh
Rocco A. Servedio[‡] Columbia University

Abstract

We initiate the study of *inverse* problems in approximate uniform generation, focusing on uniform generation of satisfying assignments of various types of Boolean functions. In such an inverse problem, the algorithm is given uniform random satisfying assignments of an unknown function f belonging to a class $\mathcal{C}$ of Boolean functions (such as linear threshold functions or polynomial-size DNF formulas), and the goal is to output a probability distribution D which is ϵ-close, in total variation distance, to the uniform distribution over $f^{-1}(1)$. Problems of this sort comprise a natural type of unsupervised learning problem in which the unknown distribution to be learned is the uniform distribution over satisfying assignments of an unknown function $f \in \mathcal{C}$.

Positive results: We prove a general positive result establishing sufficient conditions for efficient inverse approximate uniform generation for a class $\mathcal{C}$. We define a new type of algorithm called a *densifier* for $\mathcal{C}$, and show (roughly speaking) how to combine (i) a densifier, (ii) an approximate counting / uniform generation algorithm, and (iii) a Statistical Query learning algorithm, to obtain an inverse approximate uniform generation algorithm. We apply this general result to obtain a poly$(n, 1/\epsilon)$-time inverse approximate uniform generation algorithm for the class of n-variable linear threshold functions (halfspaces); and a quasipoly$(n, 1/\epsilon)$-time inverse approximate uniform generation algorithm for the class of $\mathrm{poly}(n)$-size DNF formulas.

Negative results: We prove a general negative result establishing that the existence of certain types of signature schemes in cryptography implies the hardness of certain inverse approximate uniform generation problems. We instantiate this negative result with known signature schemes from the cryptographic literature to prove (under a plausible cryptographic hardness assumption) that there are no subexponential-time inverse approximate uniform generation algorithms for 3-CNF formulas; for intersections of two halfspaces; for degree-2 polynomial threshold functions; and for monotone 2-CNF formulas.

Finally, we show that there is no general relationship between the complexity of the "forward" approximate uniform generation problem and the complexity of the inverse problem for a class $\mathcal{C}$ – it is possible for either one to be easy while the other is hard. In one direction, we show that the existence of certain types of Message Authentication Codes (MACs) in cryptography implies the hardness of certain corresponding inverse approximate uniform generation problems, and we combine this general result with recent MAC constructions from the cryptographic literature to show (under a plausible cryptographic hardness assumption) that there is a class $\mathcal{C}$ for which the "forward" approximate uniform generation problem is easy but the inverse approximate uniform generation problem is computationally hard. In the other direction, we also show (assuming the GRAPH ISOMORPHISM problem is computationally hard) that there is a problem for which inverse approximate uniform generation is easy but "forward" approximate uniform generation is computationally hard.

[*] anindya@cs.berkeley.edu. Research supported by NSF award CCF-0915929 and Umesh Vazirani's Templeton Foundation Grant 21674.

[†] ilias.d@ed.ac.uk. Most of this work was done while the author was at UC Berkeley supported by a Simons Postdoctoral Fellowship.

[‡] rocco@cs.columbia.edu. Supported by NSF grant CCF-1115703, CCF-0915929.

1 Introduction

The generation of (approximately) uniform random combinatorial objects has been an important research topic in theoretical computer science for several decades. In complexity theory, well-known results have related approximate uniform generation to other fundamental topics such as approximate counting and the power of nondeterminism [JS89, JVV86, SJ89, Sip83, Sto83]. On the algorithms side, celebrated algorithms have been given for a wide range of approximate uniform generation problems such as perfect matchings [JSV04], graph colorings (see e.g. [Jer95, Vig99, HV03]), satisfying assignments of DNF formulas [KL83, JVV86, KLM89], of linear threshold functions (i.e., knapsack instances) [MS04, Dye03] and more.

Before describing the inverse problems that we consider, let us briefly recall the usual framework of approximate uniform generation. An approximate uniform generation problem is defined by a class $\mathcal{C}$ of combinatorial objects and a polynomial-time relation $R(x, y)$ over $\mathcal{C} \times \{0, 1\}^*$. An input instance of the problem is an object $x \in \mathcal{C}$, and the problem, roughly speaking, is to output an approximately uniformly random element y from the set $R_x := \{y : R(x, y) \text{ holds}\}$. Thus an algorithm A (which must be randomized) for the problem must have the property that for all $x \in \mathcal{C}$, the output distribution of $A(x)$ puts approximately equal weight on every element of R_x. For example, taking the class of combinatorial objects to be $\{$all $n \times n$ bipartite graphs$\}$ and the polynomial-time relation R over (G, M) pairs to be "M is a perfect matching in G," the resulting approximate uniform generation problem is to generate an (approximately) uniform perfect matching in a given bipartite graph; a $\mathrm{poly}(n, \log(1/\epsilon))$-time algorithm was given in [JSV04]. As another example, taking the combinatorial object to be a linear threshold function (LTF) $f(x) = \mathrm{sign}(w \cdot x - \theta)$ mapping $\{-1, 1\}^n \to \{-1, 1\}$ (represented as a vector $(w_1, \ldots, w_n, \theta)$) and the polynomial-time relation R over (f, x) to be "x is a satisfying assignment for f," we arrive at the problem of generating approximately uniform satisfying assignments for an LTF (equivalently, feasible solutions to zero-one knapsack). A polynomial-time algorithm was given by [MS04] and a faster algorithm was subsequently proposed by [Dye03].

The focus of this paper is on *inverse* problems in approximate uniform generation. In such problems, instead of having to *output* (near-)uniform elements of R_x, the *input* is a sample of elements drawn uniformly from R_x, and the problem (roughly speaking) is to "reverse engineer" the sample and output a distribution which is close to the uniform distribution over R_x. More precisely, following the above framework, a problem of this sort is again defined by a class $\mathcal{C}$ of combinatorial objects and a polynomial-time relation R. However, now an input instance of the problem is a sample $\{y_1, \ldots, y_m\}$ of strings drawn uniformly at random from the set $R_x := \{y : R(x, y) \text{ holds}\}$, where now $x \in \mathcal{C}$ is *unknown*. The goal is to output an ϵ*-sampler* for R_x, i.e., a randomized algorithm (which takes no input) whose output distribution is ϵ-close in total variation distance to the uniform distribution over R_x. Revisiting the first example from the previous paragraph, for the inverse problem the input would be a sample of uniformly random perfect matchings of an unknown bipartite graph G, and the problem is to output a sampler for the uniform distribution over all perfect matchings of G. For the inverse problem corresponding to the second example, the input is a sample of uniform random satisfying assignments of an unknown LTF over the Boolean hypercube, and the desired output is a sampler that generates approximately uniform random satisfying assignments of the LTF.

Discussion. Before proceeding we briefly consider some possible alternate definitions of inverse approximate uniform generation, and argue that our definition is the "right" one (we give a precise statement of our definition in Section 2, see Definition 11).

One stronger possible notion of inverse approximate uniform generation would be that the output distribution should be supported on R_x and put nearly the same weight on every element of R_x, instead of just being ϵ-close to uniform over R_x. However a moment's thought suggests that this notion is too strong, since it is impossible to efficiently achieve this strong guarantee even in simple settings. (Consider, for example, the problem of inverse approximate uniform generation of satisfying assignments for an unknown LTF.

Given access to uniform satisfying assignments of an LTF f, it is impossible to efficiently determine whether f is (say) the majority function or an LTF that differs from majority on precisely one point in $\{-1,1\}^n$, and thus it is impossible to meet this strong guarantee.)

Another possible definition of inverse approximate uniform generation would be to require that the algorithm output an ϵ-approximation of the unknown object x instead of an ϵ-sampler for R_x. Such a proposed definition, though, leads immediately to the question of how one should measure the distance between a candidate object x' and the true "target" object x. The most obvious choice would seem to be the total variation distance between $\mathcal{U}_{R_x}$ (the uniform distribution over R_x) and $\mathcal{U}_{R_{x'}}$; but given this distance measure, it seems most natural to require that the algorithm actually output an ϵ-approximate sampler for R_x.

Inverse approximate uniform generation via reconstruction and sampling. While our ultimate goal, as described above, is to obtain algorithms that output a sampler, algorithms that attempt to reconstruct the unknown object x will also play an important role for us. Given $\mathcal{C}, R$ as above, we say that an (ϵ,δ)-*reconstruction algorithm* is an algorithm $A_{\mathrm{reconstruct}}$ that works as follows: for any $x \in \mathcal{C}$, if $A_{\mathrm{reconstruct}}$ is given as input a sample of $m = m(\epsilon,\delta)$ i.i.d. draws from the uniform distribution over R_x, then with probability $1-\delta$ the output of $A_{\mathrm{reconstruct}}$ is an object $\tilde{x} \in \tilde{C}$ such that the variation distance $d_{\mathrm{TV}}(\mathcal{U}_{R_x}, \mathcal{U}_{R_{\tilde{x}}})$ is at most ϵ. (Note that the class $\tilde{\mathcal{C}}$ need not coincide with the original class $\tilde{\mathcal{C}}$, so $\tilde{x}$ need not necessarily belong to $\mathcal{C}$.) With this notion in hand, an intuitively appealing schema for algorithms that solve inverse approximate uniform generation problems is to proceed in the following two stages:

1. **(Reconstruct the unknown object):** Run a reconstruction algorithm $A_{\mathrm{reconstruct}}$ with accuracy and confidence parameters $\epsilon/2, \delta/2$ to obtain $\tilde{x} \in \tilde{\mathcal{C}}$;

2. **(Sample from the reconstructed object):** Let A_{sample} be an algorithm which solves the approximate uniform generation problem $(\tilde{\mathcal{C}}, R)$ to accuracy $\epsilon/2$ with confidence $1-\delta/2$. The desired sampler is the algorithm A_{sample} with its input set to $\tilde{x}$.

We refer to this as the *standard approach* for solving inverse approximate uniform generation problems. Most of our positive results for inverse approximate uniform generation can be viewed as following this approach, but we will see an interesting exception in Section 7, where we give an efficient algorithm for an inverse approximate uniform generation problem which does not follow the standard approach.

1.1 Relation between inverse approximate uniform generation and other problems. Most of our results will deal with uniform generation problems in which the class $\mathcal{C}$ of combinatorial objects is a class of syntactically defined Boolean functions over $\{-1,1\}^n$ (such as the class of all LTFs, all $\mathrm{poly}(n)$-term DNF formulas, all 3-CNFs, etc.) and the polynomial-time relation $R(f,y)$ for $f \in \mathcal{C}$ is "y is a satisfying assignment for f." In such cases our inverse approximate uniform generation problem can be naturally recast in the language of learning theory as an unsupervised learning problem (learning a probability distribution from a known class of possible target distributions): we are given access to samples from $\mathcal{U}_{f^{-1}(1)}$, the uniform distribution over satisfying assignments of $f \in \mathcal{C}$, and the task of the learner is to construct a hypothesis distribution D such that $d_{\mathrm{TV}}(\mathcal{U}_{f^{-1}(1)}, D) \le \epsilon$ with high probability. We are not aware of prior work in unsupervised learning that focuses specifically on distribution learning problems of this sort (where the target distribution is uniform over the set of satisfying assignments of an unknown member of a known class of Boolean functions).

Our framework also has some similarities to "uniform-distribution learning from positive examples only," since in both settings the input to the algorithm is a sample of points drawn uniformly at random from $f^{-1}(1)$, but there are several differences as well. One difference is that in uniform-distribution learning from positive examples the goal is to output a hypothesis function h, whereas here our goal is to output a hypothesis *distribution* (note that outputting a function h essentially corresponds to the reconstruction problem described above). A more significant difference is that the success criterion for our framework is significantly more demanding than for uniform-distribution learning. In uniform-distribution learning of a Boolean function f over the hypercube $\{-1,1\}^n$, the hypothesis h must satisfy $\mathbf{Pr}[h(x) \neq f(x)] \leq \epsilon$, where the probability is uniform over all 2^n points in $\{-1,1\}^n$. Thus, for a given setting of the error parameter ϵ, in uniform-distribution learning the constant -1 function is an acceptable hypothesis for any function f that has $|f^{-1}(1)| \leq \epsilon 2^n$. In contrast, in our inverse approximate uniform generation framework we measure error by the total variation distance between $\mathcal{U}_{f^{-1}(1)}$ and the hypothesis distribution D, so no such "easy way out" is possible when $|f^{-1}(1)|$ is small; indeed the hardest instances of inverse approximate uniform generation problems are often those for which $f^{-1}(1)$ is a very small fraction of $\{-1,1\}^n$. Essentially we require a hypothesis with small *multiplicative* error relative to $|f^{-1}(1)|/2^n$ rather than the additive-error criterion that is standard in uniform-distribution learning. We are not aware of prior work on learning Boolean functions in which such a "multiplicative-error" criterion has been employed.

We summarize the above discussion with the following observation, which essentially says that reconstruction algorithms directly yield uniform-distribution learning algorithms:

Observation 1. *Let $\mathcal{C}$ be a class of Boolean functions $\{-1,1\}^n \to \{-1,1\}$ and let $R(f,y)$ be the relation "y is a satisfying assignment for f." Suppose there exists a $t(n,\epsilon,\delta)$-time (ϵ,δ)-reconstruction algorithm for $\mathcal{C}$ that outputs elements of $\tilde{\mathcal{C}}$. Then there is an $(O(\log(1/\delta)/\epsilon^2) + O(t(n,\epsilon,\delta/3)\cdot\log(1/\delta)/\epsilon))$-time uniform-distribution learning algorithm that outputs hypotheses in $\tilde{\mathcal{C}}$ (i.e., given access to uniform random labeled examples $(x,f(x))$ for any $f\in\mathcal{C}$, the algorithm with probability $1-\delta$ outputs a hypothesis $h \in \tilde{C}$ such that $\mathbf{Pr}[h(x)\neq f(x)] \leq \epsilon$).*

Proof. The learning algorithm draws an initial set of $O(\log(1/\delta)/\epsilon^2)$ uniform labeled examples to estimate $|f^{-1}(1)|/2^n$ to within an additive $\pm(\epsilon/4)$ with confidence $1-\delta/3$. If the estimate is less than $3\epsilon/4$ the algorithm outputs the constant -1 hypothesis. Otherwise, by drawing $O(t(n,\epsilon,\delta/3)\cdot\log(1/\delta)/\epsilon))$ uniform labeled examples, with failure probability at most $\delta/3$ it can obtain $t(n,\epsilon,\delta/3)$ positive examples (i.e., points that are uniformly distributed over $f^{-1}(1)$). Finally the learning algorithm can use these points to run the reconstruction algorithm with parameters $\epsilon, \delta/3$ to obtain a hypothesis $h \in \tilde{\mathcal{C}}$ that has $d_{\mathrm{TV}}(\mathcal{U}_{f^{-1}(1)}, \mathcal{U}_{h^{-1}(1)}) \leq \epsilon$ with failure probability at most $\delta/3$. Such a hypothesis h is easily seen to satisfy $\mathbf{Pr}[h(x)\neq f(x)] \leq \epsilon$. □

As described in the following subsection, in this paper we prove negative results for the inverse approximate uniform generation problem for classes such as 3CNF-formulas, monotone 2-CNF formulas, and degree-2 polynomial threshold functions. Since efficient uniform-distribution learning algorithms are known for these classes, these results show that the inverse approximate uniform generation problem is indeed harder than standard uniform-distribution learning for some natural and interesting classes of functions.

The problem of inverse approximate uniform generation is also somewhat reminiscent of the problem of reconstructing Markov Random Fields (MRFs) from random samples [BMS08, DMR06, Mos07]. Much progress has been made on this problem over the past decade, especially when the hidden graph is a tree. However, there does not seem to be a concrete connection between this problem and the problems we study. One reason for this seems to be that in MRF reconstruction, the task is to reconstruct the *model* and not just the distribution; because of this, various conditions need to be imposed in order to guarantee the uniqueness of the underlying model given random samples from the distribution. In contrast, in our setting the explicit goal is to construct a high-accuracy distribution, and it may indeed be the case that there is no unique underlying model (i.e., Boolean function f) given the samples received from the distribution.

1.2 Our results. We give a wide range of both positive and negative results for inverse approximate uniform generation problems. As noted above, most of our results deal with uniform generation of satisfying assignments, i.e., $\mathcal{C}$ is a class of Boolean functions over $\{-1,1\}^n$ and for $f\in\mathcal{C}$ the relation $R(f,y)$ is "y

is a satisfying assignment for f." All the results, both positive and negative, that we present below are for problems of this sort unless indicated otherwise.

Positive results: A general approach and its applications. We begin by presenting a general approach for obtaining inverse approximate uniform generation algorithms. This technique combines approximate uniform generation and counting algorithms and Statistical Query (SQ) learning algorithms with a new type of algorithm called a "densifier," which we introduce and define in Section 3. Very roughly speaking, the densifier lets us prune the entire space $\{-1,1\}^n$ to a set S which (essentially) contains all of $f^{-1}(1)$ and is not too much larger than $f^{-1}(1)$ (so $f^{-1}(1)$ is "dense" in S). By generating approximately uniform elements of S it is possible to run an SQ learning algorithm and obtain a high-accuracy hypothesis which can be used, in conjunction with the approximate uniform generator, to obtain a sampler for a distribution which is close to the uniform distribution over $f^{-1}(1)$. (The approximate counting algorithm is needed for technical reasons which we explain in Section 3.1.) In Section 3 we describe this technique in detail and prove a general result establishing its effectiveness.

In Sections 4 and 5 we give two main applications of this general technique to specific classes of functions. The first of these is the class **LTF** of all LTFs over $\{-1,1\}^n$. Our main technical contribution here is to construct a densifier for LTFs; we do this by carefully combining known efficient online learning algorithms for LTFs (based on interior-point methods for linear programming) [MT94] with known algorithms for approximate uniform generation and counting of satisfying assignments of LTFs [MS04, Dye03]. Given this densifier, our general approach yields the desired inverse approximate uniform generator for LTFs:

Theorem 2. (Informal statement) *There is a $\mathrm{poly}(n,1/\epsilon)$-time algorithm for the inverse problem of approximately uniformly generating satisfying assignments for LTFs.*

Our second main positive result for a specific class, in Section 5, is for the well-studied class $\mathbf{DNF}_{n,s}$ of all size-s DNF formulas over n Boolean variables. Here our main technical contribution is to give a densifier which runs in time $n^{O(\log(s/\epsilon))}$ and outputs a DNF formula. A challenge here is that known SQ algorithms for learning DNF formulas require time exponential in $n^{1/3}$. To get around this, we view the densifier's output DNF as an OR over $n^{O(\log(s/\epsilon))}$ "metavariables" (corresponding to all possible conjunctions that could be present in the DNF output by the densifier), and we show that it is possible to apply known *malicious noise tolerant* SQ algorithms for learning *sparse disjunctions* as the SQ-learning component of our general approach. Since efficient approximate uniform generation and approximate counting algorithms are known [JVV86, KL83] for DNF formulas, with the above densifier and SQ learner we can carry out our general technique, and we thereby obtain our second main positive result for a specific function class:

Theorem 3. (Informal statement) *There is a $n^{O(\log(s/\epsilon))}$-time algorithm for the inverse problem of approximately uniformly generating satisfying assignments for s-term DNF formulas.*

Negative results based on cryptography. In light of the "standard approach," it is clear that in order for an inverse approximate uniform generation problem $(\mathcal{C},R)$ to be computationally hard, it must be the case that either stage (1) (reconstructing the unknown object) or stage (2) (sampling from the reconstructed object) is hard. (If both stages have efficient algorithms $A_{\mathrm{reconstruct}}$ and A_{sample} respectively, then there is an efficient algorithm for the whole inverse approximate uniform generation problem that combines these algorithms according to the standard approach.) Our first approach to obtaining negative results can be used to obtain hardness results for problems for which stage (2), near-uniform sampling, is computationally hard. The approach is based on signature schemes from public-key cryptography; roughly speaking, the general result which we prove is the following (we note that the statement given below is a simplification of our actual result which omits several technical conditions; see Theorem 60 of Section 6.1 for a precise statement):

Theorem 4. (Informal statement) *Let $\mathcal{C}$ be a class of functions such that there is a parsimonious reduction from CIRCUIT-SAT to $\mathcal{C}$-SAT. Then known constructions of secure signature schemes imply that there is* *no subexponential-time algorithm for the inverse problem of approximately uniformly generating satisfying assignments to functions in $\mathcal{C}$.*

This theorem yields a wide range of hardness results for specific classes that show that our positive results (for LTFs and DNF) lie quite close to the boundary of what classes have efficient inverse approximate uniform generation algorithms. We prove:

Corollary 5. (Informal statement) *Under known constructions of secure signature schemes, there is no subexponential-time algorithm for the inverse approximate uniform generation problem for either of the following classes of functions: (i) 3-CNF formulas; (ii) intersections of two halfspaces.*

We show that our signature-scheme-based hardness approach can be extended to settings where there is no parsimonious reduction as described above. Using "blow-up"-type constructions of the sort used to prove hardness of approximate counting, we prove the following:

Theorem 6. (Informal statement) *Under the same assumptions as Corollary 5, there is no subexponential-time algorithm for the inverse approximate uniform generation problem for either of the following classes: (i) monotone 2-CNF; (ii) degree-2 polynomial threshold functions.*

It is instructive to compare the above hardness results with the problem of uniform generation of NP-witnesses. In particular, while it is obvious that no efficient randomized algorithm can produce even a single satisfying assignment of a given 3-SAT instance (assuming $\mathbf{NP} \not\subseteq \mathbf{BPP}$), the seminal results of Jerrum *et al.* [JVV86] showed that given access to an NP-oracle, it is possible to generate approximately uniform satisfying assignments for a given 3-SAT instance. It is interesting to ask whether one requires the full power of adaptive access to NP-oracles for this task, or whether a weaker form of "advice" suffices. Our hardness results can be understood in this context as giving evidence that receiving polynomially many random satisfying assignments of a 3-SAT instance does not help in further uniform generation of satisfying assignments.[1]

Our signature-scheme based approach cannot give hardness results for problems that have polynomial-time algorithms for the "forward" problem of sampling approximately uniform satisfying assignments. Our second approach to proving computational hardness can (at least sometimes) surmount this barrier. The approach is based on Message Authentication Codes in cryptography; the following is an informal statement of our general result along these lines (as before the following statement ignores some technical conditions; see Theorem 80 for a precise statement):

Theorem 7. (Informal statement) *There are known constructions of MACs with the following property: Let $\mathcal{C}$ be a class of circuits such that the verification algorithm of the MAC can be implemented in $\mathcal{C}$. Then there is no subexponential-time inverse approximate uniform generation algorithm for $\mathcal{C}$.*

We instantiate this general result with a specific construction of a MAC that is a slight variant of a construction due to Pietrzak [Pie12]. This specific construction yields a class $\mathcal{C}$ for which the "forward" approximate uniform generation problem is computationally easy, but (under a plausible computational hardness assumption) the inverse approximate uniform generation problem is computationally hard.

The above construction based on MACs shows that there are problems $(\mathcal{C}, R)$ for which the inverse approximate uniform generation problem is computationally hard although the "forward" approximate uniform generation problem is easy. As our last result, we exhibit a group-theoretic problem (based on graph automorphisms) for which the reverse situation holds: under a plausible hardness assumption the *forward*

[1] There is a small caveat here in that we are not given the 3-SAT formula *per se* but rather access to random satisfying assignments of the formula. However, there is a simple elimination based algorithm to reconstruct a high-accuracy approximation for a 3-SAT formula if we have access to random satisfying assignments for the formula.

approximate uniform generation problem is computationally hard, but we give an efficient algorithm for the *inverse* approximate uniform generation problem (which does not follow our general technique or the "standard approach").

Structure of this paper. After the preliminaries in Section 2, we present in Section 3 our general upper bound technique. In Sections 4 and 5 we apply this technique to obtain efficient inverse approximate uniform generation algorithms for LTFs and DNFs respectively. Section 6 contains our hardness results. In Section 7 we give an example of a problem for which approximate uniform generation is hard, while the inverse problem is easy. Finally, in Section 8 we conclude the paper suggesting further directions for future work.

2 Preliminaries and Useful Tools

2.1 Notation and Definitions. For $n \in \mathbb{Z}_+$, we will denote by $[n]$ the set $\{1, \ldots, n\}$. For a distribution D over a finite set $\mathcal{W}$ we denote by $D(x)$, $x \in \mathcal{W}$, the probability mass that D assigns to point x, so $D(x) \geq 0$ and $\sum_{x \in \mathcal{W}} D(x) = 1$. For $S \subseteq \mathcal{W}$, we write $D(S)$ to denote $\sum_{x \in S} D(x)$. For a finite set X we write $x \in_U X$ to indicate that x is chosen uniformly at random from X. For a random variable x, we will write $x \sim D$ to denote that x follows distribution D. Let D, D' be distributions over $\mathcal{W}$. The *total variation distance* between D and D' is $d_{\mathrm{TV}}(D, D') \overset{\text{def}}{=} \max_{S \subseteq \mathcal{W}} |D(S) - D'(S)| = (1/2) \cdot \|D - D'\|_1$, where $\|D - D'\|_1 = \sum_{x \in \mathcal{W}} |D(x) - D'(x)|$ is the L_1–distance between D and D'.

We will denote by $\mathcal{C}_n$, or simply $\mathcal{C}$, a Boolean concept class, i.e., a class of functions mapping $\{-1,1\}^n$ to $\{-1,1\}$. We usually consider syntactically defined classes of functions such as the class of all n-variable linear threshold functions or the class of all n-variable s-term DNF formulas. We stress that throughout this paper a class $\mathcal{C}$ is viewed as a *representation class*. Thus we will say that an algorithm "takes as input a function $f \in \mathcal{C}$" to mean that the input of the algorithm is a *representation* of $f \in \mathcal{C}$.

We will use the notation $\mathcal{U}_n$ (or simply $\mathcal{U}$, when the dimension n is clear from the context) for the uniform distribution over $\{-1,1\}^n$. Let $f : \{-1,1\}^n \to \{-1,1\}$. We will denote by $\mathcal{U}_{f^{-1}(1)}$ the uniform distribution over satisfying assignments of f. Let D be a distribution over $\{-1,1\}^n$ with $0 < D(f^{-1}(1)) < 1$. We write $D_{f,+}$ to denote the conditional distribution D restricted to $f^{-1}(1)$; so for $x \in f^{-1}(1)$ we have $D_{f,+}(x) = D(x)/D(f^{-1}(1))$. Observe that, with this notation, we have that $\mathcal{U}_{f^{-1}(1)} \equiv \mathcal{U}_{f,+}$.

We proceed to define the notions of approximate counting and approximate uniform generation for a class of Boolean functions:

Definition 8 (approximate counting). *Let $\mathcal{C}$ be a class of n-variable Boolean functions. A randomized algorithm $\mathcal{A}^{\mathcal{C}}_{\mathrm{count}}$ is an* efficient approximate counting algorithm for class $\mathcal{C}$, *if for any $\epsilon, \delta > 0$ and any $f \in \mathcal{C}$, on input ϵ, δ and $f \in \mathcal{C}$, it runs in time $\mathrm{poly}(n, 1/\epsilon, \log(1/\delta))$ and with probability $1 - \delta$ outputs a value $\widehat{p}$ such that*

$$\frac{1}{(1+\epsilon)} \cdot \mathbf{Pr}_{x \sim \mathcal{U}}[f(x) = 1] \leq \widehat{p} \leq (1+\epsilon) \cdot \mathbf{Pr}_{x \sim \mathcal{U}}[f(x) = 1].$$

Definition 9 (approximate uniform generation). *Let $\mathcal{C}$ be a class of n-variable Boolean functions. A randomized algorithm $\mathcal{A}^{\mathcal{C}}_{\mathrm{gen}}$ is an* efficient approximate uniform generation algorithm for class $\mathcal{C}$, *if for any $\epsilon > 0$ and any $f \in \mathcal{C}$, there is a distribution $D = D_{f,\epsilon}$ supported on $f^{-1}(1)$ with*

$$\frac{1}{1+\epsilon} \cdot \frac{1}{|f^{-1}(1)|} \leq D(x) \leq (1+\epsilon) \cdot \frac{1}{|f^{-1}(1)|}$$

for each $x \in f^{-1}(1)$, such that for any $\delta > 0$, on input ϵ, δ and $f \in \mathcal{C}$, algorithm $A^{\mathcal{C}}_{\mathrm{gen}}(\epsilon, \delta, f)$ runs in time $\mathrm{poly}(n, 1/\epsilon, \log(1/\delta))$ and either outputs a point $x \in f^{-1}(1)$ that is distributed precisely according to $D = D_{f,\epsilon}$, or outputs $\perp$. Moreover the probability that it outputs $\perp$ is at most δ.

An approximate uniform generation algorithm is said to be *fully polynomial* if its running time dependence on ϵ is $\mathrm{poly}(\log(1/\epsilon))$.

Before we define our inverse approximate uniform generation problem, we need the notion of a *sampler* for a distribution:

Definition 10. *Let D be a distribution over $\{-1,1\}^n$. A* sampler for D *is a circuit C with $m = \mathrm{poly}(n)$ input bits $z \in \{-1,1\}^m$ and n output bits $x \in \{-1,1\}^n$ which is such that when $z \sim \mathcal{U}_m$ then $x \sim D$. For $\epsilon > 0$, an* ϵ-sampler for D *is a sampler for some distribution D' which has $d_{\mathrm{TV}}(D', D) \le \epsilon$.*

For clarity we sometimes write "C is a 0-sampler for D" to emphasize the fact that the outputs of $C(z)$ are distributed *exactly* according to distribution D. We are now ready to formally define the notion of an inverse approximate uniform generation algorithm:

Definition 11 (inverse approximate uniform generation)**.** *Let $\mathcal{C}$ be a class of n-variable Boolean functions. A randomized algorithm $\mathcal{A}^{\mathcal{C}}_{\mathrm{inv}}$ is an* inverse approximate uniform generation algorithm for class $\mathcal{C}$*, if for any $\epsilon, \delta > 0$ and any $f \in \mathcal{C}$, on input ϵ, δ and sample access to $\mathcal{U}_{f^{-1}(1)}$, with probability $1 - \delta$ algorithm $A^{\mathcal{C}}_{\mathrm{inv}}$ outputs an ϵ-sampler C_f for $\mathcal{U}_{f^{-1}(1)}$.*

2.2 Hypothesis Testing. Our general approach works by generating a collection of hypothesis distributions, one of which is close to the target distribution $\mathcal{U}_{f^{-1}(1)}$. Thus, we need a way to select a high-accuracy hypothesis distribution from a pool of candidate distributions which contains at least one high-accuracy hypothesis. This problem has been well studied, see e.g. Chapter 7 of [DL01]. We use the following result which is an extension of Lemma C.1 of [DDS12a].

Proposition 12. *Let D be a distribution over a finite set $\mathcal{W}$ and $\mathcal{D}_\epsilon = \{D_j\}_{j=1}^N$ be a collection of N distributions over $\mathcal{W}$ with the property that there exists $i \in [N]$ such that $d_{\mathrm{TV}}(D, D_i) \le \epsilon$. There is an algorithm $\mathcal{T}^D$, which is given access to:*

(i) samplers for D and D_k, for all $k \in [N]$,

(ii) a $(1+\beta)$–approximate evaluation oracle $\mathrm{EVAL}_{D_k}(\beta)$, for all $k \in [N]$, which, on input $w \in \mathcal{W}$, deterministically outputs a value $\widetilde{D}^\beta_k(w)$, such that $D_k(w)/(1+\beta) \le \widetilde{D}^\beta_k(w) \le (1+\beta)D_k(w)$, where $\beta > 0$ is any parameter satisfying $(1+\beta)^2 \le 1 + \epsilon/8$,

an accuracy parameter ϵ and a confidence parameter δ, and has the following behavior: It makes

$$m = O\left((1/\epsilon^2) \cdot (\log N + \log(1/\delta))\right)$$

draws from D and from each D_k, $k \in [N]$, and $O(m)$ calls to each oracle $\mathrm{EVAL}_{D_k}(\beta)$, $k \in [N]$, performs $O(mN^2)$ arithmetic operations, and with probability $1 - \delta$ outputs an index $i^\star \in [N]$ that satisfies $d_{\mathrm{TV}}(D, D_{i^\star}) \le 6\epsilon$.

Before we proceed with the proof, we note that there are certain crucial differences between the current setting and the setting of [DDS12a, DDS12b] (as well as other related works that use versions of Proposition 12). In particular, in our setting, the set $\mathcal{W}$ is of size 2^n, which was not the case in [DDS12a, DDS12b]. Hence, we cannot assume the distributions D_i are given explicitly in the input. Thus Proposition 12 carefully specifies what kind of access to these distributions is required. Proposition 12 is an extension of similar results in the previous works; while the idea of the proof is essentially the same, the details are more involved.

Proof of Proposition 12. At a high level, the algorithm $\mathcal{T}^D$ performs a tournament by running a "competition" $\texttt{Choose-Hypothesis}^D$ for every pair of distinct distributions in the collection $\mathcal{D}_\epsilon$. It outputs a distribution $D^\star \in \mathcal{D}_\epsilon$ that was never a loser (i.e., won or achieved a draw in all its competitions). If no such distribution exists in $\mathcal{D}_\epsilon$ then the algorithm outputs "failure." We start by describing and analyzing the competition subroutine between a pair of distributions in the collection.

Lemma 13. *In the context of Proposition 12, there is an algorithm* $\texttt{Choose-Hypothesis}^D(D_i, D_j, \epsilon', \delta')$ *which is given access to*

(i) independent samples from D and D_k, for $k \in \{i, j\}$,

(ii) an evaluation oracle $\mathrm{EVAL}_{D_k}(\beta)$, for $k \in \{i, j\}$,

an accuracy parameter ϵ' and a confidence parameter δ', and has the following behavior: It uses $m' = O\left((1/\epsilon'^2)\log(1/\delta')\right)$ samples from each of D, D_i and D_j, it makes $O(m')$ calls to the oracles $\mathrm{EVAL}_{D_k}(\beta)$, $k \in \{i, j\}$, performs $O(m')$ arithmetic operations, and if some D_k, $k \in \{i, j\}$, has $d_{\mathrm{TV}}(D_k, D) \leq \epsilon'$ then with probability $1 - \delta'$ it outputs an index $k^\star \in \{i, j\}$ that satisfies $d_{\mathrm{TV}}(D, D_{k^\star}) \leq 6\epsilon'$.

Proof. To set up the competition between D_i and D_j, we consider the following subset of $\mathcal{W}$:

$$H_{ij} = H_{ij}(D_i, D_j) \stackrel{\text{def}}{=} \{w \in \mathcal{W} \mid D_i(w) \geq D_j(w)\}$$

and the corresponding probabilities $p_{i,j} \stackrel{\text{def}}{=} D_i(H_{ij})$ and $q_{i,j} \stackrel{\text{def}}{=} D_j(H_{ij})$. Clearly, it holds $p_{i,j} \geq q_{i,j}$ and by definition of the total variation distance we can write

$$d_{\mathrm{TV}}(D_i, D_j) = p_{i,j} - q_{i,j}.$$

For the purposes of our algorithm, we would ideally want oracle access to the set H_{ij}. Unfortunately though, this is not possible since the evaluation oracles are only approximate. Hence, we will need to define a more robust version of the set H_{ij} which will turn out to have similar properties. In particular, we consider the set

$$H_{ij}^\beta \stackrel{\text{def}}{=} \{w \in \mathcal{W} \mid \widetilde{D}_i^\beta(w) \geq \widetilde{D}_j^\beta(w)\}$$

and the corresponding probabilities $p_{i,j}^\beta \stackrel{\text{def}}{=} D_i(H_{ij}^\beta)$ and $q_{i,j}^\beta \stackrel{\text{def}}{=} D_j(H_{ij}^\beta)$. We claim that the difference $\Delta \stackrel{\text{def}}{=} p_{i,j}^\beta - q_{i,j}^\beta$ is an accurate approximation to $d_{\mathrm{TV}}(D_i, D_j)$. In particular, we show:

Claim 14. *We have*

$$\Delta \leq d_{\mathrm{TV}}(D_i, D_j) \leq \Delta + \epsilon/4. \tag{1}$$

Before we proceed with the proof, we stress that (1) crucially uses our assumption that the evaluation oracles provide a *multiplicative* approximation to the exact probabilities.

Proof. To show (1) we proceed as follows: Let $A = H_{ij} \cap H_{ij}^\beta$, $B = H_{ij} \cap \overline{H_{ij}^\beta}$ and $C = \overline{H_{ij}} \cap H_{ij}^\beta$. Then we can write

$$d_{\mathrm{TV}}(D_i, D_j) = (D_i - D_j)(A) + (D_i - D_j)(B)$$

and

$$\Delta = (D_i - D_j)(A) + (D_i - D_j)(C).$$

We will show that

$$0 \leq (D_i - D_j)(B) \leq \epsilon/8 \tag{2}$$

and similarly

$$-\epsilon/8 \le (D_i - D_j)(C) \le 0 \tag{3}$$

from which the claim follows. We proceed to prove (2), the proof of (3) being very similar. Let $w \in B$. Then $D_i(w) \ge D_j(w)$ (since $w \in H_{ij}$) which gives $(D_i - D_j)(B) \ge 0$, establishing the LHS of (2). We now establish the RHS. For $w \in B$ we also have that $\widetilde{D}_i^{\beta}(w) < \widetilde{D}_j^{\beta}(w)$ (since $w \in \overline{H_{ij}^{\beta}}$). Now by the definition of the evaluation oracles, it follows that $\widetilde{D}_i^{\beta}(w) \ge \frac{D_i(w)}{(1+\beta)}$ and $\widetilde{D}_j^{\beta}(w) \le (1+\beta)D_j(w)$. Combining these inequalities yields

$$D_i(w) \le (1+\beta)^2 D_j(w) \le (1+\epsilon/8) D_j(w)$$

where the second inequality follows by our choice of β. Therefore,

$$(D_i - D_j)(B) = \sum_{w \in B} (D_i(w) - D_j(w)) \le (\epsilon/8) \cdot D_j(B) \le \epsilon/8$$

as desired. □

Note that the probabilities $p_{i,j}^{\beta}$ and $q_{i,j}^{\beta}$ are not available to us explicitly. Hence, `Choose-Hypothesis` requires a way to empirically estimate each of these probability values (up to a small additive accuracy). This task can be done efficiently because we have sample access to the distributions D_i, D_j and oracle access to the set H_{ij}^{β} thanks to the $\mathrm{EVAL}_{D_k}(\beta)$ oracles. The following claim provides the details:

Claim 15. *There exists a subroutine* `Estimate`$(D_i, H_{ij}^{\beta}, \gamma, \delta)$ *which is given access to*

(i) independent samples from D_i,

(ii) an evaluation oracle $\mathrm{EVAL}_{D_k}(\beta)$, for $k \in \{i, j\}$,

an accuracy parameter γ and a confidence parameter δ, and has the following behavior: It makes $m = O\big((1/\gamma^2)\log(1/\delta)\big)$ draws from D_i and $O(m)$ calls to the oracles $\mathrm{EVAL}_{D_k}(\beta)$, $k = i, j$, performs $O(m)$ arithmetic operations, and with probability $1-\delta$ outputs a number $\widetilde{p}_{i,j}^{\beta}$ such that $|\widetilde{p}_{i,j}^{\beta} - p_{i,j}^{\beta}| \le \gamma$.

Proof. The desired subroutine amounts to a straightforward random sampling procedure, which we include here for the sake of completeness. We will use the following elementary fact, a simple consequence of the additive Chernoff bound.

Fact 16. *Let X be a random variable taking values in the range $[-1, 1]$. Then $\mathbf{E}[X]$ can be estimated to within an additive $\pm\tau$, with confidence probability $1-\delta$, using $m = \Omega((1/\tau^2)\log(1/\delta))$ independent samples from X. In particular, the empirical average $\widehat{X}_m = (1/m)\sum_{i=1}^{m} X_i$, where the X_i's are independent samples of X, satisfies $\mathbf{Pr}\left[|\widehat{X}_m - \mathbf{E}[X]| \le \tau\right] \ge 1-\delta$.*

We shall refer to this as "empirically estimating" the value of $\mathbf{E}[X]$.

Consider the indicator function $I_{H_{ij}^{\beta}}$ of the set H_{ij}^{β}, i.e., $I_{H_{ij}^{\beta}} : \mathcal{W} \to \{0, 1\}$ with $I_{H_{ij}^{\beta}}(x) = 1$ if and only if $x \in H_{ij}^{\beta}$. It is clear that $\mathbf{E}_{x \sim D_i}\left[I_{H_{ij}^{\beta}}(x)\right] = D_i(H_{ij}^{\beta}) = p_{i,j}^{\beta}$. The subroutine is described in the following pseudocode:

```
Subroutine Estimate$(D_i, H_{ij}^{\beta}, \gamma, \delta)$:
Input: Sample access to $D_i$ and oracle access to $\mathrm{EVAL}_{D_k}(\beta)$, $k = i, j$.
Output: A number $\widetilde{p}_{ij}^{\beta}$ such that with probability $1-\delta$ it holds $|\widetilde{p}_{ij}^{\beta} - D_i(H_{ij}^{\beta})| \le \gamma$.
```

```
1. Draw $m = \Theta\left((1/\gamma^2)\log(1/\delta)\right)$ samples $\mathbf{s} = \{s_\ell\}_{\ell=1}^m$ from $D_i$.
2. For each sample $s_\ell$, $\ell \in [m]$:
    (a) Use the oracles $\mathrm{EVAL}_{D_i}(\beta)$, $\mathrm{EVAL}_{D_j}(\beta)$, to approximately evaluate $D_i(s_\ell)$, $D_j(s_\ell)$.
    (b) If $\widetilde{D}_i^\beta(s_\ell) \geq \widetilde{D}_j^\beta(s_\ell)$ set $I_{H_{ij}^\beta}(s_\ell) = 1$, otherwise $I_{H_{ij}^\beta}(s_\ell) = 0$.
3. Set $\widetilde{p}_{ij}^\beta = \frac{1}{m}\sum_{\ell=1}^m I_{H_{ij}^\beta}(s_\ell)$.
4. Output $\widetilde{p}^\beta{}_{ij}$.
```

The computational efficiency of this simple random sampling procedure follows from the fact that we can efficiently decide membership in H_{ij}^β. To do this, for a given $x \in \mathcal{W}$, we make a query to each of the oracles $\mathrm{EVAL}_{D_i}(\beta)$, $\mathrm{EVAL}_{D_j}(\beta)$ to obtain the probabilities $\widetilde{D}_i^\beta(x)$, $\widetilde{D}_j^\beta(x)$. We have that $x \in H_{ij}^\beta$ (or equivalently $I_{H_{ij}^\beta}(x) = 1$) if and only if $\widetilde{D}_i^\beta(x) \geq \widetilde{D}_j^\beta(x)$. By Fact 16, applied for the random variable $I_{H_{ij}^\beta}(x)$, where $x \sim D_i$, after $m = \Omega((1/\gamma^2)\log(1/\delta))$ samples from D_i we obtain a $\pm\gamma$-additive estimate to $p_{i,j}^\beta$ with probability $1-\delta$. For each sample, we make one query to each of the oracles, hence the total number of oracle queries is $O(m)$ as desired. The only non-trivial arithmetic operations are the $O(m)$ comparisons done in Step 2(b), and Claim 15 is proved. □

Now we are ready to prove Lemma 13. The algorithm `Choose-Hypothesis`$^D(D_i, D_j, \epsilon', \delta')$ performing the competition between D_i and D_j is the following:

```
Algorithm Choose-Hypothesis$^D(D_i, D_j, \epsilon', \delta')$:
**Input:** Sample access to $D$ and $D_k$, $k = i, j$, oracle access to $\mathrm{EVAL}_{D_k}(\beta)$, $k = i, j$.
1. Set $\widetilde{p}_{i,j}^\beta$ =Estimate$(D_i, H_{ij}^\beta, \epsilon'/8, \delta'/4)$ and $\widetilde{q}_{i,j}^\beta$ =Estimate$(D_j, H_{ij}^\beta, \epsilon'/8, \delta'/4)$.
2. If $\widetilde{p}_{i,j}^\beta - \widetilde{q}_{i,j}^\beta \leq 9\epsilon'/2$, declare a draw and return either $i$ or $j$. Otherwise:
3. Draw $m' = \Theta\left((1/\epsilon'^2)\log(1/\delta')\right)$ samples $\mathbf{s}' = \{s_\ell\}_{\ell=1}^{m'}$ from $D$.
4. For each sample $s_\ell$, $\ell \in [m']$:
    (a) Use the oracles $\mathrm{EVAL}_{D_i}(\beta)$, $\mathrm{EVAL}_{D_j}(\beta)$ to evaluate $\widetilde{D}_i^\beta(s_\ell)$, $\widetilde{D}_j^\beta(s_\ell)$.
    (b) If $\widetilde{D}_i^\beta(s_\ell) \geq \widetilde{D}_j^\beta(s_\ell)$ set $I_{H_{ij}^\beta}(s_\ell) = 1$, otherwise $I_{H_{ij}^\beta}(s_\ell) = 0$.
5. Set $\tau = \frac{1}{m'}\sum_{\ell=1}^{m'} I_{H_{ij}^\beta}(s_\ell)$, i.e., $\tau$ is the fraction of samples that fall inside $H_{ij}^\beta$.
6. If $\tau > \widetilde{p}_{i,j}^\beta - \frac{13}{8}\epsilon'$, declare $D_i$ as winner and return $i$; otherwise,
7. if $\tau < \widetilde{q}_{i,j}^\beta + \frac{13}{8}\epsilon'$, declare $D_j$ as winner and return $j$; otherwise,
8. declare a draw and return either $i$ or $j$.
```

It is not hard to check that the outcome of the competition does not depend on the ordering of the pair of distributions provided in the input; that is, on inputs (D_i, D_j) and (D_j, D_i) the competition outputs the same result for a fixed set of samples $\{s_1, \ldots, s_{m'}\}$ drawn from D.

The upper bounds on sample complexity, query complexity and number of arithmetic operations can be straightforwardly verified. Hence, it remains to show correctness. By Claim 15 and a union bound, with probability at least $1 - \delta'/2$, we will have that $|\widetilde{p}^{\beta}_{i,j} - p^{\beta}_{i,j}| \le \epsilon'/8$ and $|\widetilde{q}^{\beta}_{i,j} - q^{\beta}_{i,j}| \le \epsilon'/8$. In the following, we condition on this good event. The correctness of `Choose-Hypothesis` is then an immediate consequence of the following claim.

Claim 17. *Suppose that $d_{\mathrm{TV}}(D, D_i) \le \epsilon'$. Then:*

(i) If $d_{\mathrm{TV}}(D, D_j) > 6\epsilon'$, then the probability that the competition between D_i and D_j does not declare D_i as the winner is at most $e^{-m'\epsilon'^2/8}$. (Intuitively, if D_j is very far from D then it is very likely that D_i will be declared winner.)

(ii) The probability that the competition between D_i and D_j declares D_j as the winner is at most $e^{-m'\epsilon'^2/8}$. (Intuitively, since D_i is close to D, a draw with some other D_j is possible, but it is very unlikely that D_j will be declared winner.)

Proof. Let $r^{\beta} = D(H^{\beta}_{ij})$. The definition of the variation distance implies that $|r^{\beta} - p^{\beta}_{i,j}| \le d_{\mathrm{TV}}(D, D_i) \le \epsilon'$. Therefore, we have that $|r^{\beta} - \widetilde{p}^{\beta}_{i,j}| \le |r^{\beta} - p^{\beta}_{i,j}| + |\widetilde{p}^{\beta}_{i,j} - p^{\beta}_{i,j}| \le 9\epsilon'/8$. Consider the indicator (0/1) random variables $\{Z_\ell\}_{\ell=1}^{m'}$ defined as $Z_\ell = 1$ if and only if $s_\ell \in H^{\beta}_{ij}$. Clearly, $\tau = \frac{1}{m'}\sum_{\ell=1}^{m'} Z_\ell$ and $\mathbf{E}_{\mathbf{s}'}[\tau] = \mathbf{E}_{s_\ell \sim D}[Z_\ell] = r^{\beta}$. Since the Z_ℓ's are mutually independent, it follows from the Chernoff bound that $\mathbf{Pr}[\tau \le r^{\beta} - \epsilon'/2] \le e^{-m'\epsilon'^2/8}$. Using $|r^{\beta} - \widetilde{p}^{\beta}_{i,j}| \le 9\epsilon'/8$. we get that $\mathbf{Pr}[\tau \le \widetilde{p}^{\beta}_{i,j} - 13\epsilon'/8] \le e^{-m'\epsilon'^2/8}$.

- For part (i): If $d_{\mathrm{TV}}(D, D_j) > 6\epsilon'$, from the triangle inequality we get that $p_{i,j} - q_{i,j} = d_{\mathrm{TV}}(D_i, D_j) > 5\epsilon'$ Claim 14 implies that $p^{\beta}_{i,j} - q^{\beta}_{i,j} > 19\epsilon'/4$ and our conditioning finally gives $\widetilde{p}^{\beta}_{i,j} - \widetilde{q}^{\beta}_{i,j} > 9\epsilon'/2$. Hence, the algorithm will go beyond Step 2, and with probability at least $1 - e^{-m'\epsilon'^2/8}$, it will stop at Step 6, declaring D_i as the winner of the competition between D_i and D_j.

- For part (ii): If $\widetilde{p}^{\beta}_{i,j} - \widetilde{q}^{\beta}_{i,j} \le 9\epsilon'/2$ then the competition declares a draw, hence D_j is not the winner. Otherwise we have $\widetilde{p}^{\beta}_{i,j} - \widetilde{q}^{\beta}_{i,j} > 9\epsilon'/2$ and the argument of the previous paragraph implies that the competition between D_i and D_j will declare D_j as the winner with probability at most $e^{-m'\epsilon'^2/8}$.

This concludes the proof of Claim 17. □

This completes the proof of Lemma 13. □

We now proceed to describe the algorithm $\mathcal{T}^D$ and establish Proposition 12. The algorithm performs a tournament by running the competition $\texttt{Choose-Hypothesis}^D(D_i, D_j, \epsilon, \delta/(2N))$ for every pair of distinct distributions D_i, D_j in the collection $\mathcal{D}_\epsilon$. It outputs a distribution $D^\star \in \mathcal{D}_\epsilon$ that was never a loser (i.e., won or achieved a draw in all its competitions). If no such distribution exists in $\mathcal{D}_\epsilon$ then the algorithm outputs "failure." A detailed pseudocode follows:

```
Algorithm $\mathcal{T}^D(\{D_j\}_{j=1}^N, \epsilon, \delta)$:
Input: Sample access to $D$ and $D_k$, $k \in [N]$, and oracle access to $\mathrm{EVAL}_{D_k}$, $k \in [N]$.

  1. Draw $m = \Theta\left((1/\epsilon^2)(\log N + \log(1/\delta))\right)$ samples from $D$ and each $D_k$, $k \in [N]$.
```

2. For all $i, j \in [N]$, $i \neq j$, run $\texttt{Choose-Hypothesis}^D(D_i, D_j, \epsilon, \delta/(2N))$ using this sample.
3. Output an index $i^\star$ such that $D_{i^\star}$ was never declared a loser, if one exists.
4. Otherwise, output "failure".

We now proceed to analyze the algorithm. The bounds on the sample complexity, running time and query complexity to the evaluation oracles follow from the corresponding bounds for $\texttt{Choose-Hypothesis}$. Hence, it suffices to show correctness. We do this below.

By definition, there exists some $D_i \in \mathcal{D}_\epsilon$ such that $d_{\mathrm{TV}}(D, D_i) \leq \epsilon$. By Claim 17, the distribution D_i never loses a competition against any other $D_j \in \mathcal{D}_\epsilon$ (so the algorithm does not output "failure"). A union bound over all N distributions in $\mathcal{D}_\epsilon$ shows that with probability $1 - \delta/2$, the distribution D' never loses a competition.

We next argue that with probability at least $1 - \delta/2$, every distribution $D_j \in \mathcal{D}_\epsilon$ that never loses has small variation distance from D. Fix a distribution D_j such that $d_{\mathrm{TV}}(D_j, D) > 6\epsilon$; Claim 17(i) implies that D_j loses to D_i with probability $1 - 2e^{-m\epsilon^2/8} \geq 1 - \delta/(2N)$. A union bound yields that with probability $1 - \delta/2$, every distribution D_j that has $d_{\mathrm{TV}}(D_j, D) > 6\epsilon$ loses some competition.

Thus, with overall probability at least $1 - \delta$, the tournament does not output "failure" and outputs some distribution $D^\star$ such that $d_{\mathrm{TV}}(D, D^\star)$ is at most 6ϵ. The proof of Proposition 12 is now complete. □

Remark 18. As stated Proposition 12 assumes that algorithm $\mathcal{T}^D$ has access to samplers for all the distributions D_k, so each call to such a sampler is guaranteed to output an element distributed according to D_k. Let $D_k^\perp$ be a distribution over $\mathcal{W} \cup \{\perp\}$ which is such that (i) $D_k^\perp(\perp) \leq 1/2$, and (ii) the conditional distribution $(D_k^\perp)_{\mathcal{W}}$ of $D_k^\perp$ conditioned on not outputting $\perp$ is precisely D_k. It is easy to see that the proof of Proposition 12 extends to a setting in which $\mathcal{T}^D$ has access to samplers for $D_k^\perp$ rather than samplers for D_k; each time a sample from D_k is required the algorithm can simply invoke the sampler for $D_k^\perp$ repeatedly until an element other than $\perp$ is obtained. (The low-probability event that many repetitions are ever needed can be "folded into" the failure probability δ.)

3 A general technique for inverse approximate uniform generation

In this section we present a general technique for solving inverse approximate uniform generation problems. Our main positive results follow this conceptual framework. At the heart of our approach is a new type of algorithm which we call a *densifier* for a concept class $\mathcal{C}$. Roughly speaking, this is an algorithm which, given uniform random positive examples of an unknown $f \in \mathcal{C}$, constructs a set S which (essentially) contains all of $f^{-1}(1)$ and which is such that $f^{-1}(1)$ is "dense" in S. Our main result in this section, Theorem 21, states (roughly speaking) that the existence of (i) a computationally efficient densifier, (ii) an efficient approximate uniform generation algorithm, (iii) an efficient approximate counting algorithm, and (iv) an efficient *statistical query (SQ)* learning algorithm, together suffice to yield an efficient algorithm for our inverse approximate uniform generation problem.

We have already defined approximate uniform generation and approximate counting algorithms, so we need to define SQ learning algorithms and densifiers. The *statistical query* (SQ) learning model is a natural restriction of the PAC learning model in which a learning algorithm is allowed to obtain estimates of statistical properties of the examples but cannot directly access the examples themselves. Let D be a distribution over $\{-1, 1\}^n$. In the SQ model [Kea98], the learning algorithm has access to a *statistical query oracle*, $\mathrm{STAT}(f, D)$, to which it can make a query of the form (χ, τ), where $\chi : \{-1, 1\}^n \times \{-1, 1\} \to$

$[-1,1]$ is the *query function* and $\tau > 0$ is the *tolerance*. The oracle responds with a value v such that $|\mathbf{E}_{x\sim D}\left[\chi\left(x, f(x)\right)\right] - v| \leq \tau$, where $f \in \mathcal{C}$ is the target concept. The goal of the algorithm is to output a hypothesis $h : \{-1,1\}^n \to \{-1,1\}$ such that $\mathbf{Pr}_{x\sim D}[h(x) \neq f(x)] \leq \epsilon$. The following is a precise definition:

Definition 19. *Let $\mathcal{C}$ be a class of n-variable boolean functions and D be a distribution over $\{-1,1\}^n$. An* SQ learning algorithm for $\mathcal{C}$ under D *is a randomized algorithm $\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ}}$ that for every $\epsilon, \delta > 0$, every target concept $f \in \mathcal{C}$, on input ϵ, δ and with access to oracle $\mathrm{STAT}(f, D)$ and to independent samples drawn from D, outputs with probability $1-\delta$ a hypothesis $h : \{-1,1\}^n \to \{-1,1\}$ such that $\mathbf{Pr}_{x\sim D}[h(x) \neq f(x)] \leq \epsilon$. Let $t_1(n, 1/\epsilon, 1/\delta)$ be the running time of $\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ}}$ (assuming each oracle query is answered in unit time), $t_2(n)$ be the maximum running time to evaluate any query provided to $\mathrm{STAT}(f, D)$ and $\tau(n, 1/\epsilon)$ be the minimum value of the tolerance parameter ever provided to $\mathrm{STAT}(f, D)$ in the course of $\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ}}$'s execution. We say that $\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ}}$ is* efficient *(and that $\mathcal{C}$ is* efficiently SQ *learnable with respect to distribution D), if $t_1(n, 1/\epsilon, 1/\delta)$ is polynomial in n, $1/\epsilon$ and $1/\delta$, $t_2(n)$ is polynomial in n and $\tau(n, 1/\epsilon)$ is lower bounded by an inverse polynomial in n and $1/\epsilon$. We call an* SQ *learning algorithm $\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ}}$ for $\mathcal{C}$* distribution independent *if $\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ}}$ succeeds for any distribution D. If $\mathcal{C}$ has an efficient distribution independent* SQ *learning algorithm we say that $\mathcal{C}$ is* efficiently SQ learnable (distribution independently).

We sometimes write an "(ϵ, δ)–SQ learning algorithm" to explicitly state the accuracy parameter ϵ and confidence parameter Throughout this paper, we will only deal with distribution independent SQ learning algorithms.

To state our main result, we introduce the notion of a *densifier* for a class $\mathcal{C}$ of Boolean functions. Intuitively, a densifier is an algorithm which is given access to samples from $\mathcal{U}_{f^{-1}(1)}$ (where f is an unknown element of $\mathcal{C}$) and outputs a subset $S \subseteq \{-1,1\}^n$ which is such that (i) S contains "almost all" of $f^{-1}(1)$, but (ii) S is "much smaller" than $\{-1,1\}^n$ – in particular it is small enough that $f^{-1}(1) \cap S$ is (at least moderately) "dense" in S.

Definition 20. *Fix a function $\gamma(n, 1/\epsilon, 1/\delta)$ taking values in $(0,1]$ and a class $\mathcal{C}$ of n-variable Boolean functions. An algorithm $\mathcal{A}^{(\mathcal{C},\mathcal{C}')}_{\mathrm{den}}$ is said to be* a γ-densifier for function class $\mathcal{C}$ using class $\mathcal{C}'$ *if it has the following behavior: For every $\epsilon, \delta > 0$, every $1/2^n \leq \widehat{p} \leq 1$, and every $f \in \mathcal{C}$, given as input $\epsilon, \delta, \widehat{p}$ and a set of independent samples from $\mathcal{U}_{f^{-1}(1)}$, the following holds: Let $p \overset{def}{=} \mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x) = 1]$. If $p \leq \widehat{p} < (1+\epsilon)p$, then with probability at least $1-\delta$, algorithm $\mathcal{A}^{(\mathcal{C},\mathcal{C}')}_{\mathrm{den}}$ outputs a function $g \in \mathcal{C}'$ such that:*

(a) $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}\left[g(x) = 1\right] \geq 1-\epsilon$.

(b) $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}\left[f(x) = 1\right] \geq \gamma(n, 1/\epsilon, 1/\delta)$.

We will sometimes write an "$(\epsilon, \gamma, \delta)$–densifier" to explicitly state the parameters in the definition.

Our main conceptual approach is summarized in the following theorem:

Theorem 21 (General Upper Bound)**.** *Let $\mathcal{C}, \mathcal{C}'$ be classes of n-variable boolean functions. Suppose that*

- *$\mathcal{A}^{(\mathcal{C},\mathcal{C}')}_{\mathrm{den}}$ is an $(\epsilon, \gamma, \delta)$-densifier for $\mathcal{C}$ using $\mathcal{C}'$ running in time $T_{\mathrm{den}}(n, 1/\epsilon, 1/\delta)$.*
- *$\mathcal{A}^{\mathcal{C}'}_{\mathrm{gen}}$ is an (ϵ, δ)-approximate uniform generation algorithm for $\mathcal{C}'$ running in time $T_{\mathrm{gen}}(n, 1/\epsilon, 1/\delta)$.*
- *$\mathcal{A}^{\mathcal{C}'}_{\mathrm{count}}$ is an (ϵ, δ)-approximate counting algorithm for $\mathcal{C}'$ running in time $T_{\mathrm{count}}(n, 1/\epsilon, 1/\delta)$.*
- *$\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ}}$ is an (ϵ, δ)-*SQ *learning algorithm for $\mathcal{C}$ such that: $\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ}}$ runs in time $t_1(n, 1/\epsilon, 1/\delta)$, $t_2(n)$ is the maximum time needed to evaluate any query provided to $\mathrm{STAT}(f, D)$, and $\tau(n, 1/\epsilon)$ is the minimum value of the tolerance parameter ever provided to $\mathrm{STAT}(f, D)$ in the course of $\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ}}$'s execution.*

Then there exists an inverse approximate uniform generation algorithm $\mathcal{A}_{\text{inv}}^{\mathcal{C}}$ for $\mathcal{C}$. The running time of $\mathcal{A}_{\text{inv}}^{\mathcal{C}}$ is polynomial in $T_{\text{den}}(n, 1/\epsilon, 1/\delta)$, $1/\gamma$, $T_{\text{gen}}(n, 1/\epsilon, 1/\delta)$, $T_{\text{count}}\ (n, 1/\epsilon, 1/\delta)$, $t_1(n, 1/\epsilon, 1/\delta)$, $t_2(n)$ and $1/\tau(n, 1/\epsilon)$. [2]

Sketch of the algorithm. The inverse approximate uniform generation algorithm $\mathcal{A}_{\text{inv}}^{\mathcal{C}}$ for $\mathcal{C}$ works in three main conceptual steps. Let $f \in \mathcal{C}$ be the unknown target function and recall that our algorithm $\mathcal{A}_{\text{inv}}^{\mathcal{C}}$ is given access to samples from $\mathcal{U}_{f^{-1}(1)}$.

(1) In the first step, $\mathcal{A}_{\text{inv}}^{\mathcal{C}}$ runs the densifier $\mathcal{A}_{\text{den}}^{(\mathcal{C},\mathcal{C}')}$ on a set of samples from $\mathcal{U}_{f^{-1}(1)}$. Let $g \in \mathcal{C}'$ be the output function of $\mathcal{A}_{\text{den}}^{(\mathcal{C},\mathcal{C}')}$.

Note that by setting the input to the approximate uniform generation algorithm $\mathcal{A}_{\text{gen}}^{\mathcal{C}'}$ to g, we obtain an approximate sampler C_g for $\mathcal{U}_{g^{-1}(1)}$. The output distribution D' of this sampler, is by definition supported on $g^{-1}(1)$ and is close to $D = \mathcal{U}_{g^{-1}(1)}$ in total variation distance.

(2) The second step is to run the SQ-algorithm $\mathcal{A}_{\text{SQ}}^{\mathcal{C}}$ to learn the function $f \in C$ under the distribution D. Let h be the hypothesis constructed by $\mathcal{A}_{\text{SQ}}^{\mathcal{C}}$.

(3) In the third and final step, the algorithm simply samples from C_g until it obtains an example x that has $h(x) = 1$, and outputs this x.

Remark 22. The reader may have noticed that the above sketch does not seem to use the approximate counting algorithm $\mathcal{A}_{\text{count}}^{\mathcal{C}'}$; we will revisit this point below.

Remark 23. The connection between the above algorithm sketch and the "standard approach" discussed in the Introduction is as follows: The function $g \wedge h$ essentially corresponds to the reconstructed object $\tilde{x}$ of the "standard approach." The process of sampling from C_g and doing rejection sampling until an input that satisfies h is obtained, essentially corresponds to the A_{sample} procedure of the "standard approach."

3.1 Intuition, motivation and discussion. To motivate the high-level idea behind our algorithm, consider a setting in which $f^{-1}(1)$ is only a tiny fraction (say $1/2^{\Theta(n)}$) of $\{-1,1\}^n$. It is intuitively clear that we would like to use some kind of a learning algorithm in order to come up with a good approximation of $f^{-1}(1)$, but we need this approximation to be accurate at the "scale" of $f^{-1}(1)$ itself rather than at the scale of all of $\{-1,1\}^n$, so we need some way to ensure that the learning algorithm's hypothesis is accurate at this small scale. By using a densifier to construct g such that $g^{-1}(1)$ is not too much larger than $f^{-1}(1)$, we can use the distribution $D = \mathcal{U}_{g^{-1}(1)}$ to run a learning algorithm and obtain a good approximation of $f^{-1}(1)$ at the desired scale. (Since D and D' are close in variation distance, this implies we also learn f with respect to D'.)

To motivate our use of an SQ learning algorithm rather than a standard PAC learning algorithm, observe that there seems to be no way to obtain correctly labeled examples distributed according to D. However, we show that it is possible to accurately simulate statistical queries under D having access only to random positive examples from $f^{-1}(1)$ and to unlabeled examples drawn from D (subject to additional technical caveats discussed below). We discuss the issue of how it is possible to successfully use an SQ learner in our setting in more detail below.

Discussion and implementation issues. While the three main conceptual steps (1)-(3) of our algorithm may (hopefully) seem quite intuitive in light of the preceding motivation, a few issues immediately arise in thinking about how to implement these steps. The first one concerns running the SQ-algorithm $\mathcal{A}_{\text{SQ}}^{\mathcal{C}}$ in

[2] It is straightforward to derive an explicit running time bound for $\mathcal{A}_{\text{inv}}^{\mathcal{C}}$ in terms of the above functions from our analysis, but the resulting expression is extremely long and rather uninformative so we do not provide it.

Step 2 to learn f under distribution D (recall that $D = \mathcal{U}_{g^{-1}(1)}$ and is close to D'). Our algorithm $\mathcal{A}_{\mathrm{inv}}^{\mathcal{C}}$ needs to be able to efficiently simulate $\mathcal{A}_{\mathrm{SQ}}^{\mathcal{C}}$ given its available information. While it would be easy to do so given access to random labeled examples $(x, f(x))$, where $x \sim D$, such information is not available in our setting. To overcome this obstacle, we show (see Proposition 25) that for *any* samplable distribution D, we can efficiently simulate a statistical query algorithm under D using samples from $D_{f,+}$. This does not quite solve the problem, since we only have samples from $\mathcal{U}_{f^{-1}(1)}$. However, we show (see Claim 28) that for our setting, i.e., for $D = \mathcal{U}_{g^{-1}(1)}$, we can simulate a sample from $D_{f,+}$ by a simple rejection sampling procedure using samples from $\mathcal{U}_{f^{-1}(1)}$ and query access to g.

Some more issues remain to be handled. First, the simulation of the statistical query algorithm sketched in the previous paragraph only works under the assumption that we are given a sufficiently accurate approximation $\widetilde{b}_f$ of the probability $\mathbf{Pr}_{x \sim D}[f(x) = 1]$. (Intuitively, our approximation should be smaller than the smallest tolerance τ provided to the statistical query oracle by the algorithm $\mathcal{A}_{\mathrm{SQ}}^{\mathcal{C}}$.) Second, by Definition 20, the densifier only succeeds under the assumption that it is given in its input an $(1+\epsilon)$-multiplicative approximation $\widehat{p}$ to $p = \mathbf{Pr}_{x \in \mathcal{U}_n}[f(x) = 1]$.

We handle these issues as follows: First, we show (see Claim 29) that, given an accurate estimate $\widehat{p}$ and a "dense" function $g \in \mathcal{C}'$, we can use the approximate counting algorithm $\mathcal{A}_{\mathrm{count}}^{\mathcal{C}'}$ to efficiently compute an accurate estimate $\widetilde{b}_f$. (This is one reason why Theorem 21 requires an approximate counting algorithm for $\mathcal{C}'$.) To deal with the fact that we do not a priori have an accurate estimate $\widehat{p}$, we run our sketched algorithm for all possible values of $\mathbf{Pr}_{x \sim \mathcal{U}_n}[f(x) = 1]$ in an appropriate multiplicative "grid" of size $N = O(n/\epsilon)$, covering all possible values from $1/2^n$ to 1. We thus obtain a set $\mathcal{D}$ of N candidate distributions one of which is guaranteed to be close to the true distribution $\mathcal{U}_{f^{-1}(1)}$ in variation distance. At this point, we would like to apply our hypothesis testing machinery (Proposition 12) to find such a distribution. However, in order to use Proposition 12, in addition to sample access to the candidate distributions (and the distribution being learned), we also require a *multiplicatively accurate* approximate evaluation oracle to evaluate the probability mass of any point under the candidate distributions. We show (see Lemma 39) that this is possible in our generic setting, using properties of the densifier and the approximate counting algorithm $\mathcal{A}_{\mathrm{count}}^{\mathcal{C}'}$ for $\mathcal{C}'$.

Now we are ready to begin the detailed proof of Theorem 21.

3.2 Simulating statistical query algorithms. Our algorithm $\mathcal{A}_{\mathrm{inv}}^{\mathcal{C}}$ will need to simulate a statistical query algorithm for $\mathcal{C}$, with respect to a specific distribution D. Note, however that $\mathcal{A}_{\mathrm{inv}}$ only has access to uniform positive examples of $f \in \mathcal{C}$, i.e., samples from $\mathcal{U}_{f^{-1}(1)}$. Hence we need to show that a statistical query algorithm can be efficiently simulated in such a setting. To do this it suffices to show that one can efficiently provide valid responses to queries to the statistical query oracle $\mathrm{STAT}(f, D)$, i.e., that one can simulate the oracle. Assuming this can be done, the simulation algorithm $\mathcal{A}_{\mathrm{SQ-SIM}}$ is very simple: Run the statistical query algorithm $\mathcal{A}_{\mathrm{SQ}}$, and whenever it makes a query to $\mathrm{STAT}(f, D)$, simulate it. To this end, in the following lemma we describe a procedure that simulates an SQ oracle. (Our approach here is similar to that of earlier simulation procedures that have been given in the literature, see e.g. Denis *et al.* [DGL05].)

Lemma 24. *Let $\mathcal{C}$ be a concept class over $\{-1,1\}^n$, $f \in \mathcal{C}$, and D be a samplable distribution over $\{-1,1\}^n$. There exists an algorithm* `Simulate-STAT`$_f^D$ *with the following properties: It is given access to independent samples from $D_{f,+}$, and takes as input a number $\widetilde{b}_f \in [0,1]$, a $t(n)$-time computable query function $\chi : \{-1,1\}^n \times \{-1,1\} \to [-1,1]$, a tolerance τ and a confidence δ. It has the following behavior: it uses $m = O\left((1/\tau^2)\log(1/\delta)\right)$ samples from D and $D_{f,+}$, runs in time $O\left(m \cdot t(n)\right)$, and if $|\widetilde{b}_f - \mathbf{Pr}_{x \sim D}[f(x) = 1]| \leq \tau'$, then with probability $1 - \delta$ it outputs a number v such that*

$$|\mathbf{E}_{x \sim D}\left[\chi\left(x, f(x)\right)\right] - v| \leq \tau + \tau'. \quad (4)$$

Proof. To prove the lemma, we start by rewriting the expectation in (4) as follows:

$$\mathbf{E}_{x\sim D}\left[\chi(x,f(x))\right] = \mathbf{E}_{x\sim D_{f,+}}\left[\chi(x,1)\right]\cdot\mathbf{Pr}_{x\sim D}[f(x)=1] + \mathbf{E}_{x\sim D_{f,-}}\left[\chi(x,-1)\right]\cdot\mathbf{Pr}_{x\sim D}[f(x)=-1].$$

We also observe that

$$\mathbf{E}_{x\sim D}\left[\chi(x,-1)\right] = \mathbf{E}_{x\sim D_{f,+}}\left[\chi(x,-1)\right]\cdot\mathbf{Pr}_{x\sim D}[f(x)=1] + \mathbf{E}_{x\sim D_{f,-}}\left[\chi(x,-1)\right]\cdot\mathbf{Pr}_{x\sim D}[f(x)=-1].$$

Combining the above equalities we get

$$\mathbf{E}_{x\sim D}\left[\chi(x,f(x))\right] = \mathbf{E}_{x\sim D}\left[\chi(x,-1)\right] + \mathbf{E}_{x\sim D_{f,+}}\left[\chi(x,1)-\chi(x,-1)\right]\cdot\mathbf{Pr}_{x\sim D}[f(x)=1]. \qquad (5)$$

Given the above identity, the algorithm $\texttt{Simulate-STAT}_f^D$ is very simple: We use random sampling from D to empirically estimate the expectations $\mathbf{E}_{x\sim D}\left[\chi(x,-1)\right]$ (recall that D is assumed to be a samplable distribution), and we use the independent samples from $D_{f,+}$ to empirically estimate $\mathbf{E}_{x\sim D_{f,+}}\left[\chi(x,1)-\chi(x,-1)\right]$. Both estimates are obtained to within an additive accuracy of $\pm\tau/2$ (with confidence probability $1-\delta/2$ each). We combine these estimates with our estimate $\widetilde{b_f}$ for $\mathbf{Pr}_{x\sim D}[f(x)=1]$ in the obvious way (see Step 2 of pseudocode below).

```
Subroutine $\texttt{Simulate-STAT}_f^D(D, D_{f,+}, \chi, \tau, \widetilde{b_f}, \delta)$:
Input: Independent samples from $D$ and $D_{f,+}$, query access to $\chi : \{-1,1\}^n \to \{-1,1\}$, accuracy $\tau$, $\widetilde{b_f} \in [0,1]$ and confidence $\delta$.
Output: If $|\widetilde{b_f} - \mathbf{Pr}_{x\sim D}[f(x)=1]| \leq \tau'$, a number $v$ that with probability $1-\delta$ satisfies $|\,\mathbf{E}_{x\sim D}[\chi(x,f(x))] - v| \leq \tau + \tau'$.

  1. Empirically estimate the values $\mathbf{E}_{x\sim D}[\chi(x,-1)]$ and $\mathbf{E}_{x\sim D_{f,+}}[\chi(x,1)-\chi(x,-1)]$ to within an additive $\pm\tau/2$ with confidence probability $1-\delta/2$. Let $\widetilde{E}_1, \widetilde{E}_2$ be the corresponding estimates.
  2. Output $v = \widetilde{E}_1 + \widetilde{E}_2 \cdot \widetilde{b_f}$.
```

By Fact 16, we can estimate each expectation using $m = \Theta\left((1/\tau^2)\log(1/\delta)\right)$ samples (from D, $D_{f,+}$ respectively). For each such sample the estimation algorithm needs to evaluate the function χ (once for the first expectation and twice for the second). Hence, the total number of queries to χ is $O(m)$, i.e., the subroutine $\texttt{Simulate-STAT}_f^D$ runs in time $O(m\cdot t(n))$ as desired.

By a union bound, with probability $1-\delta$ both estimates will be $\pm\tau/2$ accurate. The bound (4) follows from this latter fact and (5) by a straightforward application of the triangle inequality. This completes the proof of Lemma 24. □

Given the above lemma, we can state and prove our general result for simulating SQ algorithms:

Proposition 25. *Let $\mathcal{C}$ be a concept class and D be a samplable distribution over $\{-1,1\}^n$. Suppose there exists an* SQ*-learning algorithm $\mathcal{A}_{\mathrm{SQ}}$ for $\mathcal{C}$ under D with the following performance: $\mathcal{A}_{\mathrm{SQ}}$ runs in time $T_1 = t_1(n, 1/\epsilon, 1/\delta)$, each query provided to $\mathrm{STAT}(f,D)$ can be evaluated in time $T_2 = t_2(n)$, and the minimum value of the tolerance provided to $\mathrm{STAT}(f,D)$ in the course of its execution is $\tau = \tau(n, 1/\epsilon)$. Then, there exists an algorithm $\mathcal{A}_{\mathrm{SQ-SIM}}$ that is given access to*

(i) independent samples from $D_{f,+}$; and

(ii) a number $\widetilde{b_f} \in [0,1]$,

and efficiently simulates the behavior of $\mathcal{A}_{\rm SQ}$*. In particular,* $\mathcal{A}_{\rm SQ-SIM}$ *has the following performance guarantee: on input an accuracy* ϵ *and a confidence* δ*, it uses* $m = O\left((1/\tau^2) \cdot \log(T_1/\delta) \cdot T_1\right)$ *samples from* D *and* $D_{f,+}$*, runs in time* $T_{\rm SQ-SIM} = O\left(mT_2\right)$*, and if* $|\widetilde{b_f} - \mathbf{Pr}_{x \sim D}[f(x) = 1]| \leq \tau/2$ *then with probability* $1 - \delta$ *it outputs a hypothesis* $h : \{-1, 1\}^n \to \{-1, 1\}$ *such that* $\mathbf{Pr}_{x \sim D}[h(x) \neq f(x)] \leq \epsilon$*.*

Proof. The simulation procedure is very simple. We run the algorithm $\mathcal{A}_{\rm SQ}$ by simulating its queries using algorithm $\texttt{Simulate-STAT}_f^D$. The algorithm is described in the following pseudocode:

```
Algorithm $\mathcal{A}_{\rm SQ-SIM}(D, D_{f,+}, \epsilon, \widetilde{b_f}, \delta)$:
Input: Independent samples from $D$ and $D_{f,+}$, $\widetilde{b_f} \in [0, 1]$, $\epsilon$, $\delta > 0$.
Output: If $|\widetilde{b_f} - \mathbf{Pr}_{x \sim D}[f(x) = 1]| \leq \tau/2$, a hypothesis $h$ that with probability $1 - \delta$ satisfies $\mathbf{Pr}_{x \sim D}[h(x) \neq f(x)] \leq \epsilon$.

    1. Let $\tau = \tau(n, 1/\epsilon)$ be the minimum accuracy ever used in a query to $\mathrm{STAT}(f, D)$ during the execution of $\mathcal{A}_{\rm SQ}(\epsilon, \delta/2)$.

    2. Run the algorithm $\mathcal{A}_{\rm SQ}(\epsilon, \delta/2)$, by simulating each query to $\mathrm{STAT}(f, D)$ as follows: whenever $\mathcal{A}_{\rm SQ}$ makes a query $(\chi, \tau)$ to $\mathrm{STAT}(f, D)$, the simulation algorithm runs $\texttt{Simulate-STAT}_f^D(D, D_{f,+},\chi, \tau/2, \tau/2, \delta/(2T_1))$.

    3. Output the hypothesis $h$ obtained by the simulation.
```

Note that we run the algorithm $\mathcal{A}_{\rm SQ}$ with confidence probability $1 - \delta/2$. Moreover, each query to the $\mathrm{STAT}(f, D)$ oracle is simulated with confidence $1 - \delta/(2T_1)$. Since $\mathcal{A}_{\rm SQ}$ runs for at most T_1 time steps, it certainly performs at most T_1 queries in total. Hence, by a union bound over these events, with probability $1 - \delta/2$ all answers to its queries will be accurate to within an additive $\pm\tau/2$. By the guarantee of algorithm $\mathcal{A}_{\rm SQ}$ and a union bound, with probability $1 - \delta$, the algorithm $\mathcal{A}_{\rm SQ-SIM}$ will output a hypothesis $h : \{-1, 1\}^n \to \{-1, 1\}$ such that $\mathbf{Pr}_{x \sim D}[h(x) \neq f(x)] \leq \epsilon$. The sample complexity and running time follow from the bounds for $\texttt{Simulate-STAT}_f^D$. This completes the proof of Proposition 25. □

Proposition 25 tells us we can efficiently simulate a statistical query algorithm for a concept class $\mathcal{C}$ under a samplable distribution D if we have access to samples drawn from $D_{f,+}$ (and a very accurate estimate of $\mathbf{Pr}_{x \sim D}[f(x) = 1]$). In our setting, we have that $D = \mathcal{U}_{g^{-1}(1)}$ where $g \in \mathcal{C}'$ is the function that is output by $\mathcal{A}_{\rm den}^{(\mathcal{C}, \mathcal{C}')}$. So, the two issues we must handle are (i) obtaining samples from D, and (ii) obtaining samples from $D_{f,+}$.

For (i), we note that, even though we do not have access to samples drawn *exactly* from D, it suffices for our purposes to use a τ'-sampler for D for a sufficiently small τ'. To see this we use the following fact:

Fact 26. *Let* D, D' *be distributions over* $\{-1, 1\}^n$ *with* $d_{\rm TV}(D, D') \leq \tau'$*. Then for any bounded function* $\phi : \{-1, 1\}^n \to [-1, 1]$ *we have that* $|\,\mathbf{E}_{x \sim D}[\phi(x)] - \mathbf{E}_{x \sim D'}[\phi(x)]| \leq 2\tau'$*.*

Proof. By definition we have that

$$\begin{aligned}
|\mathbf{E}_{x\sim D}[\phi(x)] - \mathbf{E}_{x\sim D'}[\phi(x)]| &= \left|\sum_{x\in\{-1,1\}^n} \big(D(x) - D'(x)\big)\,\phi(x)\right| \\
&\leq \sum_{x\in\{-1,1\}^n} \left|\big(D(x) - D'(x)\big)\right| |\phi(x)| \\
&\leq \max\nolimits_{x\in\{-1,1\}^n} |\phi(x)| \cdot \sum_{x\in\{-1,1\}^n} \left|D(x) - D'(x)\right| \\
&\leq 1 \cdot \|D - D'\|_1 \\
&= 2d_{\mathrm{TV}}(D, D') \\
&\leq 2\tau'
\end{aligned}$$

as desired. □

The above fact implies that the statement of Proposition 25 continuous to hold with the same parameters if instead of a 0-sampler for D we have access to a τ'-sampler for D, for $\tau' = \tau/8$. The only difference is that in Step 1 of the subroutine $\texttt{Simulate-STAT}_f^D$ we empirically estimate the expectation $\mathbf{E}_{x\sim D'}[\chi(x, -1)]$ up to an additive $\pm\tau/4$. By Fact 26, this will be a $\pm(\tau/4 + 2\tau') = \pm\tau/2$ accurate estimate for the $\mathbf{E}_{x\sim D}[\chi(x, -1)]$. That is, we have:

Corollary 27. *The statement of Proposition 25 continues to hold with the same parameters if instead of a 0-sampler for D we have access to a $\tau' = \tau/8$-sampler for D.*

For (ii), even though we do not have access to the distribution $D = \mathcal{U}_{g^{-1}(1)}$ directly, we note below that we can efficiently sample from $D_{f,+}$ using samples from $\mathcal{U}_{f^{-1}(1)}$ together with evaluations of g (recall again that g is provided as the output of the densifier).

Claim 28. *Let $g : \{-1, 1\}^n \to \{-1, 1\}$ be a $t_g(n)$ time computable function such that $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[g(x) = 1] \geq \epsilon'$. There is an efficient subroutine that is given ϵ' and a circuit to compute g as input, uses $m = O((1/\epsilon')\log(1/\delta))$ samples from $\mathcal{U}_{f^{-1}(1)}$, runs in time $O(m \cdot t_g(n))$, and with probability $1 - \delta$ outputs a sample x such that $x \sim D_{f,+}$, where $D = \mathcal{U}_{g^{-1}(1)}$.*

Proof. To simulate a sample from $D_{f,+}$ we simply draw samples from $\mathcal{U}_{f^{-1}(1)}$ until we obtain a sample x with $g(x) = 1$. The following pseudocode makes this precise:

```
Subroutine Simulate-sample$^{D_{f,+}}(\mathcal{U}_{f^{-1}(1)}, g, \epsilon', \delta)$:
Input: Independent samples from $\mathcal{U}_{f^{-1}(1)}$, a circuit computing $g$, a value $\epsilon' > 0$ such that $\epsilon' \leq \mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[g(x) = 1]$ and confidence parameter $\delta$.
Output: A point $x \in \{-1, 1\}^n$ that with probability $1 - \delta$ satisfies $x \sim D_{f,+}$.

  1. Repeat the following at most $m = \Theta((1/\epsilon')\log(1/\delta))$ times:
      (a) Draw a sample $x \sim \mathcal{U}_{f^{-1}(1)}$.
      (b) If the circuit for $g$ evaluates to 1 on input $x$ then output $x$.
  2. If no point $x$ with $g(x) = 1$ has been obtained, halt and output "failure."
```

Since $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[g(x)=1]\geq \epsilon'$, after repeating this process $m=\Omega\left((1/\epsilon')\log(1/\delta)\right)$ times, we will obtain a satisfying assignment to g with probability at least $1-\delta$. It is clear that such a sample x is distributed according to $D_{f,+}$. For each sample we need to evaluate g once, hence the running time follows. □

Getting a good estimate $\widetilde{b_f}$ of $\mathbf{Pr}_{x\sim D}[f(x)=1]$. The simulations presented above require an additively accurate estimate $\widetilde{b_f}$ of $\mathbf{Pr}_{x\sim D}[f(x)=1]$. We now show that in our context, such an estimate can be easily obtained if we have access to a good estimate $\widehat{p}$ of $p=\mathbf{Pr}_{x\in\mathcal{U}_n}[f(x)=1]$, using the fact that we have an efficient approximate counting algorithm for $\mathcal{C}'$ and that $D\equiv\mathcal{U}_{g^{-1}(1)}$ where $g\in\mathcal{C}'$.

Claim 29. *Let $g:\{-1,1\}^n\to\{-1,1\}$, $g\in\mathcal{C}'$ be a $t_g(n)$ time computable function, satisfying $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[f(x)=1]\geq\gamma'$ and $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[g(x)=1]\geq 1-\epsilon'$. Let $\mathcal{A}^{\mathcal{C}'}_{\text{count}}$ be an (ϵ,δ)-approximate counting algorithm for $\mathcal{C}'$ running in time $T_{\text{count}}(n,1/\epsilon,1/\delta)$. There is a procedure* `Estimate-Bias` *with the following behavior:* `Estimate-Bias` *takes as input a value $0<\widehat{p}\leq 1$, a parameter $\tau'>0$, a confidence parameter δ', and a representation of $g\in\mathcal{C}'$.* `Estimate-Bias` *runs in time $O(t_g\cdot T_{\text{count}}(n,2/\tau',1/\delta'))$ and satisfies the following: if $p\overset{\text{def}}{=}\mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x)=1]<\widehat{p}\leq(1+\epsilon')p$, then with probability $1-\delta'$* `Estimate-Bias` *outputs a value $\widetilde{b_f}$ such that $|\widetilde{b_f}-\mathbf{Pr}_{x\sim D}[f(x)=1]|\leq\tau'$.*

Proof. The procedure `Estimate-Bias` is very simple. It runs $\mathcal{A}^{\mathcal{C}'}_{\text{count}}$ on inputs $\epsilon^\star=\tau'/2,\delta'$, using the representation for $g\in\mathcal{C}'$. Let p_g be the value returned by the approximate counter; `Estimate-Bias` returns $\widehat{p}/p_g$.

The claimed running time bound is obvious. To see that the procedure is correct, first observe that by Definition 8, with probability $1-\delta'$ we have that

$$\frac{|g^{-1}(1)|}{2^n}\cdot\frac{1}{1+\epsilon^\star}\leq p_g\leq\frac{|g^{-1}(1)|}{2^n}\cdot(1+\epsilon^\star).$$

For the rest of the argument we assume that the above inequality indeed holds. Let A denote $|g^{-1}(1)|$, let B denote $|f^{-1}(1)\cap g^{-1}(1)|$, and let C denote $|f^{-1}(1)\setminus g^{-1}(1)|$, so the true value $\mathbf{Pr}_{x\sim D}[f(x)=1]$ equals $\frac{B}{A}$ and the above inequality can be rephrased as

$$\frac{A}{1+\epsilon^\star}\leq p_g\cdot 2^n\leq A\cdot(1+\epsilon^\star).$$

By our assumption on $\widehat{p}$ we have that

$$B+C\leq\widehat{p}\cdot 2^n\leq(1+\epsilon')(B+C);$$

since $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[g(x)=1]\geq 1-\epsilon'$ we have

$$\frac{C}{B+C}\leq\epsilon'\qquad\text{(i.e., }C\leq\frac{\epsilon'}{1-\epsilon'}\cdot B\text{);}$$

and since $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[f(x)=1]\geq\gamma'$ we have

$$\frac{B}{A}\geq\gamma'.$$

Combining these inequalities we get

$$\frac{1}{1+\epsilon^\star}\cdot\frac{B}{A}\leq\frac{1}{1+\epsilon^\star}\cdot\frac{B+C}{A}\leq\frac{\widehat{p}}{p_g}\leq\frac{B}{A}\cdot(1+\epsilon')(1+\epsilon^\star)\left(1+\frac{\epsilon'}{1-\epsilon'}\right)=\frac{B}{A}\cdot(1+\epsilon^\star)$$

Hence

$$\left|\frac{B}{A}-\frac{\widehat{p}}{p_g}\right|\leq\frac{B}{A}\left(1+\epsilon^\star-\frac{1}{1+\epsilon^\star}\right)\leq\frac{2\epsilon^\star}{1+\epsilon^\star}\leq 2\epsilon^\star,$$

where we have used $B\leq A$. Recalling that $\epsilon^\star=\tau'/2$, the lemma is proved. □

3.3 An algorithm that succeeds given the (approximate) bias of f. In this section, we present an algorithm $\mathcal{A}'^{\mathcal{C}}_{\mathrm{inv}}(\epsilon, \delta, \widehat{p})$ which, in addition to samples from $\mathcal{U}_{f^{-1}(1)}$, takes as input parameters $\epsilon, \delta, \widehat{p}$. The algorithm succeeds in outputting a hypothesis distribution D_f satisfying $d_{\mathrm{TV}}(D_f, \mathcal{U}_{f^{-1}(1)}) \leq \epsilon$ if the input parameter $\widehat{p}$ is a multiplicatively accurate approximation to $\mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x) = 1]$. The algorithm follows the three high-level steps previously outlined and uses the subroutines of the previous subsection to simulate the statistical query algorithm. Detailed pseudocode follows:

```
Algorithm $\mathcal{A}'^{\mathcal{C}}_{\mathrm{inv}}(\mathcal{U}_{f^{-1}(1)}, \epsilon, \delta, \widehat{p})$:
Input: Independent samples from $\mathcal{U}_{f^{-1}(1)}$, accuracy and confidence parameters $\epsilon, \delta$, and a value $1/2^n < \widehat{p} \leq 1$.
Output: If $\mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x) = 1] \leq \widehat{p} < (1 + \epsilon)\,\mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x) = 1]$, with probability $1 - \delta$ outputs an $\epsilon$-sampler $C_f$ for $\mathcal{U}_{f^{-1}(1)}$ .

  1. [Run the densifier to obtain $g$]
     Fix $\epsilon_1 \stackrel{\mathrm{def}}{=} \epsilon/6$ and $\gamma \stackrel{\mathrm{def}}{=} \gamma(n, 1/\epsilon_1, 3/\delta)$. Run the $\gamma$-densifier $\mathcal{A}^{(\mathcal{C},\mathcal{C}')}_{\mathrm{den}}(\epsilon_1, \delta/3, \widehat{p})$ using random samples from $\mathcal{U}_{f^{-1}(1)}$. Let $g \in \mathcal{C}'$ be its output.
  2. [Run the $\mathrm{SQ}$-learner, using the approximate uniform generator for $g$, to obtain hypothesis $h$]
     (a) Fix $\epsilon_2 \stackrel{\mathrm{def}}{=} \epsilon\gamma/7$, $\tau_2 \stackrel{\mathrm{def}}{=} \tau(n, 1/\epsilon_2)$ and $m \stackrel{\mathrm{def}}{=} \Theta\big((1/\tau_2^2) \cdot \log(T_1/\delta) \cdot T_1\big)$, where $T_1 = t_1(n, 1/\epsilon_2, 12/\delta)$.
     (b) Run the generator $\mathcal{A}^{\mathcal{C}'}_{\mathrm{gen}}(g, \tau_2/8, \delta/(12m))$ $m$ times and let $S_D \subseteq \{-1,1\}^n$ be the multiset of samples obtained.
     (c) Run Simulate-sample$^{D_{f,+}}(\mathcal{U}_{f^{-1}(1)}, g, \gamma, \delta/(12m))$ $m$ times and let $S_{D_{f,+}} \subseteq \{-1,1\}^n$ be the multiset of samples obtained.
     (d) Run Estimate-Bias with parameters $\widehat{p}$, $\tau' = \tau_2/2$, $\delta' = \delta/12$ , using the representation for $g \in \mathcal{C}'$, and let $\widetilde{b_f}$ be the value it returns.
     (e) Run $\mathcal{A}_{\mathrm{SQ-SIM}}(S_D, S_{D_{f,+}}, \epsilon_2, \widetilde{b_f}, \delta/12)$. Let $h : \{-1,1\}^n \to \{-1,1\}$ be the output hypothesis.
  3. [Output the sampler which does rejection sampling according to $h$ on draws from the approximate uniform generator for $g$]
     Output the sampler $C_f$ which works as follows:
        For $i = 1$ to $t = \Theta\left((1/\gamma)\log(1/(\delta\epsilon)\right)$ do:
          (a) Set $\epsilon_3 \stackrel{\mathrm{def}}{=} \epsilon\gamma/48000$.
          (b) Run the generator $\mathcal{A}^{\mathcal{C}'}_{\mathrm{gen}}(g, \epsilon_3, \delta\epsilon/(12t))$ and let $x^{(i)}$ be its output.
          (c) If $h(x^{(i)}) = 1$, output $x^{(i)}$.
        If no $x^{(i)}$ with $h(x^{(i)}) = 1$ has been obtained, output the default element $\perp$.
     Let $\hat{D}$ denote the distribution over $\{-1,1\}^n \cup \{\perp\}$ for which $C_f$ is a 0-sampler, and let $\hat{D}'$ denote the conditional distribution of $\hat{D}$ restricted to $\{-1,1\}^n$ (i.e., excluding $\perp$).
```

We note that by inspection of the code for C_f, we have that the distribution $\hat{D}'$ is identical to $(D_{g,\epsilon_3})_{h^{-1}(1)}$, where D_{g,ϵ_3} is the distribution corresponding to the output of the approximate uniform generator when called on function g and error parameter ϵ_3 (see Definition 9) and $(D_{g,\epsilon_3})_{h^{-1}(1)}$ is D_{g,ϵ_3} conditioned on $h^{-1}(1)$.

We have the following:

Theorem 30. *Let $p \overset{def}{=} \mathbf{Pr}_{x \in \mathcal{U}_n}[f(x) = 1]$. Algorithm $\mathcal{A}'^{\mathcal{C}}_{\text{inv}}(\epsilon, \delta, \widehat{p})$ has the following behavior: If $p \leq \widehat{p} < (1+\epsilon)p$, then with probability $1-\delta$ the following both hold:*

(i) the output C_f is a sampler for a distribution $\hat{D}$ such that $d_{\text{TV}}(\hat{D}, \mathcal{U}_{f^{-1}(1)}) \leq \epsilon$; and

(ii) the functions h, g satisfy $|h^{-1}(1) \cap g^{-1}(1)|/|g^{-1}(1)| \geq \gamma/2$.

The running time of$\mathcal{A}'^{\mathcal{C}}_{\text{inv}}$ is polynomial in $T_{\text{den}}(n, 1/\epsilon, 1/\delta)$, $T_{\text{gen}}(n, 1/\epsilon, 1/\delta)$, $T_{\text{count}}(n, 1/\epsilon, 1/\delta)$, $t_1(n, 1/\epsilon, 1/\delta)$, $t_2(n)$, $1/\tau(n, 1/\epsilon)$, and $1/\gamma(n, 1/\epsilon, 1/\delta)$.

Proof. We give an intuitive explanation of the pseudocode in tandem with a proof of correctness. We argue that Steps 1-3 of the algorithm implement the corresponding steps of our high-level description and that the algorithm succeeds with confidence probability $1-\delta$.

We assume throughout the argument that indeed $\widehat{p}$ lies in $[p, (1+\epsilon)p)$. Given this, by Definition 20 with probability $1-\delta/3$ the function g satisfies properties (a) and (b) of Definition 20, i.e., $\mathbf{Pr}_{x \sim \mathcal{U}_{f^{-1}(1)}}[g(x) = 1] \geq 1-\epsilon_1$ and $\mathbf{Pr}_{x \sim \mathcal{U}_{g^{-1}(1)}}[f(x) = 1] \geq \gamma$. We condition on this event (which we denote E_1) going forth.

We now argue that Step 2 simulates the SQ learning algorithm $\mathcal{A}^{\mathcal{C}}_{\text{SQ}}$ to learn the function $f \in \mathcal{C}$ under distribution $D \equiv \mathcal{U}_{g^{-1}(1)}$ to accuracy ϵ_2 with confidence $1-\delta/3$. Note that the goal of Step (b) is to obtain m samples from a distribution D'' (the distribution "$D_{g,\tau_2/8}$" of Definition 9) such that $d_{\text{TV}}(D'', D) \leq \tau_2/8$. To achieve this, we call the approximate uniform generator for g a total of m times with failure probability $\delta/(12m)$ for each call (i.e., each call returns $\perp$ with probability at most $\delta/(12m)$). By a union bound, with failure probability at most $\delta/12$, all calls to the generator are successful and we obtain a set S_D of m independent samples from D''. Similarly, the goal of Step (c) is to obtain m samples from $D_{f,+}$ and to achieve it we call the subroutine $\texttt{Simulate-sample}^{D_{f,+}}$ a total of m times with failure probability $\delta/(12m)$ each. By Claim 28 and a union bound, with failure probability at most $\delta/12$, this step is successful, i.e., it gives a set $S_{D_{f,+}}$ of m independent samples from $D_{f,+}$. The goal of Step (d) is to obtain a value $\widetilde{b_f}$ satisfying $|\widetilde{b_f} - \mathbf{Pr}_{x \sim D}[f(x) = 1]| \leq \tau_2/2$; by Claim 29, with failure probability at most $\delta/12$ the value $\widetilde{b_f}$ obtained in this step is as desired. Finally, Step (e) applies the simulation algorithm $\mathcal{A}_{\text{SQ}-\text{SIM}}$ using the samples S_D and $S_{D_{f,+}}$ and the estimate $\widetilde{b_f}$ of $\mathbf{Pr}_{x \sim D}[f(x) = 1]$ obtained in the previous steps. Conditioning on Steps (b), (c) and (d) being successful Corollary 27 implies that Step (e) is successful with probability $1-\delta/12$, i.e., it outputs a hypothesis h that satisfies $\mathbf{Pr}_{x \sim D}[f(x) \neq h(x)] \leq \epsilon_2$. A union bound over Steps (c), (d) and (e) completes the analysis of Step 2. For future reference, we let E_2 denote the event that the hypothesis h constructed in Step 2(e) has $\mathbf{Pr}_{x \sim D}[f(x) \neq h(x)] \leq \epsilon_2$ (so we have that E_2 holds with probability at least $1-\delta/3$; we additionally condition on this event going forth). We observe that since (as we have just shown) $\mathbf{Pr}_{x \sim \mathcal{U}_{g^{-1}(1)}}[f(x) \neq h(x)] \leq \epsilon_2$ and $\mathbf{Pr}_{x \sim \mathcal{U}_{g^{-1}(1)}}[f(x) = 1] \geq \gamma$, we have $\mathbf{Pr}_{x \sim \mathcal{U}_{g^{-1}(1)}}[h(x) = 1] \geq \gamma - \epsilon_2 \geq \gamma/2$, which gives item (ii) of the theorem; so it remains to establish item (i) and the claimed running time bound.

To establish (i), we need to prove that the output distribution $\hat{D}$ of the sampler C_f is ϵ-close in total variation distance to $\mathcal{U}_{f^{-1}(1)}$. This sampler attempts to draws t samples from a distribution D' such that $d_{\text{TV}}(D', D) \leq \epsilon_3$ (this is the distribution "D_{g,ϵ_3}" in the notation of Definition 9) and it outputs one of these samples that satisfies h (unless none of these samples satisfies h, in which case it outputs a default element $\perp$). The desired variation distance bound follows from the next lemma for our choice of parameters:

Lemma 31. *Let $\hat{D}$ be the output distribution of $\mathcal{A}'^{\mathcal{C}}_{\mathrm{inv}}(\mathcal{U}_{f^{-1}(1)}, \epsilon, \delta, \widehat{p})$. If $\mathbf{Pr}_{x \sim \mathcal{U}_n}[f(x) = 1] \le \widehat{p} \le (1 + \epsilon)\,\mathbf{Pr}_{x \sim \mathcal{U}_n}[f(x) = 1]$, then conditioned on Events E_1 and E_2, we have*

$$\begin{aligned} d_{\mathrm{TV}}(\hat{D}, \mathcal{U}_{f^{-1}(1)}) &\le \frac{\epsilon}{6} + \frac{\epsilon}{6} + \frac{4\epsilon_3}{\gamma} + \epsilon_1 + \frac{\epsilon_2}{2\gamma} + \frac{\epsilon_2}{\gamma - \epsilon_2} \\ &\le \frac{\epsilon}{6} + \frac{\epsilon}{6} + \frac{\epsilon}{12000} + \frac{\epsilon}{6} + \frac{\epsilon}{14} + \frac{\epsilon}{6} < \epsilon. \end{aligned}$$

Proof. Consider the distribution $D' = D_{g,\epsilon_3}$ (see Definition 9) produced by the approximate uniform generator in Step 3 of the algorithm. Let $D'|_{h^{-1}(1)}$ denote distribution D' restricted to $h^{-1}(1)$. Let S denote the set $g^{-1}(1) \cap h^{-1}(1)$. The lemma is an immediate consequence of Claims 32, 34, 35 and 36 below using the triangle inequality (everything below is conditioned on E_1 and E_2). □

Claim 32. $d_{\mathrm{TV}}(\hat{D}, \hat{D}') \le \epsilon/6$.

Proof. Recall that $\hat{D}'$ is simply $\hat{D}$ conditioned on not outputting $\perp$.

We first claim that with probability at least $1 - \delta\epsilon/12$ all t points drawn in Step 3 of the code for C_f are distributed according to the distribution $D' = D_{g,\epsilon_3}$ over $g^{-1}(1)$. Each of the t calls to the approximate uniform generator has failure probability $\delta\epsilon/(12t)$ (of outputting $\perp$ rather than a point distributed according to D') so by a union bound no calls fail with probability at least $1 - \delta\epsilon/12$, and thus with probability at least $1 - \delta\epsilon/12$ indeed all t samples are independently drawn from such a distribution D'.

Conditioned on this, we claim that a satisfying assignment for h is obtained within the t samples with probability at least $1 - \delta\epsilon/12$. This can be shown as follows:

Claim 33. *Let $h : \{-1,1\}^n \to \{-1,1\}$ be the hypothesis output by $\mathcal{A}^{\mathcal{C}}_{\mathrm{SQ-SIM}}$. We have*

$$\mathbf{Pr}_{x \sim D'}[h(x) = 1] \ge \gamma/4.$$

Proof. First recall that, by property (b) in the definition of the densifier (Definition 20), we have $\mathbf{Pr}_{x \sim D}[f(x) = 1] \ge \gamma$. Since $d_{\mathrm{TV}}(D', D) \le \epsilon_3$, by definition we get

$$\mathbf{Pr}_{x \sim D'}[f(x) = 1] \ge \mathbf{Pr}_{x \sim D}[f(x) = 1] - \epsilon_3 \ge \gamma - \epsilon_3 \ge 3\gamma/4.$$

Now by the guarantee of Step 2 we have that $\mathbf{Pr}_{x \sim D}[f(x) \neq h(x)] \le \epsilon_2$. Combined with the fact that $d_{\mathrm{TV}}(D', D) \le \epsilon_3$, this implies that

$$\mathbf{Pr}_{x \sim D'}[f(x) \neq h(x)] \le \epsilon_2 + \epsilon_3 \le \gamma/2.$$

Therefore, we conclude that

$$\mathbf{Pr}_{x \sim D'}[h(x) = 1] \ge \mathbf{Pr}_{x \sim D'}[f(x) = 1] - \mathbf{Pr}_{x \sim D'}[f(x) \neq h(x)] \ge 3\gamma/4 - \gamma/2 \ge \gamma/4$$

as desired. □

Hence, for an appropriate constant in the big-Theta specifying t, with probability at least $1 - \delta\epsilon/12 > 1 - \delta/12$ some $x^{(i)}$ is a satisfying assignment of h. that with probability at least $1 - \delta\epsilon/12$ some $x^{(i)}$, $i \in [t]$, has $h(x) = 1$. Thus with overall failure probability at most $\delta\epsilon/6$ a draw from $\hat{D}'$ is not $\perp$, and consequently we have $d_{\mathrm{TV}}(\hat{D}, \hat{D}') \le \delta\epsilon/6 \le \epsilon/6$. □

Claim 34. $d_{\mathrm{TV}}(\hat{D}', D'|_{h^{-1}(1)}) \le \epsilon/6$.

Proof. The probability that any of the t points $x^{(1)}, \ldots, x^{(t)}$ is not drawn from D' is at most $t \cdot \delta\epsilon/(12t) < \epsilon/12$. Assuming that this does not happen, the probability that no $x^{(i)}$ lies in $h^{-1}(1)$ is at most $(1-\gamma/4)^t < \delta\epsilon/12 < \epsilon/12$ by Claim 33. Assuming this does not happen, the output of a draw from $\hat{D}$ is distributed identically according to $D'|_{h^{-1}(1)}$. Consequently we have that $d_{\mathrm{TV}}(\hat{D}, D'|_{h^{-1}(1)}) \le \epsilon/6$ as claimed. □

Claim 35. $d_{\mathrm{TV}}(D'|_{h^{-1}(1)}, \mathcal{U}_S) \le 4\epsilon_3/\gamma$.

Proof. The definition of an approximate uniform generator gives us that $d_{\mathrm{TV}}(D', \mathcal{U}_{g^{-1}(1)}) \le \epsilon_3$, and Claim 33 gives that $\mathbf{Pr}_{x \sim D'}[h(x) = 1] \ge \gamma/4$. We now recall the fact that for any two distributions D_1, D_2 and any event E, writing $D_i|_E$ to denote distribution D_i conditioned on event E, we have

$$d_{\mathrm{TV}}(D_1|_E, D_2|_E) \le \frac{d_{\mathrm{TV}}(D_1, D_2)}{D_1(E)}.$$

The claim follows since $\mathcal{U}_{g^{-1}(1)}|_{h^{-1}(1)}$ is equivalent to $\mathcal{U}_S$. □

Claim 36. $d_{\mathrm{TV}}(\mathcal{U}_S, \mathcal{U}_{f^{-1}(1)}) \le \epsilon_1 + \frac{\epsilon_2}{2\gamma} + \frac{\epsilon_2}{\gamma - \epsilon_2}$.

Proof. The proof requires a careful combination of the properties of the function g constructed by the densifier and the guarantee of the SQ algorithm. Recall that $S = g^{-1}(1) \cap h^{-1}(1)$. We consider the set $S' = g^{-1}(1) \cap f^{-1}(1)$. By the triangle inequality, we can bound the desired variation distance as follows:

$$d_{\mathrm{TV}}(\mathcal{U}_S, \mathcal{U}_{f^{-1}(1)}) \le d_{\mathrm{TV}}(\mathcal{U}_{f^{-1}(1)}, \mathcal{U}_{S'}) + d_{\mathrm{TV}}(\mathcal{U}_{S'}, \mathcal{U}_S). \tag{6}$$

We will bound from above each term of the RHS in turn. To proceed we need an expression for the total variation distance between the uniform distribution on two finite sets. The following fact is obtained by straightforward calculation:

Fact 37. *Let A, B be subsets of a finite set $\mathcal{W}$ and $\mathcal{U}_A, \mathcal{U}_B$ be the uniform distributions on A, B respectively. Then,*

$$d_{\mathrm{TV}}(\mathcal{U}_A, \mathcal{U}_B) = (1/2) \cdot \frac{|A \cap \overline{B}|}{|A|} + (1/2) \cdot \frac{|B \cap \overline{A}|}{|B|} + (1/2) \cdot |A \cap B| \cdot \left| \frac{1}{|A|} - \frac{1}{|B|} \right|. \tag{7}$$

To bound the first term of the RHS of (6) we apply the above fact for $A = f^{-1}(1)$ and $B = S'$. Note that in this case $B \subseteq A$, hence the second term of (7) is zero. Regarding the first term, note that

$$\frac{|A \cap \overline{B}|}{|A|} = \frac{|f^{-1}(1) \cap \overline{g^{-1}(1)}|}{|f^{-1}(1)|} \le \epsilon_1,$$

where the inequality follows from Property (a) of the densifier definition. Similarly, for the third term we can write

$$|A \cap B| \cdot \left| \frac{1}{|A|} - \frac{1}{|B|} \right| = |B| \cdot \left| \frac{1}{|A|} - \frac{1}{|B|} \right| = 1 - \frac{|B|}{|A|} = 1 - \frac{|f^{-1}(1) \cap g^{-1}(1)|}{|f^{-1}(1)|} \le \epsilon_1,$$

where the inequality also follows from Property (a) of the densifier definition. We therefore conclude that $d_{\mathrm{TV}}(\mathcal{U}_{f^{-1}(1)}, \mathcal{U}_{S'}) \le \epsilon_1$.

We now proceed to bound the second term of the RHS of (6) by applying Fact 37 for $A = S'$ and $B = S$. It turns out that bounding the individual terms of (7) is trickier in this case. For the first term we have:

$$\frac{|A \cap \overline{B}|}{|A|} = \frac{|f^{-1}(1) \cap g^{-1}(1) \cap \overline{h^{-1}(1)}|}{|f^{-1}(1) \cap g^{-1}(1)|} = \frac{|f^{-1}(1) \cap g^{-1}(1) \cap \overline{h^{-1}(1)}|}{|g^{-1}(1)|} \cdot \frac{|g^{-1}(1)|}{|f^{-1}(1) \cap g^{-1}(1)|} \le \frac{\epsilon_2}{\gamma},$$

where the last inequality follows from the guarantee of the SQ learning algorithm and Property (b) of the densifier definition. For the second term we have

$$\frac{|B \cap \overline{A}|}{|B|} = \frac{|\overline{f^{-1}(1)} \cap g^{-1}(1) \cap h^{-1}(1)|}{|g^{-1}(1) \cap h^{-1}(1)|}.$$

To analyze this term we recall that by the guarantee of the SQ algorithm it follows that the numerator satisfies

$$|\overline{f^{-1}(1)} \cap g^{-1}(1) \cap h^{-1}(1)| \leq \epsilon_2 \cdot |g^{-1}(1)|.$$

From the same guarantee we also get

$$|f^{-1}(1) \cap g^{-1}(1) \cap \overline{h^{-1}(1)}| \leq \epsilon_2 \cdot |g^{-1}(1)|.$$

Now, Property (b) of the densifier definition gives $|f^{-1}(1) \cap g^{-1}(1)| \geq \gamma \cdot |g^{-1}(1)|$. Combing these two inequalities implies that

$$|g^{-1}(1) \cap h^{-1}(1)| \geq |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| \geq (\gamma - \epsilon_2) \cdot |g^{-1}(1)|.$$

In conclusion, the second term is upper bounded by $(1/2) \cdot \frac{\epsilon_2}{\gamma - \epsilon_2}$.

For the third term, we can write

$$|A \cap B| \cdot \left| \frac{1}{|A|} - \frac{1}{|B|} \right| = |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| \cdot \left| \frac{1}{|f^{-1}(1) \cap g^{-1}(1)|} - \frac{1}{|g^{-1}(1) \cap h^{-1}(1)|} \right|.$$

To analyze these term we relate the cardinalities of these sets. In particular, we can write

$$\begin{aligned}
|f^{-1}(1) \cap g^{-1}(1)| &= |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| + |f^{-1}(1) \cap g^{-1}(1) \cap \overline{h^{-1}(1)}| \\
&\leq |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| + \epsilon_2 \cdot |g^{-1}(1)| \\
&\leq |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| + \frac{\epsilon_2}{\gamma} \cdot |f^{-1}(1) \cap g^{-1}(1)|
\end{aligned}$$

where the last inequlity is Property (b) of the densifier defintion. Therefore, we obtain

$$(1 - \frac{\epsilon_2}{\gamma}) \cdot |f^{-1}(1) \cap g^{-1}(1)| \leq |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| \leq |f^{-1}(1) \cap g^{-1}(1)|.$$

Similarly, we have

$$\begin{aligned}
|g^{-1}(1) \cap h^{-1}(1)| &= |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| + |\overline{f^{-1}(1)} \cap g^{-1}(1) \cap h^{-1}(1)| \\
&\leq |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| + \epsilon_2 \cdot |g^{-1}(1)| \\
&\leq |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| + \frac{\epsilon_2}{\gamma - \epsilon_2} \cdot |g^{-1}(1) \cap h^{-1}(1)|
\end{aligned}$$

and therefore

$$(1 - \frac{\epsilon_2}{\gamma - \epsilon_2}) \cdot |g^{-1}(1) \cap h^{-1}(1)| \leq |f^{-1}(1) \cap g^{-1}(1) \cap h^{-1}(1)| \leq |g^{-1}(1) \cap h^{-1}(1)|.$$

The above imply that the third term is bounded by $(1/2) \cdot \frac{\epsilon_2}{\gamma - \epsilon_2}$. This completes the proof of the claim. □

With Lemma 31 established, to finish the proof of Theorem 30 it remains only to establish the claimed running time bound. This follows from a straightforward (but somewhat tedious) verification, using the running time bounds established in Lemma 24, Proposition 25, Corollary 27, Claim 28 and Claim 29. □

3.4 Getting from $\mathcal{A}'^{\mathcal{C}}_{\mathrm{inv}}$ to $\mathcal{A}^{\mathcal{C}}_{\mathrm{inv}}$: An approximate evaluation oracle. Recall that the algorithm $\mathcal{A}'^{\mathcal{C}}_{\mathrm{inv}}$ from the previous subsection is only guaranteed (with high probability) to output a sampler for a hypothesis distribution $\hat{D}$ that is statistically close to the target distribution $\mathcal{U}_{f^{-1}(1)}$ if it is given an input parameter $\widehat{p}$ satisfying $p \leq \widehat{p} < (1+\epsilon)p$, where $p \overset{\mathrm{def}}{=} \mathbf{Pr}_{x \in \mathcal{U}_n}[f(x) = 1]$. Given this, a natural idea is to run $\mathcal{A}'^{\mathcal{C}}_{\mathrm{inv}}$ a total of $k = O(n/\epsilon)$ times, using "guesses" for $\widehat{p}$ that increase multiplicatively as powers of $1+\epsilon$, starting at $1/2^n$ (the smallest possible value) and going up to 1. This yields hypothesis distributions $\hat{D}_1, \ldots, \hat{D}_k$ where $\hat{D}_i$ is the distribution obtained by setting $\widehat{p}$ to $\widehat{p}_i \overset{\mathrm{def}}{=} (1+\epsilon)^{i-1}/2^n$. With such distributions in hand, an obvious approach is to use the "hypothesis testing" machinery of Section 2 to identify a high-accuracy $\hat{D}_i$ from this collection.

This is indeed the path we follow, but some care is needed to make the approach go through. Recall that as described in Proposition 12, the hypothesis testing algorithm requires the following:

1. independent samples from the target distribution $\mathcal{U}_{f^{-1}(1)}$ (this is not a problem since such samples are available in our framework);

2. independent samples from $\hat{D}_i$ for each i (also not a problem since the i-th run of algorithm $\mathcal{A}'^{\mathcal{C}}_{\mathrm{inv}}$ outputs a sampler for distribution $\hat{D}_i$; and

3. a $(1+O(\epsilon))$-approximate evaluation oracle $\mathrm{EVAL}_{\hat{D}_i}$ for each distribution $\hat{D}_i$.

In this subsection we show how to construct item (3) above, the approximate evaluation oracle. In more detail, we first describe a randomized procedure `Check` which is applied to the output of each execution of $\mathcal{A}'^{\mathcal{C}}_{\mathrm{inv}}$ (across all k different settings of the input parameter $\widehat{p}_i$). We show that with high probability the "right" value $\widehat{p}_{i^*}$ (the one which satisfies $p \leq \widehat{p}_{i^*} < (1+\epsilon)p$) will pass the procedure `Check`. Then we show that for each value $\widehat{p}_{i^*}$ that passed the check a simple deterministic algorithm gives the desired approximate evaluation oracle for $\hat{D}_i$.

We proceed to describe the `Check` procedure and characterize its performance.

Algorithm `Check`$(g, h, \delta', \epsilon)$:

Input: functions g and h as described in Lemma 38, a confidence parameter δ', and an accuracy parameter ϵ

Output: If $|h^{-1}(1) \cap g^{-1}(1)|/|g^{-1}(1)| \geq \gamma/2$, with probability $1-\delta'$ outputs a pair (α, κ) such that $|\alpha - |h^{-1}(1) \cap g^{-1}(1)|/|g^{-1}(1)|| \leq \mu \cdot |h^{-1}(1) \cap g^{-1}(1)|/|g^{-1}(1)|$ and $\frac{|g^{-1}(1)|}{1+\tau} \leq \kappa \leq (1+\tau)|g^{-1}(1)|$, where $\mu = \tau = \epsilon/40000$.

1. Sample $m = O(\log(2/\delta')/(\gamma\mu^2))$ points $x^1, \ldots, x^m$ from $\mathcal{A}^{\mathcal{C}'}_{\mathrm{gen}}(g, \gamma/4, \delta'/(2m))$. If any $x^j = \bot$ halt and output "failure."

2. Let α be $(1/m)$ times the number of points x^j that have $h(x) = 1$.

3. Call $\mathcal{A}^{\mathcal{C}'}_{\mathrm{count}}(\tau, \delta'/2)$ on g and set κ to 2^n times the value it returns.

Lemma 38. *Fix $i \in [k]$. Consider a sequence of k runs of $\mathcal{A}'^{\mathcal{C}}_{\mathrm{inv}}$ where in the i-th run it is given $\widehat{p}_i \overset{\mathit{def}}{=} (1+\epsilon)^{i-1}/2^n$ as its input parameter. Let g_i be the function in $\mathcal{C}'$ constructed by $\mathcal{A}'^{\mathcal{C}}_{\mathrm{inv}}$ in Step 1 of its i-th run and h_i be the hypothesis function constructed by $\mathcal{A}'^{\mathcal{C}}_{\mathrm{inv}}$ in Step 2(e) of its i-th run. Suppose* `Check` *is given as input g_i, h_i, a confidence parameter δ', and an accuracy parameter ϵ'. Then it either outputs "no"* *or a pair* $(\alpha_i, \kappa_i) \in [0,1] \times [0, 2^{n+1}]$*, and satisfies the following performance guarantee: If* $|h_i^{-1}(1) \cap g_i^{-1}(1)|/|g_i^{-1}(1)| \geq \gamma/2$ *then with probability at least* $1 - \delta'$ `Check` *outputs a pair* (α_i, κ_i) *such that*

$$\left|\alpha_i - \frac{|h_i^{-1}(1) \cap g_i^{-1}(1)|}{|g_i^{-1}(1)|}\right| \leq \mu \cdot \frac{|h_i^{-1}(1) \cap g_i^{-1}(1)|}{|g_i^{-1}(1)|} \tag{8}$$

and

$$\frac{|g_i^{-1}(1)|}{1+\tau} \leq \kappa_i \leq (1+\tau)|g_i^{-1}(1)|, \tag{9}$$

where $\mu = \tau = \epsilon/40000$.

Proof. Suppose that i is such that $|h_i^{-1}(1) \cap g_i^{-1}(1)|/|g_i^{-1}(1)| \geq \gamma/2$. Recall from Definition 9 that each point x^j drawn from $\mathcal{A}_{\text{gen}}^{\mathcal{C}'}(g_i, \gamma/4, \delta'/(2m))$ in Step 1 is with probability $1 - \delta'/(2m)$ distributed according to $D_{g_i,\gamma/4}$; by a union bound we have that with probability at least $1 - \delta'/2$ all m points are distributed this way (and thus none of them are $\perp$). We condition on this going forward. Definition 9 implies that $d_{\text{TV}}(D_{g_i,\gamma/4}, \mathcal{U}_{g_i^{-1}(1)}) \leq \gamma/4$; together with the assumption that $|h_i^{-1}(1) \cap g_i^{-1}(1)|/|g_i^{-1}(1)| \geq \gamma/2$, this implies that each x^j independently has proability at least $\gamma/4$ of having $h(x) = 1$. Consequently, by the choice of m in Step 1, a standard multiplicative Chernoff bound implies that

$$\left|\alpha_i - \frac{|h^{-1}(1) \cap g^{-1}(1)|}{|g^{-1}(1)|}\right| \leq \mu \cdot \frac{|h^{-1}(1) \cap g^{-1}(1)|}{|g^{-1}(1)|}$$

with failure probability at most $\delta'/4$, giving (8).

Finally, Definition 8 gives that (9) holds with failure probability at most $\delta'/4$. This concludes the proof. □

Next we show how a high-accuracy estimate α_i of $|h_i^{-1}(1) \cap g_i^{-1}(1)|/|g_i^{-1}(1)|$ yields a deterministic approximate evaluation oracle for $\hat{D}_i'$.

Lemma 39. *Algorithm* `Simulate-Approx-Eval` *(which is deterministic) takes as input a value* $\alpha \in [0,1]$*, a string* $x \in \{-1,1\}^n$*, a parameter* κ*, (a circuit for)* $h : \{-1,1\}^n \to \{-1,1\}$*, and (a representation for)* $g : \{-1,1\}^n \to \{-1,1\}$*,* $g \in \mathcal{C}'$*, where* h, g *are obtained from a run of* $\mathcal{A}_{\text{inv}}'^{\mathcal{C}}$*. Suppose that*

$$\left|\alpha - \frac{|h^{-1}(1) \cap g^{-1}(1)|}{|g^{-1}(1)|}\right| \leq \mu \cdot \frac{|h^{-1}(1) \cap g^{-1}(1)|}{|g^{-1}(1)|}$$

and

$$\frac{|g^{-1}(1)|}{1+\tau} \leq \kappa \leq (1+\tau)|g^{-1}(1)|$$

where $\mu = \tau = \epsilon/40000$*. Then* `Simulate-Approx-Eval` *outputs a value* ρ *such that*

$$\frac{\hat{D}'(x)}{1+\beta} \leq \rho \leq (1+\beta)\hat{D}'(x), \tag{10}$$

where $\beta = \epsilon/192$*,* $\hat{D}$ *is the output distribution constructed in Step 3 of the run of* $\mathcal{A}_{\text{inv}}^{\mathcal{C}}$ *that produced* h, g*, and* $\hat{D}'$ *is* $\hat{D}$ *conditioned on* $\{-1,1\}^n$ *(excluding* $\perp$*).*

Proof. The `Simulate-Approx-Eval` procedure is very simple. Given an input $x \in \{-1,1\}^n$ it evaluates both g and h on x, and if either evaluates to -1 it returns the value 0. If both evaluate to 1 then it returns the value $1/(\kappa\alpha)$.

For the correctness proof, note first that it is easy to see from the definition of the sampler C_f (Step 3 of $\mathcal{A}'^{\mathcal{C}}_{\mathrm{inv}}$) and Definition 9 (recall that the approximate uniform generator $\mathcal{A}^{\mathcal{C}'}_{\mathrm{gen}}(g)$ only outputs strings that satisfy g) that if $x \in \{-1,1\}^n$, $x \notin h^{-1}(1) \cap g^{-1}(1)$ then $\hat{D}$ has zero probability of outputting x, so Simulate-Approx-Eval behaves appropriately in this case.

Now suppose that $h(x) = g(x) = 1$. We first show that the value $1/(\kappa\alpha)$ is multiplicatively close to $1/|h^{-1}(1) \cap g^{-1}(1)|$. Let us write A to denote $|g^{-1}(1)|$ and B to denote $|h^{-1}(1) \cap g^{-1}(1)|$. With this notation we have

$$\left|\alpha - \frac{B}{A}\right| \leq \mu \cdot \frac{B}{A} \qquad \text{and} \qquad \frac{A}{1+\tau} \leq \kappa \leq (1+\tau)A.$$

Consequently, we have

$$B(1-\mu-\tau) \leq B \cdot \frac{1-\mu}{1+\tau} = \frac{B}{A}(1-\mu) \cdot \frac{A}{1+\tau} \leq \kappa\alpha \leq \frac{B}{A}(1+\mu) \cdot (1+\tau)A \leq B(1+2\mu+2\tau),$$

and hence

$$\frac{1}{B} \cdot \frac{1}{1+2\mu+2\tau} \leq \frac{1}{\kappa\alpha} \leq \frac{1}{B} \cdot \frac{1}{1-\mu-\tau}. \tag{11}$$

Now consider any $x \in h^{-1}(1) \cap g^{-1}(1)$. By Definition 9 we have that

$$\frac{1}{1+\epsilon_3} \cdot \frac{1}{|g^{-1}(1)|} \leq D_{g,\epsilon_3}(x) \leq (1+\epsilon_3) \cdot \frac{1}{|g^{-1}(1)|}.$$

Since a draw from $\hat{D}'$ is obtained by taking a draw from D_{g,ϵ_3} and conditioning on it lying in $h^{-1}(1)$, it follows that we have

$$\frac{1}{1+\epsilon_3} \cdot \frac{1}{B} \leq \hat{D}'(x) \leq (1+\epsilon_3) \cdot \frac{1}{B}.$$

Combining this with (11) and recalling that $\mu = \tau = \epsilon/40000$ and $\epsilon_3 = \epsilon\gamma/48000$, we get (10) as desired. □

3.5 The final algorithm: Proof of Theorem 21. Finally we are ready to give the inverse approximate uniform generation algorithm $\mathcal{A}^{\mathcal{C}}_{\mathrm{inv}}$ for $\mathcal{C}$.

```
Algorithm $\mathcal{A}^{\mathcal{C}}_{\mathrm{inv}}(\mathcal{U}_{f^{-1}(1)}, \epsilon, \delta)$
Input: Independent samples from $\mathcal{U}_{f^{-1}(1)}$, accuracy and confidence parameters $\epsilon, \delta$.
Output: With probability $1-\delta$ outputs an $\epsilon$-sampler $C_f$ for $\mathcal{U}_{f^{-1}(1)}$ .

  1. For $i = 1$ to $k = O(n/\epsilon)$:
      (a) Set $\widehat{p}_i \overset{\mathrm{def}}{=} (1+\epsilon)^{i-1}/2^n$.
      (b) Run $\mathcal{A}'^{\mathcal{C}}_{\mathrm{inv}}(\mathcal{U}_{f^{-1}(1)}, \epsilon/12, \delta/3, \widehat{p}_i)$. Let $g_i \in \mathcal{C}'$ be the function constructed in Step 1, $h_i$ be
          the hypothesis function constructed in Step 2(e), and $(C_f)_i$ be the sampler for distribution
          $\hat{D}_i$ constructed in Step 3.
      (c) Run Check$(g_i, h_i, \delta/3, \epsilon)$. If it returns a pair $(\alpha_i, \kappa_i)$ then add $i$ to the set $S$ (initially
          empty).
```

2. Run the hypothesis testing procedure $\mathcal{T}^{\mathcal{U}_{f^{-1}(1)}}$ over the set $\{\hat{D}'_i\}_{i\in S}$ of hypothesis distributions, using accuracy parameter $\epsilon/12$ and confidence parameter $\delta/3$. Here $\mathcal{T}^{\mathcal{U}_{f^{-1}(1)}}$ is given access to $\mathcal{U}_{f^{-1}(1)}$, uses the samplers $(C_f)_i$ to generate draws from distributions $\hat{D}'_i$ (see Remark 18), and uses the procedure `Simulate-Approx-Eval`$(\alpha_i, \kappa_i, h_i, g_i)$ for the $(1+\epsilon/192)$-approximate evaluation oracle $\mathrm{EVAL}_{\hat{D}'_i}$ for $\hat{D}'_i$. Let $i^\star \in S$ be the index of the distribution that it returns.

3. Output the sampler $(C_f)_{i^\star}$.

Proof of Theorem 21: Let $p \equiv \mathbf{Pr}_{x\in\mathcal{U}_n}[f(x)=1]$ denote the true fraction of satisfying assignments for f in $\{-1,1\}^n$. Let i^* be the element of $[k]$ such that $p \leq \widehat{p}_{i^*} < (1+\epsilon/6)p$. By Theorem 30, with probability at least $1-\delta/3$ we have that both

(i) $(C_f)_{i^*}$ is a sampler for a distribution $\hat{D}_{i^*}$ such that $d_{\mathrm{TV}}(\hat{D}_{i^*}, \mathcal{U}_{f^{-1}(1)}) \leq \epsilon/6$; and

(ii) $|h_{i^*}^{-1}(1) \cap g_{i^*}^{-1}(1)|/|g_{i^*}^{-1}(1)| \geq \gamma/2$.

We condition on these two events holding. By Lemma 38, with probability at least $1-\delta/3$ the procedure `Check` outputs a value α_{i^*} such that

$$\left|\alpha_{i^*} - \frac{|h_{i^*}^{-1}(1) \cap g_{i^*}^{-1}(1)|}{|g_{i^*}^{-1}(1)|}\right| \leq \mu \cdot \frac{|h_{i^*}^{-1}(1) \cap g_{i^*}^{-1}(1)|}{|g_{i^*}^{-1}(1)|}$$

for $\mu = \epsilon/40000$. We condition on this event holding. Now Lemma 39 implies that `Simulate-Approx-Eval`$((C_f)_{i^*})$ meets the requirements of a $(1+\beta)$-approximate evaluation oracle for $\mathrm{EVAL}_{\hat{D}'_{i^*}}$ from Proposition 12, for $\beta = \frac{\epsilon}{192}$. Hence by Proposition 12 (or more precisely by Remark 18) with probability at least $1-\delta/3$ the index $i^\star$ that $\mathcal{T}^{\mathcal{U}_{f^{-1}(1)}}$ returns is such that $\hat{D}'_{i^\star}$ is an $\epsilon/2$-sampler for $\mathcal{U}_{f^{-1}(1)}$ as desired.

As in the proof of Theorem 30, the claimed running time bound is a straightforward consequence of the various running time bounds established for all the procedures called by $\mathcal{A}_{\mathrm{inv}}^{\mathcal{C}}$. This concludes the proof of our general positive result, Theorem 21. □

4 Linear Threshold Functions

In this section we apply our general framework from Section 3 to prove Theorem 2, i.e., obtain a polynomial time algorithm for the problem of inverse approximate uniform generation for the class $\mathcal{C} = \mathbf{LTF}_n$ of n-variable linear threshold functions over $\{-1,1\}^n$. More formally, we prove:

Theorem 40. *There is an algorithm $\mathcal{A}_{\mathrm{inv}}^{\mathbf{LTF}}$ which is a $\mathrm{poly}(n, 1/\epsilon, \log(1/\delta))$-time inverse approximate uniform generation algorithm for the class $\mathbf{LTF}_n$.*

The above theorem will follow as an application of Theorem 21 for $\mathcal{C}' = \mathcal{C} = \mathbf{LTF}_n$. The literature provides us with three of the four ingredients that our general approach requires for LTFs – approximate uniform generation, approximate counting, and Statistical Query learning – and our main technical contribution is giving the fourth necessary ingredient, a densifier. We start by recalling the three known ingredients in the following subsection.

4.1 Tools from the literature. We first record two efficient algorithms for approximate uniform generation and approximate counting for $\mathbf{LTF}_n$, due to Dyer [Dye03]:

Theorem 41. *(approximate uniform generation for $\mathbf{LTF}_n$, [Dye03]) There is an algorithm $\mathcal{A}_{\text{gen}}^{\mathbf{LTF}}$ that on input (a weights–based representation of) an arbitrary $h \in \mathbf{LTF}_n$ and a confidence parameter $\delta > 0$, runs in time $\mathrm{poly}(n, \log(1/\delta))$ and with probability $1 - \delta$ outputs a point x such that $x \sim \mathcal{U}_{h^{-1}(1)}$.*

We note that the above algorithm gives us a somewhat stronger guarantee than that in Definition 9. Indeed, the algorithm $\mathcal{A}_{\text{gen}}^{\mathbf{LTF}}$ with high probability outputs a point $x \in \{-1,1\}^n$ whose distribution is *exactly* $\mathcal{U}_{h^{-1}(1)}$ (as opposed to a point whose distribution is *close* to $\mathcal{U}_{h^{-1}(1)}$).

Theorem 42. *(approximate counting for $\mathbf{LTF}_n$, [Dye03]) There is an algorithm $\mathcal{A}_{\text{count}}^{\mathbf{LTF}}$ that on input (a weights–based representation of) an arbitrary $h \in \mathbf{LTF}_n$, an accuracy parameter $\epsilon > 0$ and a confidence parameter $\delta > 0$, runs in time $\mathrm{poly}(n, 1/\epsilon, \log(1/\delta))$ and outputs $\widehat{p} \in [0,1]$ that with probability $1 - \delta$ satisfies $\widehat{p} \in [1-\epsilon, 1+\epsilon] \cdot \mathbf{Pr}_{x \sim \mathcal{U}_n}[h(x) = 1]$.*

We also need an efficient SQ learning algorithm for halfpaces. This is provided to us by a result of Blum et. al. [BFKV97]:

Theorem 43. *(SQ learning algorithm for $\mathbf{LTF}_n$, [BFKV97]) There is a distribution-independent SQ learning algorithm $\mathcal{A}_{\mathrm{SQ}}^{\mathbf{LTF}}$ for $\mathbf{LTF}_n$ that has running time $t_1 = \mathrm{poly}(n, 1/\epsilon, \log(1/\delta))$, uses at most $t_2 = \mathrm{poly}(n)$ time to evaluate each query, and requires tolerance of its queries no smaller than $\tau = 1/\mathrm{poly}(n, 1/\epsilon)$.*

4.2 A densifier for $\mathbf{LTF}_n$. The last ingredient we need in order to apply our Theorem 21 is a computationally efficient densifer for $\mathbf{LTF}_n$. This is the main technical contribution of this section and is summarized in the following theorem:

Theorem 44. *(efficient proper densifier for $\mathbf{LTF}_n$) Set $\gamma(\epsilon, \delta, n) \overset{def}{=} \Theta\left(\delta/(n^2 \log n)\right)$. There is an $(\epsilon, \gamma, \delta)$–densifier $\mathcal{A}_{\text{den}}^{\mathbf{LTF}}$ for $\mathbf{LTF}_n$ that, for any input parameters $0 < \epsilon, \delta$, $1/2^n \leq \widehat{p} \leq 1$, outputs a function $g \in \mathbf{LTF}_n$ and runs in time $\mathrm{poly}(n, 1/\epsilon, \log(1/\delta))$.*

Discussion and intuition. Before we prove Theorem 44, we provide some intuition. Let $f \in \mathbf{LTF}_n$ be the unknown LTF and suppose that we would like to design an $(\epsilon, \gamma, \delta)$–densifier $\mathcal{A}_{\text{den}}^{\mathbf{LTF}}$ for f. That is, given sample access to $\mathcal{U}_{f^{-1}(1)}$, and a number $\widehat{p}$ satisfying $p \leq \widehat{p} < (1+\epsilon)p$, where $p = \mathbf{Pr}_{x \in \mathcal{U}_n}[f(x) = 1]$, we would like to efficiently compute (a weights–based representation for) an LTF $g : \{-1,1\}^n \to \{-1,1\}$ such that the following conditions are satisfied:

(a) $\mathbf{Pr}_{x \sim \mathcal{U}_{f^{-1}(1)}}[g(x) = 1] \geq 1 - \epsilon$, and

(b) $\mathbf{Pr}_{x \sim \mathcal{U}_n}[g(x) = 1] \leq (1/\gamma) \cdot \mathbf{Pr}_{x \sim \mathcal{U}_n}[f = 1]$.

(While condition (b) above appears slightly different than property (b) in our Definition 20, because of property (a), the two statements are essentially equivalent up to a factor of $1/(1-\epsilon)$ in the value of γ.)

We start by noting that it is easy to handle the case that $\widehat{p}$ is large. In particular, observe that if $\widehat{p} \geq 2\gamma$ then $p = \mathbf{Pr}_{x \sim \mathcal{U}_n}[f(x) = 1] \geq \widehat{p}/(1+\epsilon) \geq \widehat{p}/2 \geq \gamma$, and we can just output $g \equiv 1$ since it clearly satisfies both properties of the definition. For the following intuitive discussion we will henceforth assume that $\widehat{p} \leq 2\gamma$.

Recall that our desired function g is an LTF, i.e., $g(x) = \mathrm{sign}(v \cdot x - t)$, for some $(v, t) \in \mathbb{R}^{n+1}$. Recall also that our densifier has sample access to $\mathcal{U}_{f^{-1}(1)}$, so it can obtain random positive examples of f, each of which gives a linear constraint over the v, t variables. Hence a natural first approach is to attempt to construct an appropriate linear program over these variables whose feasible solutions satisfy conditions (a) and (b) above. We begin by analyzing this approach; while it turns out to not quite work, it will gives us valuable intuition for our actual algorithm, which is presented further below.

Note that following this approach, condition (a) is relatively easy to satisfy. Indeed, consider any $\epsilon > 0$ and suppose we want to construct an LTF $g = \mathrm{sign}(v \cdot x - t)$ such that $\mathbf{Pr}_{x \sim \mathcal{U}_{f^{-1}(1)}}[g(x) = 1] \geq 1 - \epsilon$. This can be done as follows: draw a set S_+ of $N_+ = \Theta\left((1/\epsilon) \cdot (n^2 + \log(1/\delta))\right)$ samples from $\mathcal{U}_{f^{-1}(1)}$ and consider a linear program $\mathcal{LP}_+$ with variables $(w, \theta) \in \mathbb{R}^{n+1}$ that enforces all these examples to be positive. That is, for each $x \in S_+$, we will have an inequality $w \cdot x \geq \theta$. It is clear that $\mathcal{LP}_+$ is feasible (any weights–based representation for f is a feasible solution) and that it can be solved in $\mathrm{poly}(n, 1/\epsilon, \log(1/\delta))$ time, since it is defined by $O(N_+)$ many linear constraints and the coefficients of the constraint matrix are in $\{\pm 1\}$. The following simple claim shows that with high probability any feasible solution of $\mathcal{LP}_+$ satisfies condition (a):

Claim 45. *With probability at least* $1 - \delta$ *over the sample* S_+*,* any $g \in \mathbf{LTF}_n$ *consistent with* S_+ *satisfies condition (a).*

Proof. Consider an LTF g and suppose that it does not satisfy condition (a), i.e., $\mathbf{Pr}_{x \sim \mathcal{U}_n}[g(x) = -1 | f(x) = 1] > \epsilon$. Since each sample $x \in S_+$ is uniformly distributed in $f^{-1}(1)$, the probability it does not "hit" the set $g^{-1}(-1) \cap f^{-1}(1)$ is at most $1 - \epsilon$. The probability that *no* sample in S_+ hits $g^{-1}(-1) \cap f^{-1}(1)$ is thus at most $(1 - \epsilon)^{N_+} \leq \delta / 2^{n^2}$. Recalling that there exist at most 2^{n^2} distinct LTFs over $\{-1, 1\}^n$ [Mur71], it follows by a union bound that the probability there exists an LTF that does not satisfy condition (a) is at most δ as desired. □

The above claim directly implies that with high probability *any* feasible solution (w^*, θ^*) to $\mathcal{LP}_+$ is such that $g^*(x) = \mathrm{sign}(w^* \cdot x - \theta^*)$ satisfies condition (a). Of course, an arbitrary feasible solution to $\mathcal{LP}_+$ is by no means guaranteed to satisfy condition (b). (Note for example that the constant 1 function is certainly feasible for $\mathcal{LP}_+$.) Hence, a natural idea is to include additional constraints in our linear program so that condition (b) is also satisfied.

Along these lines, consider the following procedure: Draw a set $S_-$of $N_- = \lfloor \delta / \widehat{p} \rfloor$ uniform unlabeled samples from $\{-1, 1\}^n$ and label them negative. That is, for each sample $x \in S_-$, we add the constraint $w \cdot x < \theta$ to our linear program. Let $\mathcal{LP}$ be the linear program that contains all the constraints defined by $S_+ \cup S_-$. It is not hard to prove that with probability at least $1 - 2\delta$ over the sample S_-, we have that $S_- \subseteq f^{-1}(-1)$ and hence (any weight based representation of) f is a feasible solution to $\mathcal{LP}$. In fact, it is possible to show that *if* γ *is sufficiently small* — roughly, $\gamma \leq \delta / \left(4(n^2 + \log(1/\delta))\right)$ is what is required — then with high probability each solution to $\mathcal{LP}$ also satisfies condition (b). The catch, of course, is that the above procedure is not computationally efficient because N_- may be very large – if $\widehat{p}$ is very small, then it is infeasible even to write down the linear program $\mathcal{LP}$.

Algorithm Description. The above discussion motivates our actual densifier algorithm as follows: The problem with the above described naive approach is that it generates (the potentially very large set) S_- all at once at the beginning of the algorithm. Note that having a large set S_- is not necessarily in and of itself a problem, since one could potentially use the ellipsoid method to solve $\mathcal{LP}$ if one could obtain an efficient separation oracle. Thus intuitively, if one had an online algorithm which would generate S_- *on the fly*, then one could potentially get a feasible solution to $\mathcal{LP}$ in polynomial time. This serves as the intuition behind our actual algorithm.

More concretely, our densifier $\mathcal{A}_{\mathrm{den}}^{\mathbf{LTF}}$ will invoke a computationally efficient *online* learning algorithm for LTFs. In particular, $\mathcal{A}_{\mathrm{den}}^{\mathbf{LTF}}$ will run the online learner $\mathcal{A}_{\mathrm{MT}}^{\mathbf{LTF}}$ for a sequence of stages and in each stage it will provide as counterexamples to $\mathcal{A}_{\mathrm{MT}}^{\mathbf{LTF}}$ judiciously chosen labeled examples, which will be positive for the online learner's current hypothesis, but negative for f (with high probability). Since $\mathcal{A}_{\mathrm{MT}}^{\mathbf{LTF}}$ makes a small number of mistakes in the worst-case, this process is guaranteed to terminate after a small number of stages (since in each stage we *force* the online learner to make a mistake).

We now provide the details. We start by recalling the notion of *online learning* for a class $\mathcal{C}$ of boolean functions. In the online model, learning proceeds in a sequence of stages. In each stage the learning algorithm is given an unlabeled example $x \in \{-1,1\}^n$ and is asked to predict the value $f(x)$, where $f \in \mathcal{C}$ is the unknown target concept. After the learning algorithm makes its prediction, it is given the correct value of $f(x)$. The goal of the learner is to identify f while minimizing the total number of mistakes. We say that an online algorithm learns class $\mathcal{C}$ with mistake bound M if it makes at most M mistakes on *any* sequence of examples consistent with some $f \in \mathcal{C}$. Our densifier makes essential use of a computationally efficient online learning algorithm for the class of linear threshold functions by Maass and Turan [MT94]:

Theorem 46. *([MT94], Theorem 3.3) There exists a* $\mathrm{poly}(n)$ *time deterministic online learning algorithm* $\mathcal{A}_{\mathrm{MT}}^{\mathbf{LTF}}$ *for the class* $\mathbf{LTF}_n$ *with mistake bound* $M(n) \overset{def}{=} \Theta(n^2 \log n)$. *In particular, at every stage of its execution, the current hypothesis maintained by* $\mathcal{A}_{\mathrm{MT}}^{\mathbf{LTF}}$ *is a (weights–based representation of an) LTF that is consistent with all labeled examples received so far.*

We note that the above algorithm works by reducing the problem of online learning for LTFs to a convex optimization problem. Hence, one can use any efficient convex optimization algorithm to do online learning for LTFs, e.g. the ellipsoid method [Kha80, GLS88]. The mistake bound in the above theorem follows by plugging in the algorithm of Vaidya [Vai89, Vai96].

We now proceed with a more detailed description of our densifier followed by pseudocode and a proof of correctness. As previously mentioned, the basic idea is to execute the online learner to learn f while cleverly providing counterexamples to it in each stage of its execution. Our algorithm starts by sampling N_+ samples from $\mathcal{U}_{f^{-1}(1)}$ and making sure that these are classified correctly by the online learner. This step guarantees that our final solution will satisfy condition (a) of the densifier. Let $h \in \mathbf{LTF}_n$ be the current hypothesis at the end of this process. If h satisfies condition (b) (we can efficiently decide this by using our approximate counter for $\mathbf{LTF}_n$), we output h and terminate the algorithm. Otherwise, we use our approximate uniform generator to construct a uniform satisfying assignment $x \in \mathcal{U}_{h^{-1}(1)}$ and we label it negative, i.e., we give the labeled example $(x,-1)$ as a counterexample to the online learner. Since h does not satisfy condition (b), i.e., it has "many" satisfying assignments, it follows that with high probability (roughly, at least $1-\gamma$) over the choice of $x \in \mathcal{U}_{h^{-1}(1)}$, the point x output by the generator will indeed be negative for f. We continue this process for a number of stages. If all counterexamples thus generated are indeed consistent with f (this happens with probability roughly $1-\gamma \cdot M$, where $M = M(n) = \Theta(n^2 \log n)$ is an upper bound on the number of stages), after at most M stages we have either found a hypothesis h satisfying condition (b) or the online learner terminates. In the latter case, the current hypothesis of the online learner is identical to f, as follows from Theorem 46. (Note that the above argument puts an upper bound of $O(\delta/M)$ on the value of γ.) Detailed pseudocode follows:

```
Algorithm $\mathcal{A}_{\mathrm{den}}^{\mathbf{LTF}}(\mathcal{U}_{f^{-1}(1)}, \epsilon, \delta, \widehat{p})$:
Input: Independent samples from $\mathcal{U}_{f^{-1}(1)}$, parameters $\epsilon, \delta > 0$, and a value $1/2^n \le \widehat{p} \le 1$.
Output: If $p \le \widehat{p} \le (1+\epsilon)p$, with probability $1-\delta$ outputs a function $g \in \mathbf{LTF}_n$ satisfying conditions (a) and (b).

  1. Draw a set $S_+$ of $N_+ = \Theta\left((1/\epsilon) \cdot (n^2 + \log(1/\delta))\right)$ examples from $\mathcal{U}_{f^{-1}(1)}$.

  2. Initialize $i = 0$ and set $M \overset{\mathrm{def}}{=} \Theta(n^2 \log n)$.
     While $(i \le M)$ do the following:
```

```
    (a) Execute the $i$-th stage of $A^{\mathbf{LTF}}_{\mathrm{MT}}$ and let $h^{(i)} \in \mathbf{LTF}_n$ be its current hypothesis.
    (b) If there exists $x \in S_+$ with $h^{(i)}(x) = -1$ do the following:
        • Give the labeled example $(x, 1)$ as a counterexample to $A^{\mathbf{LTF}}_{\mathrm{MT}}$.
        • Set $i = i + 1$ and go to Step 2.
    (c) Run $\mathcal{A}^{\mathbf{LTF}}_{\mathrm{count}}(h^{(i)}, \epsilon, \delta/(4M))$ and let $\widehat{p_i}$ be its output.
    (d) Set $\gamma \overset{\text{def}}{=} \delta/(16M)$. If $\widehat{p_i} \leq \widehat{p}/\big(\gamma \cdot (1+\epsilon)^2\big)$ then output $h^{(i)}$;
    (e) otherwise, do the following:
        • Run $\mathcal{A}^{\mathbf{LTF}}_{\mathrm{gen}}(h^{(i)}, \delta/(4M))$ and let $x^{(i)}$ be its output.
        • Give the point $(x^{(i)}, -1)$ as a counterexample to $A^{\mathbf{LTF}}_{\mathrm{MT}}$.
        • Set $i = i + 1$ and go to Step 2.

3. Output the current hypothesis $h^{(i)}$ of $A^{\mathbf{LTF}}_{\mathrm{MT}}$.
```

Theorem 47. *Algorithm $\mathcal{A}^{\mathbf{LTF}}_{\mathrm{den}}(\mathcal{U}_{f^{-1}(1)}, \epsilon, \delta, \widehat{p})$ runs in time $\mathrm{poly}\,(n, 1/\epsilon, \log(1/\delta))$. If $p \leq \widehat{p} < (1+\epsilon)p$ then with probability $1 - \delta$ it outputs a vector (w, θ) such that $g(x) = \mathrm{sign}(w \cdot x - \theta)$ satisfies conditions (a) and (b) at the start of Section 4.2.*

Proof. First note that by Claim 45, with probability at least $1 - \delta/4$ over S_+ any LTF consistent with S_+ will satisfy condition (a). We will condition on this event and also on the event that each call to the approximate counting algorithm and to the approximate uniform generator is successful. Since Step 2 involves at most M iterations, by a union bound, with probability at least $1 - \delta/4$ all calls to $\mathcal{A}^{\mathbf{LTF}}_{\mathrm{count}}$ will be successful, i.e., for all i we will have that $p_i/(1+\epsilon) \leq \widehat{p_i} \leq (1+\epsilon) \cdot p_i$, where $p_i = \mathbf{Pr}_{x \in \mathcal{U}_n}[h^{(i)}(x) = 1]$. Similarly, with failure probability at most $\delta/4$, all points $x^{(i)}$ constructed by $\mathcal{A}^{\mathbf{LTF}}_{\mathrm{gen}}$ will be uniformly random over $(h^{(i)})^{-1}(1)$. Hence, with failure probability at most $3\delta/4$ all three conditions will be satisfied.

Conditioning on the above events, if the algorithm outputs a hypothesis $h^{(i)}$ in Step 2(d), this hypothesis will certainly satisfy condition (b), since $p_i \leq (1+\epsilon)\widehat{p_i} \leq \widehat{p}/\big(\gamma \cdot (1+\epsilon)\big) \leq p/\gamma$. In this case, the algorithm succeeds with probability at least $1 - 3\delta/4$. It remains to show that if the algorithm returns a hypothesis in Step 3, it will be successful with probability at least $1 - \delta$. To see this, observe that if no execution of Step 2(e) generates a point $x^{(i)}$ with $f(x^{(i)}) = 1$, all the counterexamples given to $A^{\mathbf{LTF}}_{\mathrm{MT}}$ are consistent with f. Therefore, by Theorem 46, the hypothesis of Step 3 will be identical to f, which trivially satisfies both conditions.

We claim that with overall probability at least $1 - \delta/4$ all executions of Step 2(e) generate points $x^{(i)}$ with $f(x^{(i)}) = -1$. Indeed, fix an execution of Step 2(e). Since $\widehat{p_i} > \widehat{p}/\,\big((1+\epsilon)^2 \cdot \gamma\big)$, it follows that $p \leq (4\gamma)p_i$. Hence, with probability at least $1 - 4\gamma$ a uniform point $x^{(i)} \sim \mathcal{U}_{(h^i)^{-1}(1)}$ is a negative example for f, i.e., $x^{(i)} \in f^{-1}(-1)$. By a union bound over all stages, our claim holds except with failure probability $4\gamma \cdot M = \delta/4$, as desired. This completes the proof of correctness.

It remains to analyze the running time. Note that Step 2 is repeated at most $M = O(n^2 \log n)$ times. Each iteration involves (i) one round of the online learner $\mathcal{A}^{\mathbf{LTF}}_{\mathrm{MT}}$ (this takes $\mathrm{poly}(n)$ time by Theorem 46), (ii) one call of $\mathcal{A}^{\mathbf{LTF}}_{\mathrm{count}}$ (this takes $\mathrm{poly}(n, 1/\epsilon, \log(1/\delta))$ time by Theorem 42), and (iii) one call to $\mathcal{A}^{\mathbf{LTF}}_{\mathrm{gen}}$ (this takes $\mathrm{poly}(n, 1/\epsilon, \log(1/\delta))$ time by Theorem 41). This completes the proof of Theorem 47. □

5 DNFs

In this section we apply our general positive result, Theorem 21, to give a quasipolynomial-time algorithm for the inverse approximate uniform generation problem for s-term DNF formulas. Let $\mathbf{DNF}_{n,s}$ denote the class of all s-term DNF formulas over n Boolean variables (which for convenience we think of as $0/1$ variables). Our main result of this section is the following:

Theorem 48. *There is an algorithm $\mathcal{A}_{\text{inv}}^{\mathbf{DNF}_{n,s}}$ which is an inverse approximate uniform generation algorithm for the class $\mathbf{DNF}_{n,s}$. Given input parameters ϵ, δ the algorithm runs in time $\text{poly}\left(n^{\log(s/\epsilon)}, \log(1/\delta)\right)$.*

We note that even in the standard uniform distribution learning model the fastest known running time for learning s-term DNF formulas to accuracy ϵ is $\text{poly}(n^{\log(s/\epsilon)}, \log(1/\delta))$ [Ver90, Val12]. Thus it seems likely that obtaining a $\text{poly}(n, s, 1/\epsilon)$-time algorithm would require a significant breakthrough in computational learning theory.

For our application of Theorem 21 for DNFs we shall have $\mathcal{C} = \mathbf{DNF}_{n,s}$ and $\mathcal{C}' = \mathbf{DNF}_{n,t}$ for some t which we shall specify later. As in the case of LTFs, the literature provides us with three of the four ingredients that our general approach requires for DNF — approximate uniform generation, approximate counting, and Statistical Query learning (more on this below) — and our main technical contribution is giving the fourth necessary ingredient, a densifier. Before presenting and analyzing our densifier algorithm we recall the other three ingredients.

5.1 Tools from the literature. Karp, Luby and Madras [KLM89] have given approximate uniform generation and approximate counting algorithms for DNF formulas. (We note that [JVV86] give an efficient algorithm that with high probability outputs an *exactly* uniform satisfying assignment for DNFs.)

Theorem 49. *(Approximate uniform generation for DNFs, [KLM89]) There is an approximate uniform generation algorithm $\mathcal{A}_{\text{gen}}^{\mathbf{DNF}_{n,t}}$ for the class $\mathbf{DNF}_{n,t}$ that runs in time $\text{poly}(n, t, 1/\epsilon, \log(1/\delta))$.*

Theorem 50. *(Approximate counting for DNFs, [KLM89]) There is an approximate counting algorithm $\mathcal{A}_{\text{gen}}^{\mathbf{DNF}_{n,t}}$ for the class $\mathbf{DNF}_{n,t}$ that runs in time $\text{poly}(n, t, 1/\epsilon, \log(1/\delta))$.*

The fastest known algorithm in the literature for SQ learning s-term DNF formulas under arbitrary distributions runs in time $n^{O(n^{1/3}\log s)} \cdot \text{poly}(1/\epsilon)$ [KS04], which is much more than our desired running time bound. However, we will see that we are able to use known *malicious noise tolerant* SQ learning algorithms for learning *sparse disjunctions* over N Boolean variables rather than DNF formulas. In more detail, our densifier will provide us with a set of $N = n^{O(\log(s/\epsilon))}$ many conjunctions which is such that the target function f is very close to a disjunction (which we call f') over an unknown subset of at most s of these N conjunctions. Thus intuitively any learning algorithm for disjunctions, run over the "feature space" of conjunctions provided by the densifier, would succeed if the target function were f', but the target function is actually f (which is not necessarily exactly a disjunction over these N variables). Fortunately, known results on the malicious noise tolerance of specific SQ learning algorithms imply that it is in fact possible to use these SQ algorithms to learn f to high accuracy, as we now explain.

We now state the precise SQ learning result that we will use. The following theorem is a direct consequence of, e.g., Theorems 5 and 6 of [Dec93] or alteratively of Theorems 5 and 6 of [AD98]:

Theorem 51. *(Malicious noise tolerant* SQ *algorithm for learning sparse disjunctions) Let $\mathcal{C}_{\mathbf{DISJ},k}$ be the class of all disjunctions of length at most k over N Boolean variables $x_1, \dots, x_N$. There is a distribution-independent* SQ *learning algorithm $\mathcal{A}_{\text{SQ}}^{\mathbf{DISJ}}$ for $\mathcal{C}_{\mathbf{DISJ},k}$ that has running time $t_1 = \text{poly}(N, 1/\epsilon, \log(1/\delta))$, uses at most $t_2 = \text{poly}(N)$ time to evaluate each query, and requires tolerance of its queries no smaller than $\tau = 1/\text{poly}(k, 1/\epsilon)$. The algorithm outputs a hypothesis which is a disjunction over $x_1, \dots, x_N$.*

Moreover, there is a fixed polynomial $\ell(\cdot)$ such that algorithm $\mathcal{A}_{\rm SQ}^{\mathbf{DISJ}}$ has the following property: Fix a distribution D over $\{0,1\}^N$. Let f be an N-variable Boolean function which is such that $\mathbf{Pr}_{x\sim D}[f'(x) \neq f(x)] \leq \kappa$, where $f' \in \mathcal{C}_{\mathbf{DISJ},k}$ is some k-variable disjunction and $\kappa \leq \ell(\epsilon/k) < \epsilon/2$. Then if $\mathcal{A}_{\rm SQ}^{\mathbf{DISJ}}$ is run with a $\mathrm{STAT}(f,D)$ oracle, with probability $1-\delta$ it outputs a hypothesis h such that $\mathbf{Pr}_{x\sim D}[h(x) \neq f'(x)] \leq \epsilon/2$, and hence $\mathbf{Pr}_{x\sim D}[h(x \neq f(x)] \leq \epsilon$.

(We note in passing that at the heart of Theorem 51 is an *attribute-efficient* SQ algorithm for learning sparse disjunctions. Very roughly speaking, an attribute efficient SQ learning algorithm is one which can learn a target function over N variables, which actually depends only on an unknown subset of $k \ll N$ of the variables, using statistical queries for which the minimum value of the tolerance τ is "large." The intuition behind Theorem 51 is that since the distance between f and f' is much less than τ, the effect of using a $\mathrm{STAT}(f,\mathcal{D})$ oracle rather than a $\mathrm{STAT}(f',\mathcal{D})$ oracle is negligible, and hence the SQ algorithm will succeed whether it is run with f or f' as the target function.)

5.2 A densifier for $\mathbf{DNF}_{n,s}$ and the proof of Theorem 48. In this subsection we state our main theorem regarding the existence of densifiers for DNF formulas, Theorem 52, and show how Theorem 48 follows from this theorem.

Theorem 52. *Let $\gamma(n,s,1/\epsilon,1/\delta) = 1/(4n^{2\log(2s/\ell(\epsilon/s))}\log(s/\delta))$. Algorithm $\mathcal{A}_{\rm den}^{\mathbf{DNF}_{n,s}}(\mathcal{U}_{f^{-1}(1)},\epsilon,\delta,\widehat{p})$ outputs a collection $\mathcal{S}$ of conjunctions $C_1,\dots,C_{|\mathcal{S}|}$ and has the following performance guarantee: If $p \overset{def}{=} \mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x)=1] \leq \widehat{p} < (1+\epsilon)p$, then with probability at least $1-\delta$, the function $g(x) \overset{def}{=} \vee_{i\in[|\mathcal{S}|]}C_i$ satisfies the following:*

1. $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[g(x)=1] \geq 1-\epsilon$*;*
2. $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[f(x)=1] \geq \gamma(n,s,1/\epsilon,1/\delta)$.
3. *There is a DNF $f' = C_{i_1}\vee\cdots\vee C_{i_{s'}}$, which is a disjunction of $s' \leq s$ of the conjunctions $C_1,\dots,C_{|\mathcal{S}|}$, such that $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[f'(x) \neq f(x)] \leq \ell(\epsilon/s)$, where $\ell(\cdot)$ is the polynomial from Theorem 51.*

The size of $\mathcal{S}$ and the running time of $\mathcal{A}_{\rm den}^{\mathbf{DNF}_{n,s}}(\mathcal{U}_{f^{-1}(1)},\epsilon,\delta,\widehat{p})$ is $\mathrm{poly}(n^{\log(s/\epsilon)},\log(1/\delta))$.

With a slight abuse of terminology we may rephrase the above theorem as saying that $\mathcal{A}_{\rm den}^{\mathbf{DNF}_{n,s}}$ is a (ϵ,γ,δ)-densifier for function class $\mathcal{C} = \mathbf{DNF}_{n,s}$ using class $\mathcal{C}' = \mathbf{DNF}_{n,t}$ where $t = n^{O(\log(s/\epsilon))}$. We defer the description of Algorithm $\mathcal{A}_{\rm den}^{\mathbf{DNF}_{n,s}}$ and the proof of Theorem 52 to the next subsection.

Proof of Theorem 48. The proof is essentially just an application of Theorem 21. The only twist is the use of a SQ disjunction learning algorithm rather than a DNF learning algorithm, but the special properties of Algorithm $\mathcal{A}_{\rm SQ}^{\mathbf{DISJ}}$ let this go through without a problem.

In more detail, in Step 2(e) of Algorithm $\mathcal{A}_{\rm inv}'^{\mathcal{C}}$ (see Section 3.3), in the execution of Algorithm $\mathcal{A}_{\rm SQ-SIM}$, the SQ algorithm that is simulated is the algorithm $\mathcal{A}_{\rm SQ}^{\mathbf{DISJ}}$ run over the feature space $\mathcal{S}$ of all conjunctions that are output by Algorithm $\mathcal{A}_{\rm den}^{\mathbf{DNF}_{n,s}}$ in Step 1 of Algorithm $\mathcal{A}_{\rm inv}'^{\mathcal{C}}$ (i.e., these conjunctions play the role of variables $x_1,\dots,x_N$ for the SQ learning algorithm). Property (3) of Theorem 52 and Theorem 51 together imply that the algorithm $\mathcal{A}_{\rm SQ}^{\mathbf{DISJ}}$, run on a $\mathrm{STAT}(f,\mathcal{U}_{g^{-1}(1)})$ oracle with parameters ϵ,δ, would with probability $1-\delta$ output a hypothesis h' satisfying $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[h'(x) \neq f(x)] \leq \epsilon$. Hence the hypothesis h that is output by $\mathcal{A}_{\rm SQ-SIM}$ in Step 2(e) of Algorithm $\mathcal{A}_{\rm inv}'^{\mathcal{C}}$ fulfills the necessary accuracy (with respect to f under $D = \mathcal{U}_{g^{-1}(1)}$) and confidence requirements, and the overall algorithm $A_{\rm inv}^{\mathcal{C}}$ succeeds as described in Theorem 21.

Finally, combining the running time bounds of $\mathcal{A}_{\text{den}}^{\mathbf{DNF}_{n,s}}$ and $\mathcal{A}_{\text{SQ}}^{\mathbf{DISJ}}$ with the time bounds of the other procedures described earlier, one can straightforwardly verify that the running time of the overall algorithm $\mathcal{A}_{\text{inv}}^{\mathcal{C}}$ is $\text{poly}(n^{\log(s/\epsilon)}, \log(1/\delta))$. $\square$

5.3 Construction of a densifier for $\mathbf{DNF}_{n,s}$ and proof of Theorem 52. Let $f = T_1 \vee \cdots \vee T_s$ be the target s-term DNF formula, where $T_1, \ldots, T_s$ are the terms (conjunctions). The high-level idea of our densifier is quite simple: If T_i is a term which is "reasonably likely" to be satisfied by a uniform draw of x from $f^{-1}(1)$, then T_i is at least "mildly likely" to be satisfied by $r = 2\log n$ consecutive independent draws of x from $f^{-1}(1)$. Such a sequence of draws $x^1, \ldots, x^r$ will with high probability *uniquely identify* T_i. By repeating this process sufficiently many times, with high probability we will obtain a pool $C_1, \ldots, C_{|\mathcal{S}|}$ of conjunctions which contains all of the terms T_i that are reasonably likely to be satisfied by a uniform draw of x from $f^{-1}(1)$. Theorem 52 follows straightforwardly from this.

We give detailed pseudocode for our densifier algorithm below:

```
Algorithm $\mathcal{A}_{\text{den}}^{\mathbf{DNF}_{n,s}}(\mathcal{U}_{f^{-1}(1)}, \epsilon, \delta, \widehat{p})$:
Input: Independent samples from $\mathcal{U}_{f^{-1}(1)}$, parameters $\epsilon, \delta > 0$, and a value $1/2^n < \widehat{p} \leq 1$.
Output: If $p \leq \widehat{p} \leq (1+\epsilon)p$, with probability $1-\delta$ outputs a set $\mathcal{S}$ of conjunctions $C_1, \ldots, C_{|\mathcal{S}|}$ as described in Theorem 52

  1. Initialize set $\mathcal{S}$ to $\emptyset$. Let $\ell(\cdot)$ be the polynomial from Theorem 51.
  2. For $i = 1$ to $M = 2n^{2\log(2s/\ell(\epsilon/s))}\log(s/\delta)$, repeat the following:
      (a) Draw $r = 2\log n$ satisfying assignments $x^1, \ldots, x^r$ from $\mathcal{U}_{f^{-1}(1)}$.
      (b) Let $C_i$ be the AND of all literals that take the same value in all $r$ strings $x^1, \ldots, x^r$ (note $C_i$ may be the empty conjunction). We say $C_i$ is a candidate term.
      (c) If the candidate term $C_i$ satisfies $\mathbf{Pr}_{x \sim \mathcal{U}_n}[C_i(x) = 1] \leq \widehat{p}$ then add $C_i$ to the set $\mathcal{S}$.
  3. Output $\mathcal{S}$.
```

The following crucial claim makes the intuition presented at the start of this subsection precise:

Claim 53. *Suppose T_j is a term in f such that $\mathbf{Pr}_{x \sim \mathcal{U}_{f^{-1}(1)}}[T_j(x) = 1] \geq \ell(\epsilon/s)/(2s)$. Then with probability at least $1-\delta/s$, term T_j is a candidate term at some iteration of Step 2 of Algorithm $\mathcal{A}_{\text{den}}^{\mathbf{DNF}_{n,s}}(\mathcal{U}_{f^{-1}(1)}, \epsilon, \delta, \widehat{p})$.*

Proof. Fix a given iteration i of the loop in Step 2. With probability at least

$$(\ell(\epsilon/s)/(2s))^{2\log n} = (1/n)^{2\log(2s/\ell(\epsilon/s))},$$

all $2\log n$ points $x^1, \ldots, x^{2\log n}$ satisfy T_j; let us call this event E, and condition on E taking place. We claim that conditioned on E, the points $x^1, \ldots, x^{2\log n}$ are independent uniform samples drawn from $T_j^{-1}(1)$. (To see this, observe that each x^i is an independent sample chosen uniformly at random from $f^{-1}(1) \cap T_j^{-1}$; but $f^{-1}(1) \cap T_j^{-1}(1)$ is identical to $T_j^{-1}(1)$.) Given that $x^1, \ldots, x^{2\log n}$ are independent uniform samples drawn from $T_j^{-1}(1)$, the probability that any literal which is *not* present in T_j is contained in C_i (i.e., is satisfied by all $2\log n$ points) is at most $2n/n^2 \leq 1/2$. So with overall probability at least $\frac{1}{2n^{2\log(2s/\ell(\epsilon/s))}}$, the term T_j is a candidate term at iteration i. Consequently T_j is a candidate term at some iteration with probability at least $1-\delta/s$, by the choice of $M = 2n^{2\log(2s/\ell(\epsilon/s))}\log(s/\delta)$. $\square$

Now we are ready to prove Theorem 52:

Proof of Theorem 52. The claimed running time bound of $\mathcal{A}_{\text{den}}^{\mathbf{DNF}_{n,s}}$ is easily verified, so it remains only to establish (1)-(3). Fix $\widehat{p}$ such that $p \leq \widehat{p} < (1+\epsilon)p$ where $p = \mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x)=1]$.

Consider any fixed term T_j of f such that $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[T_j(x)=1] \geq \ell(\epsilon/s)/(2s)$. By Claim 53 we have that with probability at least $1-\delta/s$, term T_j is a candidate term at some iteration of Step 2 of the algorithm. We claim that in step (c) of this iteration the term T_j will in fact be added to $\mathcal{S}$. This is because by assumption we have

$$\mathbf{Pr}_{x\sim\mathcal{U}_n}[T_j(x)=1] \leq \mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x)=1] = p \leq \widehat{p}.$$

So by a union bound, with probability at least $1-\delta$ every term T_j in f such that $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[T_j(x)=1] \geq \ell(\epsilon/s)/(2s)$ is added to $\mathcal{S}$.

Let L be the set of those terms T_j in f that have $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[T_j(x)=1] \geq \ell(\epsilon/s)/(2s)$. Let f' be the DNF obtained by taking the OR of all terms in L. By a union bound over the (at most s) terms that are in f but not in f', we have $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[f'(x)=1] \geq 1-\ell(\epsilon/s)/2$. Since g (as defined in Theorem 52 has $g(x)=1$ whenever $f'(x)=1$, it follows that $\mathbf{Pr}_{x\sim\mathcal{U}_{f^{-1}(1)}}[g(x)=1] \geq 1-\ell(\epsilon/s)/2 \geq 1-\epsilon$, giving item (1) of the theorem.

For item (2), since $f(x)=1$ whenever $f'(x)=1$, we have $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[f(x)=1] \geq \mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[f'(x)=1]$. Every x such that $f'(x)=1$ also has $g(x)=1$ so to lower bound $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[f'(x)=1]$ it is enough to upper bound the number of points in $g^{-1}(1)$ and lower bound the number of points in $f'^{-1}(1)$. Since each C_i that is added to $\mathcal{S}$ is satisfied by at most $\widehat{p}2^n \leq (1+\epsilon)p2^n$ points, we have that $|g^{-1}(1)| \leq (1+\epsilon)pM2^n$. Since at least $1-\epsilon$ of the points that satisfy f also satisfy f', we have that $|f'^{-1}(1)| \geq p(1-\epsilon)2^n$. Thus we have $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[f'(x)=1] \geq p(1-\epsilon)/((1+\epsilon)pM) = \frac{1-\epsilon}{1+\epsilon}\cdot\frac{1}{M} > \frac{1}{2M}$, giving (2).

Finally, for (3) we have that $f(x) \neq f'(x)$ only on those inputs that have $f(x)=1$ but $f'(x)=0$ (because some term outside of L is satisfied by x and no term in L is satisfied by x). Even if all such inputs x lie in $g^{-1}(1)$ (the worst case), there can be at most $(\ell(\epsilon/s)/2)p2^n$ such inputs, and we know that $|g^{-1}(1)| \geq |f^{-1}(1)| \geq p(1-\epsilon)2^n$. So we have $\mathbf{Pr}_{x\sim\mathcal{U}_{g^{-1}(1)}}[f(x)\neq f'(x)] \leq \frac{\ell(\epsilon/s)/2}{1-\epsilon} \leq \ell(\epsilon/s)$, and we have (3) as desired. □

5.4 Inverse approximate uniform generation for k-DNFs. We briefly note that our general approach immediately yields an efficient inverse approximate uniform generation algorithm for the class of k-DNFs for any constant k. Let k-$\mathbf{DNF}$ denote the class of all k-DNFs over n Boolean variables, i.e., DNF formulas in which each term (conjunction) has at most k literals.

Theorem 54. *There is an algorithm $\mathcal{A}_{\text{inv}}^{k\text{-}\mathbf{DNF}}$ which is an inverse approximate uniform generation algorithm for the class k-$\mathbf{DNF}$. Given input parameters ϵ, δ the algorithm runs in time* $\text{poly}\left(n^k, 1/\epsilon, \log(1/\delta)\right)$.

For any k-DNF f it is easy to see that $\mathbf{Pr}_{x\sim\mathcal{U}_n}[f(x)=1] \geq 1/2^k$, and consequently the constant 1 function is a γ-densifier for k-$\mathbf{DNF}$ with $\gamma = 1/2^k$. Theorem 54 then follows immediately from Theorem 21, using the algorithms for approximate uniform generation and counting of DNF formulas mentioned above [KLM89] together with well-known algorithms for SQ learning k-DNF formulas in $\text{poly}(n^k, 1/\epsilon, \log(1/\delta))$ time [Kea98].

6 Negative results for inverse approximate uniform generation

In this section, we will prove hardness results for inverse approximate uniform generation problems for specific classes $\mathcal{C}$ of Boolean functions. As is standard in computational learning theory, our hardness results are based on cryptographic hardness assumptions. The hardness assumptions we use are well studied assumptions in cryptography such as the strong RSA assumption, Decisional Diffie Hellman problem, and hardness of learning parity with noise.

As was alluded to in the introduction, in light of the standard approach, there are two potential barriers to obtaining inverse approximate uniform generation algorithms for a class $\mathcal{C}$ of functions. The first is that "reconstructing" the object from class $\mathcal{C}$ may be hard, and the second is that sampling approximately uniform random satisfying assignments from the reconstructed object may be hard. While any hard inverse approximate uniform generation problem must be hard because of one of these two potential barriers, we emphasize here that even if one of the two steps in the standard approach is shown to be hard, this does not constitute a proof of hardness of the overall inverse approximate uniform generation problem, as there is may exist some efficient algorithm for the class $\mathcal{C}$ which departs from the standard approach. Indeed, we will give such an example in Section 7, where we give an efficient algorithm for a specific inverse approximate uniform generation problem that does not follow the standard approach. (In fact, for that problem, the second step of the standard approach is provably no easier than the well-known graph automorphism problem, which has withstood several decades of effort towards even getting a sub-exponential time algorithm.)

Our hardness results come in two flavors. Our first hardness results, based on signature schemes, are for problems where it is provably hard (of course under a computational hardness assumption) to sample approximately uniform satisfying assignments. In contrast, our hardness results of the second flavor are based on Message Authentication Codes (MACs). We give such a result for a specific class $\mathcal{C}$ which has the property that it is actually easy to sample uniform satisfying assignments for functions in $\mathcal{C}$; hence, in an informal sense, it is the first step in the standard approach that is algorithmically hard for this problem. The following subsections describe all of our hardness results in detail.

6.1 Hardness results based on signature schemes. In this subsection we prove a general theorem, Theorem 60, which relates the hardness of inverse approximate uniform generation to the existence of certain secure signature schemes in cryptography. Roughly speaking, Theorem 60 says that if secure signature schemes exist, then the inverse approximate uniform generation problem is computationally hard for any class $\mathcal{C}$ which is Levin-reducible from CIRCUIT-SAT. We will use this general result to establish hardness of inverse approximate uniform generation for several natural classes of functions, including 3-CNF formulas, intersections of two halfspaces, and degree-2 polynomial threshold functions (PTFs).

We begin by recalling the definition of public key signature schemes. For an extensive treatment of signature schemes, see [Gol04]. For simplicity, and since it suffices for our purposes, we only consider schemes with deterministic verification algorithms.

Definition 55. *A signature scheme is a triple (G, S, V) of polynomial-time algorithms with the following properties :*

- **(Key generation algorithm)** *G is a randomized algorithm which on input 1^n produces a pair (pk, sk) (note that the sizes of both pk and sk are polynomial in n).*

- **(Signing algorithm)** *S is a randomized algorithm which takes as input a message m from the message space $\mathcal{M}$, a secret key sk and randomness $r \in \{0,1\}^n$, and outputs a signature $\sigma = S(m, sk, r)$.*

- **(Verification algorithm)** *V is a deterministic algorithm such that $V(m, pk, \sigma) = 1$ for every $\sigma = S(m, sk, r)$.*

We will require signature schemes with some special properties which we now define, first fixing some notation. Let (G, S, V) be a signature scheme. For a message space $\mathcal{M}$ and pair (pk, sk) of public and secret keys, we define the set $\mathcal{R}_{1,sk}$ of "valid" signed messages as the set of all possible signed messages $(m, \sigma = S(m, sk, r))$ as m ranges over all of $\mathcal{M}$ and r ranges over all of $\{0, 1\}^n$. Similarly, we define the set $\mathcal{R}_{2,pk}$ of "potential" signed messages as $\mathcal{R}_{2,pk} = \{(m, \sigma) : V(m, pk, \sigma) = 1\}$. Likewise, we define the set of valid signatures for message m, denoted $\mathcal{R}_{1,sk}(m)$, as the set of all possible pairs $(m, \sigma = S(m, sk, r))$ as r ranges over all of $\{0, 1\}^n$, and we define the set of potential signatures for message m as $\mathcal{R}_{2,pk}(m) = \{(m, \sigma) : V(m, pk, \sigma) = 1\}$.

Definition 56. *Let (G, S, V) be a signature scheme and $\mathcal{M}$ be a message space. A pair (pk, sk) of public and secret keys is said to be* (δ, η)-special *if the following properties hold :*

- *Let $\mathcal{R}_{1,sk}$ be the set of valid signed messages and $\mathcal{R}_{2,pk}$ be the set of potential signed messages. Then $\frac{|\mathcal{R}_{1,sk}|}{|\mathcal{R}_{2,pk}|} \geq 1 - \eta$.*
- *For any fixed pair $(m, \sigma) \in \mathbb{R}_{1,sk}(m)$, we have $\mathbf{Pr}_{r \in \{0,1\}^n}[\sigma = S(m, sk, r)] = \frac{1}{|\mathcal{R}_{1,sk}(m)|}$.*
- *Define two distributions D and D' over pairs (m, σ) as follows : D is obtained by choosing $m \in_U \mathcal{M}$ and choosing $\sigma \in_U \mathcal{R}_{1,sk}(m)$. D' is the distribution defined to be uniform over the set $\mathcal{R}_{1,sk}$. Then $d_{TV}(D, D') \leq \delta$.*

From now on, in the interest of brevity, $\mathcal{M}$ will denote the "obvious" message space $\mathcal{M}$ associated with a signature scheme unless mentioned otherwise. Similarly, the randomness r for the signing algorithm S will always assumed to be $r \in_U \{0, 1\}^n$.

We next recall the standard notion of existential unforgeability under RMA (Random Message Attack):

Definition 57. *A signature scheme (G, S, V) is said to be* (t, ϵ)-RMA secure *if the following holds: Let $(pk, sk) \leftarrow G(1^n)$. Let $(m_1, \ldots, m_t)$ be chosen uniformly at random from $\mathcal{M}$. Let $\sigma_i \leftarrow S(m_i, sk, r)$. Then, for any probabilistic algorithm A running in time t,*

$$\mathop{\mathbf{Pr}}_{(pk,sk),(m_1,\ldots,m_t),(\sigma_1,\ldots,\sigma_t)}[A(pk, m_1, \ldots, m_t, \sigma_1, \ldots, \sigma_t) = (m', \sigma')] \leq \epsilon$$

where $V(m', pk, \sigma') = 1$ and $m' \neq m_i$ for all $i = 1, \ldots, t$.

Next we need to formally define the notion of hardness of inverse approximate uniform generation:

Definition 58. *Let $\mathcal{C}$ be a class of n-variable Boolean functions. $\mathcal{C}$ is said to be* $(t(n), \epsilon, \delta)$-hard for inverse approximate uniform generation *if there is no algorithm A running in time $t(n)$ which is an (ϵ, δ)-inverse approximate uniform generation algorithm for $\mathcal{C}$.*

Finally, we will also need the definition of an invertible Levin reduction:

Definition 59. *A binary relation R is said to reduce to another binary relation R' by a* time-t invertible Levin reduction *if there are three algorithms α, β and γ, each running in time $t(n)$ on instances of length n, with the following property:*

- *For every $(x, y) \in R$, it holds that $(\alpha(x), \beta(x, y)) \in R'$;*
- *For every $(\alpha(x), z) \in R'$, it holds that $(x, \gamma(\alpha(x), z)) \in R$.*

Furthermore, the functions β and γ are injective maps with the property that $\gamma(\alpha(x), \beta(x, y)) = y$.

Note that for any class of functions $\mathcal{C}$, we can define the binary relation $R_{\mathcal{C}}$ as follows : $(f, x) \in R_{\mathcal{C}}$ if and only if $f(x) = 1$ and $f \in \mathcal{C}$. In this section, whenever we say that there is an invertible Levin reduction from class $\mathcal{C}_1$ to class $\mathcal{C}_2$, we mean that there is an invertible Levin reduction between the corresponding binary relations $R_{\mathcal{C}_1}$ and $R_{\mathcal{C}_2}$.

6.1.1 A general hardness result based on signature schemes.

We now state and prove our main theorem relating signature schemes to hardness of inverse approximate uniform generation:

Theorem 60. *Let (G, S, V) be a (t, ϵ)-RMA secure signature scheme. Suppose that with probability at least 99/100 a random pair $(pk, sk) \leftarrow G(1^n)$ is (δ, η)-special. Let $\mathcal{C}$ be a class of n-variable Boolean functions such that there is a Levin reduction from CIRCUIT-SAT to $\mathcal{C}$ running in time $t'(n)$. Let κ_1 and κ_2 be such that $\kappa_1 \leq 1 - 2 \cdot (2\eta + \delta + t'(n)/|\mathcal{M}|)$, $\kappa_2 \leq 1 - 2t'(n) \cdot (\eta + \delta)$ and $\epsilon \leq (1 - \kappa_1)(1 - \kappa_2)/4$. If $t_1(\cdot)$ is a time function such that $2t_1(t'(n)) \leq t(n)$, then $\mathcal{C}$ is $(t_1(n), \kappa_1, \kappa_2)$-hard for inverse approximate uniform generation.*

The high-level idea of the proof is simple: Suppose there were an efficient algorithm for the inverse approximate uniform generation problem for $\mathcal{C}$. Because of the invertible Levin reduction from CIRCUIT-SAT to $\mathcal{C}$, there is a signature scheme for which the verification algorithm (using any given public key) corresponds to a function in $\mathcal{C}$. The signed messages $(m_1, \sigma_1), \ldots, (m_t, \sigma_t)$ correspond to points from $\mathcal{U}_{f^{-1}(1)}$ where $f \in \mathcal{C}$. Now the existence of an efficient algorithm for the inverse approximate uniform generation problem for $\mathcal{C}$ (i.e. an algorithm which, given points from $\mathcal{U}_{f^{-1}(1)}$, can generate more such points) translates into an algorithm which, given a sample of signed messages, can generate a new signed message. But this violates the existential unforgeability under RMA of the signature scheme.

We now proceed to the formal proof.

Proof. Assume towards a contradiction that there is an algorithm A for inverse approximate uniform generation A_{inv} which runs in time t_1 such that with probability $1 - \kappa_2$, the output distribution is κ_1-close to the target distribution. If we can show that for any (δ, η)-special key pair (pk, sk) the resulting signature scheme is not (t, ϵ) secure, then this will result in a contradiction. We will now use algorithm A to construct an adversary which breaks the signature scheme for (δ, η)-special key pairs (pk, sk).

Towards this, fix a (δ, η)-special key pair (pk, sk) and consider the function $V_{pk} : \mathcal{M} \times \{0,1\}^* \rightarrow \{0,1\}$ defined as $V_{pk}(m, \sigma) = V(m, pk, \sigma)$. Clearly, V_{pk} is an instance of CIRCUIT-SAT (i.e. V_{pk} is computed by a satisfiable polynomial-size Boolean circuit). Since there is an invertible Levin reduction from CIRCUIT-SAT to $\mathcal{C}$, given pk, the adversary in time $t'(n)$ can compute $\Phi_{pk} \in \mathcal{C}$ with the following properties (let β and γ be the corresponding algorithms in the definition of the Levin reduction):

- For every (m, σ) such that $V_{pk}(m, \sigma) = 1$, $\Phi_{pk}(\beta(V_{pk}, (m, \sigma))) = 1$.
- For every x such that $\Phi_{pk}(x) = 1$, $V_{pk}(\gamma(\Phi_{pk}, x)) = 1$.

Recall that the adversary receives signatures $(m_1, \sigma_1), \ldots, (m_{t'(n)}, \sigma_{t'(n)})$. Let $x_i = \beta(V_{pk}, (m_i, \sigma_i))$. Let D_x be the distribution of $(x_1, \ldots, x_{t'(n)})$. We next make the following claim.

Claim 61. *Let $y_1, \ldots, y_{t'}$ be drawn uniformly at random from $\Phi_{pk}^{-1}(1)$ and let D_y be the corresponding distribution of $(y_1, \ldots, y_t)$. Then, D_y and D_x are $t'(n) \cdot (2\eta + \delta)$-close in statistical distance.*

Proof. Note that D_y and D_x are $t'(n)$-way product distributions. If $D_x^{(1)}$ and $D_y^{(1)}$ are the corresponding marginals on the first coordinate, then $t'(n) \cdot d_{TV}(D_x^{(1)}, D_y^{(1)}) \leq d_{TV}(D_x, D_y)$. Thus, it suffices to upper bound $d_{TV}(D_x^{(1)}, D_y^{(1)})$, which we now do.

$$d_{TV}(D_x^{(1)}, D_y^{(1)}) \leq \sum_{z \in supp(D_y^{(1)}) \setminus supp(D_x^{(1)})} \left| D_x^{(1)}(z) - D_y^{(1)}(z) \right| + \sum_{z \in supp(D_x^{(1)})} \left| D_x^{(1)}(z) - D_y^{(1)}(z) \right| .$$

By definition of (pk, sk) being (δ, η)-special, we get that

$$\sum_{z \in supp(D_y^{(1)}) \setminus supp(D_x^{(1)})} |D_x^{(1)}(z) - D_y^{(1)}(z)| \leq \eta.$$

To bound the next sum, let $\tau = \mathbf{Pr}[D_y^{(1)} \in supp(D_x^{(1)})]$. Note that $\tau \geq 1 - \eta$. We have

$$\begin{aligned} \sum_{z \in supp(D_x^{(1)})} \left| D_x^{(1)}(z) - D_y^{(1)}(z) \right| &\leq \sum_{z \in supp(D_x^{(1)})} \left| \tau D_x^{(1)}(z) - D_y^{(1)}(z) \right| + (1-\tau) \sum_{z \in supp(D_x^{(1)})} D_x^{(1)}(z) \\ &\leq \eta + \tau \cdot \sum_{z \in supp(D_x^{(1)})} \left| D_x^{(1)}(z) - \frac{D_y^{(1)}(z)}{\tau} \right|. \end{aligned}$$

We observe that $\frac{D_y^{(1)}(z)}{\tau}$ restricted to $supp(D_x^{(1)})$ is simply the uniform distribution over the image of the set $\mathcal{R}_{1,sk}$ and hence is the same as applying the map β on the distribution D'. Likewise $D_x^{(1)}$ is the same as applying the map β on D (mentioned in Definition 56). Hence, we have that

$$d_{TV}(D_x^{(1)}, D_y^{(1)}) \leq 2\eta + d_{TV}(D, D') \leq 2\eta + \delta.$$

□

Now, observe that the instances x_i are each of length at most $t'(n)$. Since the distributions D_x and D_y are $t'(n)\cdot(2\eta + \delta)$ close, hence our adversary can run A_{inv} in time $t(n)$ on the examples $x_1, \ldots, x_{t'(n)}$ and succeed with probability $1 - \kappa_2 - t'(n)\cdot(2\eta + \delta) \geq (1 - \kappa_2)/2$ in producing a sampler whose output distribution is κ_1-close to $\mathcal{U}_{\Phi_{pk}^{-1}(1)}$. Call this output distribution Z. Let $\beta(D)$ denote the distribution obtained by applying the map β on D. The proof of Claim 61 shows that $\beta(D)$ is $(2\eta + \delta)$-close to the distribution $\mathcal{U}_{\Phi_{pk}^{-1}(1)}$. Thus, with probability $(1-\kappa_2)/2$, Z is $(\kappa_1 + (2\eta + \delta))$-close to the distribution $\beta(D)$. By definition of D, we have

$$\mathbf{Pr}_{(m,\sigma) \in D}[\forall i \in [t'], m_i \neq m] \geq 1 - \frac{t'}{|\mathcal{M}|}.$$

Thus, with probability $\frac{1-\kappa_2}{2}$,

$$\mathbf{Pr}_{z \in Z}[z = g(m, \sigma) \text{ and } \forall i \in [t'], m_i \neq m] \geq 1 - \kappa_1 - (2\eta + \delta) - \frac{t'}{|\mathcal{M}|} \geq \frac{1 - \kappa_1}{2}$$

Thus, with overall probability $(1-\kappa_1)(1-\kappa_2)/4 \geq \epsilon$, the adversary succeeds in producing $z = g(m, \sigma)$ such that $\forall i \in [t'], m_i \neq m$. Applying the map γ on (Φ_{pk}, z), the adversary gets the pair (m, σ). Also, note that the total running time of the adversary is $t_1(t'(n)) + t'(n) \leq 2t_1(t'(n)) \leq t(n)$ which contradicts the (t, ϵ)-RMA security of the signature scheme. □

6.1.2 A specific hardness assumption.

At this point, at the cost of sacrificing some generality, we consider a particular instantiation of a signature scheme from the literature which meets our requirements. While similar signature schemes can be constructed under many different cryptographic assumptions in the literature, we forsake such generality to keep the discussion from getting too cumbersome.

To state our cryptographic assumption, we need the following notation:

- PRIMES$_k$ is the set of k-bit prime numbers.

- RSA_k is the set of all products of two primes of length $\lfloor (k-1)/2 \rfloor$.

The following cryptographic assumption (a slight variant of the standard RSA assumption) appears in [MRV99].

Assumption 1. ***The RSA$'$ $s(k)$ assumption**: Fix any $m \in RSA_k$ and let $x \in_U \mathbb{Z}_m^*$ and $p \in_U PRIMES_{k+1}$. Let A be any probabilistic algorithm running in time $s(k)$. Then,*

$$\mathbf{Pr}_{(x,p)}[A(m,x,p) = y \text{ and } y^p = x \ (mod\, m)] \leq \frac{1}{s(k)}.$$

As mentioned in [MRV99], given the present state of computational number theory, it is plausible to conjecture the RSA$'$ $s(k)$ assumption for $s(k) = 2^{k^\delta}$ for some absolute constant $\delta > 0$. For the sake of conciseness, for the rest of this section we write “Assumption 1 holds true” to mean that Assumption 1 holds true with $s(k) = 2^{n^\delta}$ for some fixed constant $\delta > 0$. (We note, though, that all our hardness results go through giving superpolynomial hardness using only $s(k) = k^{\omega(1)}$.)

Micali *et al.* [MRV99] give a construction of a “unique signature scheme” using Assumption 1:

Theorem 62. *If Assumption 1 holds true, then there is a $(t = 2^{n^\delta}, \epsilon = 1/t)$-RMA secure signature scheme (G, S, V) with the following property : For any message $m \in \mathcal{M}$, there do not exist $\sigma_1 \neq \sigma_2$ such that $V(m, \sigma_1) = V(m, \sigma_2) = 1$. In this scheme the signing algorithm S is deterministic and the message space $\mathcal{M}$ is of size 2^{n^δ}.*

The above theorem says that under the RSA$'$ $s(k)$ assumption, there is a deterministic signature scheme such that there is only one signature σ_m for every message m, and for every message m the only accepting input for V is (m, σ_m). As a consequence, the signature scheme in Theorem 62 has the property that every (pk, sk) pair that can be generated by G is $(0, 0)$-special.

Remark 63. It is important to note here that constructions of $(0, 0)$ special signature schemes are abundant in the literature. A partial list follows : Lysyanskaya [Lys02] constructed a deterministic $(0, 0)$ special signature scheme using a strong version of the Diffie–Hellman assumption. Hohenberger and Waters [HW10] constructed a scheme with a similar guarantee using a variant of the Diffie–Hellman assumption on bilinear groups. In fact, going back much further, Cramer and Shoup [CS00, Fis03] show that using the Strong RSA assumption, one can get a $(0, 0)$ special signature scheme (which however is not deterministic). We remark that the scheme as stated in [CS00] is not $(0, 0)$ special in any obvious sense, but the more efficient version in [Fis03] can be easily verified to be $(0, 0)$ special. Throughout this section, for the sake of simplicity, we use the signature scheme in Theorem 62.

Instantiating Theorem 60 with the signature scheme from Theorem 62, we obtain the following corollary:

Corollary 64. *Suppose that Assumption 1 holds true. Then the following holds : Let $\mathcal{C}$ be a function class such that there is a polynomial time (n^k-time) invertible Levin reduction from* CIRCUIT-SAT *to $\mathcal{C}$. Then $\mathcal{C}$ is $(2^{n^c}, 1 - 2^{-n^c}, 1 - 2^{-n^c})$-hard for inverse approximate uniform generation for some constant $c > 0$ (depending only on the “δ” in Assumption 1 and on k).*

6.1.3 Inverse approximate uniform generation hardness results for specific function classes whose satisfiability problem is NP-complete. In this subsection we use Corollary 64 to prove hardness results for inverse approximate uniform generation for specific function classes $\mathcal{C}$ for which there are invertible Levin reductions from CIRCUIT-SAT to $\mathcal{C}$.

Recall that a 3-CNF formula is a conjunction of clauses (disjunctions) of length 3. The following fact can be easily verified by inspecting the standard reduction from CIRCUIT-SAT to 3-CNF-SAT.

Fact 65. *There is a polynomial time invertible Levin reduction from CIRCUIT-SAT to 3-CNF-SAT.*

As a corollary, we have the following result.

Corollary 66. *If Assumption 1 holds true, then there exists an absolute constant $c > 0$ such that the class 3-CNF is $(2^{n^c}, 1 - 2^{-n^c}, 1 - 2^{-n^c})$-hard for inverse approximate uniform generation.*

Corollary 66 is interesting in light of the well known fact that the class of all 3-CNF formulas is efficiently PAC learnable from uniform random examples (in fact under any distribution).

We next observe that the problem of inverse approximate uniform generation remains hard even for 3-CNF formulas in which each variable occurs a bounded number of times. To prove this we will use the fact that polynomial time invertible Levin reductions compose:

Fact 67. *If there is a polynomial time invertible Levin reduction from CIRCUIT-SAT to $\mathcal{C}$ and a polynomial time Levin reduction from $\mathcal{C}$ to $\mathcal{C}_1$, then there is a polynomial time invertible Levin reduction from CIRCUIT-SAT to $\mathcal{C}_1$.*

The following theorem says that the inverse approximate uniform generation problem remains hard for the class of all 3-CNF formulas in which each variable occurs at most 4 times (hereafter denoted 3,4-CNF).

Theorem 68. *If Assumption 1 holds true, then there exists an absolute constant $c > 0$ such that 3,4-CNF-SAT is $(2^{n^c}, 1 - 2^{-n^c}, 1 - 2^{-n^c})$-hard for inverse approximate uniform generation.*

Proof. Tovey [Tov84] shows that there is a polynomial time invertible Levin reduction from 3-CNF-SAT to 3,4-CNF-SAT. Using Fact 67, we have a polynomial time Levin reduction from CIRCUIT-SAT to 3,4-CNF-SAT. Now the result follows from Corollary 64 □

The next theorem shows that the class of all intersections of two halfspaces over n Boolean variables is hard for inverse approximate uniform generation.

Theorem 69. *If Assumption 1 holds true, then there exists an absolute constant $c > 0$ such that $\mathcal{C} = \{$all intersections of two halfspaces over n Boolean variables$\}$ is $(2^{n^c}, 1 - 2^{-n^c}, 1 - 2^{-n^c})$-hard for inverse approximate uniform generation.*

Proof. We recall that the SUBSET-SUM problem is defined as follows : An instance Φ is defined by positive integers $w_1, \ldots, w_n, s > 0$. A satisfying assignment for this instance is given by $x \in \{0,1\}^n$ such that $\sum_{i=1}^n w_i x_i = s$. It is well known that the SUBSET-SUM problem is NP-complete and it is folklore that there is a invertible Levin reduction from 3-SAT to SUBSET-SUM. However, since it is somewhat difficult to find this reduction explicitly in the literature, we outline such a reduction.

To describe the reduction, we first define 1-in-3-SAT. An instance Ψ of 1-in-3-SAT is defined over Boolean variables $x_1, \ldots, x_n$ with the following constraints : The i^{th} constraint is defined by a subset of at most three literals over $x_1, \ldots, x_n$. An assignment to $x_1, \ldots, x_n$ satisfies Ψ if and only if for every constraint there is exactly one literal which is set to true. Schaefer [Sch78] showed that 3-SAT reduces to 1-in-3-SAT in polynomial time, and the reduction can be easily verified to be an invertible Levin reduction. Now the standard textbook reduction from 3-SAT to SUBSET-SUM (which can be found e.g. in [Pap94]) applied to instances of 1-in-3-SAT, can be easily seen to be a polynomial time invertible Levin reduction. By Fact 67, we thus have a polynomial time invertible Levin reduction from 3-CNF-SAT to SUBSET-SUM.

With this reduction in hand, it remains only to observe that that any instance of SUBSET-SUM is also an instance of "intersection of two halfspaces," simply because $\sum_{i=1}^n w_i x_i = s$ if and only if $s \leq \sum_{i=1}^n w_i \cdot x_i \leq s$. Thus, there is a polynomial time invertible Levin reduction from 3-CNF-SAT to the class of all intersections of two halfspaces. This finishes the proof. □

6.1.4 A hardness result where the satisfiability problem is in P. So far all of our hardness results have been for classes $\mathcal{C}$ of NP-complete languages. As Theorem 60 requires a reduction from CIRCUIT-SAT to $\mathcal{C}$, this theorem cannot be directly used to prove hardness for classes $\mathcal{C}$ which are not NP-hard. We next give an extension of Theorem 60 which can apply to classes $\mathcal{C}$ for which the satisfiability problem is in P. Using this result we will show hardness of inverse approximate uniform generation for MONOTONE-2-CNF-SAT. (Recall that a monotone 2-CNF formula is a conjunction of clauses of the form $x_i \vee x_j$, with no negations; such a formula is trivially satisfiable by the all-true assignment.)

We begin by defining by a notion of invertible one-many reductions that we will need.

Definition 70. *CIRCUIT-SAT is said to have an η-*almost invertible one-many reduction to a function class *$\mathcal{C}$ if the following conditions hold:*

- *There is a polynomial time computable function f such that given an instance Φ of CIRCUIT-SAT (i.e. Φ is a satisfiable circuit), $\Psi = f(\Phi)$ is an instance of $\mathcal{C}$ (i.e. $\Psi \in \mathcal{C}$ and Ψ is satisfiable).*
- *Fix any instance Φ of CIRCUIT-SAT and let $\mathcal{A} = \Psi^{-1}(1)$ denote the set of satisfying assignments of Ψ. Then $\mathcal{A}$ can be partitioned into sets $\mathcal{A}_1$ and $\mathcal{A}_2$ such that $|\mathcal{A}_2|/|\mathcal{A}| \leq \eta$ and there is an efficiently computable function $g : \mathcal{A}_1 \to \Phi^{-1}(1)$ such that $g(x)$ is a satisfying assignment of Φ for every $x \in \mathcal{A}_1$.*
- *For every y which is a satisfying assignment of Φ, the number of pre-images of y under g is exactly the same, and the uniform distribution over $g^{-1}(y)$ is polynomial time samplable.*

We next state the following simple claim which will be helpful later.

Claim 71. *Suppose there is an η-almost invertible one-many reduction from CIRCUIT-SAT to $\mathcal{C}$. Let f and g be the functions from Definition 70. Let Φ be an instance of CIRCUIT-SAT and let $\Psi = f(\Phi)$ be the corresponding instance of $\mathcal{C}$. Define distributions D_1 and D_2 as follows :*

- *A draw from D_1 is obtained by choosing y uniformly at random from $\Phi^{-1}(1)$ and then outputting z uniformly at random from $g^{-1}(y)$.*
- *A draw from D_2 is obtained by choosing z' uniformly at random from $\Psi^{-1}(1)$.*

Then we have $d_{TV}(D_1, D_2) \leq \eta$.

Proof. This is an immediate consequence of the fact that D_1 is uniform over the set $\mathcal{A}_1$ while D_2 is uniform over the set $\mathcal{A}$ (from Definition 70). □

We next have the following extension of Corollary 64.

Theorem 72. *Suppose that Assumption 1 holds true. Then if $\mathcal{C}$ is a function class such that there is an η-almost invertible one-many reduction (for $\eta = 2^{-\Omega(n)}$) from CIRCUIT-SAT to $\mathcal{C}$, then $\mathcal{C}$ is $(2^{n^c}, 1 - 2^{-n^c}, 1 - 2^{-n^c})$-hard for inverse approximate uniform generation for some absolute constant $c > 0$.*

Proof. The proof is similar to the proof of Corollary 64. Assume towards a contradiction that there is an algorithm for inverse approximation uniform generation A_{inv} for $\mathcal{C}$ which runs in time t_1 such that with probability $1 - \kappa_2$, the output distribution is κ_1-close to the target distribution. (We will set t_1, κ_1 and κ_2 later to 2^{n^c}, $1 - 2^{-n^c}$ and $1 - 2^{-n^c}$ respectively.)

Let (G, S, V) be the RMA-secure signature scheme constructed in Theorem 62. Note that (G, S, V) is a (T, ϵ)-RMA secure signature scheme where $T = 2^{n^\delta}$, $\epsilon = 1/T$ and $|\mathcal{M}| = 2^{n^\mu}$ for constant $\delta, \mu > 0$. Let (pk, sk) be a choice of key pair. We will us A_{inv} to contradict the security of (G, S, V). Towards this, consider the function $V_{pk} : \mathcal{M} \times \{0,1\}^* \to \{0,1\}$ defined as $V_{pk}(m,\sigma) = V(m,pk,\sigma)$. Clearly, V_{pk} is an instance of CIRCUIT-SAT. Consider the η-invertible one-many reduction from CIRCUIT-SAT to $\mathcal{C}$. Let α and β have the same meaning as in Definition 70. Let $\Psi = \alpha(V_{pk})$ and let $\mathcal{A}$, $\mathcal{A}_1$ and $\mathcal{A}_2$ have the same meaning as in Definition 70. The adversary receives message-signature pairs $(m_1,\sigma_1) \ldots (m_{t_1},\sigma_{t_1})$ where $m_1,\ldots,m_{t_1}$ are chosen independently at random from $\mathcal{M}$. For any i, (m_i,σ_i) is a satisfying assignment of V_{pk}. By definition, in time $t_2 = t_1 \cdot \mathrm{poly}(n)$, the adversary can sample $(z_1,\ldots,z_{t_1})$ such that $z_1,\ldots,z_{t_1}$ are independent and $z_i \sim \mathcal{U}_{\beta^{-1}(m_i,\sigma_i)}$. Note that this means that each z_i is an independent sample from $\mathcal{A}_1$ and $|z_i| = \mathrm{poly}(n)$. Note that $(z_1,\ldots,z_{t_1})$ is a t_1-fold product distribution such that if D' denotes the distribution of z_i, then by Claim 71, $d_{TV}(D',\mathcal{U}_{\Psi^{-1}(1)}) \le \eta$. Hence, if D is the distribution of $(z_1,\ldots,z_{t_1})$, then $d_{TV}(D,\mathcal{U}^t_{\Psi^{-1}(1)}) \le t_1\eta$.

Hence, the adversary can now run A_{rec} on the samples $z_1,\ldots,z_{t_1}$ and as long as $1-\kappa_2-t_1\eta \ge (1-\kappa_2)/2$, succeeds in producing a sampler with probability $(1-\kappa_2)/2$ whose output distribution (call it Z) is κ_1 close to the distribution $\mathcal{U}_{\Psi^{-1}(1)}$. Note that as $\eta = 2^{-\Omega(n)}$, for any $c > 0$, $t_1 = 2^{n^c}$ and $\kappa_2 = 1-2^{-n^c}$ satisfies this condition. Hence, we get that $d_{TV}(Z,D') \le \kappa_1 + \eta$. Now, observe that

$$\mathbf{Pr}_{\rho\in D'}[\beta(\rho) = (m,\sigma) \text{ and } m \neq m_i] = 1 - \frac{t_1}{|\mathcal{M}|}.$$

The above uses the fact that every element in the range of β has the same number of pre-images. This of course implies that

$$\mathbf{Pr}_{\rho\in Z}[\beta(\rho) = (m,\sigma) \text{ and } m \neq m_i] \ge 1 - \frac{t_1}{|\mathcal{M}|} - (\kappa_1+\eta).$$

Again as long as $\kappa_1 \le 1-2(\eta + t_1/|\mathcal{M}|)$, the adversary succeeds in getting a valid message signature pair (m,σ) with $m \neq m_i$ for any $1 \le i \le t_1$ with probability $(1-\kappa_1)/2$. Again, we can ensure $\kappa_1 \le 1-2(\eta+t_1/|\mathcal{M}|)$ by choosing c sufficiently small compared to μ. The total probability of success is $(1-\kappa_1)(1-\kappa_2)/4$ and the total running time is $t_1(\mathrm{poly}(n)) + \mathrm{poly}(n)$. Again if c is sufficiently small compared to μ and δ, then the total running time is at most $t_1(\mathrm{poly}(n)) + \mathrm{poly}(n) < T$ and the success probability is at least $(1-\kappa_1)(1-\kappa_2)/4 > \epsilon$, resulting in a contradiction. □

We now demonstrate a polynomial time η-invertible one-many reduction from CIRCUIT-SAT to MONOTONE-2-CNF-SAT for $\eta = 2^{-\Omega(n)}$. The reduction uses the "blow-up" idea used to prove hardness of approximate counting for MONOTONE-2-CNF-SAT in [JVV86]. We will closely follow the instantiation of this technique in [Wat12].

Lemma 73. *There is a polynomial time η-almost invertible one-many reduction from CIRCUIT-SAT to MONOTONE-2-CNF-SAT where $\eta = 2^{-\Omega(n)}$.*

Proof. We begin by noting the following simple fact.

Fact 74. *If there is a polynomial time invertible Levin reduction from CIRCUIT-SAT to a class $\mathcal{C}_1$ and an η-almost invertible one-many reduction from $\mathcal{C}_1$ to $\mathcal{C}_2$, then there is a polynomial time η-almost invertible one-many reduction from CIRCUIT-SAT to $\mathcal{C}_2$.*

Since there is an invertible Levin reduction from CIRCUIT-SAT to 3-CNF-SAT, by virtue of Fact 74, it suffices to demonstrate a polynomial time η-almost invertible one-many reduction from 3-CNF-SAT to MONOTONE-2-CNF-SAT. To do this, we first construct an instance of VERTEX-COVER from the 3-CNF-SAT instance. Let $\Phi = \bigwedge_{i=1}^m \Phi_i$ be the instance of 3-CNF-SAT. Construct an instance of VERTEX-COVER by introducing seven vertices for each clause Φ_i (one corresponding to every satisfying assignment of Φ_i). Now, put an edge between any two vertices of this graph if the corresponding assignments to the variables of Φ conflict on some variable. We call this graph G. We observe the following properties of this graph :

- G has exactly $7m$ vertices.
- Every vertex cover of G has size at least $6m$.
- There is an efficiently computable and invertible injection ℓ between the satisfying assignments of Φ and the vertex covers of G of size $6m$. To get the vertex cover corresponding to a satisfying assignment, for every clause Φ_i, include the six vertices in the vertex cover which conflict with the satisfying assignment.

We next do the blow-up construction. We create a new graph G' by replacing every vertex of G with a cloud of $10m$ vertices, and for every edge in G we create a complete bipartite graph between the corresponding clouds in G'. Clearly, the size of the graph G' is polynomial in the size of the 3-CNF-SAT formula. We define a map g_1 between vertex covers of G' and vertex covers of G as follows : Let S' be a vertex cover of G'. We define the set $S = g_1(S')$ in the following way. For every vertex v in the graph G, if all the vertices in the corresponding cloud in G' are in S', then include $v \in S$, else do not include v in S. It is easy to observe that g_1 maps vertex covers of G' to vertex covers of G. It is also easy to observe that a vertex cover of G of size s has $(2^{10m}-1)^{7m-s}$ pre-images under g_1.

Now, observe that we can construct a MONOTONE-2-CNF-SAT formula Ψ which has a variable corresponding to every vertex in G' and every subset S' of G' corresponds to a truth assignment $y_{S'}$ to Ψ such that $\Psi(y_{S'}) = 1$ if and only if S' is a vertex cover of G'. Because of this correspondence, we can construct a map g_1' which maps satisfying assignments of Ψ to vertex covers of G. Further, a vertex cover of size s in graph G has $(2^{10m}-1)^{7m-s}$ pre-images under g_1'. Since the total number of vertex covers of G of size s is at most $\binom{7m}{s}$, the total number of satisfying assignments of Ψ which map to vertex covers of G of size more than $6m$ can be bounded by :

$$\sum_{s=6m+1}^{7m} \binom{7m}{s} \cdot (2^{10m}-1)^{7m-s} \leq m \cdot \binom{7m}{6m+1} \cdot (2^{10m}-1)^{m-1} \leq (2^{10m}-1)^m \cdot \frac{2^{7m}}{2^{10m}-1}$$

On the other hand, since Φ has at least one satisfying assignment, hence G has at least one vertex cover of size $6m$ and hence the total number of satisfying assignments of Ψ which map to vertex covers of G of size $6m$ is at least $(2^{10m}-1)^m$. Thus, if we let $\mathcal{A}$ denote the set of satisfying assignments of Ψ and $\mathcal{A}_1$ be the set of satisfying assignment of Ψ which map to vertex covers of G of size exactly $6m$ (under g_1), then $|\mathcal{A}_1|/|\mathcal{A}| \geq 1 - 2^{-\Omega(n)}$. Next, notice that we can define the map g mapping $\mathcal{A}_1$ to the satisfying assignments of Φ in the following manner : $g(x) = \ell^{-1}(g_1(x))$. It is easy to see that this map satisfies all the requirements of the map g from Definition 70 which concludes the proof. □

Combining Lemma 73 with Theorem 72, we have the following corollary.

Corollary 75. *If Assumption 1 holds true, then MONOTONE-2-CNF-SAT is* $(2^{n^c}, 1-2^{-n^c}, 1-2^{-n^c})$ *hard for inverse approximate uniform generation for some absolute constant* $c > 0$.

As a consequence of the above result, we also get hardness for inverse approximate uniform generation of degree-2 *polynomial threshold functions (PTFs)*; these are functions of the form $\mathrm{sign}(q(x))$ where $q(x)$ is a degree-2 multilinear polynomial over $\{0,1\}^n$.

Corollary 76. *If Assumption 1 holds true, then the class of all* n*-variable degree-2 polynomial threshold functions is* $(2^{n^c}, 1-2^{-n^c}, 1-2^{-n^c})$ *hard for inverse approximate uniform generation for some absolute constant* $c > 0$.

Proof. This follows immediately from the fact that every monotone 2-CNF formula can be expressed as a degree-2 PTF. To see this, note that if $\Phi = \bigwedge_{i=1}^m (x_{i1} \vee x_{i2})$ where each x_{ij} is a 0/1 variable, then $\Phi(x)$ is true if and only if $\sum_{i=1}^m x_{i1} + x_{i2} - x_{i1} \cdot x_{i2} \geq m$. This finishes the proof. □

6.2 Hardness results based on Message Authentication Codes. All of the previous hardness results intuitively correspond to the case when the second step of our "standard approach" is algorithmically hard. Indeed, consider a class $\mathcal{C}$ of functions that has an efficient approximate uniform generation algorithm. Unless $P \neq NP$ there cannot be any Karp reduction from CIRCUIT-SAT to $\mathcal{C}$ (this would contradict the NP-completeness of CIRCUIT-SAT) and hence Theorem 60 is not applicable in this setting. In fact, even for $\eta = 1 - 1/\text{poly}(n)$ there cannot be any η-almost invertible one-many reduction from CIRCUIT-SAT to $\mathcal{C}$ unless $P \neq NP$. This makes Theorem 72 inapplicable in this setting. Thus, to prove hardness results for classes that have efficient approximate uniform generation algorithms, we need some other approach.

In this section we show that Message Authentication Codes (MAC) can be used to establish hardness of inverse approximate uniform generation for such classes. We begin by defining MACs. (We remark that we use a restricted definition which is sufficient for us; for the most general definition, see [Gol04].)

Definition 77. *A* Message Authentication Code (MAC) *is a triple* (G, T, V) *of polynomial-time algorithms with the following properties :*

- **(Key generation algorithm)** $G(\cdot)$ *is a randomized algorithm which on input* 1^n *produces a secret key* sk*;*
- **(Tagging algorithm)** T *is a randomized algorithm which takes as input message* m*, secret key* sk *and randomness* r *and outputs* $\sigma \leftarrow T(m, sk, r)$*;*
- **(Verification algorithm)** V *is a deterministic algorithm which takes as input message* m*, secret key* sk *and* σ*. If* $\sigma = T(m, sk, r)$ *for some* r *then* $V(m, sk, \sigma) = 1$*.*

For the purposes of our hardness results we require MACs with some special properties. While our hardness results can be derived from slightly more general MACs than those we specify below, we forsake some generality for the sake of clarity. For a MAC (G, T, V) and a choice of secret key sk, we say σ is a *valid tag* for message m if there exists r such that $\sigma = T(m, sk, r)$. Likewise, we say that σ is a *potential tag* for message m if $V(m, sk, \sigma) = 1$.

Definition 78. *A Message Authentication Code* (G, T, V) *over a message space* $\mathcal{M}$ *is said to be* special *if the following conditions hold : For any secret key* sk*,*

- *For every message* $m \in \mathcal{M}$*, the set of valid tags is identical to the set of potential tags..*
- *For every two messages* $m_1 \neq m_2$ *and every* σ_1, σ_2 *such that* σ_i *is a valid tag for* m_i*, we have* $\mathbf{Pr}_r[T(m_1, sk, r) = \sigma_1] = \mathbf{Pr}_r[T(m_2, sk, r) = \sigma_2]$*.. In particular, the cardinality of the set of valid tags for* m *is the same for all* m*.*

We next define the standard notion of security under Random Message attacks for MACs. As before, from now onwards, we will assume implicitly that $\mathcal{M}$ is the message space.

Definition 79. *A special MAC* (G, T, V) *is said to be* (t, ϵ)-RMA secure *if the following holds : Let* $sk \leftarrow G(1^n)$*. Let* $(m_1, \ldots, m_t)$ *be chosen uniformly at random from* $\mathcal{M}$*. Let* $\sigma_i \leftarrow T(m_i, sk, r)$*. Then for any probabilistic algorithm* A *running in time* t*,*

$$\mathop{\mathbf{Pr}}_{sk,(m_1,\ldots,m_t),(\sigma_1,\ldots,\sigma_t)}[A(m_1, \ldots, m_t, \sigma_1, \ldots, \sigma_t) = (m', \sigma')] \leq \epsilon$$

where $V(m', sk, \sigma') = 1$ *and* $m' \neq m_i$ *for all* $i = 1, \ldots, t$*.*

It is known how to construct MACs meeting the requirements in Definition 79 under standard cryptographic assumptions (see [Gol04]).

6.2.1 A general hardness result based on Message Authentication Codes.

The next theorem shows that special MACs yield hardness results for inverse approximate uniform generation.

Theorem 80. *Let $c > 0$ and (G, T, V) be a (t, ϵ)-RMA secure special MAC for some $t = 2^{n^c}$ and $\epsilon = 1/t$ with a message space $\mathcal{M}$ of size $2^{\Omega(n)}$. Let V_{sk} denote the function $V_{sk} : (m, \sigma) \mapsto V(m, sk, \sigma)$. If $V_{sk} \in \mathcal{C}$ for every s_k, then there exists $\delta > 0$ such that $\mathcal{C}$ is (t_1, κ, η)-hard for inverse approximate uniform generation for $t_1 = 2^{n^\delta}$ and $\kappa = \eta = 1 - 2^{-n^\delta}$.*

Proof. Towards a contradiction, let us assume that there is an algorithm A_{inv} for inverse approximate uniform generation of $\mathcal{C}$ which runs in time t_1 and with probability $1 - \eta$ outputs a sampler whose statistical distance is at most κ from the target distribution. (We will set t_1, κ and η later in the proof.) We will use A_{inv} to contradict the security of the MAC. Let sk be a secret key chosen according to $G(1^n)$. Now, the adversary receives message-tag pairs $(m_1, \sigma_1), \ldots, (m_{t_1}, \sigma_{t_1})$ where $m_1, \ldots, m_{t_1}$ are chosen independently at random from $\mathcal{M}$. Because the MAC is special, for each i we have that σ_i is a uniformly random valid tag for the message m_i. Hence each (m_i, σ_i) is an independent and uniformly random satisfying assignment of V_{sk}.

We can thus run A_{inv} on the samples $(m_1, \sigma_1), \ldots, (m_{t_1}, \sigma_{t_1})$ with its accuracy parameter set to κ and its confidence parameter set to $1 - \eta$. Taking $\kappa = \eta = 1 - 2^{-n^\delta}$, we can choose δ small enough compared to c, and with $t_1 = 2^{n^\delta}$ we get that the total running time of A_{inv} is at most $2^{n^c}/2$. By the definition of inverse approximate uniform generation, with probability at least $1 - \eta = 2^{-n^\delta}$ the algorithm A_{inv} outputs a sampler for a distribution Z that is $\kappa = (1 - 2^{-n^\delta})$-close to the uniform distribution over the satisfying assignments of V_{sk}. Now, observe that

$$\mathbf{Pr}_{(m,\sigma)\sim\mathcal{U}_{V_{sk}^{-1}(1)}}[m_i \neq m \text{ for all } i \in [t_1]] \geq 1 - \frac{t_1}{|\mathcal{M}|}.$$

Thus,

$$\mathbf{Pr}_{z\sim Z}[z = (m, \sigma) \text{ and } m_i \neq m \text{ for all } i \in [t_1]] \geq (1 - \kappa) - \frac{t_1}{|\mathcal{M}|}.$$

This means that with probability $(1 - \eta) \cdot ((1 - \kappa) - \frac{t_1}{|\mathcal{M}|})$, the adversary can output a forgery. It is clear that for a suitable choice of δ relative to c, recalling that $\kappa = \eta = 1 - 2^{-n^\delta}$, the probability of outputting a forgery is greater than 2^{-n^c}, which contradicts the security of the MAC. □

Unlike signature schemes, which permitted intricate reductions (cf. Theorem 60), in the case of MACs we get a hardness result for complexity class $\mathcal{C}$ only if V_{sk} itself belongs to $\mathcal{C}$. While special MACs are known to exist assuming the existence of one-way functions [Gol04], the constructions are rather involved and rely on constructions of pseudorandom functions (PRFs) as an intermediate step. As a result, the verification algorithm V also involves computing PRFs; this means that using these standard constructions, one can only get hardness results for a class $\mathcal{C}$ if PRFs can be computed in $\mathcal{C}$. As a result, the class $\mathcal{C}$ tends to be fairly complex, making the corresponding hardness result for inverse approximate uniform generation for $\mathcal{C}$ somewhat uninteresting.

One way to bypass this is to use construction of MACs which do not involve use of PRFs as an intermediate step. In recent years there has been significant progress in this area [KPC$^+$11, DKPW12]. While both these papers describe several MACs which do not require PRFs, the one most relevant for us is the MAC construction of [KPC$^+$11] based on the hardness of the "Learning Parity with Noise" (LPN) problem.

6.2.2 Some specific hardness assumptions, and a corresponding specific hardness result.

We first state a "decision" version of LPN. To do this, we need the following notation:

- Let Ber_τ denote the following distribution over $GF(2)$: If $x \leftarrow Ber_\tau$, then $\mathbf{Pr}[x = 1] = \tau$.

- For $x \in GF(2)^n$, we use $\Lambda(x,\tau,\cdot)$ to denote the distribution $(r, x \cdot r \oplus e)$ over $GF(2)^n \times GF(2)$ where $r \sim GF(2)^n$ and $e \sim Ber_\tau$ and $x \cdot r = \oplus_i x_i r_i \pmod 2$.

Assumption 2. *Let $\tau \in (0,1/2)$ and let $\mathcal{O}_{x,\tau}$ be an oracle which, each time it is invoked, returns an independent uniformly random sample from $\Lambda(x,\tau,\cdot)$. The LPN assumption states that for any* $\mathrm{poly}(n)$*-time algorithm $\mathcal{A}$,*

$$\left|\left[\mathbf{Pr}_{x \in GF(2)^n}[\mathcal{A}^{\mathcal{O}_{x,\tau}} = 1\right] - \left[\mathbf{Pr}_{x \in GF(2)^n}[\mathcal{A}^{\mathcal{O}_{x,1/2}} = 1\right]\right| \leq \epsilon$$

for some ϵ which is negligible in n.

LPN is a well-studied problem; despite intensive research effort, the fastest known algorithm for this problem takes time $2^{O(n/\log n)}$ [BKW03]. For our applications, we will need a variant of the above LPN assumption. To define the assumption, let $\Lambda(x,\ell,\tau,\cdot)$ denote the distribution over $(A, A \cdot x \oplus e)$ where A is uniformly random in $GF(2)^{\ell \times n}$ and e is uniformly random over the set $\{z \in GF(2)^\ell : wt(z) \leq \lceil \tau\ell \rceil\}$. The vector e is usually referred to as the *noise vector.*

Assumption 3. *Let $\tau \in (0,1/2)$, $\ell = c \cdot n$ for some $0 < c < 1/2$ and let $\mathcal{O}_{x,\ell,\tau}$ be an oracle which returns a uniformly random sample from $\Lambda(x,\ell,\tau,\cdot)$. Then the (t,ϵ)* exact LPN assumption *states that for any algorithm $\mathcal{A}$ running in time t,*

$$\left|\mathbf{Pr}_{x \in GF(2)^n}\left[\mathcal{A}^{\mathcal{O}_{x,\ell,\tau}} = 1\right] - \mathbf{Pr}_{x \in GF(2)^n}\left[\mathcal{A}^{\mathcal{O}_{x,\ell,1/2}} = 1\right]\right| \leq \epsilon$$

For the sake of brevity, we henceforth refer to this assumption by saying "the exact (n,ℓ,τ) LPN problem is (t,ϵ)-hard."

The above conjecture seems to be very closely related to Assumption 2, but it is not known whether Assumption 2 formally reduces to Assumption 3. Assumption 3 has previously been suggested in the cryptographic literature [KSS10] in the context of getting perfect completeness in LPN-based protocols. We note that Arora and Ge [AG11] have investigated the complexity of this problem and gave an algorithm which runs in time $n^{O(\ell)}$. We believe that the proximity of Assumption 3 to the well-studied Assumption 2, as well as the failure to find algorithms for Assumption 3, make it a plausible conjecture. For the rest of this section we use Assumption 3 with $t = 2^{n^\beta}$ and $\epsilon = 2^{-n^\beta}$ for some fixed $\beta > 0$.

We next define a seemingly stronger variant of Assumption 3 which we call *subset exact LPN*. This requires the following definitions: For $x, v \in GF(2)^n$, $\ell, d \leq n$ and $\tau \in (0,1/2)$, we define the distribution $\Lambda^a(x,v,\ell,\tau,\cdot)$ as follows :

$$\Lambda^a(x,v,\ell,\tau,\cdot) = \begin{cases} \Lambda(x \cdot v, \ell, 1/2, \cdot) & \text{if } wt(v) < d \\ \Lambda(x \cdot v, \ell, \tau, \cdot) & \text{if } wt(v) \geq d \end{cases}$$

where $x \cdot v \in GF(2)^n$ is defined by $(x \cdot v)_i = x_i \cdot v_i$. In other words, if $wt(v) \geq d$, then the distribution $\Lambda^a(x,v,\ell,\tau)$ projects x into the non-zero coordinates of v and then outputs samples corresponding to exact LPN for the projected vector. We define the oracle $\mathcal{O}^a_{x,\ell,d,\tau}(\cdot)$ which takes an input $v \in GF(2)^n$ and outputs a random sample from $\Lambda^a(x,v,\ell,\tau,\cdot)$. The subset exact LPN assumption states the following:

Assumption 4. *Let $\tau \in (0,1/2)$, $\ell = c \cdot n$ and $d = c' \cdot n$ for some $0 < c, c' < 1/2$. The (t,ϵ)*-subset exact LPN assumption *says that for any algorithm $\mathcal{A}$ running in time t,*

$$\left|\mathbf{Pr}_{x \in GF(2)^n}\left[\mathcal{A}^{\mathcal{O}^a_{x,\ell,d,\tau}} = 1\right] - \mathbf{Pr}_{x \in GF(2)^n}\left[\mathcal{A}^{\mathcal{O}^a_{x,\ell,d,1/2}} = 1\right]\right| \leq \epsilon.$$

For the sake of brevity, we henceforth refer to this assumption by saying "the subset exact (n,ℓ,d,τ) LPN problem is (t,ϵ)-hard."

Assumption 4 is very similar to the *subset LPN assumption* used in [KPC$^+$11] and previously considered in [Pie12]. The subset LPN assumption is the same as Assumption 4 but with $\ell = 1$ and the coordinates of the noise vector e being drawn independently from Ber_τ. Pietrzak [Pie12] showed that the subset LPN assumption is implied by the standard LPN assumption (Assumption 2) with a minor change in the security parameters. Along the same lines, the next lemma shows that Assumption 3 implies Assumption 4 with a minor change in parameters. The proof is identical to the proof of Theorem 1 in [Pie12] and hence we do not repeat it here.

Lemma 81. *If the exact (n, ℓ, τ) LPN problem is (t, ϵ) hard, then for any $g \in \mathbb{N}$, the subset exact $(n', \ell, n + g, \tau)$ LPN problem is (t', ϵ') hard for $n' \geq n + g$, $t' = t/2$ and $\epsilon' = \epsilon + \frac{2t}{2^{g+1}}$.*

Proof. The proof of this lemma follows verbatim from the proof of Theorem 1 in [Pie12]. The key observation is that the reduction from subset LPN to LPN in Theorem 1 in [Pie12] is independent of the noise distribution. □

From Lemma 81, we get that Assumption 3 implies Assumption 4. In particular, we can set $\ell = n/5$ and $g = n/10$, $n' \geq 11n/10$. Then we get that if the exact (n, ℓ, τ) problem is $(2^{n^\beta}, 2^{-n^\beta})$ hard for some $\beta > 0$, then the subset exact $(n', \ell, 11n/10, \tau)$ is also $(2^{n^{\beta'}}, 2^{-n^{\beta'}})$ hard for some other $\beta' > 0$. For the rest of this section, we set the value of ℓ and g as above and we assume that the subset exact $(n', \ell, 11n/10, \tau)$ is $(2^{n^{\beta'}}, 2^{-n^{\beta'}})$ hard for some $\beta' > 0$.

Now we are ready to define the following Message Authentication Code (MAC) (G, S, V), which we refer to as LPN-MAC:

- The key generation algorithm G chooses a random matrix $X \in GF(2)^{\lambda \times n}$ and a string $x' \in GF(2)^\lambda$, where $\lambda = 2n$.
- The tagging algorithm samples $R \in GF(2)^{\ell \times \lambda}$ and $e \in GF(2)^\ell$ where e is a randomly chosen vector in $GF(2)^\ell$ with at most $\lceil \tau\ell \rceil$ ones. The algorithm outputs $(R, R^T \cdot (X \cdot m + x') + e)$.
- The verification algorithm, given tag (R, Z) for message m, computes $y = Z + R^T \cdot (X \cdot m + x')$ and accepts if and only if the total number of ones in y is at most $\lceil \tau\ell \rceil$.

Note that all arithmetic operations in the description of the above MAC are done over $GF(2)$. The following theorem shows that under suitable assumptions the above MAC is special and secure as desired:

Theorem 82. *Assuming that the exact (n, ℓ, τ) problem is (t, ϵ) hard for $t = 2^{n^\beta}$ and $\epsilon = 2^{-n^\beta}$ for $\beta > 0$, LPN-MAC described above is a (t', ϵ')-RMA-secure special MAC for $t' = 2^{n^{\beta'}}$ and $\epsilon' = 2^{-n^{\beta'}}$ for some $\beta' > 0$.*

Proof. First, it is trivial to observe that the MAC described above is a special MAC. Thus, we are only left with the task of proving the security of this construction. In [KPC$^+$11] (Theorem 5), the authors show that the above MAC is secure with the above parameters under Assumption 2 provided the vector e in the description of LPN-MAC is drawn from a distribution where every coordinate of e is an independent draw from Ber_τ. (We note that the MAC of Theorem 5 in [KPC$^+$11] is described in a slightly different way, but Dodis *et al.* [DKPW12] show that the above MAC and the MAC of Theorem 5 in [KPC$^+$11] are exactly the same). Follow the same proof verbatim except whenever [KPC$^+$11] use the subset LPN assumption, we use the subset exact LPN assumption (i.e. Assumption 4), we obtain a proof of Theorem 82. □

6.2.3 A problem for which inverse approximate uniform generation is hard but approximate uniform generation is easy. Given Theorem 80, in order to come up with a problem where inverse approximate uniform generation is hard but approximate uniform generation is easy, it remains only to show that the verification algorithm for LPN-MAC can be implemented in a class of functions for which approximate uniform generation is easy. Towards this, we have the following definition.

Definition 83. *BILINEAR-MAJORITY$_{\ell,n,\lambda,\tau}$ is a class of Boolean functions such that every $f \in$BILINEAR-MAJORITY$_{\ell,n,\lambda,\tau}$, $f : GF(2)^{\ell\times\lambda} \times GF(2)^{\ell} \times GF(2)^{n} \to \{0,1\}$ is parameterized by subsets $S_1,\dots,S_\lambda \subseteq [n]$ and $x^0 \in GF(2)^{\lambda}$ and is defined as follows : On input $(R,Z,m) \in GF(2)^{\ell\times\lambda} \times GF(2)^{\ell} \times GF(2)^{n}$, define*

$$y_i = Z_i + \sum_{j=1}^{\lambda} R_{ij} \cdot (\sum_{\ell\in S_j} m_\ell + x_j^0)$$

where all the additions and multiplications are in $GF(2)$. Then $f(R,Z,m) = 1$ if and only if at most $\lceil\tau\ell\rceil$ coordinates $y_1,\dots,y_\ell$ are 1.

Claim 84. *For the LPN-MAC with parameters ℓ, n, λ and τ described earlier, the verification algorithm V can be implemented in the class BILINEAR-MAJORITY$_{\ell,n,\lambda,\tau}$.*

Proof. Consider the LPN-MAC with parameters ℓ, n, λ and τ and secret key X and x'. Now define a function f in BILINEAR-MAJORITY$_{\ell,n,\lambda,\tau}$ where $x^0 = x'$ and the subset $S_j = \{i : X_{ji} = 1\}$. It is easy to check that the corresponding $f(R,Z,m) = 1$ if and only if (R,Z) is a valid tag for message m. □

The next and final claim says that there is an efficient approximate uniform generation algorithm for BILINEAR-MAJORITY$_{\ell,n,\lambda,\tau}$:

Claim 85. *There is an algorithm which given any $f \in$BILINEAR-MAJORITY$_{\ell,n,\lambda,\tau}$ (with parameters $S_1,\dots,S_\lambda \subseteq [n]$ and $x^0 \in GF(2)^{\lambda}$) and an input parameter $\delta > 0$, runs in time $\mathrm{poly}(n,\ell,\lambda,\log(1/\delta))$ and outputs a distribution which is δ-close to being uniform on $f^{-1}(1)$.*

Proof. The crucial observation is that for any (R,m), the set $\mathcal{A}_{R,m} = \{z : f(R,Z,m) = 1\}$ has cardinality independent of R and m. This is because after we fix R and m, if we define $b_i = \sum_{j=1}^{\lambda} R_{ij} \cdot (\sum_{\ell\in S_j} m_\ell + x_j^0)$, then $y_i = Z_i + b_i$. Thus, for every fixing of R and m, since b_i is fixed, the set of those Z such that the number of y_i's which are 1 is bounded by $\tau\ell$ is independent of R and m. This implies that the following sampling algorithm returns a uniformly random element of $f^{-1}(1)$:

- Randomly sample R and m. Compute b_i as defined earlier.
- Let $a = \lceil\tau\ell\rceil$ and consider the halfspace $g(y) = sign(a - \sum_{i=1}^{\ell} y_i)$. Now, we use Theorem 41 to sample uniformly at random from $g^{-1}(1)$ and hence draw a uniformly random y from the set $\{y \in \{0,1\}^{\ell} : \sum_{i=1}^{\ell} y_i \le a\}$.
- We set $Z_i = y_i + b_i$. Output (R,Z,m).

The guarantee on the running time of the procedure follows simply by using the running time of Theorem 41. Similarly, the statistical distance of the output from the uniform distribution on $f^{-1}(1)$ is at most δ. □

7 Efficient inverse approximate uniform generation when approximate uniform generation is infeasible

In Section 4 we gave an efficient algorithm for the inverse approximate uniform generation problem for halfspaces, and in Section 5 we gave a quasi-polynomial time algorithm for the inverse approximate uniform generation problem for DNFs. Since both these algorithms follow the standard approach, both crucially use efficient algorithms for the corresponding uniform generation problems [KLM89, MS04]. In this context, it is natural to ask the following question: *Is inverse approximate uniform generation easy only if the corresponding approximate uniform generation problem is easy?*

In this section we show that the answer to this question is "no" (for at least two reasons). First, we point out that a negative answer follows easily from the well-known fact that it is computationally hard to "detect unique solutions." In more detail, we recall the definition of the UNIQUE-SAT problem. UNIQUE-SAT is a promise problem where given a CNF Φ, the task is to distinguish between the following two cases:

- Φ has no satisfying assignment; versus
- Φ has *exactly* one satisfying assignment.

In a famous result, Valiant and Vazirani [VV86] showed the following.

Theorem 86. *[VV86] There is a randomized polynomial time reduction from CNF-SAT to UNIQUE-SAT.*

Let $\mathcal{C}$ denote the class of all n-variable CNF formulas that have exactly one satisfying assignment. As an immediate corollary of Theorem [VV86] we have the following:

Corollary 87. *There is a constant $c > 0$ such that unless SAT $\in$ BPTIME$(t(n))$, there is no approximate uniform generation algorithm for $\mathcal{C}$ which runs in time BPTIME$(t(n^c))$ even for variation distance $\epsilon = 1/2$.*

On the other hand, it is clear that there is a linear time algorithm for the inverse approximate uniform generation problem for the class $\mathcal{C}$: simply draw a single example x and output the trivial distribution supported on that one example.

The above simple argument shows that there indeed exist classes $\mathcal{C}$ where inverse uniform generation is "easy" but approximate uniform generation is "hard", but this example is somewhat unsatisfying, as the algorithm for inverse approximate uniform generation is trivial. It is natural to ask the following meta-question: is there a class of functions $\mathcal{C}$ such that approximation uniform generation is hard, but inverse approximate generation is easy because of a polynomial-time algorithm that "uses its samples in a non-trivial way?" In the rest of this section we give an example of such a problem.

Efficient inverse approximate uniform generation for graph automorphism. The following problem is more naturally defined in terms of a relation over combinatorial objects rather than in terms of a function and its satisfying assignments. Let us define $\mathcal{G}_n$ to be the set of all (simple undirected) graphs over vertex set $[n]$ and $\mathbb{S}_n$ to be the symmetric group over $[n]$. We define the relation $R_{\mathrm{aut}}(G, \sigma)$ over $\mathcal{G}_n \times \mathbb{S}_n$ as follows: $R_{\mathrm{aut}}(G, \sigma)$ holds if and only if σ is an automorphism for the graph G. (Recall that "σ is an automorphism for graph G" means that (x, y) is an edge in G if and only if $(\sigma(x), \sigma(y))$ is also an edge in G.) The inverse approximate uniform generation problem for the relation R_{aut} is then as follows: There is an unknown n-vertex graph G. The algorithm receives uniformly random samples from the set $\mathrm{Aut}(G) := \{\sigma \in \mathbb{S}_n : R_{\mathrm{aut}}(G, \sigma) \text{ holds }\}$. On input ϵ, δ, with probability $1 - \delta$ the algorithm must output a sampler whose output distribution is ϵ-close to the uniform distribution over $\mathrm{Aut}(G)$.

It is easy to see that $\mathrm{Aut}(G)$ is a subgroup of $\mathbb{S}_n$, and hence the identity permutation e_n must belong to $\mathrm{Aut}(G)$. To understand the complexity of this problem we recall the graph isomorphism problem:

Definition 88. *GRAPH-ISOMORPHISM is defined as follows : The input is a pair of graphs $G_1, G_2 \in \mathcal{G}_n$ and the goal is to determine whether they are isomorphic.*

While it is known that GRAPH-ISOMORPHISM is unlikely to be NP-complete [Sch88, BHZ87], even after several decades of effort the fastest known algorithm for GRAPH-ISOMORPHISM has a running time of $2^{\tilde{O}(\sqrt{n})}$ [Bab81]. This gives strong empirical evidence that GRAPH-ISOMORPHISM is a computationally hard problem. The following claim establishes that approximate uniform generation for R_{aut} is as hard as GRAPH-ISOMORPHISM:

Claim 89. *If there is a $t(n)$-time algorithm for approximate uniform generation for the relation R_{aut} (with error $1/2$), then for some absolute constant $c > 0$ there is a $\mathrm{poly}(t(n^c))$-time randomized algorithm for GRAPH-ISOMORPHISM.*

Proof. Let A be the hypothesized $t(n)$-time algorithm, so A, run on input $(G, 1/2)$ where G is an n-node graph, returns an element $\sigma \in \mathrm{Aut}(G)$ drawn from a distribution D that has $d_{\mathrm{TV}}(D, \mathcal{U}_{\mathrm{Aut}(G)}) \leq 1/2$. Given such an algorithm A, it is easy in $O(t(n))$ time to determine (with high constant probability of correctness) whether or not $|\mathrm{Aut}(G)| > 1$. Now the claim follows from the known fact [Hof82] that there is a polynomial-time reduction from GRAPH-ISOMORPHISM to the problem of determining whether an input graph has $|\mathrm{Aut}(G)| > 1$. □

While approximate uniform generation for R_{aut} is hard, the next theorem shows that the *inverse* approximate uniform generation problem for R_{aut} is in fact easy:

Theorem 90. *There is a randomized algorithm $A^{\mathrm{aut}}_{\mathrm{inv}}$ with the following property: The algorithm takes as input $\epsilon, \delta > 0$. Given access to uniform random samples from $\mathrm{Aut}(G)$ (where G is an unknown n-node graph), $A^{\mathrm{aut}}_{\mathrm{inv}}$ runs in time $\mathrm{poly}(n, \log(1/\epsilon), \log(1/\delta))$ and with probability $1 - \delta$ outputs a sampler C_{aut} with the following property : The running time of C_{aut} is $O(n \log n + \log(1/\epsilon))$ and the output distribution of C_{aut} is ϵ-close to the uniform distribution over $\mathrm{Aut}(G)$.*

Proof. The central tool in the proof is the following theorem of Alon and Roichman [AR94]:

Theorem 91. *[AR94] Let H be any group and let $h_1, \ldots, h_k$ be chosen uniformly at random from H. Consider the set $S = \cup_{i=1}^{k} \{h_i, h_i^{-1}\}$. Then, for $k = O(\log |H| + \log(1/\delta))$, with probability at least $1 - \delta$ the Cayley graph (H, S) has its second largest eigenvalue at most $1/2$.*

We now describe our algorithm $A^{\mathrm{aut}}_{\mathrm{inv}}$. On input ϵ, δ it draws $k = O(n \log n + \log(1/\delta))$ permutations $g_1, \ldots, g_k$ from $\mathrm{Aut}(G)$. It computes $g_1^{-1}, \ldots, g_k^{-1}$ and sets $S = \cup_{i=1}^{k} \{g_i, g_i^{-1}\}$. The sampler C_{aut} is defined as follows: It uses its input random bits to perform a random walk on the Cayley graph $(\mathrm{Aut}(G), S)$, starting at e_n, for $T = O(n \log n + \log(1/\epsilon))$ steps; it outputs the element of H which it reaches at the end of the walk. (Note that in order to perform this random walk it is not necessary to have $\mathrm{Aut}(G)$ explicitly – it suffices to explicitly have the set S.)

The analysis is simple: we first observe that every graph G has an automorphism group of size $|\mathrm{Aut}(G)| \leq n!$. Theorem 91 then guarantees that with probability at least $1 - \delta$ the Cayley graph $(\mathrm{Aut}(G), S)$ has its second eigenvalue bounded by $1/2$. Assuming that the second eigenvalue is indeed at most $1/2$, standard results in the theory of random walks on graphs imply that the distribution of the location reached at the end of the walk has variation distance at most ϵ from the uniform distribution over $\mathrm{Aut}(G)$. This concludes the proof. □

8 Conclusion and future work

We have considered inverse problems in approximate uniform generation for a range of interesting and well-studied classes of functions including LTFs, DNFs, CNFs, polynomial threshold functions, and more. While our findings have determined the computational complexity of inverse approximate uniform generation for these classes, several interesting questions and directions remain to be pursued. We outline some of these directions below.

One natural goal is to extend our results (both positive and negative) to a wider range of function classes; we list several specific classes that seem particularly worthy of investigation. The first of these is the class of intersections of two monotone LTFs. We note that Morris and Sinclair [MS04] gave efficient approximate uniform generation / counting algorithms for intersections of two monotone LTFs, but on the other hand, no distribution independent PAC or SQ learning algorithm is known for this class (although quasipoly(n)-time algorithms are known if both LTFs have integer weights that are at most $\mathrm{poly}(n)$ [KOS04]). The second class is that of poly(n)-size decision trees. Our DNF result gives a quasipoly(n/ϵ)-time inverse approximate uniform generation algorithm for this class; can this be improved to $\mathrm{poly}(n, 1/\epsilon)$? We note that in order to obtain such a result one would presumably have to bypass the "standard approach," since decision trees are not known to be PAC learnable faster than quasipoly(n/ϵ)-time under the uniform distribution on $\{-1, 1\}^n$. (We further note that while [FOS08] gives a reduction from learning the uniform distribution over satisfying assignments of a decision tree to the problem of PAC learning decision trees under the uniform distribution, this reduction relies crucially on the assumption — implicit in the [FOS08] framework — that the probability mass function of the hypothesis distribution can be efficiently evaluated on any input $x \in \{-1, 1\}^n$. In our framework this assumption need not hold so the [FOS08] reduction does not apply.) Still other natural classes to investigate are context free languages (for which quasi-polynomial time uniform generation algorithms are known [GJK$^+$97]) and various classes of branching programs. It may also be of interest to consider similar problems when the underlying measure is (say) Gaussian or log-concave.

Another interesting direction to pursue is to study inverse approximate uniform generation for combinatorial problems like matching and coloring as opposed to the "boolean function satisfying assignment"–type problems that have been the main focus of this paper. We note that preliminary arguments suggest that there is a simple efficient algorithm for inverse approximate uniform generation of perfect matchings in bipartite graphs. Similarly, preliminary arguments suggest that for the range of parameters for which the "forward" approximate uniform generation problem for colorings is known to be easy (namely, the number q of allowable colors satisfies $q > 11\Delta/6$ where Δ is the degree [Vig99]), the inverse approximate uniform generation problem also admits an efficient algorithm. These preliminary results give rise to the question of whether there are similar *combinatorial* problems for which the complexity of the "forward" approximate uniform generation problem is not known and yet we can determine the complexity of inverse approximate uniform generation (like the group theoretic setting of Section 7).

Finally, for many combinatorial problems, the approximate uniform generation algorithm is to run a Markov chain on the state space. In the regimes where the uniform generation problem is hard, the Markov chain does not mix rapidly which is in turn equivalent to the existence of sparse cuts in the state space. However, an intriguing possibility arises here: If one can show that the state space can be partitioned into a small number of components such that each component has no sparse cuts, then given access to a small number of random samples from the state space (with at least one such example belonging to each component), one may be able to easily perform approximate uniform generation. Since the inverse approximate uniform generation algorithms that we consider have access to random samples, this opens the possibility of efficient approximate uniform generation algorithms in such cases. To conclude, we give an example of a natural combinatorial problem (from statistical physics) where it seems that this is essentially the situation (although we do not have a formal proof). This is the 2-D Ising model, for which the natural Glauber dynamics is known to have exponential mixing time beyond the critical temperature [Mar98]. On the other hand, it was recently shown that even beyond the critical temperature, if one fixes the boundary to have the same spin (all positive or all negative) then the mixing time comes down from exponential to quasipolynomial [LMST]. While we do not know of a formal reduction, the fact that fixing the boundary to the same spin brings down the mixing time of the Glauber dynamics from exponential to quasipolynomial is "morally equivalent" to the existence of only a single sparse cut in the state space of the graph [Sin12]. Finding other such natural examples is an intriguing goal.

Acknowledgements. We thank Alistair Sinclair for helpful conversations about approximate counting and approximate uniform generation. We thank Luca Trevisan for suggesting the use of "blow-up" constructions for hardness results. We thank Jonathan Katz, Tal Malkin, Krzysztof Pietrzak and Salil Vadhan for answering several questions about MACs and signature schemes. We are thankful to Kousha Etessami, Elchanan Mossel, Li-Yang Tan and Thomas Watson for helpful conversations about this work.

References

[AD98] J. Aslam and S. Decatur. Specification and simulation of statistical query algorithms for efficiency and noise tolerance. *Journal of Computer and System Sciences*, 56:191–208, 1998.

[AG11] Sanjeev Arora and Rong Ge. New Algorithms for Learning in Presence of Errors. In *ICALP 2011*, pages 403–415, 2011.

[AR94] N. Alon and Y. Roichman. Random Cayley Graphs and Expanders. *Random Structures and Algorithms*, 5:271–284, 1994.

[Bab81] László Babai. Moderately exponential bound for graph isomorphism. In *Proceedings of the 1981 International FCT-Conference on Fundamentals of Computation Theory*, pages 34–50, 1981.

[BFKV97] A. Blum, A. Frieze, R. Kannan, and S. Vempala. A polynomial time algorithm for learning noisy linear threshold functions. *Algorithmica*, 22(1/2):35–52, 1997.

[BHZ87] Ravi B. Boppana, Johan Hastad, and Stathis Zachos. Does co-np have short interactive proofs? *Information Processing Letters*, 25(2):127 – 132, 1987.

[BKW03] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM*, 50(4):506–519, July 2003.

[BMS08] G. Bresler, E. Mossel, and A. Sly. Reconstruction of Markov Random Fields from Samples: Some Observations and Algorithms. In *APPROX-RANDOM*, pages 343–356, 2008.

[CS00] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. *ACM Trans. Inf. Syst. Secur.*, 3(3):161–185, 2000.

[DDS12a] C. Daskalakis, I. Diakonikolas, and R.A. Servedio. Learning k-modal distributions via testing. In *SODA*, pages 1371–1385, 2012.

[DDS12b] C. Daskalakis, I. Diakonikolas, and R.A. Servedio. Learning poisson binomial distributions. In *STOC*, pages 709–728, 2012.

[Dec93] S. Decatur. Statistical queries and faulty PAC oracles. In *Proceedings of the Sixth Workshop on Computational Learning Theory*, pages 262–268, 1993.

[DGL05] F. Denis, R. Gilleron, and F. Letouzey. Learning from positive and unlabeled examples. *Theoretical Computer Science*, 348:70–83, 2005.

[DKPW12] Yevgeniy Dodis, Eike Kiltz, Krzysztof Pietrzak, and Daniel Wichs. Message Authentication, Revisited. In *ECRYPT12*, pages 355–374, 2012.

[DL01] L. Devroye and G. Lugosi. *Combinatorial methods in density estimation*. Springer Series in Statistics, Springer, 2001.

[DMR06] C. Daskalakis, E. Mossel, and S. Roch. Optimal phylogenetic reconstruction. In *STOC*, pages 159–168, 2006.

[Dye03] M. Dyer. Approximate counting by dynamic programming. In *STOC*, pages 693–699, 2003.

[Fis03] M. Fischlin. The Cramer-Shoup Strong-RSASignature Scheme Revisited. In *Public Key Cryptography - PKC 2003*, pages 116–129, 2003.

[FOS08] Jon Feldman, Ryan O'Donnell, and Rocco A. Servedio. Learning mixtures of product distributions over discrete domains. *SIAM J. Comput.*, 37(5):1536–1564, 2008.

[GJK$^+$97] V. Gore, M. Jerrum, S. Kannan, Z. Sweedyk, and S. Mahaney. A quasi-polynomial-time algorithm for sampling words from a context-free language. *Inf. Comput.*, 134(1):59–74, 1997.

[GLS88] Martin Grötschel, Lászlo Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization,* volume 2. Springer, 1988.

[Gol04] Oded Goldreich. *Foundations of Cryptography-volume 2*. Cambridge University Press, Cambridge, 2004.

[Hof82] Christoph M. Hoffmann. *Group-Theoretic Algorithms and Graph Isomorphism*, volume 136 of *Lecture Notes in Computer Science*. Springer, 1982.

[HV03] Thomas P. Hayes and Eric Vigoda. A non-markovian coupling for randomly sampling colorings. In *FOCS*, pages 618–627, 2003.

[HW10] S. Hohenberger and B. Waters. Constructing Verifiable Random Functions with Large Input Spaces. In *EUROCRYPT*, pages 656–672, 2010.

[Jer95] Mark Jerrum. A very simple algorithm for estimating the number of k-colorings of a low-degree graph. *Random Struct. Algorithms*, 7(2):157–166, 1995.

[JS89] Mark Jerrum and Alistair Sinclair. Approximating the permanent. *SIAM J. Comput.*, 18(6):1149–1178, 1989.

[JSV04] Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM*, 51(4):671–697, 2004.

[JVV86] M. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986.

[Kea98] M. Kearns. Efficient noise-tolerant learning from statistical queries. *Journal of the ACM*, 45(6):983–1006, 1998.

[Kha80] L.G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53 – 72, 1980.

[KL83] R.M. Karp and M. Luby. Monte-carlo algorithms for enumeration and reliability problems. In *FOCS*, pages 56–64, 1983.

[KLM89] R. M. Karp, M. Luby, and N. Madras. Monte-Carlo Approximation Algorithms for Enumeration Problems. *Journal of Algorithms*, 10(3):429–448, 1989.

[KOS04] A. Klivans, R. O'Donnell, and R. Servedio. Learning intersections and thresholds of halfspaces. *Journal of Computer & System Sciences*, 68(4):808–840, 2004.

[KPC$^+$11] Eike Kiltz, Krzysztof Pietrzak, David Cash, Abhishek Jain, and Daniele Venturi. Efficient Authentication from Hard Learning Problems. In *ECRYPT11*, pages 7–26, 2011.

[KS04] A. Klivans and R. Servedio. Learning DNF in time $2^{\tilde{O}(n^{1/3})}$. *Journal of Computer & System Sciences*, 68(2):303–318, 2004.

[KSS10] Jonathan Katz, Ji Sun Shin, and Adam Smith. Parallel and Concurrent Security of the HB and HB$^+$ Protocols. *Journal of Cryptology*, 23(3):402–421, 2010.

[LMST] E. Lubetzky, F. Martinelli, A. Sly, and F. L. Toninelli. Quasi-polynomial mixing of the 2D stochastic Ising model with plus boundary up to criticality. To appear in Journal of the European Mathematical Society.

[Lys02] Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In Moti Yung, editor, *Advances in Cryptology — (CRYPTO 2002)*, volume 2442 of *Lecture Notes in Computer Science*, pages 597–612. Springer-Verlag, 2002.

[Mar98] F. Martinelli. Lectures on Glauber dynamics for discrete spin models. In , volume 1717 of *Lecture Notes in Mathematics*, pages 93–191. Springer, 1998.

[Mos07] E. Mossel. Distorted Metrics on Trees and Phylogenetic Forests. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 4(1), 2007.

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable Random Functions. In *Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 120–130, 1999.

[MS04] Ben Morris and Alistair Sinclair. Random walks on truncated cubes and sampling 0-1 knapsack solutions. *SIAM J. Comput.*, 34(1):195–226, 2004.

[MT94] W. Maass and G. Turan. How fast can a threshold gate learn? In S. Hanson, G. Drastal, and R. Rivest, editors, *Computational Learning Theory and Natural Learning Systems*, pages 381–414. MIT Press, 1994.

[Mur71] S. Muroga. *Threshold logic and its applications*. Wiley-Interscience, New York, 1971.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[Pie12] Krzysztof Pietrzak. Subspace LWE. In *Theory of Cryptography Conference*, pages 548–563, 2012.

[Sch78] Thomas J. Schaefer. The Complexity of Satisfiability Problems. In *STOC*, pages 216–226, 1978.

[Sch88] Uwe Schöning. Graph Isomorphism is in the Low Hierarchy. *J. Comp. Sys. Sci.*, 37(3):312–323, 1988.

[Sin12] A. Sinclair. Personal communication. 2012.

[Sip83] M. Sipser. A complexity-theoretic approach to randomness. In *STOC*, pages 330–335, 1983.

[SJ89] Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Inf. Comput.*, 82(1):93–133, 1989.

[Sto83] L. Stockmeyer. The complexity of approximate counting. In *STOC*, pages 118–126, 1983.

[Tov84] Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984.

[Vai89] P. Vaidya. A new algorithm for minimizing convex functions over convex sets. In *Proceedings of the Thirtheth Symposium on Foundations of Computer Science*, pages 338–343, 1989.

[Vai96] P. M. Vaidya. A new algorithm for minimizing convex functions over convex sets. *Math. Prog.*, 73(3):291–341, 1996.

[Val12] G. Valiant. Finding Correlations in Subquadratic Time, with Applications to Learning Parities and Juntas. In *FOCS*, 2012.

[Ver90] Karsten A. Verbeurgt. Learning DNF under the uniform distribution in quasi-polynomial time. In Mark A. Fulk, editor, *Conference on Learning Theory*, pages 314–326. Morgan Kaufmann, 1990.

[Vig99] Eric Vigoda. Improved bounds for sampling colorings. In *FOCS*, pages 51–59, 1999.

[VV86] Leslie G. Valiant and V. V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

[Wat12] Thomas Watson. The complexity of estimating Min-entropy. Technical Report 70, Electronic Colloquium in Computational Complexity, 2012.