Restriction Access

Zeev Dvir* Anup Rao† Avi Wigderson‡ Amir Yehudayoff§

Abstract

We introduce a notion of non-black-box access to computational devices (such as circuits, formulas, decision trees, and so forth) that we call *restriction access*. Restrictions are partial assignments to input variables. Each restriction simplifies the device, and yields a new device for the restricted function on the unassigned variables. On one extreme, full restrictions (assigning all variables) correspond to evaluating the device on a complete input, yielding the result of the computation on that input, which is the same as standard black-box access. On the other extreme, empty restrictions (assigning no variables) yield a full description of the original device. We explore the grey-scale of possibilities in the middle.

Focusing on learning theory, we show that restriction access provides a setting in which one can obtain positive results for problems that have resisted attack in the black-box access model. We introduce a PAC-learning version of restriction access, and show that one can efficiently learn both decision trees and DNF formulas in this model. These two classes are not known to be learnable in the PAC model with black-box access.

Our DNF learning algorithm is obtained by a reduction to a general learning problem we call *population recovery*, in which random samples from an unknown distribution become available only after a random part of each is obliterated. Specifically, assume that every member of an unknown population is described by a vector of values. The algorithm has access to random samples, each of which is a random member of the population, whose values are given *only* on a *random* subset of the attributes. Analyzing our efficient algorithm to fully recover the unknown population calls for understanding another basic problem of independent interest: "robust *local* inversion" of matrices. The population recovery algorithm and construction of robust local inverses for some families of matrices are the main technical contributions of the paper.

We also discuss other possible variants of restriction access, in which the values to restricted variables, as well as the subset of free (unassigned) variables, are generated deterministically or randomly, in friendly or adversarial fashions. We discuss how these models may naturally suit situations in computational learning, computational biology, automated proofs, cryptography and complexity theory.

*Princeton University, Princeton NJ, zeev.dvir@gmail.com. Research partially supported by NSF grant CCF-0832797 and by the Packard fellowship.

†University of Washington, Seattle WA, anuprao@cs.washington.edu. Supported by the National Science Foundation under agreement CCF-1016565.

‡Institute for Advanced Study, Princeton NJ, avi@math.ias.edu. Research partially supported by NSF grants CCF-0832797 and DMS-0835373.

§Technion - IIT, Haifa, Israel, amir.yehudayoff@gmail.com. Horev fellow – supported by the Taub Foundation. Research supported by ISF and BSF.

1 Introduction

Situations in which we have access to a computational process and need to understand, model or imitate its behavior are practically universal. Not surprisingly, this problem has received plenty of theoretical attention. A modern, natural abstraction of the process is by using a computational model: hypothesize that a device D is performing the process. It can be a deterministic device like a circuit, a probabilistic model like a Baysian network, or a generative model like a grammar of a language. For the purpose of this initial discussion, we restrict ourselves to deterministic computational devices. For simplicity, assume that the device D computes some unknown boolean function f on n inputs (the discussion naturally extends to other families of functions).

A crucial issue is how to model the type of *access* we have to the process D. This is the main issue we wish to address here. The most common choice is the so-called black-box access, where we are allowed access to pairs of inputs and outputs of the form $(x, f(x))$. Black-box access provides very little information about D's behavior: one is given only inputs and the results of the computation. Techniques that only involve such access, therefore, tend to be quite generic.

The generality of black-box access often make it too weak. For example, under standard cryptographic hardness assumptions, black box access is not enough to efficiently learn even relatively simple models such as constant-depth threshold circuits [KV94]. Can one obtain positive results with more liberal access?

Furthermore, in many scientific investigations (see examples in Section 2) one can reasonably assume access to much more information about the computation of the device D than input-output behavior. Scientists working on such problems actually use this additional information when trying to understand the process at hand. How can one model such additional information?

1.1 Restriction Access

We introduce a new way to model access to a computational device/process using *restrictions*. This new model, which we call *restriction access*, is an extension of black-box access, and provides the learner with partial information about the "inner workings" of the device D. There are many choices that need to be made when defining this model. We wish the formalism of the model to be both *useful* and mathematically *clean*. We start with an informal description of the model, and then discuss how it may be relevant to various contexts.

We take our cue from the well-studied notion of *restriction* in computational complexity. This notion has been very useful. For example it has been used to prove computational hardness results, starting with the works of Subbotovskaya [Sub61] and Neciporuk [Nec64] in the 60's, and continuing in the early 80's with the works of Ajtai [Ajt83] and Furst et al. [FSS84].

A *restriction* ρ is a pair $\rho = (z, L)$, where $L \subset [n]$ is the set of *live* variables and $z \in \{0,1\}^{\overline{L}}$ is an assignment to all variables in $\overline{L} = [n] \setminus L$. Given $f : \{0,1\}^n \to \{0,1\}$, the function $f|_\rho : \{0,1\}^L \to \{0,1\}$ is the restriction of f obtained by fixing the variables in $\overline{L}$ to their values in z. Similarly, given a device D computing f, we can restrict it by fixing its inputs in $\overline{L}$ to their values in z. For a device, we can actually do more: we can *simplify* the device according to some fixed rule (more on the important simplification step later). The new restricted device $D|_\rho$ computes the function $f|_\rho$, but only contains partial information about D. Restriction access is a strict generalization of black-box access to D: using any natural simplification rule, if L is the empty set, then $D|_\rho$ is $f(z)$.

We illustrate this concept using *decision trees*. A decision tree is a rooted binary tree whose leaves are labelled by 0 or 1, and inner nodes are labelled by variables. An input defines a walk in

the tree from the root to a leaf, and the label of the leaf reached is the output of the computation. A restriction can be used to simplify a given decision tree using a natural simplification rule. After fixing some of the variables, we remove all subtrees that are never reached and contract any remaining paths (for a formal definition see Section 4). Thus, a restriction of a decision tree is another decision tree on the unfixed set of variables. Similarly, when dealing with circuits, for example, we shall ensure that the restriction of a circuit will also be a circuit. Thus, a restriction provides us with information on the inner workings of the computation without revealing the entire computation process.

1.2 Structure and Control of Restrictions

An important issue is consistency (or lack of it) of representations of several restrictions of a given process. Consider the formula $((x \wedge y) \vee (z \wedge (w \vee y))) \wedge \overline{y}$. If we restrict it to the partial assignment $x = 1, w = 1$ we obtain

$$\begin{aligned}
&((x \wedge y) \vee (z \wedge (w \vee y))) \wedge \overline{y} \\
&\rightarrow ((1 \wedge y) \vee (z \wedge (1 \vee y))) \wedge \overline{y} \\
&\rightarrow (y \vee (z \wedge 1)) \wedge \overline{y} \\
&\rightarrow (y \vee z) \wedge \overline{y}.
\end{aligned}$$

Should the order of literals in the formula above be respected for all restrictions? All algorithms we present work *without* this assumption. One way to formalize this is by letting an adversary "permute" the representation (e.g. the gates) of the process. In the above example the evaluation can thus take 6 different option, like $\overline{y} \wedge (y \vee z)$ and $\overline{y} \wedge (z \vee y)$, and the algorithm would have to perform well regardless of which of these representations it gets.

Another important issue is *control* over the set of live variables. The control we have on the choice of inputs x varies greatly with the process under investigation. In some cases, we have no control at all, and may assume that it is adversarial. In others, an unknown distribution governs the inputs. Such cases partially motivated the definition of the PAC-learning model [Val84]. In other scenarios, we can choose inputs by making our own queries to the system, for example, when trying to learn how to add integers from a teacher at school. Allowing such "friendly" choices of inputs is also part of the standard notions of cryptographic security.

The *easiest* learning scenario is when we are allowed to choose the restrictions adaptively. This seems too good to be true in most situations, but is certainly relevant to some of the biological systems for which we have plenty of control over the systems studied. A bit weaker is the access mentioned below in the section on identity-based encryption, in which the set L is fixed (by the designer) and the assignment (identities) can be chosen by the users trying to break the system. In the most pessimistic case, both the set of live variables and the assignment are adversarial.

Our main focus is on situations in which both the set of variables and the assignment are chosen at random. We assume that that there is *one* distribution (say ν) on all inputs $\{0,1\}^n$, and after the set of live variables L is chosen, we fix only the remaining variables according to the sample x from ν, $z = x_{\overline{L}}$. The subset L itself is chosen from a distribution (say τ) on subset of $[n]$. Thus ν, τ specify a "random restriction."

1.3 PAC-learning with Restriction Access

Since the formal statements of our technical results require lengthy definitions, we begin with an informal high-level statement of our learning theory results. We now describe two PAC-learning algorithms that use restriction access.

In the standard PAC (Probably Almost Correct) model, introduced by Valiant [Val84], the learning algorithm is given access to independent, identically distributed samples of the form $(x, f(x))$, where x is sampled from some unknown distribution ν on $\{0,1\}^n$ and $f : \{0,1\}^n \mapsto \{0,1\}$ is a *concept* we wish to learn. An algorithm is a PAC-learning algorithm if with high probability, it outputs a hypothesis $h : \{0,1\}^n \mapsto \{0,1\}$ that approximates f with respect to the distribution ν on inputs. Almost all concept-classes studied in the PAC-learning literature are defined using some natural computational device family. Here we suggest to "open the box" and give more information to the learner.

In our version of the PAC model, the learning algorithm receives samples of the form[1] $(x, \rho, D|_\rho)$, where D is some (fixed but unknown) device computing the concept f. The learning algorithm has access to some of the "inner workings" of the computation of f.

We did not yet specify *how* the restriction ρ is chosen. For PAC-learning, we choose a model in which x is sampled according to ν and then a set L is chosen so that each variable is independently kept "alive" with some probability $0 < \mu < 1$. The value of z (the assignment to the fixed variables) is chosen to be the restriction of x to $\overline{L}$. This choice of distribution over restrictions is not the only one possible, but we feel it is general enough (and sufficiently challenging). We will refer to this distribution on sets as the *μ-marginal set* distribution.

In this variant of PAC-learning we obtain two main algorithmic results:

1. A PAC-learning algorithm for decision trees from restrictions sampled according to the μ-marginal set distribution, for all $\mu > 0$. See Theorem 4.2 for the formal statement of our result.

2. A PAC-learning algorithms for DNF formulas from restrictions sampled according to the μ-marginal set distribution, for all $\mu > 0.365$. See Theorem 5.2 for a formal statement.

Both learning algorithms are efficient in that they run in polynomial time in all parameters (including $1/\mu$ for decision trees).

Researchers have been trying to find efficient PAC-learning algorithms for decision trees or DNF formulas ever since Valiant's original paper but it is still not known whether or not such efficient learners exist. Restriction access, which generalizes the standard notion of PAC-learning, could possibly lead to new insights on these open problems. For more on PAC-learning decision trees and DNF formulas see [EH89, KS04].

PAC-learning DNF formulas from restrictions is substantially more difficult than its decision-tree counterpart. The learner for DNF formulas uses a reduction to a new learning problem we call *population recovery*. Roughly, in the population recovery problem (PRP), each member of an unknown population is described by a vector of attributes. A sample from the populations is obtained by choosing a random member of the population (from some unknown distribution π on it), and then randomly obliterating some of the attributes of the chosen member. The problem

[1] Access to x is not required in an information theoretic sense. We include x since our DNF formulas learning algorithm requires access to x.

is to reconstruct the population given a sequence of independent samples, namely, to completely recover all members with non-negligible π-measure.

Our solution of PRP calls for a different object that may be of independent interest: robust local inverses of matrices. A robust local inverse is useful in the following scenario. Assume we are performing a set of measurements $\{\tilde{y}_a\}$ to estimate some unknowns $\{y_a\}$. We make these measurements since we wish to estimate some linear combination $x = \sum_a c_a y_a$ of $\{y_a\}$, where the vector (c_a) is known. If the norm of (c_a) is large, then the measurement errors may cause the natural estimate $\sum_a c_a \tilde{y}_a$ to be far from x. A robust local inverse is (roughly) a vector $(\tilde{c}_a)$ that guarantees that $\sum_a \tilde{c}_a \tilde{y}_a$ is close to x.

For more on PRP and robust local inverses see Section 6.

1.4 Related Work

In the context of learning theory, various generalizations of black-box access have been considered. Perhaps most relevant is the work of Goldman, Kwek and Scott [GKS03] that defined UAV learning. Roughly, UAV learning allows non-black-box access to the objective function f in the form of querying whether some restriction of f is constant or not. A similar model for learning was already defined by Valiant in his original paper defining PAC-learning [Val84]. Another related work is that of Angluin, Hellerstein and Karpinski [AHK93] defining a "projective equivalence" oracle, and showing how to learn read-once formulas using such an oracle.

Another non-black-box model was introduced by Angluin et. al. [AACW06] in which access to a circuit with some fixed assignment is given in the form of *Value Injection Queries* and the structure of the circuit is recovered. This model is different than restriction access in two ways. First, it allows the learner to fix wires *internal* to the circuit (and not just inputs). Second, the only value given to the learner is the final output of the circuit and not a "sub-circuit" as is done in the restriction access case.

1.5 Organization

In Section 2, we motivate restriction access by giving several different examples for possible applications. In Section 3, we give a formal description of restriction access. Section 4 describes our PAC-learning algorithm for decision trees, and Section 5 gives the algorithm for learning DNF formulas. In Section 6 we study the two notions that are the technical heart of the PAC-learning algorithms: the population recovery problem (PRP) and robust local inverses.

2 Motivation

We now describe several scenarios in which restriction access naturally occurs. In some examples the model is not perfect, and is some resolving the associated algorithmic problem may not be the best way to understand the given scenario. Nevertheless, we feel that these examples motivate a study of the restriction access model.

2.1 Scientific Modeling: Sub-systems as Restrictions

In numerous scientific endeavors, one is trying to understand a system by first understanding its subsystems and then combine the information to portray a picture of the whole system. By studying

how different parameters of the system interact, while "freezing" other parameters, scientists get to study a sub-system obtained using restrictions. When a complex system of this type is modeled by a circuit (and this is often done e.g. [Alo06]), the sub-systems may be viewed as restrictions of the original circuit.

2.2 Computational Biology: Reconstructing Phylogenetic Trees

Phylogenetic trees capture the hereditary structure of a family of extant structures (e.g. species or proteins) by viewing them as leaves of a tree. Each internal node in the tree represents the (possibly extinct) least common ancestor of its sub-tree. An internal node and its direct children depict a single evolutionary step/choice, such as a mutation. These events may be viewed as variables, and the tree as a decision tree that computes the species at the leaf from a sequence of evolutionary choices. Lots of literature is devoted to constructing such trees, and known algorithms for this problem are still quite slow when the number of leaves is large.

Consider an evolutionary process where different mutations thrive and others become extinct on different islands due to their different environments. We may want to reconstruct the whole phylogenetic tree from the ones we reconstruct for each island separately. This problem has been considered in computational biology [ASSU81, HKW99]. In our context, it can be viewed as a special case of learning decision trees using restriction access, under some unknown distribution of values to the variables (i.e. evolutionary choices). We show that this is efficiently possible, even when the number of live variables in every subtree is very small.

2.3 Automated Provers: Generalizing Proofs from Special Cases

As in science, mathematics provides numerous examples in which we are trying to generalize from special cases. Often, mathematicians try to prove a statement for a general case, while knowing how to prove it for a variety of special cases. For example, while the proof of the graph-minor theorem of Robertson and Seymour is extremely complex, it started from a few examples relating minor-free families of graphs and their topological embeddings, a connection which drives the proof.

To formalize the problem of reconstructing proofs from special cases, let us restrict ourselves to propositional proofs and their proof systems. In such systems, one starts with a *contradiction*, namely, an unsatisfiable collection of constraints on a set of boolean variables, and a proof of the fact that it has no satisfying assignments is a sequence of "proof lines." Each "line" derived from previous ones by some simple sound deduction rule, until a blatant contradiction is derived. Different propositional proof systems capture different types of reasoning (e.g. logical, geometric or algebraic) and appropriately allow different types of "lines." See the survey [BP98] for more information.

Propositional proofs are closed under restrictions: Setting some of the variables yields a restricted contradiction (which may be viewed as a "special case" of the general theorem) and similarly a restricted proof of the same type for that restricted contradiction.

What is especially nice about proofs in our context is that in the black-box model they make no sense. But with partial assignments, one can interpret the resulting proof as a proof of a special case of the contradiction.

2.4 Cryptography: Identity-Based Encryption

Cryptography provides a wealth of problems where generalizations of black-box access to computation are useful. A variety of physical attacks are known in which one can get snap-shots of memory or computation from credit cards, smart cards, shut off computers and more [Koc96, KJJ99, QS01, GMO01, ARR03]. It is not clear that the restriction access model fits these. However, restriction access presents itself more naturally in other forms of attack, for example, when the adversary has access not only to the decryption of specific chosen messages, but actually to sub-computations of the decryption algorithm. Indeed, in identity-based encryption [BF01], individual users reveal the decryption circuit to all cypher-texts whose prefix is that user's name, say. Now assume a few users collaborate. Can they break the scheme and impersonate others? Such schemes are known for tailor-made public-key encryption based on Weil-pairings. It is not known if one can achieve it for standard CCA secure (namely, that withstand black-box attacks) public key systems, for instance, ones based on RSA.

2.5 Computational Complexity: Collapse of the Polynomial-Time Hierarchy

A well-known example of the dichotomy between having black-box access and clear access to a circuit is when we assume access to a small circuit C that solves an NP-complete problem, say SAT. In this case, the very existence of such C implies the collapse of the polynomial time hierarchy PH to $\Sigma^2 \cap \Pi^2$. This class is (to the best of our knowledge) out of reach for a deterministic polynomial-time algorithm having black-box access to C (this model computes "only" P^{NP}). But given the circuit C itself, a polynomial-time algorithm can use C to eliminate quantifiers one by one, and compute all of PH. This motivates the question: At what level of restriction access (size of live-variable-set) does the phase transition occurs?

3 The Formal Setting

For a set of integers L, we denote by $f : \{0,1\}^L \mapsto \{0,1\}$ a boolean function on the $|L|$ variables $\{x_i \,|\, i \in L\}$. We start with the simplest restriction, that of functions.

Definition 3.1 (Restrictions)**.** *A* restriction ρ *is a pair* $\rho = (z, L)$*, where* $L \subset [n]$ *and* $z \in \{0,1\}^{\overline{L}}$ *is an assignment to variables outside* L*. Let* $f : \{0,1\}^{[n]} \mapsto \{0,1\}$*. The* restricted function $f|_\rho : \{0,1\}^L \mapsto \{0,1\}$ *is naturally defined by fixing the values of the variables outside* L *to the values given by* z*.*

Restriction preserve the "names" of the different variables. For example, fixing $x_1 = 1$ in the function $x_1 \wedge x_2$ does not give the same restricted function as fixing $x_2 = 1$, although in both cases the function computed is dictatorship.

We use the following, rather general, definition of a computational model. This definition allows us to define the learning model in a generic way. It is helpful to keep in mind the example of decision trees, and after each definition we shortly consider the application of that definition to decision trees.

Definition 3.2 (Computational Model)**.** *A* computational model $\mathcal{M}$ *is a collection of finite sets* $\mathcal{M} = \{\mathcal{M}_L : L \subset \mathbb{N}, |L| < \infty\}$ *such that, for each set* L*, each element* $D \in \mathcal{M}_L$ *is associated with a boolean function* $f_D : \{0,1\}^L \mapsto \{0,1\}$*. We say that* f_D *is the function* computed *by the* device

D. We also assume that the elements of $\mathcal{M}$ have some encoding as binary strings. For simplicity of notation, denote $\mathcal{M}_n = \mathcal{M}_{[n]}$.

In the decision tree example, $\mathcal{M}_L$ can be thought of as the set of decision trees over variables in L of size, say, at most $|L|^3$.

A related notion from the language of learning theory is that of a concept class, which is a class of functions rather than devices. We use devices instead of functions, as we are "opening" the black-box.

We now abstractly define restriction of computational devices. The definition does not require the simplification rule to actually "simplify" the device in any way, although any meaningful simplification rule should.

Definition 3.3 (Simplification Rule)**.** *Let $\mathcal{M}$ be a computational model, let $D \in \mathcal{M}_n$ with $f_D : \{0,1\}^n \mapsto \{0,1\}$. A* simplification *of D according to a restriction $\rho = (z, L)$ is an element $D|_\rho \in \mathcal{M}_L$ such that $f_{D|_\rho} = (f_D)|_\rho$. A* simplification rule on $\mathcal{M}$ *is a mapping $\mathcal{S}$ that, given a device D and a restriction ρ, produces $D|_\rho$.*

Again, consider the decision tree example. Here is a (loose) description of a simplification rule $\mathcal{S}_{DT}$ that gets a tree T and restriction $\rho = (z, L)$, and outputs a tree $T|_\rho$ (see Section 4 for a formal definition). The partial assignment z fixes some variables. The tree $T|_\rho$ is obtained from T by removing sub-trees that are not reachable (e.g. due to a fixed variable at the root), and contracting any remaining paths

We now define the type of (random) access a learning algorithm has to a restricted device. This definition is, of course, not the most general possible.

Definition 3.4 (Restriction Sampler)**.** *Let $\mathcal{M}$ be a computational model and let $D \in \mathcal{M}_n$. Let $\mathcal{S}$ be a simplification rule on $\mathcal{M}$. Let ν be a distribution on $\{0,1\}^n$, and let τ be a distribution on subsets of $[n]$. The* restriction sampler $\mathsf{Sampler}(\mathcal{S}, D, \nu, \tau)$ *is an oracle that computes its output in the following way:*

1. *Sample an element $x \in \{0,1\}^n$ according to ν.*
2. *Sample a set $L \in [n]$ according to τ, independently of x.*
3. *Set $z \in \{0,1\}^{\overline{L}}$ to be the values of x on the set $\overline{L}$.*
4. *Compute the simplification $D|_\rho$ with the restriction $\rho = (z, L)$ and the simplification rule $\mathcal{S}$.*
5. *Return $(x, \rho, D|_\rho)$.*

In the decision tree case, $\mathsf{Sampler}(\mathcal{S}_{DT}, T, \nu, \tau)$ outputs a random sub-decision-tree of T, together with an input and a restriction.

In most cases of interest, the device $D|_\rho$ can be efficiently evaluated on the input x. This explains why the value $f_D(x)$ is omitted from the output of the sampler.

We mainly use the following natural distributions on sets (τ in the definition above).

Definition 3.5 (Distribution on Sets)**.** *Let $\mathsf{Ind}_n(\mu)$ be the distribution on subsets of $[n]$ which samples a set by independently picking each element in $[n]$ to be in the set with probability $\mu \in (0,1)$. Let $\mathsf{Ind}(\mu) = \{\mathsf{Ind}_n(\mu) : n \in \mathbb{N}\}$.*

We are finally ready to define PAC-learning version that uses restriction access.

Definition 3.6 (PAC-learning)**.** *Let $\mathcal{M}$ be a computational model and let $\mathcal{S}$ be a simplification rule on $\mathcal{M}$. Let $\tau = \{\tau_n : n \in \mathbb{N}\}$ be a collection of distributions, τ_n is a distribution on subsets of $[n]$.*

An algorithm $\mathcal{A}$ is an (ε, δ)-PAC$_{RA}$ learning algorithm for $\mathcal{M}$ with set distribution τ and simplification rule $\mathcal{S}$ if, for every n, every $D \in \mathcal{M}_n$ and every distribution ν on $\{0,1\}^n$, when given samples from $\mathsf{Sampler}(\mathcal{S}, D, \nu, \tau_n)$ the algorithm $\mathcal{A}$ outputs with probability at least $1 - \delta$ a function $h : \{0,1\}^n \mapsto \{0,1\}$ (in some representation) such that

$$\mathbb{P}_x[h(x) \neq f_D(x)] \leq \varepsilon.$$

The learning algorithm $\mathcal{A}$ is proper *if the output hypothesis h can be computed in $\mathcal{M}$.*

4 Learning Decision Trees

A *decision tree* is a rooted binary tree where every node is labeled by either an input variable or a constant. If a node v is labeled by an input variable, then it must have two children, each corresponding to the value of the input variable labeling v. The tree computes a function by taking a walk on it according to the input, starting from the root and ending at a constant node. The size of a decision tree is defined to be the number of nodes in it.

For an increasing function $k : \mathbb{N} \mapsto \mathbb{N}$ we denote by DT$(k(n))$ the computational model which includes all decision trees on n variables and of size at most $k(n)$. We now formally define the natural simplification rule for decision trees described earlier.

Definition 4.1 (Decision Tree Simplification)**.** *Let T be a decision tree and let $\rho = (z, L)$ be a restriction. The simplification rule $\mathcal{S}_{DT}$ is specified by the following procedure: Suppose the root of the tree is labeled by x_i and has two subtrees T_0, T_1. If $i \in L$, set $T|_\rho$ as a new tree with x_i at the root and with subtrees $T_0|_\rho, T_1|_\rho$. Otherwise, define $T|_\rho$ as $T_{z_i}|_\rho$ recursively. If the root of the tree is labeled by a constant, we leave it as it is.*

After defining simplification for decision trees, we can formally state the learning algorithm.

Theorem 4.2. *There is a proper (ε, δ)-PAC$_{RA}$ learning algorithm $\mathcal{A}$ for the class* DT$(k(n))$ *with simplification rule $\mathcal{S}_{DT}$ and set distribution Ind_μ which runs in time*
$\mathsf{poly}(n, k(n), 1/\varepsilon, \log(1/\delta), 1/\mu)$.

The theorem states that we can efficiently PAC-learn decision trees using restrictions, when the underlying distribution on sets is Ind_μ. The same is true even if instead of $\mathsf{Ind}_\mu(n)$ we use any pairwise independent distribution on n-bits and marginal μ. We will prove this theorem below after the following short section which discusses decision tree reconstruction and serves as a warmup to the PAC-learning proof.

4.1 Warmup: Reconstruction of Decision Trees

We now show how to reconstruct a decision tree from restrictions that leave only two variables alive. This easy result demonstrates the versatility of our model and demonstrates certain simple techniques that can be applied in other cases too. A new notion we need to define is that of a *reconstruction* algorithm that uses restrictions. For simplicity we only define this notion only for decision trees.

Definition 4.3 (Reconstruction Algorithm)**.** *We say that an algorithm $\mathcal{A}$ has* restriction query access *to a decision tree T if $\mathcal{A}$ can pick a restriction $\rho = (z, L)$ and obtain $T|_\rho$. An algorithm $\mathcal{A}$ is a* width-w reconstruction algorithm *for decision trees if, given restriction query access, $\mathcal{A}$ reconstructs T (up to isomorphism) using restriction $\rho = (z, L)$ with $|L| = w$.*

Theorem 4.4. *There is a polynomial-time deterministic width-2 reconstruction algorithm for decision trees.*

Proof. For every node v in T, there is a directed path $\gamma^{(v)}$ from the root of T to v. This path is labeled by a sequence of variables which we denote by

$$X^{(v)} = (X_0^{(v)}, \ldots, X_{d_v-1}^{(v)}),$$

where $X_0^{(v)}$ is labeling the root of T and d_v is the *depth* of v (the label of v is not part of $X^{(v)}$). It also corresponds to a boolean assignment to the variables $X^{(v)}$ which we denote by $a^{(v)}$. The assignment $a^{(v)}$ defines the walk starting at the root and ending at v. The assignment for the root is empty.

The algorithm builds the (hypothesis) tree by performing breadth-first search on T, extending each path ending by a variable, until reaching a leaf.

Let v be a node reached in the BFS (starting at root).

We already know $X^{(v)}$ and $a^{(v)}$. (For the root both $X^{(v)}$ and $a^{(v)}$ are empty.) Denote by T_v the subtree of T rooted at v. (By assigning $X^{(v)}$ to take the values given by $a^{(v)}$, we "get access" to T_v.) Now,

find $\ell(v)$, the label of v in T:

This is done as follows. For every $i \neq j$ in $[n]$, define the restriction $\rho_{i,j}$ by: variables in $X^{(v)}$ are set according to $a^{(v)}$, both x_i, x_j are alive, and all other variables are set to zero. Let Π be the set of variables x_i not in $X^{(v)}$ so that for all x_j not in $X^{(v)} \cup \{x_i\}$, the variable x_i is labeling the root of $T|_{\rho_{i,j}}$. Clearly, if $\ell(v) \in \{0, 1\}$ then Π is empty. On the other hand, if $\ell(v) = x_i$ then x_i is in Π. Furthermore, if $\ell(v) = x_i$ then every $x_j \neq x_i$ is not in Π as seen by $\rho_{i,j}$. Finally, if Π is empty then the root of, say, $T|_{\rho_{1,2}}$ is labelled by $\ell(v)$.

□

4.2 Learning Decision Trees

The PAC-learning algorithm will follow the intuition of the reconstruction algorithm above. The random restrictions will leave (with very high probability) at least two live variables, so in principle we can use the reconstruction algorithm. The main difference is that the input distribution may be unlikely to follow some paths in the tree all the way to the leaves, and these parts cannot be reconstructed. However, for PAC- learning we don't need to discover them anyway. The formal details follow.

Proof of Theorem 4.2. We will make repeated use of the Chernoff-Hoeffding bound.

Lemma 4.5. *If $r_1, \ldots, r_m$ are independent random variables distributed like r so that almost surely $|r - \mathbb{E}\, r| \leq B$, then for every $\gamma > 0$, $\mathbb{P}\left(\left|\sum_{s\in[m]} r_t/m - \mathbb{E}\, r\right| \geq \gamma\right) \leq 2\exp(-2\gamma^2 m/B^2)$.*

When dealing with t-wise independence, we use the following well-known estimate (see, e.g, [BR94]).

Lemma 4.6. *Let $m, h \in \mathbb{N}$ with $h \geq 4$ even. Let $z_1, \ldots, z_m$ be h-wise independent zero-one random variables, and let $z = \sum_{\ell \in [m]} z_\ell$. Then for every $\gamma > 0$, $\mathbb{P}[|z - \mathbb{E}\, z| \geq \gamma] < 2\left(\frac{mh}{\gamma^2}\right)^{h/2}$.*

Let ν be a distribution on $\{0,1\}^n$ and let $\mu \in (0,1)$. Let x be chosen according to ν and ρ defined using x and $\mathsf{Ind}(\mu)$ as described above. Our goal is to PAC-learn the decision tree T of size k given random queries of the form $(x, \rho, T|_\rho)$.

The algorithm builds the hypothesis tree h, which with high probability will be a subtree of T, containing all paths that are followed with probability at least ε/k under the (unknown) input distribution ν.

For every node v in T, define $\gamma^{(v)}, X^{(v)}, a^{(v)}$ as in the reconstruction algorithm above. Perform breadth-first search on T, extending each path ending by a variable, until reaching a leaf. Let v be any node in T reached in the search (which starts at the root). For the root both $X^{(v)}$ and $a^{(v)}$ are empty. So, assume that we already know $X^{(v)}$ and $a^{(v)}$. We first figure out if the node v is useful in terms of PAC-learning. Define the event

$$E_v = \{x \text{ agrees with } a^{(v)} \text{ on } X^{(v)}\}.$$

Call v *useful* if

$$\mathbb{P}_x\left[E_v\right] \geq \varepsilon/k.$$

The root is always useful.

Check whether v is useful,

using Lemma 4.5 by making $\mathsf{poly}(k/\varepsilon, \log(1/\delta))$ queries. The probability of failing this check is at most δ.

If v is not useful, set v to be a leaf labeled 0 in the hypothesis h.

If v is useful, we can condition on the event E_v without changing the running time and query complexity by much.

If v is useful, find out its label according to the following three cases:

Denote by T_v the subtree of T rooted at v. Conditioned on E_v, we can get access to $T_v|_\rho$. Make $S = \mathsf{poly}(\log(k/\delta), 1/\mu)$ queries, conditioned on E_v, and let R_v be the set of variables x_j so that (i) x_j appeared as the label of the root of $T_v|_\rho$ in at least one of these queries and (ii) whenever x_j appeared it was as the label of the root of $T_v|_\rho$.

(a) If $|R_v| = 0$, set the label of v to be the value of T_v on the first query made.

We claim that with probability $1 - \delta$, if $|R_v| = 0$ then T_v computes a constant function, specifically, it has only one node. To prove this it suffices to show that if v's label is a variable x_j, then the probability that x_j is not in R_v is at most δ. Indeed, if v's label is x_j, then the probability that x_j appears as the root of $T_v|_\rho$ is μ. In addition, whenever x_j appears in $T_v|_\rho$ it appears as v's label. Therefore, if x_j is v's label, the probability that $|R_v| = 0$ is at most $(1-\mu)^S \leq \delta$.

(b) If $|R_v| > 1$, abort and output $h \equiv 0$.

We claim that this happens with probability at most δ. When v's label is a constant, the size of R_v is always zero. We have argued that, otherwise, v's label is in R_v with high probability. Assume that v's label is $x(v)$. By the union bound, it suffices to show that for every $x_j \neq x(v)$, the probability that x_j is in R_v is at most δ/k. Indeed, conditioned on $\{x_j \text{ is in } T_v|_\rho\}$, the probability that $x(v)$ is in $T_v|_\rho$ is μ (here we use pair-wise independence). The probability that $x_j \in R_v$ is thus at most $(1-\mu)^S \leq \delta/k$.

Finally,

$$\boxed{\text{(c) If } |R_v| = 1, \text{ set } v\text{'s label to be the single variable } x(v) \text{ in } R_v.}$$

After finding out v's label, do the following.

$$\boxed{\text{If } v \text{ is labeled by a constant } b, \text{ set } v \text{ to be a leaf of } h \text{ labeled } b,}$$

and

$$\boxed{\text{otherwise, set } X^{(v)}_{d_v} = x(v) \text{ and continue the breadth-first search to construct } h.}$$

Concluding By the union bound, the total probability of making a wrong decision is at most $O(\delta k)$, so, by rescaling δ, the probability of error is as claimed. Assuming all decisions are correct, the hypothesis h is an ε-approximation to T. Indeed, denote by V the set of nodes that the algorithm called not useful. Thus,

$$\mathbb{P}_x[h(x) \neq T(x)] \leq \sum_{v \in V} \mathbb{P}_x[E_v] \leq k \cdot \varepsilon/k = \varepsilon.$$

□

Comment The algorithm performs well regardless of the labels at the leaves of the tree.

5 Learning DNF Formulas using Population Recovery

We move on to learning DNF formulas using restriction access. This will be quite a bit more involved. We start by formally defining the model and the result. In subsection 5.1 we define a new learning problem, Population Recovery, which will be key to learning DNFs, but is also intersting in its own right. Then in subsection 5.2 we show how to learn DNFs using the population recovery solver (to be described in the next section).

A DNF formula is a boolean formula of the form $F = \bigvee_{i \in [k]} T_i$ where each term T_i is an AND of several literals (a variable or its negation). DNF formulas are boolean formulas of depth 2.

In our definition of DNFs, we allow terms to be repeated (for example, T_1 may be equal to T_2) although there is no computational importance to this fact. However, the order of the terms is not given, e.g., T_1 is not before nor after T_2. In other words, a DNF formula can be thought of as a multi-set of terms.

For an increasing function $k : \mathbb{N} \mapsto \mathbb{N}$ we denote by $\text{DNF}(k(n))$ the computational model which includes all DNF formulas with n variables and at most $k(n)$ clauses.

A simplification rule can be defined for DNF formulas in the following way.

Definition 5.1 (DNF Formula Simplification)**.** *Let F be a DNF formula and let $\rho = (z, L)$ be a restriction. The simplification rule $\mathcal{S}_{DNF}$ is defined using as follows: Write*

$$F = \bigvee_{i \in [k]} T_i$$

with every term *T_i of the form*

$$T_i = \bigwedge_{j \in S_i} \ell_{i,j},$$

where $S_i \subseteq [n]$ and every $\ell_{i,j}$ is a literal. For a restriction $\rho = (z, L)$, define $F|_\rho$ as follows. For a literal $\ell = x_j^e$ with $e \in \{-1, 1\}$ (-1 denotes negation), define

$$\ell|_\rho = \begin{cases} \ell & \text{if } j \in L, \\ z_j^e & \text{if } j \notin L. \end{cases}$$

For a term $T = \bigwedge_{j \in S} \ell_j$, define

$$T|_\rho = \begin{cases} 0 & \text{if there exists } j \in S \text{ so that } \ell_j|_\rho = 0, \\ 1 & \text{if for all } j \in S \text{ we have } \ell_j|_\rho = 1, \\ \bigwedge_{j \in L} \ell_j|_\rho & \text{otherwise.} \end{cases}$$

Define

$$F|_\rho = \begin{cases} 1 & \text{if there exists } i \text{ so that } T_i|_\rho = 1, \\ 0 & \text{if for all } i \text{ we have } T_i|_\rho = 0, \\ \bigvee_{i : T_i|_\rho \neq 0} T_i|_\rho & \text{otherwise.} \end{cases}$$

Observe that in $F|_\rho$ there could be several $T_i|_\rho$ that are the same, even if all terms of F are different. Such terms are provided with repetitions. Also observe that given ρ, x and $F|_\rho$ we can efficiently compute $F(x)$ where x is the input defining ρ.

The following theorem says that we can learn DNF formulas from restrictions as long as a (sufficiently large) constant fraction of the variables are "alive."

Theorem 5.2. *For all $\mu \geq 0.365$, there is a proper (ε, δ)-PAC$_{RA}$ learning algorithm $\mathcal{A}$ for the class* DNF$(k(n))$ *with simplification rule $\mathcal{S}_{DNF}$ and set distribution Ind_μ which runs in time* $\mathsf{poly}(n, k(n), 1/\varepsilon, \log(1/\delta))$.

Similar to the case of decision trees, the algorithm works even when the underlying distribution on sets is only t-wise independent with $t \geq C \log(nk(n)/\varepsilon)$, $C > 0$ a universal constant.

The proof of the above theorem, which is given below, is by reduction to the *population recovery* problem.

5.1 Population Recovery

We start with a rough description of the learning scenario of the population recovery problem (PRP). There is some unknown population of k individuals numbered 1 to k. Each individual $i \in [k]$ has a set of attributes encoded as a vector of values $M_i \in \Sigma^n$, where Σ is some finite alphabet. There is some (unknown) distribution π on the population. The learner has access to random and partial samples from the population. Each sample is obtained by choosing a random individual according to π, and then randomly obliterating some of the attributes of that individual. The leaner's goal is to reconstruct the population and all of its attributes.

Definition 5.3 (Population Recovery Sampler)**.** *Let M be a $k \times n$ matrix over an alphabet Σ. Let $0 < \mu < 1$ and π be a probability distribution over k. Define $\mathsf{Sampler}_{PRP}(M, \mu, \pi)$ as the following procedure that samples restrictions of M. First, pick a row R of M according to π, namely, R is row i of M with i distributed according to π. Second, pick J according to $\mathsf{Ind}_n(\mu)$. Third, replace all entries in R whose indices are outside J by the symbol $*$, assumed not to be in Σ. Finally, output the random vector in $(\Sigma \cup \{*\})^n$ thus obtained.*

The problem is formally defined as follows.

Definition 5.4 (PRP)**.** *The* population recovery problem *is, given a* noticeability *parameter $p > 0$ and access to samples from $\mathsf{Sampler}_{PRP}(M, \mu, \pi)$, without knowing M, μ, π or even k, output (exactly) all rows i of M for which $\pi(i) \geq p$.*

The following theorem states a solution to PRP, for some range of marginals.

Theorem 5.5. *For all $\mu > 0.365$, there is an algorithm that runs in time $\mathsf{poly}(n, k, 1/p, \log(1/\delta))$, and with probability at least $1 - \delta$ solves PRP with noticeability $p > 0$.*

The proof of the theorem actually follows by a reduction to an easier problem, the individual recovery problem (IRP), we presently define. We mention that the PRP solver actually work when the underlying distribution on J is only t-wise independent for t roughly $\log(nk)$. For more details, see the proofs below.

5.2 Learning DNF Formulas

Proof of Theorem 5.2. Let ν be an arbitrary distribution on inputs in $\{0,1\}^n$ and $\mu > 0.365$. Let $x \sim \nu$ and $\rho \sim (\nu, \mu)$. Our goal is to PAC-learn F given random queries of the form $(x, \rho, F|_\rho)$.

To learn F we shall employ the population recovery solver. As we shall see, the reduction to the population recovery works only when F does not contain *short* terms, i.e., terms with at most $Q = O(\log(k/\varepsilon)/\mu)$ literals. To overcome this, we partition the learning procedure to two parts. Write $F = F_1 \vee F_2$, where F_1 consists of all short terms in F (with at most Q literals) and F_2 consists of all long terms (with more than Q literals). In the first part we learn F_1 by brute force. This is possible as with non-negligible probability every short term appears in $F|_\rho$. In the second part, we use the fact that we already learned all short terms to reduce the learning process to that of learning just F_2. We then learn F_2 by a simple reduction to the population recovery solver.

Pre-processing We start with a pre-processing step: understanding whether F is close to being 1 or not. This can be easily done using black-box access. Make $q_0 = O(\log(1/\delta)/\varepsilon^2)$ queries and denote by $v_1, v_2, \ldots, v_{q_0} \in \{0,1\}$ the value of F on these queries. Denote $n_0 = \sum_{j \in [q_0]} v_i/q_0$. Make the following decision:

$$\boxed{\text{If } n_0 \geq 1 - \varepsilon/2, \text{ output the hypothesis } h_0 \equiv 1.}$$

By Lemma 4.5, if $\mathbb{P}_x[F(x) = 1] \leq 1 - \varepsilon$, then

$$\mathbb{P}[\text{output } h_0] = \mathbb{P}[n_0 < 1 - \varepsilon/2] \leq \delta.$$

Similarly, if $\mathbb{P}_x[F(x) = 1] \geq 1 - \varepsilon/4$, then

$$\mathbb{P}\left[\, \mathbb{P}_x[F(x) \neq h_0] \geq \varepsilon \right] \leq \delta.$$

For the rest of the proof we can thus assume that we either already output the correct answer or

$$\mathbb{P}_x[F(x) = 0] > \varepsilon/4 \tag{1}$$

(this holds with probability at least $1 - \delta$).

Learning short terms The intuition behind this part of the learning process is that for every short term T in F, with at least polynomially small probability, all the literals in T are alive (i.e. in L). Therefore, if we make polynomially many queries, with high probability, every short term will appear fully. This process may introduce some terms that are not in F, and we need to filter them out (the extra terms are restrictions of longer terms).

Make $q_1 = \mathsf{poly}(\mu^Q, k, 1/\varepsilon, \log(2/\delta))$ queries, and let I_0 be the set of terms of length at most Q that appear in these q_1 queries. The size of I_0 is at most $q_1 k$. Some of the terms in I_0 may not be part of F. We shall filter them later on. We claim that with very high probability, every term T_i in F of length at most Q is in I_0. The probability that such a term T_i appears in a single query is at least

$$\mathbb{P}_x[F(x) = 0] \cdot \mu^Q > (\varepsilon/4)\mu^Q.$$

The choice of q_1 thus implies that the probability that T_i does not appear in any of the q_1 queries is at most $(1 - (\varepsilon/4)\mu^Q)^{q_1} \leq \delta/k$. The union bound implies our claim.

We now filter all short terms that should not be in the hypothesis. This is done by black-box access. Ask another $q_1' = \mathsf{poly}(q_1, k, 1/\varepsilon, \log(1/\delta))$ queries. For every $T \in I_0$, let N_T be the number of queries among the q_1' asked so that $T(x) > F(x)$ with x from that query. Let I_1 be the set of term T in I_0 so that $N_T = 0$, and define the hypothesis h_1 as

$$h_1 = \bigvee_{T \in I_1} T.$$

If I_1 is empty, set $h_1 = 0$. Intuitively, the hypothesis h_1 should be a good approximation to F_1. If a term $T \in I_0$ admits $\mathbb{P}_x[T(x) > F(x)] \geq \varepsilon/(2q_1 k)$, then

$$\mathbb{P}[T \in I_1] \leq (1 - \varepsilon/(2q_1 k))^{q_1'} \leq \delta/(q_1 k).$$

The union bound hence implies that

$$\mathbb{P}\left[\, \mathbb{P}_x[h_1(x) > F(x)] \geq \varepsilon/4 \right] \leq \delta. \tag{2}$$

Similar to the check whether F is 1 or not, with probability at least $1 - \delta$ of success,

check whether $\mathbb{P}_x[h_1(x) < F(x)] \leq \varepsilon/2$. If this holds output h_1.

The correctness in this case follows by (2). Indeed, if $\mathbb{P}_x[h_1(x) < F(x)] \leq \varepsilon/2$ then with probability at least $1 - 2\delta$,

$$\mathbb{P}_x[h_1(x) \neq F(x)] \leq \varepsilon.$$

For the rest of the proof we can thus assume that

$$\mathbb{P}_x[h_1(x) < F(x)] > \varepsilon/2. \tag{3}$$

Learning long terms via population recovery In this part we use the population recovery solver to learn all long terms in F. As we shall explain, the population recovery solver will work only as long as all terms in F are long. Intuitively, as we have already learned all short terms in F, we will be able to assume that F contains only long terms. This, in turn, will enable us to reduce the learning task to population recovery. The main helpful observation behind this reduction is that, with very high probability, all long terms are alive (i.e., have at least one live literal in them).

The matrix M underlying the population recovery consists of encodings of terms in F and a special row of $\perp$ symbols. Recall $F = \bigvee_{i\in[k]} T_i$. For every term T_i in F, there is a row $M_i \in \{-1,0,1\}^n$ that encodes T_i defined by $M_{i,j} = e_{i,j}$ if $j \in S_i$ where $T_i = \bigwedge_{j\in S_i} x_j^{e_{i,j}}$, and $M_{i,j} = 0$ otherwise. The row M_{k+1} encodes "skip" and is defined as $(\perp, \perp, \ldots, \perp)$.

We now implicitly explain how the probabilities $\pi(1), \ldots, \pi(k+1)$ are defined. Let $(x, \rho, F|_\rho)$ be a query. If either $h_1(x) \neq 0$ or $F(x) \neq 1$, then do not produce any sample to the population recovery solver. Otherwise, $h_1(x) = 0$ and $F(x) = 1$. If $F|_\rho$ contains a literal ($F|_\rho \neq 1$), then choose I uniformly at random from all $i \in [k]$ so that $T_i|_\rho \neq 1$, and produce as a sample to population recovery the encoding of $T_I|_\rho$, that is, if $T_I|_\rho = \bigwedge_{j\in S} x_j^{e_j}$, then produce the sample v defined by for all $j \in [n]$,

$$v_j = \begin{cases} e_j & \text{if } j \in S, \\ * & \text{if } j \notin L, \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

(observe that L is known from ρ). Otherwise, produce "skip", that is, the sample v defined by for all $j \in [n]$,

$$v_j = \begin{cases} \perp & \text{if } j \in L, \\ * & \text{if } j \notin L. \end{cases}$$

Summarizing, the queries produced for the population recovery are (roughly) generated by

$$\boxed{\text{condition on } \{h_1(x) = 0, F(x) = 1, F|_\rho \neq 1\} \text{ and produce noisy encoding of } T_I|_\rho.}$$

We conditioned on the event

$$\mathcal{E} = \{h_1(x) = 0, F(x) = 1\}.$$

Inequality (3) tells us that the probability of $\mathcal{E}$ is at least $\varepsilon/2$, and so this conditioning does not affect the running time and query complexity by much.

What are the $\pi(i)$'s? Denote by σ_x the set of terms T_i in F so that $T_i(x) = 1$. Observe that if $F|_\rho \neq 1$ for some x that agrees with ρ, then every $T_i \in \sigma_x$ has a living variables (since otherwise $T_i|_\rho = 1$ and so $F|_\rho = 1$). For a term T_i in F, we thus have

$$\pi(i) = \sum_{\ell\in[k]} \frac{1}{\ell} \mathbb{P}_{x,\rho}\left[F|_\rho \neq 1, |\sigma_x| = \ell, T_i \in \sigma_x \middle| \mathcal{E}\right].$$

This shows that the $*$ entries may not be independent of the choice of the terms we see (since the event $\{F|_\rho \neq 1\}$ depends on the positions of the $*$'s). Our goal is to show that, nevertheless, the $*$ entries are close to being independent of these terms. If this is the case, the population recovery algorithm will serve its purpose.

Recall that (with probability $1-\delta$) h_1 contains all clauses of length at most Q in F. For every fixed $x \in \mathcal{E}$, therefore, every term in σ_x is of length more than Q. So, even for every fixed $x \in \mathcal{E}$,

$$\mathbb{P}_\rho \left[F|_\rho = 1 \middle| x \right] \leq k(1-\mu)^Q \leq \mathsf{poly}(1/k, \varepsilon), \tag{5}$$

for a small polynomial to be determined (this inequality determines Q). Specifically,

$$\left| \pi(i) - \sum_{\ell \in [k]} \frac{1}{\ell} \mathbb{P}\left[|\sigma_x| = \ell, T_i \in \sigma_x \middle| \mathcal{E} \right] \right| \leq \mathsf{poly}(1/k, \varepsilon).$$

The events $\{|\sigma_x| = \ell, T_i \in \sigma_x\}$ and $\mathcal{E}$ are measurable with respect to x, and so are independent of $\{j : \rho_j = *\}$. The joint distribution of the row of M chosen and the set $\{j : \rho_j = *\}$, is hence statistically close to being independent. This statistical distance can be chosen to be at most $p/2 := \varepsilon/(16k^2)$, for small enough $\mathsf{poly}(1/k, \varepsilon)$ above.

Run the population recovery solver with noticeability $p = \varepsilon/(8k^2)$ and probability of error δ.

Theorem 5.5 implies that (with probability at least $1-\delta$) the PRP solver outputs all rows i of M with $\pi(i) \geq p$, as long as the distribution of the $*$ entries is independent from the distribution on rows of M. As discussed above, the $*$ entries are not independent of the row chosen, so the theorem as stated does not hold in this case. However, the distribution of samples produced to the PRP solver are statistically close to satisfying the assumptions required for the PRP solver to perform well. It can be verified that the population recovery solver performs well, even under this more general conditions.

Denote by I_2 the set of terms the population recovery solver produced as output, and denote

$$h_2 = \bigvee_{T \in I_2} T.$$

If I_2 is empty, set $h_2 = 0$.

Output $h = h_1 \vee h_2$.

By correctness of the PRP solver, every term in h_2 is also in F, and so $h_2 < F$ as functions. Equation (3) thus implies

$$\mathbb{P}_x[h(x) > F(x)] \leq \varepsilon/2.$$

It remains to show that h is almost always not smaller than F. With probability at least $1-\delta$, every term in F not in I_2 satisfies $p_i < 2p$. Every such term thus satisfies, using (5),

$$\begin{aligned} 2p &> \sum_{\ell \in [k]} \frac{1}{\ell} \mathbb{P}_{x,\rho}\left[T_i \in \sigma_x, F|_\rho \neq 1, |\sigma_x| = \ell \middle| \mathcal{E} \right] \\ &\geq \sum_{\ell \in [k]} \frac{1}{2k} \mathbb{P}_x\left[T_i \in \sigma_x, |\sigma_x| = \ell \middle| \mathcal{E} \right] \\ &= \frac{1}{2k} \mathbb{P}_x\left[T_i(x) = 1 \middle| \mathcal{E} \right]. \end{aligned}$$

In other words, for every such T_i we have

$$\mathbb{P}\left[T_i(x) = 1 \middle| h_1(x) < F(x) \right] = \mathbb{P}\left[T_i(x) = 1 \middle| \mathcal{E} \right] < \varepsilon/(2k).$$

The union bound thus implies

$$\mathbb{P}\left[h(x) < F(x)\right] \leq \mathbb{P}\left[h_2(x) < F(x) \middle| h_1(x) < F(x)\right] \leq \varepsilon/2.$$

□

6 Solving PRP using Robust Local Inverses

In this section we describe the main technical contribution of this work: A solution to a the population recovery problem (PRP) for some range of parameters. A key ingredient of the proof is a construction of a robust local inverse for some matrix, as we explain below. The plan of this section is as follows.

In subsection 6.1 we reduce PRP to a simpler problem, called Individual Recovery (IRP) - determining the probability of a fixed, known row of the matrix. In subsection 6.2 we define robust local inverses, and state their existence (and properties) for a certain matrix we will use. These inverses are used in turn to solve IRP in subsection 6.3. Finally we show how to construct these robust local inverses in subsection 6.4.

6.1 Individual Recovery

We now describe a (somewhat) easier version of PRP, which we call individual recovery problem (IRP). Roughly, IRP just requires to compute the π-probability of a given vector of attributes, which may or may not be in the underlying database. We consider IRP for two reasons. One is that PRP can be easily reduced to IRP. The second is that, as IRP is simpler, the IRP solver explains the main ideas of the PRP solver in a clear way.

Definition 6.1 (IRP)**.** *The* individual recovery problem (IRP) *is, given an* accuracy *parameter $p > 0$, a vector $v \in \Sigma^n$ and access to samples from $\mathsf{Sampler}_{PRP}(M, \mu, \pi)$, without knowing M, μ, π, output, up to an additive factor $p > 0$, the sum $P(v) = \sum_{i \in [k]: M_i = v} \pi(i)$.*

To solve IRP, that is, to estimate $P(v)$, we would like to just count the average number of times v occurs. But, because entries are obliterated, this task is not straightforward. Here is a simple example demonstrating some of the difficulties. Assume that M is an $n+1 \times n$ matrix obtained by adding to the identity one extra row of zeros (i.e., if $1 \leq i \leq n$ then $M_{i,j} = 1$ if $i = j$ and $M_{i,j} = 0$ otherwise, and if $i = n+1$ then $M_{i,j} = 0$ for all j). The sample $(0, *, 0, 0, *, 0, \ldots, 0)$, for example, can originate from at least three different rows in M. Assume π is uniform on $[n]$, and π' is uniform on $[n+1]$. Let S be the output of $\mathsf{Sampler}_{\mathrm{PRP}}(M, 1/2, \pi)$ and S' be the output of $\mathsf{Sampler}_{\mathrm{PRP}}(M, 1/2, \pi')$. The IRP solver with accuracy $1/n^3$ and $v = 0$ easily distinguishes between S and S'. To do so, it needs to estimate certain statistics of S and S'. The IRP algorithm does exactly that: it estimate some statistics and then use them to estimate $P(v)$.

Theorem 6.2. *For all $\mu > 0.365$, there is an algorithm that runs in time $\mathsf{poly}(1/p, \log(2/\delta))$, and with probability at least $1 - \delta$ solves IRP with accuracy $p > 0$.*

The IRP solver easily yields the PRP solver, stated in Theorem 5.5.

Proof of Theorem 5.5. The idea is to use IRP iteratively for n times. At each time $j \in [n]$, reconstruct one more *column* of M.

The algorithm starts with a short pre-processing stage: Estimate Σ by

let $\tilde{\Sigma} \subseteq \Sigma$ be the set of all alphabet symbols that appeared in $\mathsf{poly}(1/p, \log(nk/\delta))$ queries.

For a vector v in Σ^j for $j \in [n]$, denote by $P(v)$ the sum of $\pi(i)$ over all rows $i \in [k]$ that agree with v on indices in $[j]$. The algorithm shall construct sets $S_1, \ldots, S_n$. Each set S_j will contain, w.h.p., all prefixes of length j of rows in M.

For $j = 1$, run the IRP algorithm with noticeability $p/4$ to estimate $P(\sigma)$ for all σ in $\tilde{\Sigma}$, and

set S_1 to be all values σ in $\tilde{\Sigma}$ for which the IRP algorithm says $P(\sigma) > p/2$.

For $j > 1$, we thus obtained a set S_{j-1}.

Define the set S'_j to be the concatenation of every vector v in S_{j-1} and an element of $\tilde{\Sigma}$.

The size of S'_j is $|S_{j-1}| \cdot |\tilde{\Sigma}| \leq k^2 n$. Run the IRP algorithm with noticeability $p/4$ on every v in S'_j to estimate $P(v)$, and

define S_j to be the set of v in S'_j so that $P(v) > p/2$.

Finally,

output S_n.

Running time: The number of applications of the IRP algorithm is at most $k^2 n^2$ times. So, the total running time is at most $\mathsf{poly}(n, k, 1/p, \log(1/\delta))$ as stated.

Probability of error: Error can occur in one of two places: in the pre-processing stage and in one of the applications of the IRP solver.

The size of $\tilde{\Sigma}$ is at most nk. For every alphabet symbol σ in Σ that appears in a row i of M with $p_i \geq p$, the probability that σ does not appear in all the S queries is at most $(1 - p\mu)^S \leq \delta/(nk)$. The union bound thus implies that the probability that there is an alphabet symbol that we missed is at most δ.

At each application of IRP, the probability of making an error is δ. By the union bound, the total error is thus at most $k^2 n^2 \delta$.

By scaling δ appropriately, the total error of PRP is at most δ (and the change in the running time is too small to matter). □

An important ingredient of the IRP solver is a robust local inverse of a specific matrix. Before describing the solution to IRP, we thus define and state the main result concerning robust local inverses. Later, we describe the IRP solver, and only then prove the results for robust local inverses.

6.2 Robust Local Inverses

Roughly, a *robust local inverse of matrix M at the vector v* is a vector u so that

- the norm $\|u\|$ is not too large, and
- the norm $\|Mu - v\|$ is very small, say, at most a given ε.

When all the eigenvalues of M are bounded away from zero (more precisely when the condition number of M is sufficiently small), M has local inverses at all vectors. In our case M will be the "binomial" matrix, which has exponentially small eigenvalues, but nevertheless has robust local inverses at some vectors.

Let $\mu > 0$ and define $B = B_n(\mu)$ as the $n+1$ by $n+1$ matrix whose (i,j) entry, $i,j \in \{0,1,\ldots,n\}$, is

$$B_{i,j} = \mu^j (1-\mu)^{i-j} \binom{i}{j}$$

(where $\binom{0}{0} = 1$). We call B the *μ-binomial* matrix. The matrix B is invertible and its eigenvalues are μ^i, $i \in \{0,1,\ldots,n\}$. We consider robust local inverses for B at e_0, where $(e_0)_j$ is 1 if $j = 0$ and 0 otherwise.

Consider the unique vector $w = w_n(\mu)$ such that $Bw = e_0$. This vector is

$$w_j = (-1)^j ((1-\mu)/\mu)^j, \tag{6}$$

indeed, for every $i \in \{0,1,\ldots,n\}$,

$$(Bw)_i = (1-\mu)^i \sum_j (-1)^j \binom{i}{j} = \begin{cases} 1 & \text{if } i = 0, \\ 0 & \text{if } i \neq 0. \end{cases}$$

For $\mu \geq 1/2$, the norm of w is polynomial in n, so although B has exponentially small eigenvalues, it also has good robust local inverses at e_0. For $\mu < 1/2$, on the other hand, the norm of w is exponentially large. Nevertheless, we construct good robust local inverses at e_0 (for $\mu \geq 0.365$).

For $\varepsilon > 0$, define

$$\mathsf{BN}(\varepsilon; \mu, n) = \min\{\|u\| : \|Bu - e_0\| \leq \varepsilon\} \tag{7}$$

where $\|u\|$. The minimum is well defined, due to compactness. The discussion above implies that $\mathsf{BN}(0; \mu, n)$ is exponential in n for $\mu < 1/2$. The following theorem, proved is Section 6.4 below, states the existence of good robust local inverses.

Theorem 6.3. *For every $\mu \geq 0.365$, integer n and $\varepsilon > 0$,*

$$\mathsf{BN}(\varepsilon; \mu, n) \leq \mathsf{poly}(1/\varepsilon).$$

In fact, we construct an explicit vector $u = u(\varepsilon; \mu, n)$ so that $\|Bu - e_0\| \leq \mathsf{poly}(1/\varepsilon)$. This choice of u has the following two advantages: (1) its norm is Holder continuous in μ, that is, for every $\mu' \geq \mu$,

$$\|u(\mu) - u(\mu')\| \leq \mathsf{poly}(1/\varepsilon) \sqrt{|\mu' - \mu|},$$

and (2) it has logarithmic support, namely, $\{j : u_j \neq 0\} \subset \{0,1,\ldots,O(\log(2/\varepsilon))\}$.

In general, we expect the degree of the polynomials $\mathsf{poly}(1/\varepsilon)$ and the size of the support in the theorem to depend on μ. The reason we do not get a dependency on μ is that μ is bounded away from zero.

The two extra properties of u are advantages in the following sense. Holder continuity tells us that even if we do not know μ exactly, but rather have some good estimate μ' for μ, then $u(\mu')$ will do as good job as $u(\mu)$. Logarithmic support tells us that u in fact "works" for a wider class of

matrices: as long as the first $O(\log(1/\varepsilon))$ columns in a matrix B' are the same as B, we have that $B'u$ is close to e_0 as well.

Observe that by solving a convex program, we can approximate an optimal u given ε, μ, n. For our purposes this is not enough, and the important part is the asymptotic behavior of $\mathsf{BN}(\varepsilon; \mu, n)$ as ε tends to zero. So Theorem 6.3 is interesting even without constructing u explicitly. In particular, just the knowledge of a polynomial upper bound, even without explicit u, yields efficient population recovery solver. Unfortunately, we do not know how to use this information.

The condition $\mu \geq 0.365$ is a consequence of our methods, and we do not believe it to be tight. Our methods, in fact, imply a slightly smaller constant than 0.365, but they do not work for much smaller marginals.

We now proceed to prove Theorem 6.3. Our goal is, for $\mu \geq 0.365$ and $\varepsilon > 0$, to construct a vector u of small norm so that $\|Bu - e_0\| \leq \varepsilon$, where B is the μ-binomial matrix. We start with an example, motivating our construction. Recall the unique vector v so that $Bv = e_0$ given in (6),

$$v_j = (-1)^j((1-\mu)/\mu)^j, \quad j \in \{0, 1, \ldots, n\}.$$

This is the only vector in $\mathsf{BN}(\varepsilon; \mu, n)$ for $\varepsilon = 0$. The "problem" with this vector is that for $\mu < 1/2$, its norm is exponentially large. Our construction will be a carefully chosen "perturbation" of v.

6.3 IRP Solver

Proof of Theorem 6.2. By moving from PRP to IRP, we are left with the crux of the algorithm: estimating the contribution of a single individual to the database, that is, estimating

$$P(v) = \sum_{i \in [k] : M_i = v} \pi(i).$$

The algorithm consists of three steps: making queries, estimating the marginal μ, and estimating $P(v)$. Of the three steps, the last is the most important and complicated.

Make queries. The algorithm starts by asking S queries so that whenever it estimates some quantity it can assume that the estimate is indeed good.

$$\boxed{\text{Make } S = \mathsf{poly}(1/p, \log(2/\delta)) \text{ queries } \kappa(1), \kappa(2), \ldots, \kappa(S).}$$

Estimate the marginal Let $\alpha = \mathsf{poly}(p)$ be a small error so that the following holds (specifically, so that (9) below holds). Estimate μ via the average number of times the symbol $*$ occurs,

$$\boxed{\tilde{\mu} = \textstyle\sum_{s \in [S]} \mathbf{1}_{\{\kappa(s)_1 \neq *\}}/S.}$$

If $\tilde{\mu} < 0.365$, set $\tilde{\mu} = 0.365$. Lemma 4.5 implies that

$$\mathbb{P}[|\tilde{\mu} - \mu| > \alpha] \leq \delta. \tag{8}$$

Estimate $P(v)$ As previously described, the algorithm estimate certain natural statistics. Due to sampling errors, it is not completely straightforward, however, how to use these statistics. This use will require a robust local inverse for a matrix (which we later describe).

We are interested in the hamming distance from v. For simplicity we make the following abuse of notation. We think of a vector w with entries in $\Sigma \cup \{*\}$ also as the subset of $[n]$ of all $j \in [n]$ so that $w_j \notin \{v_j, *\}$. Specifically, $|w|$ is the number of entries j with $w_j \neq *$ in which w and v do not agree. This definition, of course, depends on the vector v.

The statistics we shall gather will be of the average number of queries that are of distances from v: For every $a \in \{0, 1, \ldots, n\}$, define Y_a as the probability that a query κ (distributed according to $\mathsf{Sampler}_{\text{PRP}}(M, \mu, \pi)$) has distance a from the vector v,

$$Y_a = \mathbb{E}\, \mathbf{1}_{\{|\kappa|=a\}}.$$

It is easy to check that $P(v)$ is a linear combination of $Y_0, \ldots, Y_n$ (more details follow). Due to sampling errors, we shall only have an estimate of the Y_a's, and so to compute $P(v)$ we need to use a linear combination of small norm. Such a linear combination is available via a robust local inverse.

In fact, we shall only use the first $A + 1$ of the Y_a's, where

$$A = \lceil C_A \log(2/p) \rceil,$$

$C_A > 0$ is a universal constant to be determined (C_A is universal only since μ is bounded away from zero). For every $a \in \{0, 1, \ldots, A\}$, compute

$$\tilde{Y}_a = \sum_{s \in [S]} \mathbf{1}_{\{|\kappa(s)|=a\}} / S.$$

Estimate $P(v)$ using a linear combination of $\tilde{Y}_a$: The linear combination is the robust local inverse $\tilde{u} = u(\varepsilon; \tilde{\mu}, t)$ given by Theorem 6.3 above with $\varepsilon = p/(10(A+1))$ and $t = \lceil 3A/\mu^2 \rceil$. Estimate $P(v)$ by

$$\boxed{\text{output } \tilde{P}(v) = \textstyle\sum_{a \in \{0,1,\ldots,A\}} \tilde{u}_a \tilde{Y}_a.}$$

The algorithm is thus defined. It remains to prove correctness. The following proposition is the basic observation behind the choices we made. The proposition says that we can find a vector u so that the inner product of u and Y is close to $P(v)$. The vector $\tilde{u}$ the algorithm uses is an estimation of the "real" u, since $\tilde{\mu}$ is only an estimate of μ.

Proposition 6.4. *Let $u = u(\varepsilon; \mu, t)$ be as given by Theorem 6.3 with $\varepsilon = p/(10(A+1))$. Then,*

$$\left| P(v) - \sum_{a \in \{0,1,\ldots,A\}} u_a Y_a \right| \leq (A+1) \left(\varepsilon + \frac{18}{\mu^2} \left(\frac{3}{4} \right)^{\frac{A}{2}} \right) \leq p/6.$$

Before proving the proposition, we show how it completes the proof. Equation (8) says that $|\tilde{\mu} - \mu|$ is, w.h.p., small. Theorem 6.3 thus implies that $\|u - \tilde{u}\|$ is, w.h.p., small as well. Formally,

with probability at least $1-\delta$,

$$\begin{aligned}&\left|\sum_{a\in\{0,1,\ldots,A\}} u_a Y_a - \sum_{a\in\{0,1,\ldots,A\}} \tilde{u}_a Y_a\right| \\ &\le \|u-\tilde{u}\|\cdot\|Y\| \\ &\le \mathsf{poly}(1/p)\sqrt{|\mu-\tilde{\mu}|}(A+1) \le p/6. \end{aligned} \tag{9}$$

Lemma 4.5 and the union bound (applied to the approximation of Y by $\tilde{Y}$) imply that with probability at least $1-\delta$,

$$\begin{aligned}&\left|\sum_{a\in\{0,1,\ldots,A\}} \tilde{u}_a Y_a - \tilde{P}(v)\right| \\ &\qquad \le \|\tilde{u}\|\cdot\|Y-\tilde{Y}\| \le \mathsf{poly}(1/p)\|Y-\tilde{Y}\| \le p/6.\end{aligned}$$

Proposition 6.4 and the triangle inequality finally imply that

$$\mathbb{P}\left[\left|P(v)-\tilde{P}(v)\right| > p/3\right] \le O(\delta),$$

which completes the proof.

Remark 6.5. *The algorithm works as long as the obliteration process is t-wise independent with $t = \lceil 3A/\mu^2 \rceil$.*

Proof of Proposition 6.4. Linearity of expectation implies that for every $a \in \{0,1,\ldots,A\}$,

$$Y_a = \mathbb{E}\,\mathbf{1}_{\{|\kappa|=a\}} = \sum_{i\in[k]} p_i\,\mathbb{P}[|M_i\cap J| = a],$$

where J is the random subset of $[n]$ with marginals μ defining κ (recall that we think of M_i as a subset of $[n]$ consisting of the entries in which M_i and v do not agree).

As long as J is t-wise independent, for every i with $|M_i| = q \le t$, we have

$$\mathbb{P}[|M_i\cap J| = a] = \mu^a(1-\mu)^{q-a}\binom{q}{a}.$$

For other i's, by Lemma 4.6, it is unlikely that $|M_i\cap J| = a$. Write

$$Y_a = \sum_{q\in\{0,1,\ldots,t\}} \mu^a(1-\mu)^{q-a}\binom{q}{a}Q_q + \sum_{r\in\{t+1,t+2,\ldots,n\}} R_r(a),$$

where

$$Q_q = \sum_{i\in[k]:|M_i|=q} p_i$$

and

$$R_r(a) = \sum_{i\in[k]:|M_i|=r} p_i\,\mathbb{P}[|M_i\cap J| = a].$$

Since $t \geq 3A/\mu^2$, we have $\mu r - A \geq 2\mu r/3$ for every $r > t$. For all $r \in \{t+1, t+2, \ldots, n\}$, Lemma 4.6 with $m = r$ and $h = A$ thus implies that for all $i \in [k]$ with $|M_i| = r$,

$$\mathbb{P}\left[\left||M_i \cap J| - \mu r\right| \geq \mu r - a\right] \leq 2\left(\frac{9rA}{4\mu^2 r^2}\right)^{\frac{A}{2}} = 2\left(\frac{9A}{4\mu^2 r}\right)^{\frac{A}{2}}.$$

So

$$\begin{aligned}
\sum_{r\in\{t+1,t+2,\ldots,n\}} R_r(a) &\leq 2\left(\frac{9A}{4\mu^2}\right)^{\frac{A}{2}} \sum_{r=t+1}^{\infty} r^{-\frac{A}{2}} \\
&\leq 2\left(\frac{9A}{4\mu^2}\right)^{\frac{A}{2}} \int_{\xi=t}^{\infty} \xi^{-\frac{A}{2}} d\xi \\
&= 2\left(\frac{9A}{4\mu^2}\right)^{\frac{A}{2}} \frac{1}{\frac{A}{2}-1} t^{1-\frac{A}{2}} \\
&\leq \frac{6A}{\mu^2(\frac{A}{2}-1)}\left(\frac{3}{4}\right)^{\frac{A}{2}} \\
&< \frac{18}{\mu^2}\left(\frac{3}{4}\right)^{\frac{A}{2}}.
\end{aligned}$$

Denote

$$Z_a = \sum_{q\in\{0,1,\ldots,t\}} \mu^a (1-\mu)^{q-a} \binom{q}{a} Q_q.$$

For every $a \in \{0, 1, \ldots, A\}$, therefore,

$$|Y_a - Z_a| \leq \frac{18}{\mu^2}\left(\frac{3}{4}\right)^{\frac{A}{2}}.$$

In matrix notation, we can write

$$Z = B^T Q,$$

where $Z = (Z_0, Z_1, \ldots, Z_A)$, $Q = (Q_0, Q_1, \ldots, Q_t)$, and B is a $t+1$ by $A+1$ binomial matrix with marginal μ. By choice of u, we know that Bu is ε-close to the vector $(1, 0, 0, \ldots, 0)$. Recall that $Q_0 = P(v)$. Thus,

$$\begin{aligned}
&\left|\sum_{a\in\{0,1,\ldots,A\}} u_a Z_a - P(v)\right| = \left|u^T Z - P(v)\right| \\
&= \left|u^T B^T Q - P(v)\right| \leq \varepsilon(A+1).
\end{aligned}$$

Finally, the triangle inequality implies

$$\left|\sum_{a\in\{0,1,\ldots,A\}} u_a Y_a - P(v)\right| \leq (A+1)\left(\varepsilon + \frac{18}{\mu^2}\left(\frac{3}{4}\right)^{\frac{A}{2}}\right).$$

□
□

6.4 Construction of Robust Local Inverses

Proof of Theorem 6.3. Let $A = A(\varepsilon)$ be an *accuracy* parameter to be determined below. Define u as follows

$$u_j = u_j(\mu) = (-1)^j((1-\mu)/\mu)^j p(j), \quad j \in \{0, 1, \ldots, n\}, \tag{10}$$

where

$$p(j) = 2^{-A} \sum_{\ell \in \{j, j+1, \ldots, n\}} \binom{A}{\ell}.$$

Roughly, the vector u is a "composition" of the "optimal" v from (6) with a $1/2$ binomial distribution over a universe of size A.

As $u_0 = 1$, we have $(Bu)_0 = 1$. It thus suffices to prove

$$\sum_{1 \le m \le n} \big((Bu)_m\big)^2 \le C \cdot c^A \tag{11}$$

with $C > 0$ and $c < 1$ universal constants. Indeed, if this inequality holds we can choose

$$A = \lceil \log_c(\varepsilon/C) \rceil$$

so that

$$\|Bu - e_0\| \le C \cdot c^A \le \varepsilon$$

and

$$\|u\| \le A(1/\mu)^A \le (1/\varepsilon)^{O(\log(2/\mu))}.$$

Before proving (11), we show that the two advantages of u indeed hold. The mean value theorem implies that for every $\nu \ge \mu$, there exists $\mu \le \sigma \le \nu$ so that

$$\begin{aligned}
&\|u(\mu) - u(\nu)\|^2 \\
&= \frac{d}{d\nu}\|u(\mu) - u(\nu)\|^2\big|_{\nu=\sigma} \cdot |\nu - \mu| \\
&= \left.\frac{\sum_{j=0}^{A} 2jp(j)\big((\frac{1-\mu}{\mu})^j - (\frac{1-\nu}{\nu})^j\big)(\frac{1-\nu}{\nu})^{j-1}}{\nu^2}\right|_{\nu=\sigma} \cdot |\nu - \mu| \\
&\le 2A^2(1/\mu)^{2A+2} \cdot |\nu - \mu|.
\end{aligned}$$

The choice of A thus implies that for every $\mu' \ge \mu$,

$$\|u(\mu) - u(\mu')\| \le (1/\varepsilon)^{O(\log(2/\mu))}\sqrt{|\mu' - \mu|}.$$

In addition, the support of u is indeed $\{0, 1, \ldots, A\}$ with $A = O(\log(2/\varepsilon))$.

We now prove (11). Using the identity

$$\sum_{j \in \{0,1,\ldots,\ell\}} (-1)^j \binom{m}{j} = (-1)^\ell \binom{m-1}{\ell},$$

we have that for every $m \in [n]$,

$$\begin{aligned}(Bu)_m &= \sum_{j\in\{0,1,\ldots,n\}} \mu^j(1-\mu)^{m-j}\binom{m}{j}u_j \\ &= (1-\mu)^m \sum_j \binom{m}{j}(-1)^j 2^{-A} \sum_{\ell\in\{j,j+1,\ldots,n\}} \binom{A}{\ell} \\ &= (1-\mu)^m \sum_{\ell\in\{0,1,\ldots,n\}} 2^{-A}\binom{A}{\ell} \sum_{j\in\{0,1,\ldots,\ell\}} \binom{m}{j}(-1)^j \\ &= (1-\mu)^m \sum_\ell (-1)^\ell 2^{-A}\binom{A}{\ell}\binom{m-1}{\ell}.\end{aligned}$$

We evaluate the sum (11) is three different regions.

Region one: $\mathbf{m < A/6}$. As $\binom{A}{\ell}$ is monotone increasing in ℓ for $\ell \le A/6$, the triangle inequality implies

$$\begin{aligned}|(Bu)_m| &= \left|(1-\mu)^m \sum_\ell (-1)^\ell 2^{-A}\binom{A}{\ell}\binom{m-1}{\ell}\right| \\ &\le \sum_\ell 2^{-A}\binom{A}{\ell}\binom{m-1}{\ell} \\ &\le 2^{-A+w} A\binom{A}{w}\end{aligned}$$

for $w = \lfloor A/6 \rfloor$. We use well-known properties of the entropy function to upper bound $\binom{A}{w}$. For a random variable r taking values in a set X, the *entropy* function is $H(r) = -\sum_{x\in X} \mathbb{P}[r = x] \log \mathbb{P}[r = x]$. For two random variables r and r', convexity implies $H(r, r') \le H(r) + H(r')$. By standard notation, $H(\beta)$ is the entropy of a zero-one random variable with expectation $0 \le \beta \le 1$. If $r = (r_1, \ldots, r_A) \in \{0,1\}^A$ is random variable distributed uniformly in the set of all vectors with w ones, then

$$\log\binom{A}{w} = H(r) \le \sum_{i\in[A]} H(r_i) \le AH(1/6). \tag{12}$$

Therefore,

$$\sum_{0\le m<A/6} \big((Bu)_m\big)^2 \le (A/6+1)A2^{-A(5/6-H(1/6))} \le C_1 \cdot c^A,$$

$C_1 > 0$ a universal constant.

Region two: $\mathbf{m > 5.96 \cdot A}$. The triangle inequality implies

$$\begin{aligned}|(Bu)_m| &= \left|(1-\mu)^m \sum_\ell (-1)^\ell 2^{-A}\binom{A}{\ell}\binom{m-1}{\ell}\right| \\ &\le (1-\mu)^m \sum_\ell 2^{-A}\binom{A}{\ell}\binom{m-1}{\ell}.\end{aligned}$$

Since $\sum_\ell 2^{-A}\binom{A}{\ell} = 1$ and the maximal value of $\binom{m-1}{\ell}$ for $\ell \le A$ is attained at $\ell = A$,

$$|(Bu)_m| \le (1-\mu)^m \binom{m-1}{A} \le (1-\mu)^m \binom{m}{w}$$

with $w = \lfloor m/5.96 \rfloor$. Entropy arguments, similar to (12), thus imply

$$|(Bu)_m| \le (1-\mu)^m 2^{mH(1/5.96)} \le \left((1-\mu)2^{H(1/5.96)}\right)^m \le c^m,$$

as $\mu \ge 0.365$. Therefore,

$$\sum_{5.96A<m\le n} \left((Bu)_m\right)^2 \le \sum_{m=A}^{\infty} c^m \le C_2 \cdot c^A,$$

$C_2 > 0$ a universal constant.

The hardest case to analyze is

case three: $\mathbf{A/6 \le m \le 5.96 \cdot A}$. In this case, we use the following well-known equalities (see, e.g., [AS64]) and the following lemma. Recall Euler's Gamma function

$$\Gamma(z) = \int_0^\infty w^{z-1} e^{-w} dw.$$

It satisfies $\Gamma(z+1) = z!$ for every integer $z \ge 1$. Stirling's approximation: $\Gamma(z) = \Theta(e^{-z} z^{z-1/2})$, for every real number $z \ge 1$.

Lemma 6.6. *For every two integers g and h that are at most n,*

$$\left| \sum_{\ell \in \{0,1,\ldots,n\}} (-1)^\ell \binom{g}{\ell}\binom{h}{\ell} \right| \le 2^{g+h} \mathcal{B}((g+1)/2, (h+1)/2),$$

where $\mathcal{B}$ is the Beta function, that is, $\mathcal{B}(z,w) = \Gamma(z)\Gamma(w)/\Gamma(z+w)$.

Before proving the lemma, we show how it completes the proof. By Lemma 6.6,

$$\begin{aligned} |(Bu)_m| &= (1-\mu)^m 2^{-A} \left| \sum_\ell (-1)^\ell \binom{A}{\ell}\binom{m-1}{\ell} \right| \\ &\le (1-\mu)^m 2^{m-1} \mathcal{B}((A+1)/2, m/2). \end{aligned} \tag{13}$$

We use Stirling's approximation to upper bound the Beta function,

$$\begin{aligned} &\mathcal{B}((A+1)/2, m/2) \le \mathcal{B}\left(\frac{A}{2}, m/2\right) \\ &\le O\left(e^{\frac{-A}{2}} \left(\frac{A}{2}\right)^{\frac{A-1}{2}} e^{-\frac{m}{2}} \left(\frac{m}{2}\right)^{\frac{m-1}{2}} e^{\frac{A+m}{2}} \left(\frac{A+m}{2}\right)^{-\frac{A+m+1}{2}} \right) \\ &\le O\left(A^{\frac{A}{2}} m^{\frac{m}{2}} (A+m)^{-\frac{A+m}{2}} \right). \end{aligned}$$

Upper bound the square of last quantity,

$$\begin{aligned} &A^A m^m (A+m)^{-(A+m)} \\ &\quad = e^{A\ln A + m\ln m - (A+m)\ln(A+m)} \\ &\quad = e^{m((A/m)\ln(A/(A+m)) + \ln(m/(A+m)))}. \end{aligned}$$

Thinking of m as $m = \xi A$,

$$\begin{aligned} A^A m^m (A+m)^{-(A+m)} &= e^{m((1/\xi)\ln(1/(1+\xi)) + \ln(\xi/(1+\xi)))} \\ &:= e^{mE(\xi)}, \end{aligned}$$

where

$$E(\xi) = (1/\xi)\ln(1/(1+\xi)) + \ln(\xi/(1+\xi)).$$

The derivative of $E(\xi)$ is positive for $\xi > 0$, as $\frac{d}{d\xi}E(\xi) = (1/\xi^2)\log(1+\xi)$. Since $m \le 5.96 \cdot A$, we can thus conclude

$$A^A m^m (A+m)^{-(A+m)} \le e^{mE(5.96)}. \tag{14}$$

Equations (13) and (14) thus imply

$$|(Bu)_m| \le C \cdot \big((1-\mu)2e^{E(5.96)/2}\big)^m \le C \cdot c^m,$$

as $\mu \ge 0.365$. Concluding,

$$\sum_{A/6 \le m \le 5.96\cdot A} \big((Bu)_m\big)^2 \le C_3 \cdot c^A,$$

$C_3 > 0$ a universal constant.

Proof of Lemma 6.6. For simplicity of notation, define

$$Q = \sum_{\ell} (-1)^\ell \binom{g}{\ell}\binom{h}{\ell},$$

the quantity we wish to estimate. As $\int_{z=0}^1 e^{2\pi i \alpha z} dz = \mathbf{1}_{\{\alpha=0\}}$ (in words, the integral is one if alpha is zero, and the integral is zero otherwise), here $i = \sqrt{-1}$,

$$\begin{aligned} Q &= \sum_{x\in\{0,1\}^g, y\in\{0,1\}^h} (-1)^{|x|} \mathbf{1}_{\{|x|=|y|\}} \\ &= \sum_{x\in\{0,1\}^g, y\in\{0,1\}^h} (-1)^{|x|} \int_{z=0}^1 e^{2\pi i(|x|-|y|)z} dz, \end{aligned}$$

where $|x|$ is the number of ones in x. Changing order of summation,

$$Q = \int_{z=0}^1 \sum_{x\in\{0,1\}^g, y\in\{0,1\}^h} (-1)^{|x|} e^{2\pi i(|x|-|y|)z} dz.$$

For every fixed z, the inner sum can be written as a product,

$$Q = \int_{z=0}^{1} \prod_{\ell\in[g]} \Big(\sum_{x_\ell\in\{0,1\}} (-1)^{x_\ell} e^{2\pi i z x_\ell} \Big) \cdot \prod_{j\in[h]} \Big(\sum_{y_j\in\{0,1\}} e^{-2\pi i z y_j} \Big) dz$$
$$= \int_{z=0}^{1} \Big(1 - e^{2\pi i z}\Big)^g \Big(1 + e^{-2\pi i z}\Big)^h dz.$$

Multiplying and dividing by the same number,

$$Q = \int_{z=0}^{1} \Big(e^{-\pi i z} - e^{\pi i z}\Big)^g \Big(e^{\pi i z} + e^{-\pi i z}\Big)^h e^{\pi i z(g-h)} dz$$
$$= \int_{z=0}^{1} \big(-2i\sin(\pi z)\big)^g \big(2\cos(\pi z)\big)^h e^{\pi i z(g-h)} dz.$$

Applying the triangle inequality,

$$|Q| \le 2^{g+h} \int_{z=0}^{1} \big(\sin(\pi z)\big)^g \big(\cos(\pi z)\big)^h dz.$$

By symmetry,

$$|Q| \le 2^{g+h+1} \int_{z=0}^{1/2} \big(\sin(\pi z)\big)^g \big(\cos(\pi z)\big)^h dz.$$

Renaming $w = \sin^2(\pi z)$, since $dw/dz = 2\sin(\pi z)\cos(\pi z)\pi = 2\pi\sqrt{w(1-w)}$,

$$|Q| \le 2^{g+h+1} \int_{w=0}^{1} w^{g/2}(1-w)^{h/2} \frac{dw}{2\pi\sqrt{w(1-w)}}$$
$$\le 2^{g+h} \int_{w=0}^{1} w^{(g-1)/2}(1-w)^{(h-1)/2} dw$$

$$= 2^{g+h}\mathcal{B}((g+1)/2, (h+1)/2).$$

The last equality is by definition, see [AS64]. □

□

7 Conclusions and Open Problems

In this paper, we introduced a new model of access to computational devices, which could potentially lead to good mathematical models for many situations.

We explored the problems of learning decision tree and DNF formulas in this model, showing how to get algorithms for settings where algorithms with black-box access are not known.

One question left open by our work is whether DNF formulas can be learned in our model when the fraction μ of *live* variables is allowed to be an arbitrarily small constant. Our algorithm currently requires $\mu > 0.365$. Specifically, it is left open if PRP can be solved (or robust local inverses for the binomial matrix can be constructed) for arbitrary μ.

References

[AACW06] Dana Angluin, James Aspnes, Jiang Chen, and Yinghua Wu. Learning a circuit by injecting values. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, STOC '06, pages 584–593, New York, NY, USA, 2006. ACM.

[AHK93] Dana Angluin, Lisa Hellerstein, and Marek Karpinski. Learning read-once formulas with queries. *J. ACM*, 40(1):185–210, 1993.

[Ajt83] Mikls Ajtai. Σ_1^1-formulae on finite structures. *Annals of Pure and Applied Logic*, 24:1–48, 1983.

[Alo06] Uri Alon. *An Introduction to Systems Biology: Design Principles of Biological Circuits*. CRCpress, 2006.

[ARR03] Dakshi Agrawal, Josyula R. Rao, and Pankaj Rohatgi. Multi-channel attacks. *Lecture Notes in Computer Science*, 2779:2–16, 2003.

[AS64] Milton Abramowitz and Irene A. Stegun, editors. *Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables*, volume 55 of *Applied Mathematics Series*. National Bureau of Standards, Washington, D.C., 1964. Reprinted by Dover, New York.

[ASSU81] Alfred V. Aho, Yehoshua Sagiv, Thomas G. Szymanski, and Jeffrey D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM J. Comput*, 10(3):405–421, 1981.

[BF01] Dan Boneh and Matthew Franklin. Identity-based encryption from the Weil Pairing. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO ' 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 2001.

[BP98] Paul Beame and Toniann Pitassi. Propositional proof complexity: Past, present, future. *Bulletin of the EATCS*, 65:66–89, June 1998.

[BR94] Mihir Bellare and John Rompel. Randomness-efficient oblivious sampling. In *35th Annual Symposium on Foundations of Computer Science*, pages 276–287, Santa Fe, New Mexico, 20–22 November 1994. IEEE.

[EH89] Andrzej Ehrenfeucht and David Haussler. Learning decision trees from random examples. *Information and Computation*, 82(3):231–246, 1989.

[FSS84] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

[GKS03] S. A. Goldman, S. S. Kwek, and S. D. Scott. Learning from examples with unspecified attribute values. *Information and Computation*, 180(2):82–100, 2003.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. *Lecture Notes in Computer Science*, 2162:251–261, 2001.

[HKW99] Monika Rauch Henzinger, Valerie King, and Tandy Warnow. Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. *Algorithmica*, 24(1):1–13, 1999.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. *Lecture Notes in Computer Science*, 1109:104–113, 1996.

[KS04] A. Klivans and R. Servedio. Learning dnf in time $2^{O(n^{1/3})}$. *Journal of Computer and System Sciences*, 68(2), 2004.

[KV94] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM*, 41(1):67–95, 1994.

[Nec64] È.I. Neciporuk. Synthesis of circuits from threshold logic elements. *Soviet Mathematics — Doklady*, 5(1):163–166, 1964.

[QS01] Jean-Jacques Quisquater and David Samyde. Electro magnetic analysis (EMA): Measures and counter-measures for smart cards. *Lecture Notes in Computer Science*, 2140:200–210, 2001.

[Sub61] B. A. Subbotovskaya. Realizations of linear functions by formulas using +,*,-. *Soviet Mathematics — Doklady*, 1961.

[Val84] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.