Computational Sample Complexity and Attribute-Efficient Learning

Rocco A. Servedio*

Division of Engineering and Applied Sciences

Harvard University

Cambridge, MA 02138

rocco@deas.harvard.edu

July 26, 1999

*Supported in part by an NSF Graduate Fellowship and by grants NSF-CCR-95-04436 and ONR-N00014-96-1-0550.

Proposed running head: Sample complexity and attribute-efficiency

Author to whom proofs should be sent:

Rocco Servedio

Division of Engineering and Applied Sciences

40 Oxford Street

Harvard University

Cambridge, MA 02138

Abstract

Two fundamental measures of the efficiency of a learning algorithm are its running time and the number of examples it requires (its sample complexity). In this paper we demonstrate that even for simple concept classes, an inherent tradeoff can exist between running time and sample complexity. We present a concept class of 1-decision lists and prove that while a computationally unbounded learner can learn the class from $O(1)$ examples, under a standard cryptographic assumption any polynomial-time learner requires almost $\Theta(n)$ examples. Using a different construction, we present a concept class of k-decision lists which exhibits a similar but stronger gap in sample complexity. These results strengthen the results of Decatur, Goldreich and Ron [9] on distribution-free computational sample complexity and come within a logarithmic factor of the largest possible gap for concept classes of k-decision lists. Finally, we construct a concept class of decision lists which can be learned attribute-efficiently and can be learned in polynomial time but cannot be learned attribute-efficiently in polynomial time. This is the first result which shows that attribute-efficient learning can be computationally hard. The main tools used are one-way permutations, error-correcting codes and pseudorandom generators.

List of symbols:

Θ capital theta

Ω capital omega

$\prod$ product (capital pi)

ϵ epsilon

δ delta

α alpha

τ tau

$\overline{x}$ x bar

$\mathcal{D}$ calligraphic D

$\mathcal{I}$ calligraphic I

$\mathcal{T}$ calligraphic T

$\mathcal{S}$ calligraphic S

$\mathcal{C}$ calligraphic C

$\mathcal{G}$ calligraphic G

$\mathcal{U}$ calligraphic U

$\mathcal{Z}$ calligraphic Z

O capital oh

0 zero

o lowercase oh

ℓ lowercase ell

1 one

∞ infinity

$\langle$ left angle

$\rangle$ right angle

$\circ$ circle

$\rightarrow$ right arrow

$*$ asterix

$\subseteq$ subset

$\in$ element of

$\overbrace{\hspace{3em}}$ overbrace

1 Introduction

A broad research goal in computational learning theory is to discover fast (i.e., polynomial-time) learning algorithms for various concept classes. Another broal goal is to discover algorithms which can learn from few examples. This paper studies how these two goals can sometimes be mutually exclusive.

In Valiant's Probably Approximately Correct model of concept learning [29], the *sample complexity* of a concept class C is the minimum number of labelled examples which any successful learning algorithm for C must require. Lower bounds on sample complexity were first given by Ehrenfeucht et. al. in [10], where it was shown that any algorithm which learns a concept class of Vapnik-Chervonenkis dimension d must use $\Omega(d/\epsilon)$ examples. Similar bounds were subsequently established in [18] for a generalization of the PAC model to learning probabilistic concepts. However, these results do not address the question of how many examples a *polynomial-time* learning algorithm must require. (Of course, since drawing an example takes at least one time step, a polynomial-time learning algorithm can require at most polynomially many examples.)

The first indication that polynomial-time learning might be computationally hard for the unrestricted class of boolean circuits of polynomial size was given by Valiant in his original paper [29]. Kearns and Valiant [19] subsequently established the existence of concept classes of polynomial-size boolean formulae which are hard for any polynomial-time algorithm to learn but are learnable from polynomially many examples by a computationally unbounded algorithm. Their results were later refined and extended by Kharitonov [21].

Decatur, Goldreich and Ron [9] were the first to study concept classes in which polynomial-time learning is doable but requires more examples than learning using a computationally unbounded algorithm. Among other results, they proved the following theorem:

Theorem 1 *Let $p(n)$ be any polynomial such that $p(n) \geq n$. If one-way functions exist, then there is a concept class C of polynomial-size circuits such that*

- *any polynomial-time PAC learning algorithm for C must use $\Omega(p(n)/\epsilon)$ examples,*
- *there is a computationally unbounded PAC learning algorithm for C which uses $O(n/\epsilon)$ examples.*

The proof of Theorem 1 relies essentially on the idea that a pseudorandom generator can be used to hide information from a computationally bounded learner but not from a computationally unbounded learner.

The first contribution of the present paper is to strengthen the results of Decatur et. al. by establishing stronger gaps of this sort and showing that such gaps hold even for concept classes whose concepts have an extremely simple and natural representation as decision lists; we do this via two constructions. Our first construction yields a concept class whose concepts are 1-decision lists and which has the following property: a computationally unbounded learner can learn the class from $O(1/\epsilon)$ examples, but under a standard cryptographic assumption any polynomial-time learner requires almost $\Theta(n/\epsilon)$ examples. This construction uses error-correcting codes and requires only very basic cryptography (the notion of a one-way function). Our second construction makes more extensive use of cryptographic machinery to prove the following result: for any $k \geq 1$ there is a concept class of k-decision lists which a computationally unbounded algorithm can learn from $O(1/\epsilon)$ examples, but under a widely held cryptographic assumption any polynomial-time learner requires $\Theta(n^k/\epsilon)$ examples. This is within a logarithmic factor of the largest possible gap for concept classes of k-decision lists.

Our last main result concerns *attribute-efficient* learning. Loosely speaking, a concept class C is said to be attribute-efficiently learnable if there is a learning algorithm for C which requires only $\text{poly}(\text{size}(c), \log n)/\epsilon$ examples to learn any concept $c \in C$ over n variables (we give a precise definition in Section 5). Attribute-efficient learning algorithms are particularly useful when the target concept depends on few variables but n, the total number of variables, is large. Results of Haussler [16] and Littlestone [22] yield attribute-efficient learning algorithms for k-CNF and k-DNF formulae; more recent results on attribute-efficiency can be found in [4, 7, 28]. Blum [2] and Valiant [30] have each posed the question of whether there exists a polynomial-time attribute-efficient learning algorithm for the concept class of 1-decision lists of length k. Such an algorithm would run in time $\text{poly}(n)$ (this is unavoidable since each example is of length n) but would require only $\text{poly}(k, \log n)/\epsilon$ examples, and could potentially be a useful tool in machine learning.

We take a step toward answering Blum and Valiant's question by providing the first proof that attribute-efficient learning can be computationally hard. We do this by exhibiting a concept class of decision lists which can be learned in polynomial time and can be learned by a computationally unbounded attribute-efficient learning algorithm but cannot (under a plausible cryptographic assumption) be learned in polynomial time by any attribute-efficient algorithm.

A common paradigm for concepts and examples is used throughout this paper. In each of the concept classes which we consider, each concept is associated with a secret key; it is easy to exactly identify the target concept if this key is known. Also, in each of our constructions examples come in two types, which we call *useful* and *useless*. Useful examples each contain an encrypted version of the secret key as well as a small amount of unencrypted information about the target concept. Useless examples all have label 0 and contain no information about the target concept.

Our constructions are based on the following simple idea: a computationally unbounded learning algorithm can decrypt the secret key and hence can learn the target concept exactly from a single useful example. Consequently, such a learning algorithm requires few examples. On the other hand, a polynomial-time learner cannot decrypt the secret key; instead, it can only use the small amount of unencrypted information in each useful example. Hence a polynomial-time learner will need many useful examples in order to acquire a significant amount of information about the target concept.

The remainder of the paper is structured as follows. Section 2 contains preliminary definitions which we use throughout the paper. In Section 3 we exhibit a concept class of 1-decision lists which has a substantial gap between its information-theoretic and computational sample complexities. Section 4 contains analogous results (obtained using a different construction) for a concept class of k-decision lists. In Section 5 we show that attribute-efficient learning of polynomial-time learnable concept classes can be computationally hard. Section 6 concludes with some open problems.

2 Preliminaries

In the boolean PAC learning model, a *concept* $c : \{0,1\}^n \rightarrow \{0,1\}$ is a boolean function and a *concept class* C is a collection of concepts. The learner has access to an *example oracle* $EX(c, \mathcal{D}_n)$ which, on each call, takes one time step and outputs a labelled boolean example $\langle x, c(x) \rangle$ where x is drawn from the distribution $\mathcal{D}_n$ over $\{0,1\}^n$. Given two boolean functions h, c and a distribution $\mathcal{D}_n$ over $\{0,1\}^n$, we say that h *is* ϵ*-accurate under* $\mathcal{D}_n$ *with respect to* c if $\Pr_{x \in \mathcal{D}_n}[h(x) \neq c(x)] < \epsilon$;

alternatively, such a function h is said to *ϵ-approximate the concept c under $\mathcal{D}_n$*. An algorithm L is said to be a *PAC learning algorithm for concept class C* if the following condition holds: for every distribution $\mathcal{D}_n$, for every $c \in C$ and for every $0 < \epsilon, \delta < 1$, if L is given access to $EX(c, \mathcal{D}_n)$ then with probability at least $1 - \delta$, algorithm L outputs a hypothesis h which ϵ-approximates c under $\mathcal{D}_n$. See [20] for a thorough discussion of PAC learning.

The following definitions are from [9]: The *distribution free information theoretic sample complexity* of a concept class C, denoted $\mathcal{ITSC}(C; n, \epsilon)$, is the minimum sample size (as a function of n and ϵ) needed for PAC learning the class C with accuracy ϵ and confidence $\delta = 9/10$, where no computational limitations exist on the learning algorithms which may be used. The *distribution free computational sample complexity* of a concept class C, denoted $\mathcal{CSC}(C; n, \epsilon)$, is the minimum sample size (as a function of n and ϵ) needed for PAC learning the class C with accuracy ϵ and confidence $\delta = 9/10$, where the learning algorithm must operate in polynomial (in n and $1/\epsilon$) time.

A *k-decision list of length ℓ* over the boolean variables $x_1, \dots, x_n$ is a boolean function L which is represented by a list of ℓ pairs $(m_1, b_1), (m_2, b_2), \dots, (m_\ell, b_\ell)$, where each m_i is a conjunction of at most k literals over $x_1, \dots, x_n$ and each b_i is either 0 or 1. Given any $x \in \{0,1\}^n$, the value of $L(x)$ is b_i if i is the smallest index such that m_i is satisfied; if no m_i is satisfied then $L(x) = 0$.

We write $x \circ y$ to denote the concatenation of binary strings x, y and $|x|$ to denote the length of x. We say that a permutation $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is *length-preserving* if $|f(x)| = |x|$ for all $x \in \{0,1\}^*$.

A *length-preserving one-way permutation* is a length-preserving permutation f which has the following properties: there is a deterministic polynomial-time algorithm which computes f, but for sufficiently large n there is no probabilistic polynomial-time algorithm which inverts f on a

1/poly(n) fraction of $\{0,1\}^n$.

3 A Construction Using Error-Correcting Codes

In this section we prove the following theorem:

Theorem 2 *Let $0 < \tau < 1$ be any constant. If length-preserving one-way permutations exist, then there is a concept class C_τ which has*

$$\mathcal{ITSC}(C_\tau; n, \epsilon) = O(1/\epsilon)$$

and

$$\Omega(n^{1-\tau}/\epsilon) = \mathcal{CSC}(C_\tau; n, \epsilon) = O(n/\epsilon)$$

where each concept in C_τ is a 1-decision list over $\{0,1\}^n$.

3.1 Error-correcting codes

We need some basic terminology from the theory of error-correcting codes. As in [26, 27] we say that a *binary code of block length ℓ and rate r_ℓ* is a code in which codewords are ℓ bits long, where $r_\ell \cdot \ell$ positions are "message bits" that can be filled with any combination of 0's and 1's and the remaining $(1 - r_\ell)\ell$ positions have their contents determined by the message bits. Let $A_\ell : \{0,1\}^{r_\ell \cdot \ell} \to \{0,1\}^\ell$ be a binary code of block length ℓ and rate r_ℓ; for $x \in \{0,1\}^{r_\ell \cdot \ell}$, the j-th bit of the ℓ-bit string $A_\ell(x)$ is denoted by $A_\ell(x)_j$.

We say that the code A_ℓ has *minimum relative distance* δ_ℓ if any pair of distinct codewords $\{A_\ell(x), A_\ell(y)\}$ has Hamming distance at least $\delta_\ell \cdot \ell$. For $\alpha_\ell < \delta_\ell/2$, we say that an algorithm D

is an α_ℓ-*decoding algorithm for* A_ℓ if, when D is given a string $z \in \{0,1\}^\ell$ which has Hamming distance at most $\alpha_\ell \cdot \ell$ from some codeword $A_\ell(x)$, the algorithm D outputs x.

The papers [26, 27] each contain versions of the following important theorem:

Theorem 3 [Sipser, Spielman] *There exists a polynomial-time-constructible family* $\{A_\ell\}_{\ell=1}^{\infty}$ *of binary error-correcting codes, where each* A_ℓ *is a function from* $\{0,1\}^{r_\ell \cdot \ell}$ *to* $\{0,1\}^\ell$*, with the following properties:*

- $\lim_{\ell \to \infty} r_\ell > 0$, $\lim_{\ell \to \infty} \delta_\ell > 0$ *and* $\lim_{\ell \to \infty} \alpha_\ell > 0$,
- *For each* ℓ*, there is an* α_ℓ*-decoding algorithm for* A_ℓ *which runs in time poly*(ℓ).

Recall that in the PAC framework, a learning algorithm succeeds if it can construct a hypothesis which agrees with the target concept on all but a small fraction of points. In the construction which we use to prove Theorem 2, such a hypothesis will yield a string z which is close to a codeword $A_\ell(x)$. By the polynomial-time decoding algorithm of Theorem 3, the ability to find an accurate hypothesis in polynomial time would thus imply the ability to find x in polynomial time. However, we will show that this is impossible (under a cryptographic assumption) if few examples have been seen.

3.2 The Concept Class C_τ

Before giving a formal description of the concept class C_τ, we mention that in this concept class the secret key for each concept is composed of many small subkeys, each of which is encrypted separately. The reason is that each useful example will contain a small amount of unencrypted information about exactly one of the subkeys. Hence, unless many useful examples have been seen, there will exist subkeys about which no unencrypted information has been revealed.

Before we can describe the concept class C_τ, we must first specify some numerical parameters. Let $\{A_\ell\}_{\ell=1}^{\infty}$ be a fixed family of error-correcting codes with the properties stated in Theorem 3 (so the block length is r_ℓ, the minimum relative distance is δ_ℓ, and there is a poly(ℓ)-time α_ℓ-decoding algorithm for A_ℓ). Given a positive integer m, let $q = m^{\frac{1-\tau}{\tau}}$, let ℓ be the smallest integer such that $r_\ell \cdot \ell \geq m$, and let $n = mq + q\ell$. The following facts can be easily verified:

Fact 4 $\alpha_\ell = \Theta(1)$, $r_\ell = \Theta(1)$ *(follows from Theorem 3).*

Fact 5 $\ell = \Theta(m)$ *(follows from definition of ℓ and Fact 4).*

Fact 6 $n = \Theta(mq)$ *(follows from definition of n and Fact 5).*

Fact 7 $m = \Theta(n^\tau)$, $q = \Theta(n^{1-\tau})$ *(follows from definition of q and Fact 6).*

Let f be a fixed length-preserving one-way permutation. The set $(\{0,1\}^m)^q$ will be our set of secret keys; each secret key $v = (v^1, \ldots, v^q) \in (\{0,1\}^m)^q$ is composed of q subkeys each of which is m bits long. The class C_τ has a concept c_v for each secret key v.

We now describe a concept c_v over $\{0,1\}^n$. If c_v is the target concept, then an example $x \in \{0,1\}^n$ is said to be *useful* if $x_1 \cdots x_{mq} = f(v^1) \circ \cdots \circ f(v^q)$ and is *useless* otherwise. Given an example $x \in \{0,1\}^n$, let $i(x) \in \{1, \ldots, q\}, j(x) \in \{1, \ldots, \ell\}$ be such that $x_{mq+(i(x)-1)\ell+j(x)}$ is the first bit of $x_{mq+1} \cdots x_{mq+q\ell}$ whose value is 1. (If $x_{mq+1} = \cdots = x_{mq+q\ell} = 0$ then $i(x) = j(x) = 0$.) Figure 1 illustrates the structure of a useful example. The concept c_v is defined as follows:

- $c_v(x) = 0$ if x is useless,
- $c_v(x) = A_\ell(v^{i(x)})_{j(x)}$, the $j(x)$-th bit of $A_\ell(v^{i(x)})$, if x is useful and $i(x), j(x) \geq 1$. If x is useful and $i(x) = j(x) = 0$ then $c_v(x) = 0$.

3.3 Proof of Theorem 2

First we establish that c_v is a 1-decision list. For each $1 \leq k \leq mq$, let ℓ_k denote the literal $\overline{x}_k$ if the k-th bit of $f(v^1) \circ \cdots \circ f(v^q)$ is 1, and let ℓ_k denote x_k otherwise. Then the following is seen to be a 1-decision list which computes c_v :

$$(\ell_1, 0), \ldots, (\ell_{mq}, 0), (x_{mq+1}, A_\ell(v^1)_1), (x_{mq+2}, A_\ell(v^1)_2), \ldots,$$

$$(x_{mq+(i-1)\ell+j}, A_\ell(v^i)_j), \ldots, (x_{mq+q\ell}, A_\ell(v^q)_\ell).$$

This is because the first mq pairs ensure that all useless examples will be labelled 0, and the ordering of the last $q\ell$ pairs ensures that the label of each useful example will be as described in Section 3.2.

To prove the information-theoretic sample complexity upper bound, we must show that under any distribution at most $O(1/\epsilon)$ examples are required. Since each positive example contains $f(v^1) \circ \ldots \circ f(v^q)$, a computationally unbounded learner can learn the target concept exactly from a single positive example by inverting the one-way permutation f to find each v^i and then computing each $A_\ell(v^i)$. Such a learner can thus make, say, $20/\epsilon$ calls to the oracle $EX(c, \mathcal{D}_n)$ and output the identically zero hypothesis if all examples are negative, otherwise output the correct hypothesis as described above. A simple calculation shows that this algorithm finds an ϵ-accurate hypothesis with high probability, and hence $\mathcal{ITSC}(C_\tau; n, \epsilon) = O(1/\epsilon)$.

It remains to bound the computational sample complexity of C_τ; we begin with the simpler upper bound. We say that a 1-decision list over $\{0,1\}^n$ is *well-structured* if its length is exactly n and it has the following structure: for $1 \leq t \leq mq$ the t-th pair of the decision list has x_t or $\overline{x}_t$ as its conjunction and has 0 as its output bit, and for $mq+1 \leq t \leq mq+q\ell$ the t-th term of the decision list has x_t as its conjunction. Given a sample S of examples which are labelled according to the concept c_v, it is easy for a polynomial-time algorithm to find a well-structured 1-decision list which is consistent with S. Any positive example of S identifies the first mq literals of the well-structured 1-decision list, and each useful example provides the output bit for one of the last $q\ell$ pairs (note that it is possible to identify useful examples as long as S contains at least one positive example). Since there are 2^n well-structured 1-decision lists, Occam's Razor [6] immediately implies that $O(n/\epsilon)$ examples suffice for this polynomial-time learning algorithm.

Now we show the lower bound on $\mathcal{CSC}(C_\tau; n, \epsilon)$. The idea of the proof is as follows: we will exhibit a particular distribution on $\{0,1\}^n$ and show that any polynomial-time learning algorithm for C_τ which learns to accuracy ϵ using $q\alpha_\ell/18\epsilon$ examples drawn from this distribution can be used to invert the one-way permutation f in polynomial time with nonnegligible success probability. This contradiction implies that every polynomial-time learning algorithm must use more than $q\alpha_\ell/18\epsilon$ examples. Since Facts 4 and 7 together imply that $q\alpha_\ell/18\epsilon = \Theta(n^{1-\tau}/\epsilon)$, this will prove that $\mathcal{CSC}(C_\tau; n, \epsilon) = \Omega(n^{1-\tau}/\epsilon)$ as desired.

Let $\mathcal{D}_n$ be the distribution on $\{0,1\}^n$ which assigns weight $3\epsilon/(\alpha_\ell \cdot q\ell)$ to each of the $q\ell$ useful examples

$$\{f(v^1) \circ \cdots f(v^q) \circ 0^k 1 0^{q\ell-k-1}\}_{k=0}^{q\ell-1}.$$

and assigns the remaining $1 - 3\epsilon/\alpha_\ell$ weight to the single useless example 0^n. (Recall from Section 3.1 that α_ℓ is the frequency of errors up to which the error-correcting codes of [26, 27] can be successfully decoded using a poly(ℓ)-time algorithm.) Note that under this distribution, each bit of each $A_\ell(v^i)$ is equally likely to occur as the label of a useful example.

Let S be a sample of $q\alpha_\ell/18\epsilon$ examples which are drawn from $EX(c, \mathcal{D}_n)$. Since the expected number of useful examples in S is $q/6$, a simple application of Chernoff bounds (see, e.g., [1, 20]) shows that with overwhelmingly high probability the sample S will contain at least one useful example. Since each useful example contains $f(v^1) \circ \cdots f(v^q)$ as its mq-bit prefix, it follows that with overwhelmingly high probability a polynomial-time learning algorithm which has access to S can identify the strings $f(v^1), \ldots, f(v^q)$.

Now suppose that a polynomial-time learning algorithm could achieve an ϵ-accurate hypothesis from the sample S. Since the learning algorithm knows $f(v^1), \ldots, f(v^q)$, the algorithm can apply its ϵ-accurate hypothesis to each of the $q\ell$ useful examples described above. The algorithm can thus construct $B^1, \ldots, B^q$ in polynomial time, where each B^i is an ℓ-bit string which is the learning algorithm's "guess" at the string $A_\ell(v^i)$. Since by assumption the hypothesis is ϵ-accurate under $\mathcal{D}_n$, at most an $\alpha_\ell/3$ fraction of the $q\ell$ total bits in the strings $B^1, \ldots, B^q$ can be incorrect. By Markov's inequality, at least $2/3$ of the B^i's must each have at most $\alpha_\ell \cdot \ell$ incorrect bits; consequently, by using the polynomial-time decoding algorithm for A_ℓ, the learning algorithm can find at least $2/3$ of the subkeys $\{v^1, \ldots, v^q\}$ in polynomial time. However, since as noted earlier the expected number of useful examples in S is $q/6$, by a straightforward application of Chernoff bounds it is extremely unlikely that S contained more than $q/3$ useful examples. As a result, we have that with very high probability the polynomial-time learner has received no information at all (other than $f(v^i)$) for at least $2/3$ of the subkeys. It follows that the poly(n)-time learner was able to invert f on at least $1/3$ of the $f(v^i)$'s "from scratch." Since each subkey v^i is $m = \Theta(n^\tau)$ bits long, though, our poly(n)-time learner is also a poly(m)-time algorithm; but this contradicts the fact that f is one-way. Hence $\mathcal{CSC}(C_\tau; n, \epsilon) > q\alpha_\ell/18\epsilon = \Omega(n^{1-\tau}/\epsilon)$. ■

4 A Stronger Gap

In this section we prove the following:

Theorem 8 *Let $k \geq 1$ be any integer. If length-preserving one-way permutations exist, then there is a concept class C_k which has*

$$\mathcal{ITSC}(C_k; n, \epsilon) = O(1/\epsilon)$$

and

$$\mathcal{CSC}(C_k; n, \epsilon) = \Theta(n^k/\epsilon)$$

where each concept in C_k is a k-decision list over $\{0,1\}^n$.

This strengthens the result of Decatur et. al. [9] on distribution-free computational versus information-theoretic sample complexity in two ways: we improve the upper bound on information-theoretic sample complexity from $O(n/\epsilon)$ to $O(1/\epsilon)$, and we prove this stronger gap for the much simpler class of k-decision lists (rather than poly-size circuits).

4.1 Cryptographic Preliminaries

The cryptographic definitions we present in this section are slightly more general than the standard definitions (we will need this extra generality in Section 5). Throughout this section the function $q(\cdot)$ denotes an arbitrary nondecreasing integer-valued function which satisfies $q(n) \geq n$. The standard cryptographic definitions are obtained if $q(n)$ is taken to be a polynomial in n (the reader is encouraged to verify this for herself). Intuitively, the faster $q(n)$ grows, the less plausible are the resulting cryptographic assumptions. In Section 5 we will take $q(n)$ to be a function which grows very slightly faster than any polynomial in n; this is a stronger-than-standard cryptographic assumption, but as we discuss in Section 5, we believe that it is still quite a reasonable assumption.

The notation "$x \in \mathcal{D}_n$" means that x is selected from the set $\{0,1\}^n$ according to distribution $\mathcal{D}_n$; we write $\mathcal{U}_n$ to denote the uniform distribution over $\{0,1\}^n$.

Definition 9 *A length-preserving permutation f is said to be* $q(n)$-one-way *if the following conditions hold:*

- *there is a deterministic poly-time algorithm which computes $f(x)$,*
- *for all probabilistic poly$(q(n))$-time algorithms A, for all polynomials Q, for all sufficiently large n, we have*

$$\Pr_{x \in \mathcal{U}_n}[A(f(x)) = x] < \frac{1}{Q(q(n))}.$$

Definition 10 *Let f be a length-preserving permutation. A polynomial-time computable predicate $B : \{0,1\}^* \to \{0,1\}$ is said to be a $q(n)$*-hard-core predicate of f *if the following condition holds: for all probabilistic poly$(q(n))$-time decision algorithms A, for all polynomials Q, for all sufficiently large n, we have*

$$\Pr_{x \in \mathcal{U}_n}[A(f(x)) = B(x)] < \frac{1}{2} + \frac{1}{Q(q(n))}.$$

Suppose that g is a length-preserving poly(n)-one-way permutation. Let $x = p \circ y$ where $|p| = |y| = n$, and let f be the function defined as $f(x) = p \circ g(y)$. It is easy to check that f is also a length-preserving poly(n)-one-way permutation. Goldreich and Levin [14] have shown that $B(x) = \sum_{i=1}^{n} p_i y_i \pmod 2$ is a poly(n)-hard-core predicate for f (see Appendix C.2 of [13] for a very readable proof of this result). An entirely straightforward modification of their proof shows that if g is a length-preserving $q(n)$-one-way permutation, then f is a length-preserving $q(n)$-one-way permutation and $B(x)$ is a $q(n)$-hard-core predicate for f.

Definition 11 *A family of probability distributions* $\{\mathcal{X}_{q(n)}\}$ *on* $\{0,1\}^{q(n)}$ *is* $q(n)$-pseudorandom *if* $\{\mathcal{X}_{q(n)}\}$ *is poly($q(n)$)-time indistinguishable from* $\{\mathcal{U}_{q(n)}\}$*. That is, for all probabilistic poly($q(n)$)-time decision algorithms A, for all polynomials Q, for all sufficiently large n, we have*

$$\left| \Pr_{z \in \mathcal{X}_{q(n)}} [A(z) = 1] - \Pr_{z \in \mathcal{U}_{q(n)}} [A(z) = 1] \right| < \frac{1}{Q(q(n))}.$$

Definition 12 *A poly($q(n)$)-time deterministic algorithm* $G : \{0,1\}^n \to \{0,1\}^{q(n)}$ *is said to be a* $q(n)$-pseudorandom generator *if* $\{\mathcal{G}_{q(n)}\}$ *is a $q(n)$-pseudorandom family of distributions, where* $\mathcal{G}_{q(n)}$ *is the distribution on* $\{0,1\}^{q(n)}$ *obtained as follows: to select* $z \in \mathcal{G}_{q(n)}$*, pick* $x \in \mathcal{U}_n$ *and set* $z = G(x)$*. We write* $G(z)_i$ *to denote the* i*-th bit of* $G(z)$*.*

Now we can state the following useful theorem:

Theorem 13 *Let f be a length-preserving $q(n)$-one-way permutation and let B be a $q(n)$-hard-core predicate of f. Let* $G : \{0,1\}^n \to \{0,1\}^{q(n)}$ *be defined as follows:*

$$G(x) = B(x) \circ B(f(x)) \circ B(f(f(x))) \circ \cdots \circ B(f^{q(n)-1}(x)).$$

Then G is a $q(n)$-pseudorandom generator. Moreover, the distributions

$$\{G(z) \circ f^{q(n)}(z)\}_{z \in \mathcal{U}_n}$$

and

$$\{w \circ f^{q(n)}(z)\}_{w \in \mathcal{U}_{q(n)}, z \in \mathcal{U}_n}$$

are poly$(q(n))$-time indistinguishable.

In the case where $q(n)$ is a polynomial, this theorem is a standard result (see, e.g., Proposition 3.17 of [12]). This construction of a pseudorandom generator, along with the definition of a pseudorandom generator, is originally from [5]. The proof of the more general theorem which is stated above (where $q(n)$ need not be a polynomial) is a straighforward modification of the proof of the standard result, entirely analogous to the modification of the Goldreich-Levin theorem mentioned above.

Observation 14 *We note that by Theorem 13, even if a poly$(q(n))$-time algorithm is given $f^{q(n)}(z)$ along with some bits of $G(z)$, the algorithm still cannot predict the unseen bits of $G(z)$ with accuracy significantly better than $1/2$. This is because the ability to do such prediction would violate the poly$(q(n))$-time indistinguishability which is asserted in Theorem 13, since clearly no poly$(q(n))$-time algorithm (in fact, no algorithm at all) can successfully predict the unseen bits of a uniformly selected random string.*

4.2 The Concept Class C_k

Let f be a length-preserving one-way permutation. The set $\{0,1\}^m$ will be our set of secret keys. As discussed in Section 4.1 we can suppose without loss of generality that f has a hard-core predicate. Let G be the $\binom{n}{k}$-pseudorandom generator associated with f whose existence is asserted by Theorem 13 (so G maps inputs of length m to outputs of length $\binom{m}{k}$.) Let $n = 2m$. For $1 \leq i \leq \binom{m}{k}$, let T_i

denote the i-th k-element subset of the set $\{m+1,\ldots,2m\}$ under some fixed and easily computable ordering (e.g., lexicographic), and let z_i be the conjunction $\prod_{j\in T_i} x_j$. Given any input $x \in \{0,1\}^n$, let $i(x)$ be the the smallest index in $\{1,\ldots,\binom{m}{k}\}$ such that $z_{i(x)}$ is satisfied by x (if no $z_{i(x)}$ is satisfied by x for $1 \le i(x) \le \binom{m}{k}$ then let $i(x) = 0$).

The class C_k has a concept c_v for each secret key $v \in \{0,1\}^m$. If c_v is the target concept, then an example x is *useful* if $x_1 \cdots x_m = f^{\binom{m}{k}}(v)$ and is *useless* otherwise. (As in Section 4.1, $f^{\binom{m}{k}}(v)$ denotes the result of applying f exactly $\binom{m}{k}$ times to v.) The concept c_v is defined as follows:

- $c_v(x) = 0$ if x is useless,
- $c_v(x) = G(v)_{i(x)}$, the $i(x)$-th bit of $G(v)$, if x is useful and $i(x) \ge 1$. If x is useful and $i(x) = 0$ then $c_v(x) = 0$.

4.3 Proof of Theorem 8

First we show that c_v is a k-decision list. For each $1 \le j \le m$, let ℓ_j denote the literal $\overline{x}_j$ if the j-th bit of $f^{\binom{m}{k}}(v)$ is 1, and let ℓ_j denote x_j otherwise. The following k-decision list of length $m + \binom{m}{k}$ computes c_v :

$$(\ell_1, 0), \ldots, (\ell_m, 0), (z_1, G(v)_1), \ldots, (z_{\binom{m}{k}}, G(v)_{\binom{m}{k}}).$$

To bound $\mathcal{ITSC}(C_k; n, \epsilon)$, note that upon receiving a single positive example, an unbounded learner can invert $f^{\binom{m}{k}}(v)$ to find v (this is possible since f is a permutation) and thus learn the target concept c_v exactly. As in the proof of Theorem 2, it follows that $\mathcal{ITSC}(C_k; n, \epsilon) = O(1/\epsilon)$.

An analogous argument to the computational sample complexity upper bound proof of Theorem 2 establishes that $\mathcal{CSC}(C_k; n, \epsilon) = O(\binom{m}{k}/\epsilon) = O(n^k/\epsilon)$.

For the computational lower bound, consider the distribution $\mathcal{D}_n$ over $\{0,1\}^n$ which assigns weight $1-6\epsilon$ to the single useless example 0^n and assigns weight $6\epsilon/\binom{m}{k}$ to each of the $\binom{m}{k}$ useful examples

$$\{f^{\binom{m}{k}}(v) \circ T_i\}_{i=1}^{\binom{m}{k}}$$

(here we are viewing each T_i as an m-bit string in the obvious way). Let S be a sample of $\binom{m}{k}/24\epsilon$ examples which are drawn from $EX(c, \mathcal{D}_n)$. By Theorem 13, we have that the string-valued random variables

$$\{G(z) \circ f^{\binom{m}{k}}(z)\}_{z \in \mathcal{U}_m}$$

and

$$\{w \circ f^{\binom{m}{k}}(z)\}_{w \in \mathcal{U}_{\binom{m}{k}}, z \in \mathcal{U}_m}$$

are polynomial-time indistinguishable. Consequently, even though a polynomial-time learner which has drawn the sample S may discover $f^{\binom{m}{k}}(v)$ from any positive example, by Observation 14 such a learner cannot predict the bits of $G(v)$ which it has not seen with accuracy significantly better than 1/2. Since the expected number of useful examples in S is $\binom{m}{k}/4$, a straightforward application of Chernoff bounds shows that with very high probability S will contain fewer than $\binom{m}{k}/2$ useful examples, and thus with very high probability the polynomial-time learner will have seen at most half of the $\binom{m}{k}$ bits of $G(v)$. Since useful examples which correspond to the unseen bits of $G(v)$ have weight at least 3ϵ under the distribution $\mathcal{D}_n$, the polynomial-time learner's overall error rate will exceed ϵ (with very high probability it will be at least $3\epsilon/2$). Hence $\binom{m}{k}/24\epsilon$ examples do not suffice for polynomial-time learnability, and we have that $\mathcal{CSC}(C_k; n, \epsilon) \geq \binom{m}{k}/24\epsilon = \Theta(n^k/\epsilon)$. ■

It is interesting to contrast the bounds given in Theorem 8 with other known bounds. The upper bound on information-theoretic sample complexity which is given in Theorem 8 is the best possible for nontrivial concept classes. Rivest's polynomial-time algorithm for learning k-decision lists [25] requires $O(\frac{n^k}{\epsilon} \min\{\log n, \log \frac{1}{\epsilon}\})$ examples; thus our lower bound on computational sample complexity could be improved by at most a logarithmic factor for concept classes of k-decision lists. Ehrenfeucht et. al. [10] have shown that $\Omega(n^k/\epsilon)$ examples are required for information-theoretic reasons for learning k-decision lists. Our Theorem 8 shows that $\Omega(n^k/\epsilon)$ examples can be required for learning subclasses of k-decision lists for computational reasons even in the absence of any information-theoretic barriers to learning from fewer examples.

5 Hardness of Attribute-Efficient Learning

We now turn our attention to *attribute-efficient* learning algorithms. These algorithms require very few examples relative to the total number of input variables (i.e., attributes), and hence have exceptionally good performance over high-dimensional input spaces which contain many irrelevant attributes. This property has led researchers to apply attribute-efficient learning algorithms to real-world problems such as calendar scheduling [3], text categorization [8], and context-sensitive spelling correction [11].

Attribute-efficiency has chiefly been studied in the *on-line mistake-bound* model of concept learning which was introduced in [22, 23]. In this model learning proceeds in a series of trials, where in each trial the learner is given an unlabelled boolean example $x \in \{0,1\}^n$ and must predict the value $c(x)$; after each prediction the learner is told the true value of $c(x)$ and can update its hypothesis. The mistake bound of a learning algorithm on a target concept c is measured by the worst-case number of mistakes that the algorithm makes over all (possibly infinite) sequences of examples, and the mistake bound of a learning algorithm on a concept class C is the worst-case mistake bound across all concepts $c \in C$. A learning algorithm L for a concept class C over $\{0,1\}^n$ is said to run in polynomial time if the mistake bound of L on C is $\mathrm{poly}(n)$ and the time required by L to make its prediction and update its hypothesis on each example is $\mathrm{poly}(n)$.

A boolean function c over $x_1, \ldots, x_n$ is said to *depend* on a variable x_i if there are two vectors $y, z \in \{0,1\}^n$ which have $y_j = z_j$ for all $j \neq i$, $y_i \neq z_i$, and $c(y) \neq c(z)$. Let C be a class of boolean functions on $x_1, \ldots, x_n$ each of which depends on at most r variables and each of which has a description of length at most s under some reasonable encoding scheme. Following [4], we say that a learning algorithm L for C in the mistake-bound model is *attribute-efficient* if the mistake bound of L on any concept $c \in C$ is $\mathrm{poly}(r, s, \log n)$.

In this section we provide strong evidence that there are concept classes learnable in polynomial time for which attribute-efficient learning is information-theoretically possible but computationally hard. We do this by proving the following theorem:

Theorem 15 *For any integer $c \geq 2$, let $\log(c,n)$ denote*

$$\overbrace{\log \log \cdots \log}^{c} n.$$

Let $q(c,n) = n^{\log(c,n)}$. If there is some integer $c \geq 2$ such that length-preserving $q(c,n)$-one-way permutations exist, then there exists a concept class C of $O(\log(c,n))$-decision lists which has the following properties in the mistake-bound model:

- *A computationally unbounded learner can learn C with at most 1 mistake,*
- *C can be learned in polynomial time,*
- *C cannot be learned in polynomial time by an attribute-efficient learning algorithm.*

While the existence of length-preserving $q(c,n)$-one-way permutations is not a standard cryptographic assumption, we believe that it is still a very reasonable assumption. As we discuss in Appendix B, if there does not exist a collection of $q(c,n)$-one-way permutations (where each permutation in the collection is defined over a finite domain)[1], then there must exist algorithms for factoring Blum integers which are far more powerful than any currently known algorithms for this well-studied problem.

5.1 Proof of Theorem 5

First we define the concept class C. This construction is similar to the construction of Section 4.2 but with some different parameters.

Let f be a length-preserving $q(c,n)$-one-way permutation; as before, the set $\{0,1\}^m$ will be our set of secret keys. Let G be the $q(c,n)$-pseudorandom generator whose existence is guaranteed by Theorem 13. Let n be such that $m = n^{1/\log(c,n)}$, and let $k(n)$ denote the least integer such that $\binom{m}{k(n)} \geq q(c,m)$. The following claims can be easily verified:

Claim 16 *For q, m, and k as defined above, we have:*

1. $q(c,m) = n^{1-o(1)}$.

2. $k(n) = O(\log(c,n))$.

Proof: See Appendix A. ■

For $i = 1, \ldots, q(c,m)$ let T_i denote the i-th $k(n)$-element subset of the set $\{m+1, \ldots, 2m\}$ and

[1] This is a slightly different notion of a one-way function than the notion which we have been using thus far in the paper. See Section 2.4.2 of [12] for a discussion of the relationship between these two notions.

let z_i be the conjunction $\prod_{j \in T_i} x_j$. Given any input $x \in \{0,1\}^n$, let $i(x)$ be the the smallest index in $\{1, \ldots, q(c,m)\}$ such that $z_{i(x)}$ is satisfied by x (if no z_i is satisfied by x then $i(x) = 0$).

For each secret key $v \in \{0,1\}^m$, there exists a corresponding concept $c_v \in C$. If c_v is the target concept, then an example x is *useful* if $x_1 \cdots x_m = f^{q(c,m)}(v)$ and is *useless* otherwise. The concept c_v is defined as follows:

- $c_v(x) = 0$ if x is useless,
- $c_v(x) = G(v)_{i(x)}$, the $i(x)$-th bit of $G(v)$, if x is useful and $i(x) \geq 1$. If x is useful and $i(x) = 0$ then $c_v(x) = 0$.

Now we prove that C has the properties listed in Theorem 15. The first property is easy: a computationally unbounded learner can achieve a mistake bound of 1 by predicting 0 until it makes a mistake. From this positive example the unbounded learner can compute v (by inverting $f^{q(c,m)}(v)$) and hence can exactly identify the target concept.

For the second property, note that the concept c_v can be represented as a $O(\log(c,n))$-decision list of length at most $m + q(c,m)$. As in the computational sample complexity upper bound of Theorem 2, a polynomial-time algorithm can learn the first m pairs of the target decision list from a single positive example, and will make at most one mistake for each of the last $q(c,m)$ pairs of the decision list. Since $q(c,m) = n^{1-o(1)}$, such an algorithm will make poly(n) mistakes, and it follows that C can be learned in polynomial time.

Now suppose that there is a polynomial-time attribute-efficient learning algorithm for the concept class C. Since each concept c_v has an m-bit description (the string v), we have that $s = O(m)$. Each function c_v depends only on the variables $x_1, \ldots, x_{2m}$, so r is also $O(m)$. Hence any attribute-efficient learning algorithm for C must have mistake bound $\mathrm{poly}(m, \log n) = \mathrm{poly}(m)$.

Consider the $q(c,m)$-long sequence S of useful examples $\{f^{q(c,m)}(v) \circ T_i \circ 0^{n-2m}\}_{i=1}^{q(c,m)}$. From Theorem 13, we have that no $\mathrm{poly}(q(c,m))$-time learning algorithm can predict an unseen bit of $G(v)$ with accuracy significantly better than $1/2$. Since $q(c,m) = n^{1-o(1)}$, we have that $\mathrm{poly}(q(c,m)) = \mathrm{poly}(n)$. Consequently, any $\mathrm{poly}(n)$-time learning algorithm will have probability $1/2$ of making a mistake on each example in the sequence S; it follows that with very high probability, any $\mathrm{poly}(n)$-time algorithm will make $\Theta(q(c,m))$ mistakes on S. But this means that no polynomial-time attribute-efficient learning algorithm can exist for C, since $\mathrm{poly}(m) = o(q(c,m))$. ■

6 Conclusion

We have demonstrated the existence of various subclasses of k-decision lists which can be information-theoretically learned from a constant number of examples but which requires any polynomial-time learner to use $\Theta(n^k)$ examples. We have also shown that under a plausible cryptographic assumption, attribute-efficient learning is computationally hard but information-theoretically possible for a polynomial-time learnable class whose concepts are $O(\log(c,n))$-decision lists.

Many directions remain for future research. For one thing, it would be interesting to see if gaps such as the ones we have demonstrated in Sections 3 and 4 can be shown for concept classes whose concepts are even simpler than decision lists, and to determine whether the cryptographic assumptions which are used to establish these gaps can be weakened. In a similar vein, it would be nice to be able to replace the cryptographic assumption which is used to prove Theorem 15 with a more standard (i.e., weaker) cryptographic assumption.

As noted in Section 5, each concept of the class described there has a natural m-bit representation. However, to represent a concept in this class as a decision list requires more than m bits; our attribute-efficient hardness result relies on the m-bit representation. It would be very interesting to see an attribute-efficient hardness result for a concept class of decision lists where the description length of a concept is taken to be the length of the decision list which computes it.

A related goal is to prove computational hardness results for attribute-efficient learning of simpler concept classes. In particular, let L_k denote the class of 1-decision lists of length k. Using the Halving Algorithm [22] the class L_k can be learned with $O(k \log n)$ mistakes, but no algorithm running in time $n^{O(1)}$ is known which makes $\mathrm{poly}(k, \log n)$ mistakes (there is a polynomial-time algorithm which make $O(kn)$ mistakes [2], and Littlestone's Winnow algorithm [22] makes $2^{O(k)} \log n$ mistakes). Perhaps techniques such as those used in this paper can help resolve whether L_k is attribute-efficiently learnable in polynomial time.

7 Acknowledgements

We thank Amos Beimel and Les Valiant for helpful suggestions which greatly improved the structure of the paper.

References

[1] D. Angluin and L. G. Valiant, Fast probabilistic algorithms for Hamiltonian circuits and matchings, *J. Comput. System Sci.* **48** (1979), 155–193.

[2] A. Blum, On-line algorithms in machine learning, available at http://www.cs.cmu.edu/~avrim/Papers/pubs.html, 1996.

[3] A. Blum, Empirical support for winnow and weighted-majority algorithms: results on a calendar scheduling domain, *Machine Learning* **26** (1997), 5–23.

[4] A. Blum, L. Hellerstein, and N. Littlestone, Learning in the presence of finitely or infinitely many irrelevant attributes, *J. Comput. System Sci.* **50** (1995), 32–40.

[5] M. Blum and S. Micali, How to generate cryptographically strong sequences of pseudo-random bits, *SIAM J. Comput.* **13** (1984), 850-864.

[6] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, Occam's razor, *Inform. Process. Lett.* **24** (1987), 377–380.

[7] N. Bshouty and L. Hellerstein, Attribute efficient learning with queries, *J. Comput. System Sci.* **56** (1998), 310–319.

[8] I. Dagan, Y. Karov, and D. Roth, Mistake-driven learning in text categorization, in "2nd Conf. on Empirical Methods in Natural Language Processing (EMNLP-97)," 1997.

[9] S. E. Decatur, O. Goldreich, and D. Ron, Computational sample complexity, in "Proc. Tenth Ann. Conf. on Comp. Learning Theory," ACM Press, New York, 1997, 130–142.

[10] A. Ehrenfeucht, D. Haussler, M. Kearns, and L. Valiant, A general lower bound on the number of examples needed for learning, *Inform. and Comput.* **82**(3) (1989), 247–261.

[11] A.R. Golding and D. Roth, A winnow-based approach to spelling correction, *Machine Learning* **34** (1999), 107-130.

[12] O. Goldreich, Foundations of cryptography (Fragments of a Book), available at `http://www.wisdom.weizmann.ac.il/~oded/frag.html`, 1995.

[13] O. Goldreich, "Modern cryptography, probabilistic proofs and pseudo-randomness," Algorithms and Combinatorics series (Vol. 17), Springer, 1998.

[14] O. Goldreich and L. Levin, A hard-core predicate for all one-way functions, in "Proc. 21st Ann. Symp. on Theory of Comp.," ACM Press, New York, 1995, 25–32.

[15] S. Goldwasser and M. Bellare, Lecture notes on cryptography, available at `http://www-cse.ucsd.edu/users/mihir/papers/gb.html`, 1996.

[16] D. Haussler, Quantifying inductive bias: AI learning algorithms and Valiant's learning framework, *Artificial Intelligence* **36**(2), (1988), 177–221.

[17] M. Kearns, M. Li, L. Pitt, and L. Valiant, Recent results on boolean concept learning, in "Proc. 4th Int. Workshop on Machine Learning," Morgan Kaufmann, Los Altos, CA, 1987, 337–352.

[18] M. J. Kearns and R. E. Schapire, Efficient distribution-free learning of probabilistic concepts, *J. Comput. System Sci.* **48** (1994), 464-497.

[19] M. Kearns and L. G. Valiant, Cryptographic limitations on learning boolean formulae and finite automata, *J. ACM* **41**(1), (1994), 67–95.

[20] M. Kearns and U. Vazirani, "An introduction to computational learning theory," MIT Press, Cambridge, MA, 1994.

[21] M. Kharitonov, Cryptographic lower bounds for learnability of boolean functions on the uniform distribution, *J. Comput. System Sci.* **50** (1995), 600–610.

[22] N. Littlestone, Learning quickly when irrelevant attributes abound: a new linear-threshold learning algorithm, *Machine Learning* **2** (1988), 285–318.

[23] N. Littlestone, "Mistake bounds and logarithmic linear-threshold learning algorithms," Ph.D. thesis, Technical Report UCSC-CRL-89-11, Univ. of Calif., Santa Cruz, 1989.

[24] M. Rabin, Digitalized signatures as intractable as factorization, Technical Report MIT/LCS/TR-212, MIT Lab. for Comp. Sci., 1979.

[25] R. L. Rivest, Learning decision lists, *Machine Learning* **2**(3) (1987), 229–246.

[26] M. Sipser and D. A. Spielman, Expander codes, *IEEE Trans. Inf. Theory* **42**(6) 1996, 1710-1722.

[27] D. A. Spielman, Linear-time encodable and decodable error-correcting codes, *IEEE Trans. Inf. Theory* **42**(6) 1996, 1723-1731.

[28] R. Uehara, K. Tsuchida, and I. Wegener, Optimal attribute-efficient learning of disjunction, parity, and threshold functions, in "Proc. 3rd European Conf. on Comp. Learning Theory," LNAI, Vol. 1208, Springer, pp. 1761-184, 1997.

[29] L. G. Valiant, A theory of the learnable, *Comm. ACM* **27**(11) (1984), 1134–1142.

[30] L. G. Valiant, Projection learning, in "Proc. Eleventh Ann. Conf. on Comp. Learning Theory," ACM Press, New York, 1998, 287–293.

A Proof of Claim 16

In this appendix we prove Claim 16. We first remind the reader of what this claim says:

Claim 16 *For q, m, and k as defined in Section 5.1, we have:*

1. $q(c, m) = n^{1-o(1)}$.

2. $k(n) = O(\log(c, n))$.

Proof: Recall that $\log(2, n) = \log\log n$ and $\log(c, n) = \log(c-1, \log n)$ for $c > 2$. We first note that

$$\begin{aligned}
\log(c, m) &= \log(c, n^{1/\log(c,n)}) \\
&= \log(c-1, \log(n^{1/\log(c,n)})) \\
&= \log(c-1, \log n / \log(c, n)) \\
&= \log(c-2, \log(\log n / \log(c, n))) \\
&= \log(c-2, \log\log n - \log(c+1, n))
\end{aligned}$$

It follows that

$$\begin{aligned}
q(c, m) &= m^{\log(c,m)} \\
&= \left(n^{1/\log(c,n)}\right)^{\log(c-2, \log\log n - \log(c+1,n))} \\
&= n^{1-o(1)}
\end{aligned} \tag{1}$$

which proves the first part of the claim.

For the bound on $k(n)$, recall that $k(n)$ is the least integer such that $\binom{m}{k(n)} \geq q(c, m)$. We prove that $k(n) \leq 2\log(c, n)$ by showing that $\binom{m}{2\log(c,n)} \geq q(c, m)$. To see this, note that by the standard inequality $\binom{x}{y} \geq (x/y)^y$, we have

$$\binom{m}{2\log(c,n)} = \binom{n^{1/\log(c,n)}}{2\log(c,n)} \geq \left(\frac{n^{1/\log(c,n)}}{2\log(c,n)}\right)^{2\log(c,n)} = \frac{n^2}{(2\log(c,n))^{2\log(c,n)}} = n^{2-o(1)}.$$

Since $q(c,m) = n^{1-o(1)}$, the claim is proved. ∎

B Factoring Blum Integers and One-Way Permutations

Let J_n be the set of all n-bit primes which are congruent to 3 mod 4. An *n-bit Blum integer* is an integer of the form $N = p_1 \cdot p_2$ where $p_1 \neq p_2$ and $p_1, p_2 \in J_{n/2}$. Given an n-bit Blum integer N, let Q_N denote the set of quadratic residues modulo N, and let $f_N : Q_N \to Q_N$ be the function $f_N(z) = z^2 \bmod N$. Blum and Williams have noted that f_N is a permutation on Q_N (see Lemma 2.3.29 of [15] for a proof).

As discussed in Section 2.4.3 of [12], it is widely believed that the collection

$$\{f_N\}_{Blum} \equiv \{f_N : N \text{ is a Blum integer}\}$$

has the following properties:

1. Given n, it is computationally easy to uniformly select a random n-bit Blum integer N (with negligible error probability) by taking $N = p_1 \cdot p_2$, where p_1, p_2 are uniformly selected $n/2$-bit primes with $p_1 < p_2$ and $p_1 \equiv p_2 \equiv 3 \bmod 4$ (this assumes that the set $J_{n/2}$ of such primes is non-negligibly dense in the set of $n/2$-bit integers).

2. Given an n-bit Blum integer N, it is easy to uniformly select a quadratic residue $r \bmod N$ (this can be done by squaring a randomly chosen element of Z_N^*).

3. For every sufficiently large n, for every probabilistic poly(n)-time algorithm A', for every polynomial Q, given N and r selected as described above, we have

$$\Pr[A'(f_N(r), N) = r] < \frac{1}{Q(n)}.$$

As in [12], we say that $\{f_N\}_{Blum}$ is a *collection of one-way permutations.*

Now consider the following variant of Property 3:

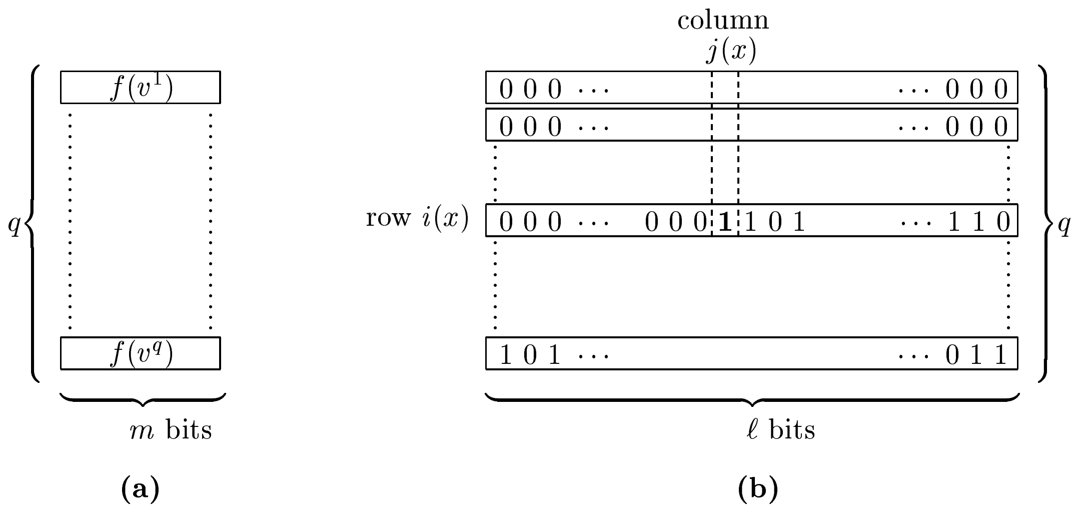
3$'$. For every sufficiently large n, for every probabilistic poly($q(n)$)-time algorithm A', for every polynomial Q, given N and r selected as described above, we have

$$\Pr[A'(f_N(r), N) = r] < \frac{1}{Q(q(n))}.$$

If $\{f_N\}_{Blum}$ satisfies Properties 1, 2 and 3$'$ then we say that $\{f_N\}_{Blum}$ is a *collection of* $q(n)$-*one-way permutations.*

Rabin's results in [24] yield the following: Suppose that A' is a probabilistic poly($q(n)$)-time algorithm which, when given as input the pair $(f_N(r), N)$ with N and r selected as described above, outputs r with probability at least $1/\text{poly}(q(n))$. Then there is a poly($q(n)$)-time algorithm A which, when given a uniformly selected n-bit Blum integer N, factors N with success probability at least $1/\text{poly}(q(n))$.

Thus, if $\{f_N\}_{Blum}$ is not a collection of $q(c,n)$-one-way permutations, then there is a $q(c,n)$-time algorithm which factors randomly selected Blum integers with success probability at least $1/\text{poly}(q(c,n))$. This would be an extremely surprising result, since to date the best algorithms for factoring n-bit Blum integers require time $2^{O(\sqrt{n \log n})}$, which is a much faster-growing function than $q(c,n)$ for all $c \geq 2$ (recall that $q(2,n) = n^{\log\log n}$, $q(3,n) = n^{\log\log\log n}$, etc.).



$f(v^1)$
$f(v^q)$
q
m bits
(a)
column
$j(x)$
row $i(x)$
0 0 0 ⋯ ⋯ 0 0 0
0 0 0 ⋯ ⋯ 0 0 0
0 0 0 ⋯ 0 0 0 **1** 1 0 1 ⋯ 1 1 0
1 0 1 ⋯ ⋯ 0 1 1
q
ℓ bits
(b)

Figure 1

Figure 1: A useful example $\langle x, A_\ell(v^{i(x)})_{j(x)}\rangle$. Part **(a)** depicts the mq-bit prefix of x; since x is useful this must be $f(v^1) \circ \cdots \circ f(v^q)$. Part **(b)** depicts the $q\ell$-bit suffix $x_{mq+1} \ldots x_n$, where the bit $x_{mq+(r-1)\ell+c}$ is in row r and column c for $1 \le r \le q$, $1 \le c \le \ell$. As shown in **(b)**, the values of $i(x)$ and $j(x)$ are determined by the location of the first 1 in the $q\ell$-bit suffix of x.