Cryptographic Limitations on Learning Boolean Formulae and Finite Automata

MICHAEL KEARNS

AT&T Bell Laboratories, Murray Hill, New Jersey

AND

LESLIE VALIANT

Harvard University, Cambridge, Massachusetts

Abstract. In this paper, we prove the intractability of learning several classes of Boolean functions in the distribution-free model (also called the Probably Approximately Correct or PAC model) of learning from examples. These results are *representation independent*, in that they hold regardless of the syntactic form in which the learner chooses to represent its hypotheses.

Our methods reduce the problems of cracking a number of well-known public-key cryptosystems to the learning problems. We prove that a polynomial-time learning algorithm for Boolean formulae, deterministic finite automata or constant-depth threshold circuits would have dramatic consequences for cryptography and number theory. In particular, such an algorithm could be used to break the RSA cryptosystem, factor Blum integers (composite numbers equivalent to 3 modulo 4), and detect quadratic residues. The results hold even if the learning algorithm is only required to obtain a slight advantage in prediction over random guessing. The techniques used demonstrate an interesting duality between learning and cryptography.

We also apply our results to obtain strong intractability results for approximating a generalization of graph coloring.

Categories and Subject Descriptors: E.3 [**Data Encryption**]: *public key cryptosystems*; F [Theory]; I.2.6 [**Artificial Intelligence**]: Learning—*concept learning*, *connectionism and neural nets*

General Terms: Theory

This research was conducted while M. Kearns was at Harvard University and supported by an AT&T Bell Laboratories scholarship.

L. Valiant was supported by Office of Naval Research (ONR) grant N00014-85-K-0445, National Science Foundation (NSF) grants DCR 86-06366 and CCR 89-02500, DAAL grant 03-86-K-0171, Defense Advanced Research Projects Agency (DARPA) grant AFOSR 89-0506, and by SERC.

Authors' addresses: M. Kearns, AT & T Bell Laboratories, 600 Mountain Avenue, Room 2A-423, Murray Hill, NJ 07974; L. Valiant, Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138

1. *Introduction*

In this paper, we prove *representation-independent* hardness results for the distribution-free learning of several representation classes whose efficient learnability has thus far been unresolved.[1] Informally, a representation-independent hardness result states that learning is difficult regardless of the *form* in which a learning algorithm represents its hypothesis, provided this hypothesis meets the quite reasonable constraint of being evaluatable in polynomial time (i.e., having an equivalent polynomial-size Boolean circuit). In contrast, a *representation-based* hardness result states only that learning is difficult when the hypothesis is constrained to meet some (usually strong) structural or syntactic restrictions.

We prove representation-independent hardness results for the distribution-free learning of several simple representation classes, including polynomial-size Boolean formulae, acyclic deterministic finite automata, and constant-depth threshold circuits (which may be regarded as a form of simplified neural networks). These hardness results are based on assumptions regarding the intractability of specific number-theoretic problems of interest in cryptography, namely factoring Blum integers, inverting the RSA function, and recognizing quadratic residues. Thus, a polynomial-time learning algorithm in the distribution-free model for any of the named representation classes using *any* polynomial-time evaluatable hypothesis representation would immediately yield a polynomial-time algorithm for all of these cryptographic problems, which have defied efficient solution for decades, and are widely believed to be intractable.

For practical purposes, the efficient learnability of a representation class must be considered unresolved until a polynomial-time learning algorithm is discovered or until a representation-independent hardness result is proved. This is because a representation-based result stating that the class C is not efficiently learnable by the class H (modulo some complexity-theoretic assumption such as RP $\neq$ NP) still leaves open the possibility that C is efficiently learnable by a different hypothesis class H'. Indeed, this possibility has been realized for several natural target classes: for instance, it is known that for any fixed constant natural number $k \geq 2$, the problem of learning 2-term disjunctive normal form (DNF) formulae in the distribution-free model is NP-hard if the learning algorithm is restricted to represent its hypothesis in 2-term DNF form, but there is a polynomial-time learning algorithm if we relax this restriction [Pitt and Valiant, 1988]. A similar result holds for Boolean threshold functions [Pitt and Valiant, 1988].

The only previous representation-independent hardness results for distribution-free learning follow from the elegant work of Goldreich et al. [1986] on constructing random functions. Their functions have many properties stronger than those mentioned here, but for our purposes we may state their result formally as follows: Let $\mathrm{CKT}_n^{p(n)}$ denote the class of Boolean circuits over n inputs with at most $p(n)$ gates, and let $\mathrm{CKT}^{p(n)} = \bigcup_{n \geq 1} \mathrm{CKT}_n^{p(n)}$. Then it is shown by Goldreich et al. [1986] that if there exists a one-way function, then for some polynomial $p(n)$, $\mathrm{CKT}^{p(n)}$ is not learnable in polynomial time (by *any* polynomial-time evaluatable representation class). Pitt and Warmuth [1988] then used this result to construct other hard-to-learn representation classes.

[1] The distribution-free model of learning that we use and will define shortly is often also referred to as the *probably approximately correct* or PAC model of learning.

For definitions and a discussion of one-way functions, we refer the reader to Yao [1982], Blum and Micali [1984], Levin [1985], and Goldreich et al. [1986].

Note that in any reasonable model of learning, we intuitively do not expect to find polynomial-time learning algorithms for classes of representations that are not polynomial-time evaluatable, since a learning algorithm may not even have enough time to write down a good hypothesis. More formally, Schapire [1989] has shown that any representation class that is not evaluatable in polynomial time cannot be learned in polynomial time in the distribution-free model.

Thus, we may informally interpret the result of Goldreich et al. as stating that not everything with a small (polynomial-size) circuit representation is efficiently learnable (assuming there is a one-way function). However, there is a large gap in computational power between the class of polynomial-size circuits and the classes that have been the subject of intense scrutiny within the computational learning theory community of late (e.g., DNF, decision trees, Boolean formulae, classes based on finite automata, restricted classes of circuits). In this paper, we prove hardness results similar to those of Goldreich et al., but for much less powerful representation classes, thus, clarifying the limits of efficient learnability.

The intuition behind the approach taken to obtain these results is contained in the following analogy. Consider a computer system with two users, Alice and Bob. Alice and Bob wish to communicate via an insecure channel, and it is assumed that Eve the eavesdropper is listening to this channel. We make no assumptions about Eve's behavior other than a polynomial bound on her computing resources. In this cryptographic setting, Alice and Bob wish to communicate *privately* in spite of Eve's nosy presence.

A classic solution to Alice and Bob's problem is the *one-time pad*. Here Alice and Bob would physically meet in a secure room (away from Eve) and compile a large common table of random bits. Then, after separating, Bob, to send a bit b to Alice, chooses the next random bit c from the common list and sends the bit $b \oplus c$ to Alice. It is easily verified that if the bit c is uniformly distributed then the encoded bit $b \oplus c$ is also uniformly distributed, regardless of the value of the cleartext message bit b. Thus, Eve, regardless of computation time, is probably unable to gain any information about the cleartext messages from listening to the channel between Alice and Bob. Alice, however, also knows the random bit c, and so may decode by computing $(b \oplus c) \oplus c = b$.

There are some obvious practical problems with the one-time pad. Foremost among these is the need for Alice and Bob to meet in person and compile the table of random bits; in a network of thousands of computers, having every pair of users meet clearly defeats the point of using computers in the first place. In response to complaints such as these and also more subtle security concerns, the field of *public-key cryptography* was created by Diffie and Hellman [1976].

Public-key cryptography solves the problem of Alice and Bob via the use of *trapdoor functions*. Informally, a trapdoor function is one that can be computed in polynomial time (i.e., it is easy to compute $f(x)$ on input x) but cannot be inverted in polynomial time (i.e., it is hard to compute x on input $f(x)$)—unless one is the "creator" of the function, in which case one possesses a piece of "trapdoor" information that makes inversion possible in polynomial time. Now rather than meeting with Bob in person, Alice "creates" a trapdoor function f and *publishes* a program for computing f (which reveals no information about

f^{-1}) in a directory that is available to everyone—Bob and Eve included. To send the message x to Alice, Bob simply computes $f(x)$ and sends it to Alice. Eve, seeing only $f(x)$ on the channel and not possessing the trapdoor, is unable to recover the message x in polynomial time. Alice, being the creator of f and thus having the trapdoor, can efficiently invert Bob's ciphertext and recover x.

Our approach is based on viewing Eve as a learning algorithm. Note that since a program for f is available to Eve, she may create as many pairs of the form $(f(x), x)$ that she likes simply by choosing x and then computing $f(x)$. If we set $y = f(x)$, we see that such pairs have the form $(y, f^{-1}(y))$, and can thus be regarded as "examples" of the inverse function f^{-1}. Thus, from the learning perspective, public-key cryptography assumes the existence of functions that are not learnable from examples, since if Eve could learn f^{-1} efficiently from examples of its input-output behavior, she could then decode messages sent from Bob to Alice! Furthermore, note that the inverse function f^{-1} is "simple" in the sense that it does have a small circuit (determined by the trapdoor, which Alice has access to and uses for decoding); thus, from an information-theoretic standpoint the learning problem is "fair," as opposed to the one-time pad, where there is no small circuit underlying the communication between Alice and Bob, just a large random bit table.

Thus, we see that recent developments in the theory of cryptography provide us with simple functions that are difficult to learn. Our approach in this paper is based on refining the functions provided by cryptography in an attempt to find the *simplest* functions that are difficult to learn.

The outline of the paper is as follows: In Section 2, we provide definitions for the distribution-free model of learning, adapted from Valiant [1984]. Then in Section 3, we discuss previous hardness results for learning, both of the representation-based and representation-independent type. Section 4 gives the needed definitions and background from cryptography.

In Section 5, we develop simple representation classes based on cryptographic functions and prove that learning these classes is as difficult as breaking the associated cryptosystems. In Section 6, these results are applied to prove the difficulty of learning Boolean formulae, finite automata, and threshold circuits. In Section 7, we give a generalized method for proving hardness results for learning based on *any* trapdoor function. Section 8 applies our learning results to give strong hardness results for approximating the optimal solution for various combinatorial optimization problems, including a generalization of graph coloring.

2. *Definitions for Distribution-Free Learning*

2.1. Representing Subsets of a Domain

2.1.1. *Concept Classes and Their Representation.* Let X be a set called a *domain* (also sometimes referred to as the *instance space*). We think of X as containing encodings of all objects of interest to us in our learning problem. For example, each instance in X may represent a different object in a particular room, with discrete attributes representing properties such as color, and continuous values representing properties such as height. The goal of a learning algorithm is to infer some unknown subset of X, called a *concept*, chosen from a known *concept class*.

For computational purposes, we always need a way of *naming* or *representing* concepts. Thus, we formally define a *representation class over* X to be a pair (σ, C), where $C \subseteq \{0,1\}^*$ and σ is a mapping $\sigma: C \rightarrow 2^X$ (here 2^X denotes the power set of X). In the case that the domain X has real-valued components, we sometimes assume $C \subseteq (\{0,1\} \cup R)^*$, where R is the set of real numbers. For $c \in C$, $\sigma(c)$ is called a *concept* over X; the image space $\sigma(C)$ is the *concept class* that is *represented* by (σ, C). For $c \in C$, we define $pos(c) = \sigma(c)$ (the *positive examples* of c) and $neg(c) = X - \sigma(c)$ (the *negative examples* of c). The domain X and the mapping σ will usually be clear from the context, and we simply refer to the *representation class* C. We sometimes use the notation $c(x)$ to denote the value of the characteristic function of $\sigma(c)$ on the domain point x; thus, $x \in pos(c)$ ($x \in neg(c)$, respectively) and $c(x) = 1$ ($c(x) = 0$, respectively) are used interchangeably. We assume that domain points $x \in X$ and representations $c \in C$ are efficiently encoded using any of the standard schemes (see Garey and Johnson [1979], and denote by $|x|$ and $|c|$ the length of these encodings measured in bits (or in the case of real-valued domains, some other reasonable measure of length that may depend on the model of arithmetic computation used; see Aho et al. [1974]).

2.1.2. *Parameterized Representation Classes.* In this paper, we study *parameterized* classes of representations. Here we have a stratified domain $X = \bigcup_{n \geq 1} X_n$ and representation class $C = \bigcup_{n \geq 1} C_n$. The parameter n can be regarded as an appropriate measure of the complexity of concepts in $\sigma(C)$ (such as the number of domain attributes), and we assume that for a representation $c \in C_n$ we have $pos(c) \subseteq X_n$ and $neg(c) = X_n - pos(c)$. For example, X_n may be the set $\{0,1\}^n$, and C_n the class of all Boolean formulae over n variables whose length is at most n^2. Then, for $c \in C_n$, $\sigma(c)$ would contain all satisfying assignments of the formula c.

2.1.3. *Efficient Evaluation of Representations.* In general, we are primarily concerned with learning algorithms that are computationally efficient. In order to prevent this demand from being vacuous, we need to ensure that the *hypotheses* output by a learning algorithm can be efficiently evaluated as well. Thus, if C is a representation class over X, we say that C is *polynomially evaluatable* if there is a polynomial-time *evaluation algorithm* A that on input a representation $c \in C$ and a domain point $x \in X$ outputs $c(x)$. Note that if a class C is polynomially evaluatable, then each representation $c \in C$ has an equivalent polynomial-size circuit, obtained by hard-wiring the representation input of A to be c, and converting the resulting polynomial-time algorithm (now accepting the single input $x \in X$) to a polynomial-size circuit using standard techniques. All representation classes considered here are polynomially evaluatable. It is worth mentioning at this point that Schapire [1989] has shown that if a representation class is not polynomially evaluatable, then it is not efficiently learnable in our model. Thus, perhaps not surprisingly we see that classes that are not polynomially evaluatable are not only "unfair" as learning problems but also intractable.

Samples. A *labeled example* from a domain X is a pair $\langle x, b \rangle$, where $x \in X$ and $b \in \{0,1\}$. A *labeled sample* $S = \langle x_1, b_1 \rangle, \ldots, \langle x_m, b_m \rangle$ from X is a finite sequence of labeled examples from X. If C is a representation class, a *labeled example of* $c \in C$ is a labeled example $\langle x, c(x) \rangle$, where $x \in X$. A *labeled* *sample of* c is a labeled sample S where each example of S is a labeled example of c. In the case, where all labels b_i or $c(x_i)$ are 1 (0, respectively), we may omit the labels and simply write S as a list of points $x_1, \ldots, x_m$, and we call the sample a *positive* (*negative*, respectively) sample.

We say that a representation h and an example $\langle x, b \rangle$ *agree* if $h(x) = b$; otherwise, they *disagree*. We say that a representation h and a sample S are *consistent* if h agrees with each example in S; otherwise, they are *inconsistent*.

2.2. Distribution-Free Learning

2.2.1. *Distributions on examples.* On any given execution, a learning algorithm for a representation class C will be receiving examples of a single distinguished representation $c \in C$. We call this distinguished c the *target representation*. Examples of the target representation are generated probabilistically as follows: Let D_c^+ be a fixed but arbitrary probability distribution over *pos*(c), and let D_c^- be a fixed but arbitrary probability distribution over *neg*(c). We call these distributions the *target distributions*. When learning c, learning algorithms will be given access to two oracles, *POS* and *NEG*, that behave as follows: Oracle *POS* (*NEG*, respectively) returns in unit time a positive (negative, respectively) example of the target representation, drawn randomly according to the target distribution D_c^+ (D_c^-, respectively). The distribution-free model is sometimes defined in the literature with a single target distribution over the entire domain; the learning algorithm is then given labeled examples of the target concept drawn from this distribution. These models, however, are equivalent with respect to polynomial-time computation, in the sense that any class learnable in polynomial time in one model is learnable in polynomial time in the other model, as shown by Haussler et al. [1988].

Given a fixed target representation $c \in C$, and given fixed target distributions D_c^+ and D_c^-, there is a natural measure of the *error* (with respect to c, D_c^+ and D_c^-) of a representation h from a representation class H. We define $e_c^+(h) = D_c^+(neg(h))$ (i.e., the weight of the set *neg*(h) under the probability distribution D_c^+) and $e_c^-(h) = D_c^-(pos(h))$ (the weight of the set *pos*(h) under the probability distribution D_c^-). Note that $e_c^+(h)$ (respectively, $e_c^-(h)$) is simply the probability that a random positive (respectively, negative) example of c is identified as negative (respectively, positive) by h. If both $e_c^+(h) < \epsilon$ and $e_c^-(h) < \epsilon$, then we say that h is an *ϵ-good* hypothesis (with respect to c, D_c^+, and D_c^-); otherwise, h is *ϵ-bad*. We define the *accuracy* of h to be the value $\min(1 - e_c^+(h), 1 - e_c^-(h))$.

It is worth noting that our definitions so far assume that the hypothesis h is deterministic. However, this need not be the case; for example, we can instead define $e_c^+(h)$ to be the probability that h classifies a random positive example of c as negative, where the probability is now over both the random example and the coin flips of h. All of the results presented here hold under these generalized definitions.

When the target representation c is clear from the context, we drop the subscript c and simply write D^+, D^-, e^+, and e^-.

In the definitions that follow, we demand that a learning algorithm produce with high probability an ϵ-good hypothesis regardless of the target representation and target distributions. Although at first this may seem like a strong criterion, note that the error of the hypothesis output is always measured with respect to the same target distributions on which the algorithm was trained. Thus, while it is true that certain examples of the target representation may be extremely unlikely to be generated in the training process, these same examples intuitively may be "ignored" by the hypothesis of the learning algorithm, since they contribute a negligible amount of error.

2.2.2. *Learnability.* Let C and H be representation classes over X. Then C is *learnable from examples by* H if there is a (probabilistic) algorithm A with access to *POS* and *NEG*, taking inputs ϵ, δ, with the property that for any target representation $c \in C$, for any target distributions D^+ over $pos(c)$ and D^- over $neg(c)$, and for any inputs $0 < \epsilon$, $\delta < 1$, algorithm A halts the outputs a representation $h_A \in H$ that with probability greater than $1 - \delta$ satisfies $e^+(h_A) < \epsilon$ and $e^-(h_A) < \epsilon$.

We call C the *target class* and H the *hypothesis class*; the output $h_A \in H$ is called the *hypothesis* of A. A will be called a *learning algorithm* for C. If C and H are polynomially evaluatable, and A runs in time polynomial in $1/\epsilon, 1/\delta$ then we say that C is *polynomially learnable from examples by* H; if C is parameterized, we also allow the running time of A to have polynomial dependence on the parameter n.

We drop the phrase "from examples" and simply say that C is *learnable by* H, and C is *polynomially learnable by* H. We say C is *polynomially learnable* to mean that C is polynomially learnable by H for some polynomially evaluatable H. We sometimes call ϵ the *accuracy parameter* and δ the *confidence parameter*.

Thus, we ask that for any target representation and any target distributions, a learning algorithm finds an ϵ-good hypothesis with probability at least $1 - \delta$. A primary goal of research in this model is to discover which representation classes C are polynomially learnable.

Note that in the above definitions, we allow the learning algorithm to output hypotheses from some class H that is possibly different from C, as opposed to the natural choice $C = H$. Although, in general, we assume that H is at least as powerful as C (i.e., $C \subseteq H$), we see that in some cases for computational reasons we may not wish to restrict H beyond it being polynomially evaluatable. If the algorithm produces an accurate and easily evaluated hypothesis, then our learning problem is essentially solved, and the actual form of the hypothesis is of secondary concern.

We refer to this model as the *distribution-free* model, to emphasize that we seek algorithms that work for any target distributions. It is also known in the literature as the *probably approximately correct* model. We also occasionally refer to the model as that of *strong learnability* (to mean learnability by some polynomially evaluatable representation class H), in contrast with the notion of *weak learnability* defined below.

2.2.3. *Weak Learnability.* We also consider a distribution-free model in which the hypothesis of the learning algorithm is required to perform only slightly better than random guessing.

Let C and H be representation classes over X. Then C is *weakly learnable from examples by* H if there is a polynomial p and a (probabilistic) algorithm A with access to *POS* and *NEG*, taking input δ, with the property that for any target representation $c \in C$, for any target distributions D^+ over $pos(c)$ and D^- over $neg(c)$, and for any input value $0 < \delta < 1$, algorithm A halts and outputs a representation $h_A \in H$ that with probability greater than $1 - \delta$ satisfies $e^+(h_A) < 1/2 - 1/p(n)$ and $e^-(h_A) < 1/2 - 1/p(n)$.

Thus, the accuracy of h_A must be at least $1/2 + 1/p(n)$. A will be called a *weak learning* algorithm for C. If C and H are polynomially evaluatable, and A runs in time polynomial in $1/\delta$ and $|n|$ we say that C is *polynomially weakly learnable by H* and C is *polynomially weakly learnable* if it is weakly learnable by H for some polynomially evaluatable H.

We may intuitively think of weak learning as the ability to detect some slight bias separating positive and negative examples, where the advantage gained over random guessing diminishes as the complexity of the problem grows. Our main use of the weak learning model is in proving the strongest possible hardness results.

2.3. Some Representation Classes.

We now define the representation classes whose learnability we study. In this paper, the domain X_n is always $\{0,1\}^n$ and the mapping σ simply maps each circuit to its set of satisfying assignments. The classes defined below are all parameterized; for each class, define the subclasses C_n, and then C is defined by $C = \bigcup_{n \geq 1} C_n$.

Boolean Formulae. The representation class BF_n consists of all Boolean formulae over the Boolean variables $x_1, \ldots, x_n$.

Boolean Circuits. The representation class CKT_n consists of all Boolean circuits over input variables $x_1, \ldots, x_n$.

Threshold Circuits. A *threshold gate* over input variables $x_1, \ldots, x_n$ is defined by a value $1 \leq t \leq n$ such that the gate outputs 1 if and only if at least t of the input bits are set to 1. We let TC_n denote the class of all circuits of threshold gates over $x_1, \ldots, x_n$. For constant d, $d\mathrm{TC}_n$ denotes the class of all threshold circuits in TC_n with depth at most d.

Acyclic Finite Automata. The representation class ADFA_n consists of all deterministic finite automata that accept only strings of length n, that is, all deterministic finite automata M such that the language $L(M)$ accepted by M satisfies $L(M) \subseteq \{0,1\}^n$.

We also frequently discuss computations performed by the circuit class $\mathrm{NC}^1 = \bigcup_{n \geq 1} \mathrm{NC}^1_n$, where NC^1_n is the class of circuits consisting of AND, OR, and NOT gates of fan-in two having size polynomial in n and depth logarithmic in n.

2.4. Other Definitions and Notation

2.4.1. *Chernoff Bounds.*

We shall make extensive use of the following bounds on the area under the tails of the binomial distribution. For $0 \leq p \leq 1$ and m a positive integer, let $LE(p, m, r)$ denote the probability of at most r successes in m independent trials of a Bernoulli variable with probability of success p, and let $GE(p, m, r)$ denote the probability of at least r successes. Then for $0 \leq \alpha \leq 1$,

Fact CB1. $LE(p, m, (1 - \alpha)mp) \leq \exp(-\alpha^2 mp/2)$.

Fact CB2. $GE(p, m, (1 + \alpha)mp) \leq \exp(-\alpha^2 mp/3)$.

These bounds in the form they are stated are from the paper of Angluin and Valiant [1979] and follow from Chernoff [1952]. Although we make frequent use of Fact CB1 and Fact CB2, we do so in varying levels of detail, depending on the complexity of the calculation involved. However, we are primarily interested in Chernoff bounds for the following consequence of Fact CB1 and Fact CB2: Given an event E of probability p, we can obtain an estimate $\hat{p}$ of p by drawing m points from the distribution and letting $\hat{p}$ be the frequency with which E occurs in this sample. Then, for m polynomial in $1/p$ and $1/\alpha$, $\hat{p}$ satisfies $p/2 < \hat{p} < 2p$ with probability at least $1 - \alpha$. If we also allow m to depend polynomially on $1/\beta$, we can obtain an estimate $\hat{p}$ such that $p - \beta < \hat{p} < p + \beta$ with probability at least $1 - \alpha$.

2.4.2. *Notational Conventions.* Let $E(x)$ be an event and $\psi(x)$ a random variable that depend on a parameter x that takes on values in a set X. Then, for $X' \subseteq X$, we denote by $\mathbf{Pr}_{x \in X'}[E(x)]$ the probability that E occurs when x is drawn uniformly at random from X'. Similarly, $\mathbf{E}_{x \in X'}[\psi(x)]$ is the expected value of ψ when x is drawn uniformly at random from X'. We also need to work with distributions other than the uniform distribution; thus, if P is a distribution over X, we use $\mathbf{Pr}_{x \in P}[E(x)]$ and $\mathbf{E}_{x \in P}[\psi(x)]$ to denote the probability of E and the expected value of ψ, respectively, when x is drawn according to the distribution P. When E or ψ depend on several parameters that are drawn from different distributions, we use multiple subscripts. For example, $\mathbf{Pr}_{x_1 \in P_1, x_2 \in P_2, x_3 \in P_3}[E(x_1, x_2, x_3)]$ denotes the probability of event E when x_1 is drawn from distribution P_1, x_2 from P_2, and x_3 from P_3 (all draws being independent).

3. *Previous Hardness Results for Learning*

The initial paper defining the distribution-free model [Valiant, 1984] gave the first polynomial-time learning algorithms in this model. It showed that the class of monomials is polynomially learnable, as are the classes kCNF and kDNF (with time complexity $O(n^k)$). For each of these algorithms, the hypothesis class is the same as the target class; that is, in each case, C is polynomially learnable by C.

Pitt and Valiant [1988] subsequently observed that the classes k-TERM-DNF and k-CLAUSE-CNF, when viewed as functions, are properly contained within the classes kCNF and kDNF, respectively. Combined with the results above [Valiant, 1984], this shows that for fixed k, the class k-TERM-DNF is polynomially learnable by kCNF, and the class k-CLAUSE-CNF is polynomially learnable by kDNF. More surprisingly, Pitt and Valiant prove that for any fixed $k \geq 2$, learning k-TERM-DNF by k-TERM-DNF and learning k-CLAUSE-CNF by k-CLAUSE-CNF are NP-hard problems.

These results are important in that they demonstrate the tremendous computational advantage that may be gained by a judicious change of hypothesis representation. This can be viewed as a limited but provable confirmation of the rule of thumb in artificial intelligence that *representation is important*. By moving to a more powerful hypothesis class H instead of insisting on the more "natural" choice $H = C$, we move from an NP-hard problem to a polynomial-time solution. This may be explained intuitively by the observation that while the constraint $H = C$ may be significant enough to render the learning task intractable, a richer hypothesis representation allows a greater latitude for expressing the learned formula.

In discussing hardness results, we distinguish between two types: *representation-based* hardness results and *representation-independent* hardness results. Briefly, representation-based hardness results state that for some *fixed* representation classes C and H, learning C by H is hard in some computational sense (such as NP-hardness). Thus, the aforementioned result on the difficulty of learning k-TERM-DNF by k-TERM-DNF is representation-based. In contrast, a representation-independent hardness result says that, for fixed C and *any* polynomially evaluatable H, learning C by H is hard.

Representation-based hardness results are interesting for a number of reasons. They can be used to give formal verification to the importance of hypothesis representation, and for practical reasons it is important to study the *least* expressive class H that can be used to learn C, since the choice of hypothesis representation can greatly affect resource complexity (such as the number of examples required) even for those classes already known to be polynomially learnable.

However, since a representation-based hardness result dismisses the polynomial learnability of C only with respect to the *fixed* hypothesis class H, such results leave something to be desired in the quest to classify learning problems as "easy" or "hard." For example, we may be perfectly willing to settle for an efficient algorithm learning C by H for some more expressive H if we know that learning C by C is NP-hard. Thus, for practical purposes we must regard the polynomial learnability of C as not entirely resolved until we either find an efficient learning algorithm or we prove that learning C by H is hard for *any* reasonable H, that is, until we prove a representation-independent hardness result for C.

Gold [1978] gave the first representation-based hardness results that apply to the distribution-free model of learning. He proves that the problem of finding the smallest deterministic finite automaton consistent with a given sample is NP-complete; the results of Haussler et al. [1988] can be easily applied to Gold's result to prove that learning deterministic finite automata of size n by deterministic finite automata of size n cannot be accomplished in polynomial time unless RP = NP. There are some technical issues involved in properly defining the problem of learning finite automata in the distribution-free model; see Pitt and Warmuth [1988] for details. Gold's results were improved by Li and Vazirani [1988] who show that finding an automaton 9/8 larger than the smallest consistent automaton is still NP-complete.

As we have already discussed, Pitt and Valiant [1988] prove that for $k \geq 2$, learning k-TERM-DNF by k-TERM-DNF is NP-hard by giving a randomized reduction from a generalization of the graph coloring problem. Even stronger, for $k \geq 6$, they prove that even if the hypothesis DNF formulae is allowed to have $2k - 3$ terms, k-TERM-DNF cannot be learned in polynomial time unless RP = NP. These results hold even when the target formulae are restricted to be monotone and the hypothesis formulae is allowed to be nonmonotone. Dual results hold for the problem of learning k-CLAUSE-CNF. Pitt and Valiant also prove that μ-formulae (Boolean formulae in which each variable occurs at most once, sometimes called *read-once*) cannot be learned by μ-formulae in polynomial time, and that Boolean threshold functions cannot be learned by Boolean threshold functions in polynomial time, unless RP = NP.

Pitt and Warmuth [1989] dramatically improved the results of Gold by proving that deterministic finite automata of size n cannot be learned in polynomial time by deterministic finite automata of size n^{α} for any fixed value $\alpha \geq 1$ unless RP = NP. Their results leave open the possibility of an efficient learning algorithm using deterministic finite automata whose size depends on ϵ and δ, or an algorithm using some entirely different representation of the sets accepted by automata. This possibility is addressed and dismissed (modulo cryptographic assumptions) by the results in this paper.

Hancock [1989] has shown that learning decision trees of size n by decision trees of size n cannot be done in polynomial time unless RP = NP. Representation-based hardness results for learning various classes of neural networks can also be derived from the results of Judd [1988] and Blum and Rivest [1988].

The first representation-independent hardness results for the distribution-free model follow from the work of Goldreich et al. [1986], whose true motivation was to find easy-to-compute functions whose output on random inputs appears random to all polynomial-time algorithms. A simplified and weakened statement of their result is that the class of polynomial-size Boolean circuits is not polynomially learnable by *any* polynomially evaluatable H, provided that there exists a one-way function (see Yao [1982]). Pitt and Warmuth [1988] defined a general notion of reducibility for learning and gave a number of other representation classes that are not polynomially learnable under the same assumption by giving reductions from the learning problem for polynomial-size circuits. One of the main contributions of the research presented here is representation-independent hardness results for much simpler classes than those addressed by Goldreich et al. [1986] or Pitt and Warmuth [1988], among these the classes of Boolean formulae, acyclic deterministic finite automata, and constant-depth threshold circuits.

4. *Background and Definitions from Cryptography*

4.1. Some Basic Number Theory. For an introduction to number theory that is relevant to cryptography, we refer the reader to the work of Angluin [1982] and Kranakis [1986]. For N a natural number, Z_N will denote the ring of integers modulo N, and Z_N^* will denote the multiplicative group modulo N. Thus, $Z_N = \{x\colon 0 \leq x \leq N-1\}$ and $Z_N^* = \{x\colon 1 \leq x \leq N-1$ and $gcd(x,N) = 1\}$, where $gcd(x,N)$ denotes the greatest common divisor of x and N. The *Euler quotient function* φ is defined by $\varphi(N) = | Z_N^* |$. For $x \in Z_N^*$, we say that x is a *quadratic residue* modulo N if there is an $a \in Z_N^*$ such that $x = a^2 \, mod N$. We denote by QR_N the set of all quadratic residues in Z_N^*. For a prime p and $x \in Z_p^*$, we define the *Legendre symbol* of x with respect to p by $L(x,p) = 1$ if x is a quadratic residue modulo p, and $L(x,p) = -1$ otherwise. For $N = p \cdot q$, where p and q are prime, we define the *Jacobi symbol* of $x \in Z_N^*$ with respect to N by $J(x,N) = L(x,p) \cdot L(x,q)$. Since x is a quadratic residue modulo N if and only if it is a quadratic residue modulo p and modulo q, it follows that $J(x,N) = -1$ implies that x is not a quadratic residue modulo N. However, $J(x,N) = 1$ does not necessarily imply that x is a quadratic residue mod N. For any integer N, we define the set $Z_N^*(+1) = \{x \in Z_N^*\colon J(x,N) = 1\}$. A *Blum integer* is an integer of the form $p \cdot q$, where p and q are primes both congruent to 3 modulo 4.

We make use of the following facts from number theory:

Fact NT1. On inputs x and N, $gcd(x,N)$ can be computed in polynomial time.

Fact NT2. For p a prime and $x \in Z_p^*$, $L(x, p) = x^{(p-1)/2} \, mod p$.
Fact NT3. On inputs x and N, $J(x, N)$ can be computed in polynomial time.
Fact NT4. For $N = p \cdot q$ where p and q are prime, $|Z_N^*(+1)| = |Z_N^*|/2$ and $|QR_N| = |Z_N^*|/4$.
Fact NT5. For any $x \in Z_N^*$, $x^{\varphi(N)} = 1 \, mod N$.

4.2. THE RSA ENCRYPTION FUNCTION. Let p and q be primes of length l, and let $N = p \cdot q$. Let e be an *encrypting exponent* such that $gcd(e, \varphi(N)) = 1$ and d a *decrypting exponent* such that $d \cdot e = 1 \, mod \, \varphi(N)$. The existence of such a d is guaranteed for all elements e for which $gcd(e, \varphi(N)) = 1$. The *RSA encryption function* [Rivest et al., 1978] is then defined for all $x \in Z_N$ by

$$RSA(x, N, e) = x^e \, mod N.$$

Note that decryption can be accomplished by exponentiation mod N:

$$(x^e)^d = x^{e\,d} \, mod N = x^{1 + i \cdot \varphi(N)} \, mod N = x \, mod N$$

for some natural number i by Fact NT5 because $e \cdot d = 1 \, mod \, \varphi(N)$.

Thus, following the informal intuition of Section 1, we think of Alice as generating the product $N = p \cdot q$; since she also knows p and q, she can generate both e (which she publishes along with N, thus yielding an encryption program) and d (the "trapdoor", which she keeps private).

There is currently no known polynomial-time algorithm for *inverting* the RSA encryption function—that is, the problem of computing x on inputs $RSA(x, N, e)$, N and e. Furthermore, the following result from Alexi et al. [1988] indicates that determining the least significant bit of x is as hard as inverting RSA (which amounts to determining *all* the bits of x).

THEOREM 1 [ALEXI ET AL. 1988]. *Let x, N, and e be as above. Then with respect to probabilistic polynomial-time reducibility, the following problems are equivalent*:

(1) *On input $RSA(x, N, e)$, N and e, output x.*
(2) *On input $RSA(x, N, e)$, N and e, output $LSB(x)$ with probability exceeding $1/2 + 1/p(l)$, where p is any fixed polynomial, $l = log N$ is the length of N, and $LSB(x)$ denotes the least significant bit of x. The probability is taken over x chosen uniformly from Z_N and any coin tosses of A.*

4.3. THE RABIN AND MODIFIED RABIN ENCRYPTION FUNCTIONS. The *Rabin encryption function* [Rabin, 1979] is specified by two primes p and q of length l. For $N = p \cdot q$ and $x \in Z_N^*$, we define

$$R(x, N) = x^2 \, mod N.$$

In this scheme, the trapdoor is the factorization of N, which allows Alice to compute square roots modulo N, and thus to decrypt. Known results regarding the security of the Rabin function include the following:

THEOREM 2 [RABIN 1979]. *Let x and N be as above. Then with respect to probabilistic polynomial-time reducibility, the following problems are equivalent*:

(1) *On input N, output a nontrivial factor of N.*
(2) *On input N and $R(x, N)$, output a y such that $R(y, N) = R(x, N)$.*

Furthermore, this reduction still holds when N is restricted to be a Blum integer in both problems. The *modified Rabin encryption function* [Alexi et al., 1988] is specified by two primes p and q of length l, both congruent to 3 modulo 4. Let $N = p \cdot q$ (thus, N is a Blum integer). We define a subset M_N of Z_N^* by

$$M_N = \left\{ x : 0 \leq x \leq \frac{N}{2} \text{ and } x \in Z_N^*(+1) \right\}.$$

For $x \in M_N$, the modified Rabin encryption function is then

$$MR(x, N) = x^2 \, mod N \quad \text{if} \quad x^2 \, mod N \in M_N,$$
$$MR(x, N) = (N - x^2) \, mod N, \quad \text{otherwise}.$$

This defines a 1–1 map from M_N onto M_N.

THEOREM 3 [ALEXI ET AL. 1988]. *Let x and N be as above. Then with respect to probabilistic polynomial-time reducibility, the following problems are equivalent*:

(1) *On input* $MR(x, N)$ *and* N, *output* x.
(2) *On input* $MR(x, N)$ *and* N, *output* $LSB(x)$ *with probability exceeding* $1/2 + 1/p(l)$, *where* p *is any fixed polynomial and* $l = \log N$ *is the length of* N. *The probability is taken over* x *chosen uniformly from* M_N *and any coin tosses of* A.

For Blum integers, $R(x, N)$ is a 1–1 mapping of QR_N. Hence, if $MR(x, N)$ is invertible, then we can invert $R(x, N)$ by attempting to invert MR for both the values $R(x, N)$ and $N - R(x, N)$, and succeeding for just the right one of these. Hence, Theorems 2 and 3 together imply that Problem (2) in Theorem 3 is equivalent to factoring Blum integers (with respect to probabilistic polynomial-time reducibility), a problem for which no polynomial-time algorithm is known.

4.4. THE QUADRATIC RESIDUE ASSUMPTION. Let $N = p \cdot q$, where p and q are primes of length l. For each $x \in Z_N^*(+1)$, define $QR(x, N) = 1$ if x is a quadratic residue mod N and $QR(x, N) = 0$ otherwise. Then, the *Quadratic Residue Assumption* states that if A is any probabilistic polynomial-time algorithm that takes N and x as input, then for infinitely many N we have

$$\mathbf{Pr}[A(N, x) = QR(x, N)] < \frac{1}{2} + \frac{1}{p(l)},$$

where p is any fixed polynomial. The probability is taken over x chosen uniformly from the set $Z_N^*(+1)$ and any coin tosses of A. As in the Rabin scheme, knowledge of the factors of N allows Alice to compute square roots modulo N and thus to determine if an element is a quadratic residue.

5. *Hard Learning Problems Based on Cryptographic Functions*

In this section, we construct hard learning problems based on the number-theoretic encryption functions described above. For each such function, we first define a representation class based on the function. For each possible target representation in this class, we then describe the *relevant examples* for this representation. These are the only examples with nonzero probability in the hard target distributions we define. We then proceed to prove the difficulty of even *weakly* learning the representation class under the chosen distributions, based on some standard cryptographic assumption on the security of the underlying encryption function. Finally, we show the ease of *evaluating* the representation class: More precisely, we show that each representation in the class can be computed by an NC^1 circuit (a polynomial-size, log-depth circuit of standard fan-in 2 Boolean gates). In Section 6, we apply these results to prove that weakly learning Boolean formulae, finite automata, constant-depth threshold circuits, and a number of other representation classes is hard under cryptographic assumptions.

We adopt the following notation: If $a_1, \dots, a_m$ are natural numbers, then $binary(a_1, \dots, a_m)$ is the binary representation of the sequence $a_1, \dots, a_m$. The relevant examples we construct will be of the form

$$\langle binary(a_1, \dots, a_m), b \rangle,$$

where b is a bit indicating whether the example is positive or negative. We denote by $powers(z, N)$ the sequence of natural numbers

$$z \bmod N, z^2 \bmod N, z^4 \bmod N, \dots, z^{2^{\lceil \log N \rceil}} \bmod N,$$

which are the first $\lceil \log N \rceil + 1$ successive square powers of z modulo N.

In the following subsections, we define representation classes C_n based on the number-theoretic function families described above. Representations in C_n will be over the domain $\{0, 1\}^n$; relevant examples with length less than n will implicitly be assumed to be padded to length n. Since only the relevant examples will have nonzero probability, we assume that all nonrelevant examples are negative examples of the target representation.

5.1. A Learning Problem Based on RSA

5.1.1. *The Representation Class* C_n. Let l be the largest natural number satisfying $4l^2 + 8l + 2 \leq n$. Each representation in C_n is defined by a triple (p, q, e) and this representation will be denoted $r_{(p,q,e)}$. Here p and q are primes of exactly l bits and $e \in Z^*_{\varphi(N)}$, where $N = p \cdot q$ (thus, $gcd(e, \varphi(N)) = 1$).

5.1.2. *Relevant Examples for* $r_{(p,q,e)} \in C_n$. A relevant example of $r_{(p,q,e)} \in C_n$ is of the form

$$\langle binary(powers(RSA(x, N, e), N), N, e), LSB(x) \rangle,$$

where $x \in Z_N$. Note that since the length of N is at most $2l + 1$, the length of such an example in bits is at most $(2l + 1)(2l + 1) + (2l + 1) + (2l + 1) = 4l^2 + 8l + 2 \leq n$. The target distribution D^+ for $r_{(p,q,e)}$ is uniform over the relevant positive examples of $r_{(p,q,e)}$ (i.e., those for which $LSB(x) = 1$) and the target distribution D^- is uniform over the relevant negative examples (i.e., those for which $LSB(x) = 0$).

5.1.3. *Difficulty of Weakly Learning* $C = \bigcup_{n \geq 1} C_n$. Suppose that A is a polynomial-time weak learning algorithm for C. We now describe how we can use algorithm A to invert the RSA encryption function. Let N be the product of two unknown l-bit primes p and q, and let $e \in Z^*_{\varphi(N)}$. Then given only N and e, we run algorithm A. Each time A requests a positive example of

$r_{(p,q,e)}$, we uniformly choose an $x \in Z_N$ such that $LSB(x) = 1$ and give the example

$$\langle \mathit{binary}(\mathit{powers}(RSA(x, N, e), N), N, e), 1\rangle$$

to A. Note that we can generate such an example in polynomial time on input N and e. This simulation generates the target distribution D^+. Each time that A requests a negative example of $r_{(p,q,e)}$, we uniformly choose an $x \in Z_N$ such that $LSB(x) = 0$ and give the example

$$\langle \mathit{binary}(\mathit{powers}(RSA(x, N, e), N), N, e), 0\rangle$$

to A. Again, we can generate such an example in polynomial time, and this simulation generates the target distribution D^-. Let h_A be the hypothesis output by algorithm A following this simulation. Then given $r = RSA(x, N, e)$ for some unknown x chosen uniformly from Z_N, $h_A(\mathit{binary}(\mathit{powers}(r, N), N, e)) = LSB(x)$ with probability at least $1/2 + 1/p(l)$ for some polynomial p by the definition of weak learning because n and l are polynomially related. Thus, we have a polynomial advantage for inverting the least significant bit of RSA. This allows us to invert RSA by the results of Alexi et al. [1988], given as Theorem 1.

5.1.4. *Ease of Evaluating* $r_{(p,q,e)} \in C_n$. For each $r_{(p,q,e)} \in C_n$, we show that $r_{(p,q,e)}$ has an equivalent NC^1 circuit. More precisely, we give a circuit that has depth $O(\log n)$ and size polynomial in n, and outputs the value of $r_{(p,q,e)}$ on inputs of the form

$$\mathit{binary}(\mathit{powers}(r, N), N, e),$$

where $N = p \cdot q$ and $r = RSA(x, N, e)$ for some $x \in Z_N$. Thus, the representation class $C = \bigcup_{n \geq 1} C_n$ is contained in (nonuniform) NC^1.

Since $e \in Z^*_{\varphi(N)}$, there is a $d \in Z^*_{\varphi(N)}$ such that $e \cdot d = 1 \; mod \; \varphi(N)$ (d is just the decrypting exponent for e). Thus, $r^d \, mod N = x^{e\,d} \, mod N = x \, mod N$. Hence, the circuit for $r_{(p,q,e)}$ simply multiplies together the appropriate powers of r (which are always explicitly provided in the input) to compute $r^d \, mod N$, and outputs the least significant bit of the resulting product. This is an NC^1 step by the iterated product circuits of Beame et al. [1986].

5.2. A Learning Problem Based on Quadratic Residues

5.2.1. *The Representation Class* C_n. Let l be the largest natural number satisfying $4l^2 + 6l + 2 \leq n$. Each representation in C_n is defined by a pair of l-bit primes (p, q) and this representation will be denoted $r_{(p,q)}$.

5.2.2. *Relevant Examples for* $r_{(p,q)} \in C_n$. For a representation $r_{(p,q)} \in C_n$, let $N = p \cdot q$. We consider only points $x \in Z^*_N(+1)$. A relevant example of $r_{(p,q)}$ is then of the following form:

$$\langle \mathit{binary}(\mathit{powers}(x, N), N), QR(x, N)\rangle.$$

Note that the length of such an example in bits is at most $4l^2 + 6l + 2 \leq n$. The target distribution D^+ for $r_{(p,q)}$ is uniform over the relevant positive examples of $r_{(p,q)}$ (i.e., those for which $QR(x, N) = 1$) and the target distribution D^- is uniform over the relevant negative examples (i.e., those for which $QR(x, N) = 0$).

5.2.3. *Difficulty of Weakly Learning* $C = \bigcup_{n \geq 1} C_n$. Suppose that A is a polynomial-time weak learning algorithm for C. We now describe how we can use algorithm A to recognize quadratic residues. Let N be the product of two unknown l-bit primes p and q. Given only N as input, we run algorithm A. Every time A requests a positive example of $r_{(p,q)}$, we uniformly choose $y \in Z_N^*$ and give the example

$$\langle \mathit{binary}(\mathit{powers}(y^2 \, \mathit{mod} N, N), N), 1\rangle$$

to A. Note that such an example can be generated in polynomial time on input N. This simulation generates the target distribution D^+.

In order to generate the negative examples for our simulation of A, we uniformly choose $u \in Z_N^*$ until $J(u, N) = 1$. By Fact NT4, this can be done with high probability in polynomial time. The probability is $1/2$ that such a u is a nonresidue modulo N. Assuming we have obtained a nonresidue u, every time A requests a negative example of $r_{(p,q)}$, we uniformly choose $y \in Z_N^*$ and give to A the example

$$\langle \mathit{binary}(\mathit{powers}(uy^2 \, \mathit{mod} N, N), N), 0\rangle,$$

which can be generated in polynomial time. Note that, if u actually is a nonresidue, then this simulation generates the target distribution D^-, and this run of A will with high probability produce an hypothesis h_A with accuracy at least $1/2 + 1/p(l)$ with respect to D^+ and D^-, for some polynomial p (call such a run a *good* run). On the other hand, if u is actually a residue, then A has been trained improperly (i.e., A has been given positive examples when it requested negative examples), and no performance guarantees can be assumed. The probability of a good run of A is at least $1/2(1 - \delta)$.

We thus simulate A as described above many times, testing each hypothesis to determine if the run was a good run. To test if a good run has occurred, we first determine if h_A has accuracy at least $1/2 + 1/2p(l)$ with respect to D^+. This can be determined with high probability by generating D^+ as above and estimating the accuracy of h_A using Fact CB1 and Fact CB2. Assuming h_A passes this test, we now would like to test h_A against the simulated distribution D^-; however, we do not have direct access to D^- since this requires a nonresidue *mod* N. Thus, we instead estimate the probability that h_A classifies an example as positive when this example is drawn from the uniform distribution over *all* relevant examples (both positive and negative). This can be done by simply choosing $x \in Z_N^*$ uniformly and computing $h_A(\mathit{binary}(\mathit{powers}(x, N), N))$. The probability that h_A classifies such examples as positive is near $1/2$ if and only if h_A has nearly equal accuracy on D^+ and D^-. Thus, by estimating the accuracy of h_A on D^+, we can estimate the accuracy of h_A on D^- as well, without direct access to a simulation of D^-.

We continue to run A and test until a good run of A is obtained with high probability. Then given x chosen randomly from Z_N^*,

$$h_A(\mathit{binary}(\mathit{powers}(x, N), N)) = QR(x, N)$$

with probability at least $1/2 + 1/p(l)$, contradicting the Quadratic Residue Assumption.

5.2.4. *Ease of Evaluating* $r_{(p,q)} \in C_n$. For each $r_{(p,q)} \in C_n$, we give an NC^1 circuit for evaluating the concept represented by $r_{(p,q)}$ on an input of the form

$$binary(powers(x, N), N),$$

where $N = p \cdot q$ and $x \in Z_N^*$. This circuit has four phases:

Phase I. Compute the powers

$$x \bmod p, x^2 \bmod p, x^4 \bmod p, \ldots, x^{2^{2l}} \bmod p$$

and the powers

$$x \bmod q, x^2 \bmod q, x^4 \bmod q, \ldots, x^{2^{2l}} \bmod q.$$

Note that the length of N is $2l$. Since for any $a \in Z_N^*$ we have that $a \bmod p = (a \bmod N) \bmod p$, these powers can be computed from the input $binary(powers(x, N), N)$ by parallel $\bmod p$ and $\bmod q$ circuits. Each such circuit involves only a division step followed by a multiplication and a subtraction. The results of Beame et al. [1986] imply that these steps can be carried out by an NC^1 circuit.

Phase II. Compute $x^{(p-1)/2} \bmod p$ and $x^{(q-1)/2} \bmod q$. These can be computed by multiplying the appropriate powers $\bmod p$ and $\bmod q$ computed in Phase I. Since the iterated product of l numbers each of length l bits can be computed in NC^1 by the results of Beame et al. [1986], this is also an NC^1 step.

Phase III. Determine if $x^{(p-1)/2} = 1 \bmod p$ or $x^{(p-1)/2} = -1 \bmod p$, and if $x^{(q-1)/2} = 1 \bmod q$ or $x^{(q-1)/2} = -1 \bmod q$. That these are the only cases follows from Fact NT2; furthermore, this computation determines whether x is a residue $\bmod p$ and $\bmod q$. Given the outputs of Phase II, this is clearly an NC^1 step.

Phase IV. If the results of Phase III were $x^{(p-1)/2} = 1 \bmod p$ and $x^{(q-1)/2} = 1 \bmod q$, then output 1, otherwise output 0. This is again an NC^1 step.

5.3. A Learning Problem Based on Factoring Blum Integers

5.3.1. *The Representation Class* C_n. Let l be the largest natural number satisfying $4l^2 + 6l + 2 \leq n$. Each representation in C_n is defined by a pair of l-bit primes (p, q), both congruent to 3 modulo 4, and this representation will be denoted $r_{(p,q)}$. Thus, the product $N = p \cdot q$ is a Blum integer.

5.3.2. *Relevant Examples for* $r_{(p,q)} \in C_n$. We consider points $x \in M_N$. A relevant example of $r_{(p,q)} \in C_n$ is then of the form

$$\langle binary(powers(MR(x, N), N), N), LSB(x) \rangle.$$

The length of this example in bits is at most $4l^2 + 6l + 2 \leq n$. The target distribution D^+ for $r_{(p,q)}$ is uniform over the relevant positive examples (i.e., those for which $LSB(x) = 1$) and the target distribution D^- is uniform over the relevant negative examples (i.e., those for which $LSB(x) = 0$).

5.3.3. *Difficulty of Weakly Learning* $C = \bigcup_{n \geq 1} C_n$. Suppose that A is a polynomial-time weak learning algorithm for C. We now describe how to use A to factor Blum integers. Let N be a Blum integer. Given only N as input, we run algorithm A. Every time A requests a positive example, we choose

$x \in M_N$ uniformly such that $LSB(x) = 1$, and give the example

$$\langle binary(powers(MR(x, N), N), N), 1\rangle$$

to A. Such an example can be generated in polynomial time on input N. This simulation generates the distribution D^+. Every time A requests a negative example, we choose $x \in M_n$ uniformly such that $LSB(x) = 0$, and give the example

$$\langle binary(powers(MR(x, N), N), N), 0\rangle$$

to A. Again, this example can be generated in polynomial time. This simulation generates the distribution D^-. When algorithm A has halted, $h_A(binary(powers(r, N), N)) = LSB(x)$ with probability $1/2 + 1/p(l)$ for $r = MR(x, N)$ and x chosen uniformly from M_N. This implies that we can factor Blum integers by the results of Rabin [1979] and Alexi et al. [1988] given in Theorems 2 and 3.

5.3.4. *Ease of Evaluating* $r_{\langle p,q\rangle} \in C_n$. For each $r_{\langle p,q\rangle} \in C_n$, we give an NC^1 circuit for evaluating the concept represented by $r_{\langle p,q\rangle}$ on an input of the form

$$binary(powers(r, N), N),$$

where $N = p \cdot q$ and $r = MR(x, N)$ for some $x \in M_N$. This is accomplished by giving an NC^1 implementation of the first three steps of the root-finding algorithm of Adleman et al. [1977], as it is described by Angluin [1982]. Note that if we let $a = x^2 \, mod N$, then either $r = a$ or $r = (N - a) \, mod N$ according to the definition of the modified Rabin function. The circuit has four phases:

Phase I. Determine if the input r is a quadratic residue $mod N$. This can be done using the given powers of r and $r_{\langle p,q\rangle}$ using the NC^1 circuit described in quadratic residue-based scheme of Section 5.2. Note that since p and q are both congruent to $3 \, mod \, 4$, $(N - a) \, mod N$ is never a quadratic residue $mod N$ (see Angluin [1982]). If it is decided that $r = (N - a) \, mod N$, generate the intermediate output $a \, mod N$. This can clearly be done in NC^1. Also, notice that for any z, $z^{2i} = (N - z)^{2i} \, mod N$ for $i \geq 1$. Hence, these powers of r are identical in the two cases. Finally, recall that the NC^1 circuit for quadratic residues produced the powers of $r \, mod p$ and the powers of $r \, mod q$ as intermediate outputs, so we may assume that the powers

$$a, a^2 \, mod p, a^4 \, mod p, \ldots, a^{2^{2l}} \, mod p$$

and

$$a, a^2 \, mod q, a^4 \, mod q, \ldots, a^{2^{2l}} \, mod q$$

are also available.

Phase II. Let l_p (respectively, l_q) be the largest positive integer such that $2^{l_p} \mid (p - 1)$ (respectively, $2^{l_q} \mid (q - 1)$). Let $Q_p = (p - 1)/2^{l_p}$ (respectively, $Q_q = (q - 1)/2^{l_q}$). Using the appropriate powers of $x^2 \, mod p$ and $mod q$, compute $u = a^{(Q_p+1)/2} \, mod p$ and $v = a^{(Q_q+1)/2} \, mod q$ with NC^1 iterated product circuits. Since p and q are both congruent to $3 \, mod \, 4$, u and $p - u$ are square roots of $a \, mod q$, and v and $q - v$ are square roots of $a \, mod q$ by the results of Adleman et al. [1977] (see also Angluin [1982]).

Phase III. Using Chinese remaindering, combine u, $p - u$, v and $q - v$ to compute the four square roots of $a \bmod N$ (see, e.g., Kranakis [1986]). Given p and q, this requires only a constant number of multiplication and addition steps, and so is computed in NC^1.

Phase IV. Find the root from Phase III that is in M_N, and output its least significant bit.

6. *Learning Small Boolean Formulae, Finite Automata, and Threshold Circuits is Hard*

The results of Section 5 show that for some fixed polynomial $q(n)$, learning NC^1 circuits of size at most $q(n)$ is computationally as difficult as the problems of inverting RSA, recognizing quadratic residues, and factoring Blum integers. However, there is a polynomial $p(n)$ such that any NC^1 circuit of size at most $q(n)$ can be represented by a Boolean formulae of size at most $p(n)$. Thus we have proved the following:

THEOREM 4. *Let $BF_n^{p(n)}$ denote the class of Boolean formulae over n variables of size at most $p(n)$, and let $BF^{p(n)} = \bigcup_{n \geq 1} BF_n^{p(n)}$. Then for some polynomial $p(n)$, the problems of inverting the RSA encryption function, recognizing quadratic residues and factoring Blum integers are probabilistic polynomial-time reducible to weakly learning $BF^{p(n)}$.*

In fact, we can apply the substitution arguments of Kearns et al. [1987] to show that Theorem 4 holds even for the class of monotone Boolean formulae in which each variable appears at most once.

Pitt and Warmuth [1988] show that if the class ADFA is polynomially weakly learnable, then the class BF is polynomially weakly learnable. Combining this with Theorem 4, we have:

THEOREM 5. *Let $ADFA_n^{p(n)}$ denote the class of deterministic finite automata of size at most $p(n)$ that only accept strings of length n, and let $ADFA^{p(n)} = \bigcup_{n \geq 1} ADFA_n^{p(n)}$. Then for some polynomial $p(n)$, the problems of inverting the RSA encryption function, recognizing quadratic residues and factoring Blum integers are probabilistic polynomial-time reducible to weakly learning $ADFA^{p(n)}$.*

Note that combined with the algorithm of Angluin for learning DFAs using membership and equivalence queries (which can be replaced by random examples), Theorem 5 demonstrates a provable increase in the learner's power when membership queries are added.

Using results of Chandra et al., [1984], Beame et al. [1986], and Reif [1987], it can be shown that the representations described in Section 5 can each be computed by a polynomial-size, constant-depth threshold circuit. Thus, we have:

THEOREM 6. *For some fixed constant natural number d, let $dTC_n^{p(n)}$ denote the class of threshold circuits over n variables with depth at most d and size at most $p(n)$, and let $dTC^{p(n)} = \bigcup_{n \geq 1} dTC_n^{p(n)}$. Then for some polynomial $p(n)$, the problems of inverting the RSA encryption function, recognizing quadratic residues and factoring Blum integers are probabilistic polynomial-time reducible to weakly learning $dTC^{p(n)}$.*

It is important to reiterate that these hardness results hold regardless of the hypothesis representation class of the learning algorithm; that is, Boolean formulae, DFAs, and constant-depth threshold circuits are not weakly learnable by *any* polynomially evaluatable representation class (under standard cryptographic assumptions). We note that no NP-hardness results are known for these classes even if we restrict the hypothesis class to be the same as the target class and insist on strong learnability rather than weak learnability. It is also possible to give reductions showing that many other interesting classes (e.g., CFGs and NFAs) are not weakly learnable, under the same cryptographic assumptions. In general, any representation class whose computational power subsumes that of NC^1 is not weakly learnable; however, more subtle reductions are also possible. In particular, our results resolve a problem posed by Pitt and Warmuth [1988] by showing that under cryptographic assumptions, the class of all languages accepted by logspace Turing machines is not weakly learnable.

Pitt and Warmuth [1988] introduce a general notion of reduction between learning problems, and a number of learning problems are shown to have equivalent computational difficulty (with respect to probabilistic polynomial-time reducibility); thus, if the learning problem for a representation class C_1 reduces to the learning problem for a representation class C_2, then a polynomial-time learning algorithm for C_2 in the distribution-free model implies a polynomial-time learning algorithm for C_1. Learning problems are then classified according to the complexity of their *evaluation problem*, the problem of evaluating a representation on an input example. In Pitt and Warmuth [1988], the evaluation problem is treated as a uniform problem (i.e., one algorithm for evaluating all representations in the class); by treating the evaluation problem nonuniformly (e.g., a separate circuit for each representation), we were able to show that NC^1 contains a number of presumably hard-to-learn classes of Boolean functions. By giving reductions from NC^1 to other classes of representations, we thus clarify the boundary of what is efficiently learnable.

7. *A Generalized Construction Based on Any Trapdoor Function*

Let us now give a brief summary of the techniques that were used in Sections 5 and 6 to obtain hardness results for learning based on cryptographic assumptions. In each construction (RSA, quadratic residue and factoring Blum integers), we began with a candidate *trapdoor function* family, informally a family of functions each of whose members f is easy to compute (i.e., given x, it is easy to compute $f(x)$), hard to invert (i.e., given only $f(x)$, it is difficult to compute x), but easy to invert given a secret "key" to the function [Yao, 1982] (the *trapdoor*). We then constructed a learning problem in which the complexity of inverting the function *given* the trapdoor key corresponds to the complexity of the representations being learned, and learning from random examples corresponds to inverting the function *without* the trapdoor key. Thus, the learning algorithm is essentially required to learn the inverse of a trapdoor function, and the small representation for this inverse is simply the secret trapdoor information.

To prove hardness results for the simplest possible representation classes, we then eased the computation of the inverse given the trapdoor key by providing the powers of the original input in each example. This additional information provably does not compromise the security of the original function. A key property of trapdoor functions exploited by our constructions is the ability to generate random examples of the target representation without the trapdoor key; this corresponds to the ability to generate encrypted messages given only the public key in a public-key cryptosystem.

By assuming that specific functions such as RSA are trapdoor functions, we were able to find modified trapdoor funcitons whose inverse computation given the trapdoor could be performed by very simple circuits. This allows us to prove hardness results for specific representation classes that are of interest in computational learning theory. Such specific intractability assumptions appear necessary since the weaker and more general assumption that there exists a trapdoor family that can be computed (in the forward direction) in polynomial time does not allow us to say anything about the hard-to-learn representation class other than it having polynomial-size circuits.

However, the summary above suggests a general method for proving hardness results for learning: To show that a representation class C is not learnable, find a trapdoor function whose inverse can be computed by C given the trapdoor key. In this section, we formalize these ideas and prove a theorem demonstrating that this is indeed a viable approach.

We use the following definition for a family of trapdoor functions, which can be derived from Yao [1982]: Let $P = \{P_n\}$ be a family of probability distributions, where for $n \geq 1$ the distribution P_n is over pairs $(k, k') \in \{0,1\}^n \times \{0,1\}^n$. We think of k as the n-bit *public key* and k' as the associated n-bit *private key*. Let $Q = \{Q_k\}$ be a family of probability distributions parameterized by the public key k, where if $|k| = n$, then Q_k is a distribution over $\{0,1\}^n$. We think of Q as a distribution family over the *message space*. The function f: $\{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ maps an n-bit public key k and an n-bit *cleartext message* x to the *ciphertext* $f(k, x)$. We call the triple (P, Q, f) an $\alpha(n)$*-strong trapdoor scheme* if it has the following properties:

(i) There is probabilistic polynomial-time algorithm G (the *key generator*) that on input 1^n outputs a pair (k, k') according to the distribution P_n. Thus, pairs of public and private keys are easily generated.

(ii) There is a probabilistic polynomial-time algorithm M (the *message generator*) that on input k outputs x according to the distribution Q_k. Thus, messages are easily generated given the public key k.

(iii) There is a polynomial-time algorithm E that on input k and x outputs $f(k, x)$. Thus, encryption is easy.

(iv) Let A be any probabilistic polynomial-time algorithm. Perform the following experiment: Draw a pair (k, k') according to P_n, and draw x according to Q_k. Give the inputs k and $f(k, x)$ to A. Then, the probability that $A(k, f(k, x)) \neq x$ is at least $\alpha(n)$. Thus, decryption from only the public key and the ciphertext is hard.

(v) There is a polynomial-time algorithm D that on input k, k' and $f(k, x)$ outputs x. Thus, decryption given the private key (or *trapdoor*) is easy.

As an example, consider the RSA cryptosystem [Rivest et al., 1978]. Here the distribution P_n is uniform over all (k, k') where $k' = (p, q)$ for n-bit primes p and q and $k = (p \cdot q, e)$ with $e \in Z^*_{\varphi(p \cdot q)}$. The distribution Q_k is uniform over $Z_{p \cdot q}$, and $f(k, x) = f((p \cdot q, e), x) = x^e \bmod p \cdot q$.

We now formalize the notion of the inverse of a trapdoor function being computed in a representation class. Let $C = \bigcup_{n \geq 1} C_n$ be a parameterized Boolean representation class. We say that a trapdoor scheme (P, Q, f) is *invertible in C given the trapdoor* if for any $n \geq 1$, for any pair of keys $(k, k') \in \{0, 1\}^n \times \{0, 1\}^n$, and for any $1 \leq i \leq n$, there is a representation $c^i_{(k,k')} \in C_n$ that on input $f(k, x)$ (for any $x \in \{0, 1\}^n$) outputs the ith bit of x.

THEOREM 7. *Let p be any polynomial, and let $\alpha(n) \geq 1/p(n)$. Let (P, Q, f) be an $\alpha(n)$-strong trapdoor scheme, and let C be a parameterized Boolean representation class. Then if (P, Q, f) is invertible in C given the trapdoor, C is not polynomially learnable.*

PROOF. Let A be any polynomial-time learning algorithm for C. We use algorithm A as a subroutine in a polynomial-time algorithm A' that with high probability outputs x on input k and $f(k, x)$, thus contradicting condition (iv) in the definition of a trapdoor scheme.

Let (k, k') be n-bit public and private keys generated by the distribution P_n. Let x be an n-bit message generated according to the distribution Q_k. Then on input k and $f(k, x)$, algorithm A' behaves as follows: for $1 \leq i \leq n$, algorithm A' simulates algorithm A, choosing accuracy parameter $\epsilon = \alpha(n)/n$. For the ith run of A, each time A requests a positive example, A' generates random values x' from the distribution Q_k (this can be done in polynomial time by condition (ii) in the definition of trapdoor scheme) and computes $f(k, x')$ (this can be done in polynomial time by condition (iii) in the definition of trapdoor scheme). If the ith bit of $f(k, x')$ is 1, then A' gives x' as a positive example to A; similarly, A' generates negative examples for the ith run of A by drawing x' such that the ith bit of $f(k, x')$ is 0. If after $O(1/\epsilon \ln n/\delta)$ draws from Q_k, A' is unable to obtain a positive (respectively, negative) example for A, then A' assumes that with high probability a random x' results in the ith bit of $f(k, x')$ being 0 (respectively, 1), and terminates this run by setting h^i_k to the hypothesis that is always 0 (respectively, 1). The probability that A' terminates the run incorrectly can be shown to be smaller than δ/n by application of Fact CB1 and Fact CB2.

Note that all of the examples given to the ith run of A are consistent with a representation in C_n, since the ith bit of $f(k, \cdot)$ is computed by the representation $c^i_{(k,k')}$. Thus, with high probability A outputs an ϵ-good hypothesis h^i_k. To invert the original input $f(k, x)$, A' simply outputs the bit sequence $h^1_k(f(k, x)) \cdots h^n_k(f(k, x))$. The probability that any bit of this string differs from the corresponding bit of x is at most $n\epsilon < \alpha(n)$, contradicting the assumption that (P, Q, f) is an $\alpha(n)$-strong trapdoor scheme. □

8. *Application: Hardness Results for Approximation Algorithms*

In this section, we digress from learning briefly and apply the results of Section 6 to prove that under cryptographic assumptions, certain combinatorial optimization problems, including a natural generalization of graph coloring, cannot be efficiently approximated even in a very weak sense. These results show that, for these problems, it is difficult to find a solution that approximates the optimal solution even within a factor that grows rapidly with the input size. Such results are infrequent in complexity theory, and seem difficult to obtain for natural problems using presumably weaker assumptions such as P $\neq$ NP.

We begin by stating a needed theorem of Blumer et al. [1987] known as *Occam's Razor*. Their result essentially gives an upper bound on the sample size required for learning C by H, and shows that the general technique of finding an hypothesis that is both consistent with the sample drawn and significantly shorter than this sample is sufficient for distribution-free learning. Thus, if one can efficiently perform *data compression* on a random sample, then one can learn efficiently.

THEOREM 8 [BLUMER ET AL., 1987]. *Let C and H be polynomially evaluatable parameterized Boolean representation classes. Fix $\alpha \geq 1$ and $0 \leq \beta \leq 1$, and let A be an algorithm that on input a labeled sample S of some $c \in C_n$, consisting of m positive examples of c drawn from D^+ and m negative examples of c drawn from D^-, outputs an hypothesis $h_A \in H_n$ that is consistent with S and satisfies $|h_A| \leq n^\alpha m^\beta$, where $|h_A|$ is the length of the representation h_A in bits. Then A is a learning algorithm for C by H; the sample size required is*

$$m = O\left(\frac{1}{\epsilon} log \frac{1}{\delta} + \left(\frac{n^\alpha}{\epsilon} log \frac{n^\alpha}{\epsilon}\right)^{1/(1-\beta)}\right).$$

Let $|S| = mn$ denote the number of bits in the sample S. Note that if A instead outputs h_A satisfying $|h_A| \leq n^{\alpha'} |S|^\beta$ for some fixed $\alpha' \geq 1$ and $0 \leq \beta < 1$ then $|h_A| \leq n^{\alpha'}(mn)^\beta = n^{\alpha' + \beta} m^\beta$, so A satisfies the condition of Theorem 8 for $\alpha = \alpha' + \beta$. This formulation of Occam's Razor will be of particular use to us.

Let C and H be polynomially evaluatable parameterized Boolean representation classes, and define the *Consistency Problem* $Con(C, H)$ as follows:

The Consistency Problem Con(C, H).

Input. A labeled sample S of some $c \in C_n$.

Output. $h \in H_n$ such that h is consistent with S and $|h|$ is minimized.

We use $opt_{\text{Con}}(S)$ to denote the size of the smallest hypothesis in H that is consistent with the sample S, and $|S|$ to denote the number of bits in S. Using the results of Section 6 and Theorem 8, we immediately obtain proofs of the following theorems:

THEOREM 9. *Let BF_n denote the class of Boolean formulae over n variables, and let $BF = \bigcup_{n \geq 1} BF_n$. Let H be any polynomially evaluatable parameterized Boolean representation class. Then the problems of inverting the RSA encryption function, recognizing quadratic residues and factoring Blum integers are probabilistic polynomial-time reducible to the problem of approximating the optimal solution of an instance S of $Con(BF, H)$ by an hypothesis h satisfying*

$$|h| \leq (opt_{Con}(S))^\alpha |S|^\beta$$

for any $\alpha \geq 1$ and $0 \leq \beta < 1$.

THEOREM 10. *Let $ADFA_n$ denote the class of deterministic finite automata accepting only strings of length n, and let $ADFA = \bigcup_{n \geq 1} ADFA_n$. Let H be any polynomially evaluatable parameterized Boolean representation class. Then invert-*

ing the RSA encryption function, recognizing quadratic residues and factoring Blum integers are probabilistic polynomial-time reducible to approximating the optimal solution of an instance S of Con(ADFA, H) by an hypothesis h satisfying

$$| h | \leq (opt_{Con}(S))^{\alpha} | S |^{\beta}$$

for any $\alpha \geq 1$ *and* $0 \leq \beta < 1$.

THEOREM 11. *Let* dTC_n *denote the class of threshold circuits over n variables with depth at most d, and let* $dTC = \bigcup_{n \geq 1} dTC_n$. *Let H be any polynomially evaluatable parameterized Boolean representation class. Then for some constant* $d \geq 1$, *the problems of inverting the RSA encryption function, recognizing quadratic residues and factoring Blum integers are probabilistic polynomial-time reducible to the problem of approximating the optimal solution of an instance S of Con(dTC, H) by an hypothesis h satisfying*

$$| h | \leq (opt_{Con}(S))^{\alpha} | S |^{\beta}$$

for any $\alpha \geq 1$ *and* $0 \leq \beta < 1$.

These theorems demonstrate that the results of Section 6 are, in some sense, not dependent upon the particular models of learnability that we study, since we are able to restate the hardness of learning in terms of standard combinatorial optimization problems. Using a generalization of Theorem 8 [Blumer et al., 1989], we can in fact prove Theorems 9, 10, and 11 for the *Relaxed Consistency Problem*, where the hypothesis found must agree with only a fraction $1/2 + 1/p(opt_{\text{Con}}(S), n)$ for any fixed polynomial p. The central idea of the proof is the same: since the results of Blumer et al. [1989] demonstrate that for sufficient sample size, solution of the relaxed consistency problem implies weak learning, and we have shown weak learning to be as hard as the cryptographic problems for the various representation classes, the relaxed consistency problem is as hard as the cryptographic problems. Using the results of Goldreich et al. [1986], it is also possible to show similar hardness results for the Boolean circuit consistency problem *Con*(CKT, CKT) using the weaker assumption that there exists a one-way function.

It is interesting to contrast Theorem 10 with similar results obtained by Pitt and Warmuth [1989]. They also prove hardness results for the problem of finding small deterministic finite automata consistent with a labeled sample, but based on the weaker assumption P ≠ NP. However (using the notation of Theorem 10), their results only hold for a more restricted range of α and β, and require the restriction that H be the class of deterministic finite automata. We refer the reader to their paper for details.

Note that Theorem 11 addresses the optimization problem *Con*(dTC, TC) as a special case. This problem is essentially that of finding a set of weights in a neural network that yields the desired input–output behavior, sometimes referred to as the *loading problem*. Theorem 11 states that even if we allow a much larger net than is actually required, finding these weights is computationally intractable, even for only a constant number of "hidden layers". This result should be contrasted with those of Judd [1988] and Blum and Rivest [1988], which rely on the weaker assumption P ≠ NP but do not prove hardness of relaxed consistency and do not allow the hypothesis network to be substantially larger than the smallest consistent network. We also make no assumptions on the topology of the output circuit.

Theorems 9, 10, and 11 are interesting for at least two reasons. First, they suggest that it is possible to obtain stronger hardness results for combinatorial optimization approximation algorithms by using stronger complexity-theoretic assumptions. Such results seem difficult to obtain using only the assumption P $\neq$ NP. Second, these results provide us with natural examples of optimization problems for which it is hard to approximate the optimal solution even within a multiplicative factor that grows as a function of the input size. Several well-studied problems apparently have this property, but little has been proven in this direction. Perhaps the best example is graph coloring, where the best polynomial-time algorithms require approximately $n^{1-1/(k-1)}$ colors on k-colorable n-vertex graphs (see Wigderson [1982] and Blum [1989]) but coloring has been proven NP-hard only for $(2 - \epsilon)k$ colors for any $\epsilon > 0$ (see Garey and Johnson [1979]). Thus, for 3-colorable graphs we only know that 5-coloring is hard, but the best algorithm requires roughly $O(n^{0.4})$ colors on n-vertex graphs! This leads us to look for approximation-preserving reductions from our provably hard optimization problems to other natural problems.

We now define a class of optimization problems that we call *formula coloring* problems. Here we have variables $y_1, \dots, y_m$ assuming natural number values, or *colors*. We regard an assignment of colors to the y_i (called a *coloring*) as a partition P of the variable set into equivalence classes; thus two variables have the same color if and only if they are in the same equivalence class. We consider Boolean formulae that are formed using the standard basis over atomic elements of the form $(y_i = y_j)$ and $(y_i \neq y_j)$, where the predicate $(y_i = y_j)$ is satisfied if and only if y_i and y_j are assigned the same color.

A *model* for such a formula $F(y_1, \dots, y_m)$ is a coloring of the variables $y_1, \dots, y_m$ such that F is satisfied. A *minimum model* for the F is a model using the fewest colors. For example, the formula

$$(y_1 = y_2) \vee ((y_1 \neq y_2) \wedge (y_3 \neq y_4))$$

has as a model the two-color partition $\{y_1, y_3\}, \{y_2, y_4\}$ and has a minimum model the one-color partition $\{y_1, y_2, y_3, y_4\}$.

We are interested in the problem of finding minimum models for certain restricted classes of formulae. For $F(y_1, \dots, y_m)$, a formula as described above, and P a model of F, we let $|P|$ denote the number of colors in P and $opt_{\mathrm{FC}}(F)$ the number of colors in a minimum model of F.

We first show how graph coloring can be exactly represented as a formula coloring problem. If G is a graph, then for each edge (v_i, v_j) in G, we conjunct the expression $(y_i \neq y_j)$ to the formula $F(G)$. Then $opt_{\mathrm{FC}}(F(G))$ is exactly the number of colors required to color G. Similarly, by conjuncting expressions of the form

$$((y_1 \neq y_2) \vee (y_1 \neq y_3) \vee (y_2 \neq y_3))$$

we can also exactly represent the *3-hypergraph coloring* problem (where each hyperedge contains 3 vertices) as a formula coloring problem.

To prove our hardness results, we consider a generalization of the graph coloring problem:

THE FORMULA COLORING PROBLEM *FC*:

Input: A formula $F(y_1, \dots, y_m)$ that is a conjunction only of expressions of the form $(y_i \neq y_j)$ (as in the graph coloring problem) *or* of the form $((y_i \neq y_j) \vee (y_k = y_l))$.

Output: A minimum model for F.

We show that approximating an optimal solution to this problem is as hard as approximating the consistency problem *Con*(DFA, DFA), where DFA is the class of deterministic finite automata. Note that this problem is at least as hard to approximate as *Con*(ADFA, H), which we have already proven an approximation hardness result in Theorem 10.

THEOREM 12. *There is a polynomial-time algorithm A that on input an instance S of the problem Con(DFA, DFA) outputs an instance $F(S)$ of the formula coloring problem such that S has a k-state consistent hypothesis $M \in DFA$ if and only if $F(S)$ has a model of k colors.*

PROOF. Let S contain the labeled examples

$$\langle w_1, b_1 \rangle, \langle w_2, b_2 \rangle, \dots, \langle w_m, b_m \rangle$$

where each $w_i \in \{0,1\}^n$ and $b_i \in \{0,1\}$. Let w_i^j denote the jth bit of w_i. We create a variable z_i^j for each $1 \leq i \leq n$ and $0 \leq j \leq m$. Let M be a smallest DFA consistent with S. Then we interpret z_i^j as representing the state that M is in immediately after reading the bit w_i^j on input w_i. The formula $F(S)$ will be over the z_i^j and is constructed as follows: For each i_1, i_2 and j_1, j_2 such that $0 \leq j_1, j_2 < n$ and $w_{i_1}^{j_1+1} = w_{i_2}^{j_2+1}$ we conjunct the predicate

$$\left(\left(z_{i_1}^{j_1} = z_{i_2}^{j_2}\right) \to \left(z_{i_1}^{j_1+1} = z_{i_2}^{j_2+1}\right)\right)$$

to $F(S)$. Note that this predicate is equivalent to

$$\left(\left(z_{i_1}^{j_1} \neq z_{i_2}^{j_2}\right) \vee \left(z_{i_1}^{j_1+1} + z_{i_2}^{j_2+1}\right)\right)$$

and thus has the required form. These formulae are designed to encode the constraint that if M is in the same state in two different computations on input strings from S, and the next input symbol is the same in both strings, then the next state in each computation must be the same.

For each i_1, i_2 $(1 \leq i_1, i_2 \leq m)$ such that $b_{i_1} \neq b_{i_2}$, we conjunct the predicate $(z_{i_1}^n \neq z_{i_2}^n)$. These predicates are designed to encode the constraint that the input strings in S that are accepted by M must result in different final states than those strings in S that are rejected by M.

We first prove that if M has k states, then $opt_{\mathrm{FC}}(F(S)) \leq k$. In particular, let P be the k-color partition that assigns $z_{i_1}^{j_1}$ and $z_{i_2}^{j_2}$ the same color if and only if M is in the same state after reading $w_{i_1}^{j_1}$ on input w_{i_1} and after reading $w_{i_2}^{j_2}$ on input w_{i_2}. We show that P is a model of $F(S)$. A conjunct

$$\left(\left(z_{i_1}^{j_1} = z_{i_2}^{j_2}\right) \to \left(z_{i_1}^{j_1+1} = z_{i_2}^{j_2+1}\right)\right)$$

of $F(S)$ cannot be violated by P since this conjunct appears only if $w_{i_1}^{j_1+1} = w_{i_2}^{j_2+1}$; thus, if state $z_{i_1}^{j_1}$ is equivalent to state $z_{i_2}^{j_2}$, then state $z_{i_1}^{j_1+1}$ must be equivalent to state $z_{i_2}^{j_2+1}$ since M is deterministic. A conjunct

$$\left(z_{i_1}^{n} \neq z_{i_2}^{n}\right)$$

of $F(S)$ cannot be violated by P since this conjunct appears only if $b_{i_1} \neq b_{i_2}$, and if state $z_{i_1}^{n}$ is equivalent to state $z_{i_2}^{n}$ then w_{i_1} and w_{i_2} are either both accepted or both rejected by M, which contradicts M being consistent with S.

For the other direction, we show that, if $opt_{\mathrm{FC}}(F(S)) \leq k$, then there is a k-state DFA M' that is consistent with S. M' is constructed as follows: The k states of M' are labeled with the k equivalence classes (colors) $X_1, \dots, X_k$ of the variables z_i^j in a minimum model P' for $F(S)$. There is a transition from state X_p to state X_q if and only if there are i, j such that $z_i^j \in X_p$ and $z_i^{j+1} \in X_q$; this transition is labeled with the symbol w_i^{j+1}. We label X_p an *accepting* (respectively, *rejecting*) state, if for some variable $z_i^n \in X_p$ we have $b_i = 1$ (respectively, $b_i = 0$).

We first argue that no state X_p of M' can be labeled both an accepting and rejecting state. For if $b_i = 1$ and $b_j = 0$, then the conjunct $(z_i^n \neq z_j^n)$ appears in $F(S)$; hence, z_i^n and z_j^n must have different colors in P'.

Next we show that M is in fact deterministic. For suppose that some state X_p has transitions to X_q and X_r, and that both transitions are labeled with the same symbol. Then there exist i_1, i_2 and j_1, j_2 such that $z_{i_1}^{j_1} \in X_p$ and $z_{i_1}^{j_1+1} \in X_q$, and $z_{i_2}^{j_2} \in X_p$ and $z_{i_2}^{j_2+1} \in X_r$. Furthermore, we must have $w_{i_1}^{j_1+1} = w_{i_2}^{j_2+1}$ since both transitions have the same label. But then the conjunct

$$\left(\left(z_{i_1}^{j_1} = z_{i_2}^{j_2}\right) \rightarrow \left(z_{i_1}^{j_1+1} = z_{i_2}^{j_2+1}\right)\right)$$

must appear in $F(S)$, and this conjunct is violated P', a contradiction. Thus, M' is deterministic.

These arguments prove that M' is a well-defined DFA. To see that M' is consistent with S, consider the computation of M' on any w_i in S. The sequence of states visited on this computation is just $EC_{P'}(z_i^1), \dots, EC_{P'}(z_i^n)$, where $EC_{P'}(z_i^j)$ denotes the equivalence class of the variable z_i^j in the coloring P'. The final state $EC_{P'}(z_i^n)$ is by definition of M' either an accept state or a reject state according to whether $b_i = 1$ or $b_i = 0$. □

Note that if $|S|$ is the number of bits in the sample S and $|F(S)|$ denotes the number of bits in the formula $F(S)$, then in Theorem 12 we have $|F(S)| = \Theta(|S|^2 \log |S|) = O(|S|^{2+\gamma})$ for any $\gamma > 0$ for $|S|$ sufficiently large. This means that if an algorithm colors $F(S)$ using at most $opt_{\mathrm{FC}}(F(S))^{\alpha} |F(S)|^{\beta}$ for some $\alpha \geq 1$ and $\beta < \frac{1}{2}$, then for $|S|$ sufficiently large we can use the reduction of Theorem 12 to find a DFA consistent with S that has at most $k^{\alpha} |S|^{\beta'}$ for some $\beta' < 1$, contradicting Theorem 10. Thus we have:

THEOREM 13. *The problems of inverting the RSA encryption function, recognizing quadratic residues and factoring Blum integers are polynomial-time reducible to approximating the optimal solution to an instance F of the formula coloring problem by a model P of F satisfying*

$$|P| \leq opt_{FC}(F)^{\alpha} |F|^{\beta}$$

for any $\alpha \geq 1$ *and* $0 \leq \beta < 1/2$.

Figure 1 summarizes hardness results for coloring a formula F using at most $f(opt_{\mathrm{FC}}(F))g(|F|)$ colors for various functions f and g, where an entry

Difficulty of coloring F using $A \cdot B$ colors	$A = 1$	$A = \mid F \mid^{1/29}$	$A = \mid F \mid^{0.499}$	$A = \mid F \mid$
$B = opt_{\text{FC}}(F)$	NP-hard	NP-hard	Factoring	P
$B = 1.99 \cdots opt_{\text{FC}}(F)$	NP-hard	Factoring	Factoring	P
$B = (opt_{\text{FC}}(F))^{\alpha}$ any fixed $\alpha \geq 0$	NP-hard	Factoring	Factoring	P

FIG. 1. Difficulty of approximating the formula coloring problem using at most $A \cdot B$ colors on input formula F. The constant $0.499 \cdots$ is intended to indicate any value strictly smaller than $1/2$; the constant $1/29$ is determined from the paper of Pitt and Warmuth.

"NP-hard" indicates that such an approximation is NP-hard, "Factoring" indicates that such an approximation is as hard as factoring Blum integers (or recognizing quadratic residues or inverting the RSA function), and "P" indicates there is a polynomial-time algorithm achieving this approximation factor. The NP-hardness results follow from Garey and Johnson [1979] and Pitt and Warmuth [1989].

REFERENCES

ADLEMAN, L., MANDERS, K., AND MILLER, G. 1977. On taking roots in finite fields. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*. IEEE, New York, pp. 175–178.

AHO, A., HOPCROFT, J., AND ULLMAN, J. 1974. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass.

ALEXI, W., CHOR, B., GOLDREICH, O., AND SCHNORR, C. P. 1988. RSA and Rabin functions: Certain parts are as hard as the whole. *SIAM J. Comput. 17*, 2, 194–209.

ANGLUIN, D. 1982. Lecture notes on the complexity of some problems in number theory. Tech Rep. TR-243. Comput. Sci. Dept., Yale Univ., New Haven, Conn.

ANGLUIN, D. 1987. Learning regular sets from queries and counterexamples. *Inf. Comput. 75*, 87–106.

ANGLUIN, D., AND KHARITONOV, M. 1991. When won't membership queries help? In *Proceedings of the 23rd ACM Symposium on the Theory of Computing* (New Orleans, La., May 6–8) ACM, New York, pp. 444–454.

ANGLUIN, D., AND LAIRD, P. 1988. Learning from noisy examples. *Mach. Learn. 2*, 319–342.

ANGLUIN, D., AND VALIANT, L. C. 1979. Fast probabilistic algorithms for Hamiltonian circuits and matchings. *J. Comput. Syst. Sci. 18*, 155–193.

BEAME, P. W., COOK, S. A., AND HOOVER, H. J. 1986. Log depth circuits for division and related problems. *SIAM J. Comput. 15*, 4 (1986), 994–1003.

BLUM, A. 1989. An $\tilde{O}(n^{0.4})$-approximation algorithm for 3-coloring. In *Proceedings of the 21st ACM Symposium on the Theory of Computing* (Seattle, Wash., May 15–17). ACM, New York, pp. 535–542.

BLUM, A., AND RIVEST, R. L. 1988. Training a 3-node neural network is NP-complete. In *Proceedings of the 1988 Workshop on Computational Learning Theory*. Morgan-Kaufmann, San Mateo, Calif., pp. 9–18.

BLUM, M., AND MICALI, S. 1984. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput. 13*, 4, 850–864.

BLUMER, A., EHRENFEUCHT, A., HAUSSLER, D., AND WARMUTH, M. 1987. Occam's razor. *Inf. Proc. Lett. 24*, 377–380.

BLUMER, A., EHRENFEUCHT, A., HAUSSLER, D., AND WARMUTH, M. 1989. Learnability and the Vapnik–Chervonenkis dimension. *J. ACM 36*, 4, (Oct.) 929–965.

CHANDRA, A. K., STOCKMEYER, L. J., AND VISHKIN, U. 1984. Constant depth reducibility. *SIAM J. Comput. 13*, 2, 423–432.

CHERNOFF, H. 1952. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Ann. Math. Stat. 23*, 493–509.

DIFFIE, W., AND HELLMAN, M. 1976. New directions in cryptography. *IEEE Trans. Inf. Theory 22*, 644–654.

GAREY, M., AND JOHNSON, D. 1979. *Computers and intractability: A guide to the theory of NP-completeness*. Freeman.

GOLD, E. M. 1978. Complexity of automaton identification from given data. *Inf. Cont. 37*, 302–320.

GOLDREICH, O., GOLDWASSER, S., AND MICALI, S. 1986. How to construct random functions. *J. ACM 33*, 4 (Oct.), 792–807.

HANCOCK, T. 1989. On the difficulty of finding small consistent decision trees. Harvard University, unpublished manuscript.

HAUSSLER, D., KEARNS, M., LITTLESTONE, N., AND WARMUTH, M. 1988. Equivalence of models for polynomial learnability. In *Proceedings of the 1988 Workshop on Computational Learning Theory*. Morgan-Kaufmann, San Mateo, Calif., pp. 42–55.

JUDD, S. 1984. Learning in neural networks. In *Proceedings of the 1988 Workshop on Computational Learning Theory*. Morgan-Kaufmann, San Mateo, Calif., pp. 2–8.

KEARNS, M., LI, M., PITT, L., AND VALIANT, L. 1987. On the learnability of Boolean formulae. In *Proceedings of the 19th ACM Symposium on the Theory of Computing* (New York, N.Y., May 25–27). ACM, New York, pp. 285–295.

KEARNS, M., AND PITT, L. 1989. A polynomial-time algorithm for learning *k*-variable pattern languages from examples. In *Proceedings of the 1989 Workshop on Computational Learning Theory*. Morgan-Kaufmann, San Mateo, Calif., pp. 57–71.

KRANAKIS, E. 1986. *Primality and cryptography*. Wiley, New York.

LEVIN, L. A. 1985. One-way functions and pseudorandom generators. In *Proceedings of the 17th ACM Symposium on the Theory of Computing* (Providence, R.I., May 6–8), ACM, New York, pp. 363–365.

LI, M., AND VAZIRANI, U. 1988. On the learnability of finite automata. In *Proceedings of the 1988 Workshop on Computational Learning Theory*. Morgan-Kaufmann, San Mateo, Calif., pp. 359–370.

PITT, L., AND VALIANT, L. G. 1988. Computational limitations on learning from examples. *J. ACM 35*, 4 (Oct.) 965–984.

PITT, L., AND WARMUTH, M. K. 1988. Reductions among prediction problems: on the difficulty of predicting automata. In *Proceedings of the 3rd IEEE Conference on Structure in Complexity Theory*. IEEE, New York, pp. 60–69.

PITT, L., AND WARMUTH, M. K. 1989. The minimum consistent DFA problem cannot be approximated within any polynomial. In *Proceedings of the 21st ACM Symposium on the Theory of Computing* (Seattle, Wash., May 15–17). ACM, New York, pp. 421–432.

RABIN, M. O., 1979. Digital signatures and public key functions as intractable as factoring. Tech. Rep. TM-212. Lab. Comput. Sci., MIT Cambridge, Mass.

REIF, J. 1987. On threshold circuits and polynomial computations. In *Proceedings of the 2nd Structure in Complexity Theory Conference*. pp. 118–125.

RIVEST, R. L., SHAMIR, A., AND ADLEMAN, L. 1978. A method for obtaining digital signatures and public key cryptosystems. *Commun. ACM 21*, 2 (Feb.), 120–126.

SCHAPIRE, R. 1989. On the strength of weak learnability. In *Proceedings of the 30th IEEE Symposium on the Foundations of Computer Science*. IEEE, New York, pp. 28–33.

VALIANT, L. G. 1989. A theory of the learnable. *Commun. ACM 27*, 11 (Nov.) 1134–1142.

WIGDERSON, A. 1983. A new approximate graph coloring algorithm. In *Proceedings of the 14th ACM Symposium on the Theory of Computing* (San Francisco, Calif., May 5–7). ACM, New York, pp. 325–329.

YAO, A. C. 1982. Theory and application of trapdoor functions. In *Proceedings of the 23rd IEEE Symposium on the Foundations of Computer Science*. IEEE, New York, pp. 80–91.

RECEIVED NOVEMBER 1988; REVISED JULY 1992; ACCEPTED JULY 1992