INFORMATION AND COMPUTATION **138**, 23–48 (1997)
ARTICLE NO. IC972648

Efficient Learning of Typical Finite Automata from Random Walks

Yoav Freund and Michael Kearns

AT&T Labs, 180 Park Avenue, Florham Park, New Jersey 07932
E-mail: {yoav,mkearns}@research.att.com

Dana Ron

Massachusetts Institute of Technology, Cambridge, Massachusetts 02139
E-mail: danar@theory.lcs.mit.edu

Ronitt Rubinfeld

Cornell University, Ithaca, New York 14853
E-mail: ronitt@cs.cornell.edu

Robert E. Schapire

AT&T Labs, 180 Park Avenue, Florham Park, New Jersey 07932
E-mail: schapire@research.at.com

and

Linda Sellie

University of Chicago, Chicago, Illinois 60637
E-mail: sellie@cs.uchicago.edu

This paper describes new and efficient algorithms for learning deterministic finite automata. Our approach is primarily distinguished by two features: (1) the adoption of an average-case setting to model the ''typical'' labeling of a finite automaton, while retaining a worst-case model for the underlying graph of the automaton, along with (2) a learning model in which the learner is not provided with the means to experiment with the machine, but rather must learn solely by observing the automaton's output behavior on a random input sequence. The main contribution of this paper is in presenting the first efficient algorithms for learning non-trivial classes of automata in an entirely passive learning model. We adopt an on-line learning model in which the learner is asked to predict the output of the next state, given the next symbol of the random input sequence; the goal of the learner is to make as few prediction mistakes as possible. Assuming the learner has a means of resetting the target machine to a fixed start state, we first present an efficient algorithm that makes an expected polynomial number of mistakes in this model. Next, we show how this first algorithm can be used as a subroutine by a second algorithm that also makes a polynomial number of mistakes even in the absence of a reset. Along the way, we prove a number of combinatorial results for randomly labeled automata. We also show that the labeling of the states and the bits of the input sequence need not be truly random, but merely *semi-random*. Finally, we discuss an extension of our results to a model in which automata are used to represent distributions over binary strings.

1. INTRODUCTION

In this paper, we describe new and efficient algorithms for learning deterministic finite automata. Our approach is primarily distinguished by two features:

- The adoption of an average-case setting to model the "typical" *labeling* of a finite automaton, while a worst-case model is retained for the underlying *graph* of the automaton.
- A learning model in which the learner is not provided with the means to experiment with the machine, but rather must learn solely by observing the automaton's output behavior on a random input sequence.

Viewed another way, we may think of the learner as a robot taking a random walk in a finite-state environment whose topology may be adversarially chosen, but where the sensory information available to the robot from state to state has limited dependence.

An important feature of our algorithms is their robustness to a weakening of the randomness assumptions. For instance, it is sufficient that the states be labeled in a manner that is both partially adversarial and partially random; this is discussed further momentarily.

One of our main motivations in studying a model mixing worst-case and average-case analyses was the hope for efficient passive learning algorithms that remained in the gap between the pioneering work of Trakhtenbrot and Barzdin' [25] and the intractability results discussed in the history section below for passive learning in models where both the state graph and the labeling are worst-case. In the former work, there is an implicit solution to the problem of efficient passive learning when both the graph and labeling are chosen randomly, and there are also many exponential-time algorithms in mixed models similar to those we consider. The choice of a random state graph, however, tends to greatly simplify the problem of learning, due in part to the probable proximity of all states to one another. The problem of efficient learning when the graph is chosen adversarially but the labeling randomly was essentially left open by Trakhtenbrot and Barzdin'. In providing efficient algorithms for passive learning in this case, we demonstrate that the topology of the graph cannot be the only source of the apparent worst-case difficulty of learning automata passively (at least for the random walk model we consider), but rather that the difficulty seems to arise from particular combinations of topology and labeling.

We give algorithms that learn with respect to a worst-case choice of the underlying directed state graph (transition function) of the target automaton along with a random choice of the $\{+, -\}$-labeling (output function) of the states. Throughout most of the paper, we assume that the label at each state is determined by the outcome of an unbiased coin flip; however, our algorithms are robust in the sense that they continue to work even when there is only limited independence among the state labels. Limited independence is formalized using the semi-random model of Santha and Vazirani [24], in which each label is determined by the outcome of a coin flip of variable bias chosen by an omniscient adversary to be between Δ and $1-\Delta$. In addition to investigations of their properties as a computational resource [6, 24, 27, 28], semi-random sources have also been used as a model for studying the complexity of graph coloring that falls between worst-case and average-case (random) models [5], and as a model for biased random walks on graphs [3].

In our setting, the learner observes the behavior of the unknown machine on a random walk. (As for the random labeling function, the walk may actually be only semi-random.) At each step, the learner must predict the output of the machine (the current state label) when it is fed the next randomly chosen input symbol. The goal of the learner is to minimize the expected number of *prediction mistakes*, where the expectation is taken over the choice of the random walk.

Our first algorithm, for any state graph, and with high probability over the random labeling of the state graph, will make only an expected polynomial number of mistakes. In fact, we show that this algorithm has the stronger property of *reliability* [22]: if allowed to output either a $\{+, -\}$-prediction or the special symbol "?" (called a *default mistake*) the algorithm will make no prediction mistakes and only an expected polynomial number of default mistakes. In other words, every $\{+, -\}$-prediction made by the algorithm will be correct.

This first algorithm assumes that the target machine is returned to a fixed start state following each default mistake. The random walk observed by the learner is then continued from this start state. Thus, the learner is essentially provided with a *reset* mechanism (but is charged one default mistake each time it is used), so the data seen by the learner can be thought of as a sample of finite length input/output behaviors of the target machine. This view allows us to prove performance bounds in an average-case version of the popular PAC model of learning.

In our second algorithm, we are able to remove the need for the reset. The second algorithm thus learns by observing the output of a single, unbroken random walk. For this, we sacrifice reliability, but are nevertheless able to prove polynomial bounds on the absolute number of prediction mistakes and the expected number of default mistakes. The removal of the reset mechanism is particularly important in the motivation offered above of a robot exploring an environment; in such a setting, each step of the robot's random walk is irreversible and the robot must learn to "orient" itself in its environment solely on the basis of its observations.

Finally we give a modification of our algorithm which applies to the setting of learning probability distributions over binary strings. The algorithm of Ron *et al.* [23] which learns acyclic probabilistic finite automata builds on the algorithm given here. Their algorithm has been successfully applied to the problem of handwriting recognition and the modeling of phonemes. to construct multiple-pronunciation models for spoken words.

Following a history of the problem of learning finite automata and the definitions of our models, the paper is organized into three technical sections: one describing each of the two algorithms and a third describing extensions. Each of the two algorithm sections consists of two parts. In the first part, we define "nice" combinatorial properties of finite automata that hold with high probability over a random (or semi-random) labeling of any state graph. The second part then describes how the algorithm exploits these properties in order to efficiently learn the target automaton.

For our first algorithm, which assumes the reset mechanism, the important combinatorial object is the *signature* of a state of the machine. Informally, the signature of a particular state describes the input/output behavior of the machine for some fixed "neighborhood" of the given state. Our algorithm exploits a theorem stating that with high probability the signature of every state is unique.

For our first algorithm, which assumes the reset mechanism, the important combinatorial object is the *signature* of a state of the machine. Informally, the signature of a state q is a complete description of the output behavior of all states within a small distance of q. Our algorithm exploits a theorem stating that with high probability the signature of every state is unique.

For our second algorithm, which eliminates the reset mechanism, the important combinatorial object is the *local homing sequence*, which is related to but weaker than the homing sequences used by Rivest and Schapire [20]. Informally, a (local) homing sequence is an input sequence which, when executed, may allow the learner to determine "where it is" in the machine based on the observed output sequence. The algorithm hinges on our theorem stating that with high probability a short local homing sequence exists for every state, and proceeds to identify this sequence by simulating many copies of our first algorithm.

In the final sections, we explore both the relaxation of randomness to semi-randomness already mentioned and a modification of our algorithms for learning probability distributions over binary strings.

2. HISTORY OF THE PROBLEM

The problem of learning finite automata has an extensive history. To understand this history, we broadly divide results into those addressing the *passive* learning of finite automata, in which the learner has no control over the data it receives, and those addressing the *active* learning of finite automata, in which we introduce mechanisms for the learner to experiment with the target machine.

The intractability results for various passive learning models begin with the work of Gold [11] and Angluin [2], who proved that the problem of finding the smallest automaton consistent with a set of accepted and rejected strings is NP-complete. This result left open the possibility of efficiently approximating the smallest machine, which was later dismissed in a very strong sense by the NP-hardness results of Pitt and Warmuth [18, 19]. Such results imply the intractability of learning finite automata (when using finite automata as the hypothesis representation) in a variety of passive learning models, including the well-studied "probably approximately correct" (or PAC) model introduced by Valiant [26] and the mistake-bound models of Littlestone [16] and Haussler *et al.* [12].

These results demonstrated the intractability of passively learning finite automaton when we insist that the hypothesis constructed by the learner also be a finite automaton, but did not address the complexity of passively learning finite automata by more powerful representations. Although such changes of hypothesis representation can in some instances provably reduce the complexity of certain learning problems from NP-hard to polynomial time [17], Kearns and Valiant [14] demonstrated that this is not the case for finite automata by proving that passive learning in the PAC model by any reasonable representation is as hard as breaking various cryptographic protocols that are based on factoring. This again implies intractability for the same problem in the mistake-bound models.

The situation becomes considerably brighter when we turn to the problem of actively learning finite automata. Angluin [1], elaborating on an algorithm of Gold [10], proved that if a learning algorithm is provided with both passive counterexamples to its current hypothesis automaton (that is, arbitrary strings on which the hypothesis automaton disagrees with the target) and the ability to actively query the target machine on any string of the algorithm's choosing (known as *membership queries*), then finite automata are learnable in polynomial time. This result provides an efficient algorithm for learning finite automata in the PAC model augmented with membership queries. Together with the results of Kearns and Valiant [14], this separates (under cryptographic assumptions) the PAC model and the PAC model with membership queries, so experimentation provably helps for learning finite automata in the PAC setting.

The Angluin and Gold algorithm essentially assumes the existence of an experimentation mechanism that can be *reset*: on each membership query x, the target automaton is executed on x and the final state label is given to the learner; the target machine is then reset in preparation for the next query. Rivest and Schapire [20, 21] considered the natural extension in which we regard the target automaton as representing some aspect of the learner's physical environment, and in which experimentation is allowed, but without a reset. The problem becomes more difficult since the learner is not directly provided with the means to "orient" itself in the target machine. Nevertheless, Rivest and Schapire extend Angluin's algorithm and provide a polynomial time algorithm for inferring any finite automaton from a single continuous walk on the target automaton. Variants of this algorithm have recently been examined by Dean *et al.* [7].

All of the results discussed above, whether in a passive or an active model, have considered the worst-case complexity of learning: to be considered efficient, algorithms must have small running time on any finite automaton. However, average-case models have been examined in the extensive work of Trakhtenbrot and Barzdin' [4, 25]. In addition to providing a large number of extremely useful theorems on the combinatorics of finite automata, Trakhtenbrot and Barzdin' also give many polynomial time and exponential time inference algorithms in both worst-case models, and models in which some property of the target machine (such as the labeling or the graph structure) is chosen randomly. For an interesting empirical study of the performance of one of these algorithms, see Lang's paper [15] on experiments he conducted using automata that were chosen partially or completely at random.

The primary lesson to be gleaned from the previous work on learning finite automata is that passive learning of automata tends to be computationally difficult. Thus far, only the introduction of active experimentation has allowed us to ease this intractability. The main contribution of this paper is in presenting the first efficient algorithms for learning non-trivial classes of automata in an entirely passive learning model.

3. PRELIMINARIES

A deterministic finite automaton is a tuple $M = (Q, \tau, \gamma, q_0)$. Here Q is a finite non-empty set of n *states*; $\tau\colon Q \times \{0, 1\} \to Q$ is the *transition function*; $\gamma\colon Q \to \{+, -\}$ is the *labeling function*; and $q_0 \in Q$ is the designated *start state*. Notice that here we have assumed an *input alphabet* of $\{0, 1\}$ and an *output alphabet* of $\{+, -\}$ for simplicity; the results presented here all generalize to larger input and output alphabets.

We adopt the following notational conventions: For q a state of M and $x \in \{0, 1\}^*$ we denote by $qx \in Q$ the state of M reached by executing the walk x from state q (as defined by τ). We denote by $q\langle x\rangle$ the sequence of length $|x| + 1$ of $\{+, -\}$ labels observed along this walk. Finally, we write $x^{(i)}$ to denote the length i prefix of x.

The state set Q and the transition function τ taken together (but without the state labeling γ) define the *underlying automaton graph* $G_M(Q, \tau) = G_M$ of machine M. Thus, throughout the paper G_M denotes a directed graph on the states in Q, with each directed edge labeled by either a 0 or a 1, and with each state having exactly one outgoing 0-edge and one outgoing 1-edge.

In all of the learning models considered in this paper, we give algorithms for learning with respect to a worst-case underlying automaton graph G_M, but with respect to a random labeling γ of G_M. Thus we may think of the target machine M as being defined by the combination of an adversary who chooses the underlying automaton graph G_M, followed by a randomly chosen labeling γ of G_M. Here by random we shall always mean that each state $q \in Q$ is randomly and independently assigned a label $+$ or $-$ with equal probability. Since all of our algorithms will depend in some way on special properties that for any fixed G_M hold with high probability (where this probability is taken over the random choice of the labeling γ), we make the following general definition.

Definition 1. Let $P_{n,\delta}$ be any predicate on n-state finite automata which depends on n and a confidence parameter δ (where $0 \leqslant \delta \leqslant 1$). We say that *uniformly almost all* automata have property $P_{n,\delta}$ if the following holds: for all $\delta > 0$, for all $n > 0$ and for any n-state underlying automaton graph G_M, if we randomly choose $\{+, -\}$ labels for the states of G_M, then with probability at least $1 - \delta$, $P_{n,\delta}$ holds for the resulting finite automaton M.

The expression "uniformly almost all automata" is borrowed from Trakhtenbrot and Barzdin' [25] and was used by them to refer to a property holding with high probability for any fixed underlying graph. (The term "uniformly" thus indicates that the graph is chosen in a worst-case manner.)

Throughout the paper δ quantifies confidence only over the random choice of labeling for the target automaton M. We will require our learning algorithms, when given δ as input, to "succeed" (where success will be defined shortly) for uniformly almost all automata. Thus, for any fixed underlying automaton graph G_M, the algorithms must succeed with probability $1-\delta$, where this probability is over the random labeling.

In this paper we shall primarily consider two basic models for learning finite automata: one model in which the learner is given a mechanism for resetting the target machine to its initial state, and one model in which such a mechanism is absent. In both models the learner will be expected to make continuous predictions on an infinitely long random walk over the target machine, while being provided feedback after each prediction.

More precisely, in both models the learner is engaged in the following unending protocol: at the tth *trial*, the learner is asked to predict the $\{+, -\}$ label of M at the *current state* $r_t \in Q$ of the random walk (the current state is the start state q_0 at trial 0 and is updated following each trial in a manner described momentarily). The prediction p_t of the learner is an element of the set $\{+, -, ?\}$, where we interpret a prediction "?" as an admission of confusion on the learner's part. After making its prediction, the learner is told the correct label $l_t \in \{+, -\}$ of the current state r_t and therefore knows whether its prediction was correct. Note that the learner sees only the state labels, not the state names.

The two models we consider differ only in the manner in which the current state is updated following each trial. Before describing these update rules, we observe that there are two types of mistakes that the learner may make. The first type, called a *prediction mistake*, occurs when the learner outputs a prediction $p_t \in \{+, -\}$ on trial t and this prediction differs from the correct label l_t. The second type of mistake, called a *default mistake*, occurs any time the algorithm chooses to output the symbol "?." Note that default mistakes are preferable, since in this case the algorithm explicitly admits its inability to predict the output.

We are now ready to discuss the two current-state update rules we will investigate. In both models, under normal circumstances the random walk proceeds forward from the current state. Thus, the current state r_t is updated to r_{t+1} by selecting an input bit $b_{t+1} \in \{0, 1\}$ at random, and setting $r_{t+1} = r_t b_{t+1}$. The learner is provided with the bit b_{t+1} and the protocol proceeds to trial $t+1$.

However, in the *Reset-on-Default Model*, any default mistake by the learner (that is, any trial t such that the learner's prediction p_t is "?") causes the target machine to be reset to its initial state: on a "?" prediction we reinitialize the current state r_{t+1} to be q_0 and arbitrarily set $b_{t+1} = \lambda$ to indicate that the random walk has been reinitialized to proceed from q_0. Thus, by committing a default mistake the learner may "reorient" itself in the target machine.

In contrast, in the more difficult *No-Reset Model*, the random walk proceeds forward from the current state (that is, $r_{t+1} = r_t b_{t+1}$ for a random bit b_{t+1}) regardless of the prediction made by the learner.

Finally, we turn to the question of an appropriate definition of efficiency in our models. Since the trial sequence is infinite, we measure efficiency by the amount of computation per trial. Thus we say that a learning algorithm in either the Reset-on-Default Model or the No-Reset Model is *efficient* if the amount of computation on each trial is bounded by a fixed polynomial in the number of states n of the target machine and the quantity $1/\delta$.

In this paper we describe two main algorithms, both of which take the number of states n and the confidence parameter δ as input and are efficient in the sense just defined. The first algorithm works in the Reset-on-Default Model, and for uniformly almost all target automata M (that is, for any underlying automaton graph G_M and with probability at least $1-\delta$ over the random labeling), the algorithm makes no prediction mistakes, and the expected number of default mistakes is polynomial in n and $1/\delta$ (where the expectation is taken only over the infinite input bit sequence $b_1 b_2 \cdots$). The second algorithm works in the No-Reset Model and is based on the first algorithm; for uniformly almost all target automata, the expected total number of mistakes that it makes is polynomial in n and $1/\delta$.

4. LEARNING IN THE RESET-ON-DEFAULT MODEL

The main result of this section is an algorithm for learning uniformly almost all automata in the Reset-on-Default model. We state this result formally:

THEOREM 1. *There exists an algorithm that takes n and the confidence parameter δ as input, that is efficient, and in the Reset-on-Default Model, for uniformly almost all n-state automata, the algorithm makes no prediction mistakes and an expected number of default mistakes that is at most $O((n^5/\delta^2)\log(n/\delta))$ (where this expectation is taken over choice of the random walk).*

As mentioned before, we first describe the combinatorial properties on which our algorithm is based, followed by the algorithm itself.

4.1. Combinatorics

For the following definitions, let M be a fixed automaton with underlying automaton graph G_M, and let q, q_1, and q_2 be states of M.

DEFINITION 2. The *d-tree* of q is a complete binary tree of depth d with a state of M at each node. The root contains state q, and if p_v is the $\{0, 1\}$-path from the root of the tree to a node v, then v contains the state qp_v of M.

Note that the same state can occur several times in a signature. d-tree.

DEFINITION 3. The *d-signature* of q is a complete binary tree of depth d with a $\{+, -\}$ label at each node. It is obtained by taking the d-tree of q and replacing the state of M contained at each node by the corresponding label of that state in M.

We omit the depth of the d-signature when it is clear from context.

Note that since a learning algorithm never sees the state names encountered on a random walk, the d-tree of a state contains information that is inaccessible to the learner; however, since the learner sees the state labels, the d-signature is accessible in principle.

Definition 4. A string $x \in \{0, 1\}^*$ is a distinguishing string for q_1 and q_2 if $q_1\langle x\rangle \neq q_2\langle x\rangle$.

The statement of the key combinatorial theorem needed for our algorithm follows. This theorem is also presented by Trakhtenbrot and Barzdin' [25, Theorem 5.2] but we include our proof in Appendix A for completeness.

Theorem 2. *For uniformly almost all automata, every pair of inequivalent states have a distinguishing string of length at most* $2\log(n^2/\delta)$. *Thus for* $d \geqslant 2\log(n^2/\delta)$, *uniformly almost all automata have the property that the d-signature of every state is unique.*

4.2. Algorithm

For every state q in M let $\Sigma(q)$ be the d-signature of q for $d = 2\log(n^2/\delta)$. We assume henceforth that all signatures are unique; that is, $\Sigma(q) = \Sigma(q')$ if and only if q and q' are indistinguishable in M. From Theorem 2, this will be the case for uniformly almost all automata.

The main idea of our algorithm is to identify every state with its signature, which we have assumed is unique. If we reach the same state often enough, then the signature of that state can be discovered allowing us to determine its identity. As will be seen, this ability to determine the identity of states of the machine allows us also to reconstruct the automaton's transition function.

An *incomplete* d-signature of a state q is a complete binary tree of depth d in which some nodes are unlabeled, but the labeled nodes have the same label as in the d-signature of q.

The essence of our algorithm is the gradual construction of $M' = (Q', \tau', \gamma', q'_0)$, the hypothesis automaton. Each state $q' \in Q'$ can be viewed formally as a distinct symbol (such as an integer), and each has associated with it a complete signature (which, in fact, will turn out to be the complete signature $\Sigma(q)$ of some state q in the target machine M). In addition, the algorithm maintains a set Q'_{inc} consisting of states (again, arbitrary distinct symbols) whose signatures are incomplete, but which are in the process of being completed. Once the signature of a state in Q'_{inc} is completed, the state may be promoted to membership in Q'.

During construction of M', the range of the transition function τ' is extended to include states in $Q' \cup Q'_{\text{inc}}$. Thus, transitions may occur to states in either Q' or Q'_{inc}, but no transitions occur out of states in Q'_{inc}. As described below, predictions are made using the partially constructed machine M' and the incomplete signatures in Q'_{inc}.

Initially, Q' is empty and $Q'_{\text{inc}} = \{q'_0\}$ where q'_0 is the distinguished start state of M'. For any state $q' \in Q'$ of the target machine, $\Sigma'(q')$ denotes the (possibly partial) signature the learner associates with q'.

We will argue inductively that at all times M' is homomorphic to a partial subautomaton of M. More precisely, we will exhibit the existence at all times of a mapping φ: $Q' \cup Q'_{\mathrm{inc}} \to Q$ with the properties that (1) $\varphi(\tau'(q', b)) = \tau(\varphi(q'), b)$, (2) $\gamma'(q') = \gamma(\varphi(q'))$, and (3) $\varphi(q'_0) = q_0$ for all $q' \in Q'$ and $b \in \{0, 1\}$. (Technically, we have assumed implicitly (and without loss of generality) that M is reduced in the sense that all its states are distinguishable.)

Here is a more detailed description of how our learning algorithm makes its predictions and updates its data structures. The algorithm is summarized in Fig. 1. Initially, and each time that a reset is executed (following a default mistake), we reset M' to its start state q'_0. The machine M' is then simulated on the observed random input sequence, and predictions are made in a straightforward manner using the constructed output function γ'. From our inductive assumptions on φ, it follows easily that no mistakes occur during this simulation of M'. This simulation continues until a state q' is reached with an incomplete signature, that is, until we reach a state $q' \in Q'_{\mathrm{inc}}$.

At this point, we follow a path through the incomplete signature of q' beginning at the root node and continuing as dictated by the observed random input sequence. At each step, we predict the label of the current node. We continue in this fashion until we reach an unlabeled node, or until we "fall off" of the signature tree (that is, until we attempt to exit a leaf node). In either case, we output "?" and so

```
1. $Q' \leftarrow \emptyset$; $Q'_{\mathrm{inc}} \leftarrow \{q'_0\}$.
2. $q' \leftarrow q'_0$.
3. While $q' \notin Q'_{\mathrm{inc}}$ do the following:
   On observing input symbol $b$, set $q' \leftarrow \tau'(q', b)$, and predict $\gamma'(q')$.
4. Traverse the path through $\Sigma'(q')$ as dictated by the input sequence. At each step, predict the label
   of the current node. Continue until an unlabeled node is reached, or until the maximum depth of
   the tree is exceeded.
5. Predict "?." If at an unlabeled node of $\Sigma'(q')$, then label it with the observed output symbol.
6. If $\Sigma'(q')$ is complete then "promote" $q'$ as follows:
   (a) $Q'_{\mathrm{inc}} \leftarrow Q'_{\mathrm{inc}} - \{q'\}$
   (b) if $\Sigma'(q') = \Sigma'(X1)r'$ for some $r' \in Q'$ then
         — find $s', b$ such that $\tau'(s', b) = q'$
         — $\tau'(s', b) \leftarrow r'$.
   (c) else
         — $Q' \leftarrow Q' \cup \{q'\}$
         — create new states $r'_0$ and $r'_1$
         — $Q'_{\mathrm{inc}} \leftarrow Q'_{\mathrm{inc}} \cup \{r'_0, r'_1\}$
         — partially fill in signatures of $r'_0, r'_1$ using $\Sigma'(q')$
         — $\tau'(q', b) \leftarrow r'_b$ for $b \in \{0, 1\}$.
7. Go to step 2.
```

FIG. 1. Pseudocode for algorithm **Reset**.

incur a default mistake. If we currently occupy an unlabeled node of the incomplete signature, then we label this node with the observed output symbol.

Our inductive assumptions imply that the signature $\Sigma'(q')$ built up by this process is in fact $\Sigma(\varphi(q'))$, the true signature of $\varphi(q')$ in M'. This means that no prediction mistakes occur while following a path in the incomplete signature $\Sigma'(q')$.

Once a signature for some state q' in Q'_{inc} is completed, we "promote" the state to Q'. We first remove q' from Q'_{inc}, and we then wish to assign q' a new identity in Q' based on its signature. More precisely, suppose first that there exists a state $r' \in Q'$ whose signature matches that of q' (so that $\Sigma'(q') = \Sigma'(r')$). Then, from the foregoing comments, it must be that $\Sigma(\varphi(q')) = \Sigma(\varphi(r'))$, which implies (by the assumed uniqueness of signatures) that $\varphi(q') = \varphi(r')$. We therefore wish to identify q' and r' as equivalent states in M' by updating our data structures appropriately. Specifically, from our construction below, there must be some (unique) state s' and input symbol b for which $\tau'(s', b) = q'$; we simply replace this assignment with $\tau'(s', b) = r'$ and discard state q' entirely. Note that this preserves our inductive assumptions on φ.

Otherwise, it must be the case that the signature of every state in Q' is different from that of q'; similar to the argument above, this implies that $\varphi(q')$ does not occur in $\varphi(Q')$ (the image of Q' under φ). We therefore wish to view q' as a new state of M'. We do so by adding q' to Q', and by setting $\gamma'(q')$ to be the label of the root node of $\Sigma'(q')$. Finally, we create two new states r'_0 and r'_1 which we add to Q'_{inc}. These new states are the immediate successors of q', so we set $\tau'(q', b) = r'_b$ for $b \in \{0, 1\}$. Note that the incomplete signatures of r'_0 and r'_1 can be partially filled in using $\Sigma'(q')$; specifically, all of their internal nodes can be copied over using the fact that the node reached in $\Sigma'(r'_b)$ along some path x must have the same label as the node reached in $\Sigma'(q')$ along path bx (since $r'_b = \tau'(q', b)$).

As before, it can be shown that these changes to our data structures preserve our inductive assumptions on φ.

We call the algorithm described above Algorithm **Reset**. The inductive arguments made above on φ imply the reliability of **Reset**:

LEMMA 3. *If every state in M has a unique d-signature, then Algorithm* **Reset** *makes no prediction mistakes.*

LEMMA 4. *The expected number of default mistakes made by Algorithm* **Reset** *is $O((n^5/\delta^2)\log(n/\delta))$.*

Proof. We treat the completion of the start state's signature as a special case since this is the only case in which the entire signature must be completed (recall that in every other case, only the leaf nodes are initially empty). In order to simplify the analysis for the start state, let us require that the learner label the nodes in $\Sigma'(q'_0)$ level by level according to their distance from the root. The learner thus waits until it is presented with all strings of length i before it starts labeling nodes on level $i+1$. Clearly the expected number of default mistakes made by this method is an upper bound on the number of default mistakes made in completing $\Sigma'(q'_0)$. We thus define, for every $0 \leqslant i \leqslant d$, a random variable X_i that represents the number of default mistakes encountered during the labeling of nodes at level i.

For each q' (other than q'_0) added to Q'_{inc}, let $Y_{q'}$ be a random variable that represents the number of times that state q' is reached in our simulation of M' before the signature of q' is completed, that is, until every leaf node of $\Sigma'(q')$ is visited.

The expected number of default mistakes made is then the sum of the expectations of the random variables defined above. Computing the expectation of each of these variables in turn reduces to the so-called *Coupon Collector's Problem* [9]: there are N types of coupons, and at each step we are given a uniformly chosen coupon. What is the expected number of steps before we obtain at least one coupon of each type? The answer to this is $\sum_{i=1}^{N}(N/i)$ and a good upper bound is $N(\ln N+1)$.

Thus, the expected number of default mistakes is

$$\sum_{i=1}^{d}\mathbf{E}[X_i]+\sum_{q'}\mathbf{E}[Y_{q'}]\leqslant\sum_{i=1}^{d}2^i(\ln 2^i+1)+2n2^d(\ln 2^d+1)$$
$$=O((n^5/\delta^2)\cdot\log(n/\delta)),$$

where the sum indexed by q' represents a sum over all q' (excluding q'_0) ever added to Q'_{inc}. There are at most $2n$ such q' since, by construction, each is of the form $\tau'(q'_1, b)$ for some $q'_1\in Q'$ and $b\in\{0,1\}$, and since $|Q'|\leqslant n$. ▌(Lemma 4)

Finally, the amount of computation done by Algorithm **Reset** in every trial is clearly bounded by a polynomial in n and $1/\delta$.

In Appendix B we show that Algorithm **Reset**, although derived in the Reset-on-Default model, can be modified to learn finite automata in an average-case, PAC-like model.

5. LEARNING WITHOUT A RESET

In this section, we consider the problem of learning in the No-Reset Model described in Section 3.

The main result of this section is an algorithm for effectively learning uniformly almost all automata in this model:

THEOREM 5. *There exists an algorithm that takes n and the confidence parameter δ as input, is efficient, and in the No-Reset Model, for uniformly almost all n-state automata the algorithm makes at most $n2^l$ prediction mistakes and an expected number of default mistakes that is at most*

$$O(n2^l(l2^l+1)(n^5(2/\delta)^2\log(2n/\delta))),$$

where

$$l=2\log(2n^2/\delta)+\frac{4\log^2(2n^2/\delta)}{\log(n)-\log(\log(2n^2/\delta))-2}$$

and where the expectation is taken over the choice of a random walk. In particular, if $\delta=n^{-c}$ for some constant c, then the number of prediction mistakes and the expected number of default mistakes is polynomial in n.

Throughout this section, we assume that the target machine is *strongly connected* (that is, every state is reachable from every other state). We make this assumption without loss of generality since the machine will eventually fall into a strongly connected component from which escape is impossible.

As in the last section, we begin with the relevant combinatorics, followed by description and analysis of our algorithm.

5.1. Combinatorics

Learning is considerably more difficult in the absence of a reset. Intuitively, given a reset, the learner can more easily relate the information it receives to the structure of the unknown machine, since it knows that each random walk following a default mistake begins again at a fixed start state. In contrast, without a reset, the learner can easily "get lost" with no obvious means of reorienting itself.

In a related setting, Rivest and Schapire [20] introduced the idea of using a *homing sequence* for learning finite automata in the absence of a reset. Informally, a homing sequence is a sequence of input symbols that is guaranteed to "orient" the learner; that is, by executing the homing sequence, the learner can determine where it is in the automaton, and so can use it in lieu of a reset.

In our setting, the learner has no control over the inputs that are executed. Thus, for a homing sequence to be useful, it must have a significant probability of being executed on a random walk, that is, it must have length roughly $O(\log n)$. In general, every machine has a homing sequence of length n^2, and one might hope to prove that uniformly almost all automata have "short" homing sequences. We have been unable to prove this latter property, and it may well be false.

Instead, we introduce the related notion of a *local* homing sequence. This is a sequence of inputs that is guaranteed to orient the learner, but only if the observed output sequence matches a particular pattern. In contrast, an ordinary homing sequence orients the learner after any possible output sequence.

More formally, a *homing sequence* is an input sequence h with the property that $q_1\langle h\rangle = q_2\langle h\rangle$ implies $q_1 h = q_2 h$ for all states q_1 and q_2. Thus, by observing the output sequence, one can determine the final state reached at the end of the sequence. A *local homing sequence for state* q is an input sequence h for which $q\langle h\rangle = q'\langle h\rangle$ implies $qh = q'h$ for all states q'. Thus, if the observed output sequence is $q\langle h\rangle$, then the final state reached at the end of the sequence must be qh; however, if the output sequence is something different, then nothing is guaranteed about the final state.

We will see that uniformly almost all automata have "short" local homing sequences for every state. To prove this, we will find the following lemma to be useful.

We say that an input sequence s is an *r-exploration sequence for state q* if at least r distinct states are visited when s is executed from q. Note that this property has nothing to do with the machine's labeling, but only with its architecture.

LEMMA 6. *Every strongly connected graph G_M has an r-exploration sequence of length at most $r + r^2/(\log(n/r) - 1)$ for each of its n states.*

Proof. Let q be a vertex of G_M for which we wish to construct an r-exploration sequence.

Suppose first that there exists a state q' at distance at least r from q (where the distance is the length of the shortest path from q to q'). Let s be the shortest path from q to q'. Then all the states on the s-walk from q are distinct, and so the length-r prefix of s is an r-exploration sequence. Thus, for the remainder of the proof, we can assume without loss of generality that every state q' is within distance r of q.

Suppose now that there exists a pair of states q' and q'' which are such that the distance from q' to q'' has distance at least r. Then, similar to what was done before, we let s be the shortest path from q to q' (which we assumed has length at most r) followed by the shortest path from q' to q''. Then the length-$2r$ prefix of s is an r-exploration sequence for q. Therefore, we can henceforth assume without loss of generality that all pairs of states q' and q'' are within distance r of one another.

We construct a path s sequentially. Initially, s is the empty string. Let T denote the set of states explored when s is executed from q; thus, initially, $T = \{q\}$.

The construction repeatedly executes the following steps until $|T| \geqslant r$: Let $T = \{t_1, t_2, ..., t_k\}$. Then $|T| = k < r$ (since we're not done). Our construction grows a tree rooted at each t_i representing the states reachable from t_i. Each tree is grown to maximal size maintaining the condition that no state appears twice in the forest of k trees. Each node t in each tree has at most one child for each input symbol b; if present, this b-child is that state which is reached from t by executing b. We add such a b-child provided it has not appeared elsewhere in the forest. Nodes are added to the forest in this manner until no more nodes can be added.

Since we assume G_M is strongly connected, it is not hard to see that every state will eventually be reached in this manner, that is, that the total number of nodes in the forest is n. Since there are $k < r$ trees, this means that some tree has at least n/r nodes, and so must include a node at depth at least $\log(n/r) - 1$. In other words, there is a path y from t_i (the root of this tree) to some other state t of length at least $\log(n/r) - 1$. So we append to the end of s the shortest path x from qs (the state where we left off) to t_i, followed by the path y from t_i to t; that is, we replace s with sxy.

Note that $|x| \leqslant r$ since the machine has diameter at most r, so the length of s increases by at most $r + |y|$. Moreover, by extending s, we have added at least $|y| \geqslant \log(n/r) - 1$ states to T. This implies that the total number of times that we repeat this procedure is at most $r/(\log(n/r) - 1)$. Also, we have thus argued that the difference between the length of s and $|T|$ increases on each iteration by at most r. Thus, the final length of s is at most $r + r^2/(\log(n/r) - 1)$. ∎ (Lemma 6)

We are now ready to prove that uniformly almost all automata have short local homing sequences.

LEMMA 7. *For uniformly almost all strongly connected automata, every state has a local homing sequence of length* $r + r^2/(\log(n/r) - 1)$ *where* $r = 2\log(n^2/\delta)$.

Proof. Let G_M be a strongly connected graph. Let q be a vertex of G_M. By Lemma 6, we know that there exists an r-exploration sequence s for q of length $r + r^2/(\log(n/r) - 1)$.

Let q' be another vertex of G_M. If $qs \neq q's$ then with probability at least $1-2^{-r/2}$, we have that $q\langle s\rangle \neq q'\langle s\rangle$. This follows from a similar argument to that used in the proof of Theorem 2. Thus, the probability that s is not a local homing sequence for q is at most $n2^{-r/2}$.

Therefore, the probability that any state q does not have a local homing sequence of the stated length is at most $n^2 2^{-r/2} = \delta$. ▮(Lemma 7)

Note, in particular, that if $\delta = n^{-c}$ where c is a positive constant then the length bound given in Lemma 7 is $O(\log n)$.

5.2. Algorithm

In this section, we show that local homing sequences can be used to derive an algorithm that efficiently learns almost all automata in the No-Reset Model.

Informally, suppose we are given a "short" local homing sequence h for some "frequently" visited state q, and suppose further that we know the output $q\langle h\rangle$ produced by executing h from q. In this case, we can use the learning algorithm **Reset** constructed in the previous section for the Reset-on-Default Model to construct a learning algorithm for the No-Reset Model. The main idea is to simulate **Reset**, but to use our knowledge about h in lieu of a reset. Recall that because h is a local homing sequence for q, whenever we observe the execution of h with output $q\langle h\rangle$, we know that the automaton must have reached state qh. Thus, we can use qh as the start state for our simulation of **Reset**, and we can simulate each reset required by **Reset** by waiting for the execution of h with output $q\langle h\rangle$. Note that from our assumptions, we will not have to wait too long for this to happen since we assumed that q is "frequently" visited, and since we also assumed that h is "short" (so that the probability that h is executed once we reach q is reasonably large).

There are two problems with this strategy. The first problem is determining what it means for a state to be "frequently" visited. This problem could be avoided if we had a "short" local homing sequence h_q for every state q, along with its associated output sequence $q\langle h_q\rangle$. In this case, we could simulate several separate copies of algorithm **Reset**, each corresponding to one state of the machine. The copy corresponding to state q has start state qh_q and is "activated" when h_q is executed with output $q\langle h_q\rangle$. Note that when this event is observed, the learner can conclude that the machine has actually reached qh_q, the start state for the corresponding copy of **Reset**. Thus, to simulate a reset, we wait for one of the local homing sequences h_q to be executed with output $q\langle h_q\rangle$. This resets us to one of the copies of **Reset**, so we always make some progress on one of the copies. Also, regardless of our current state q, we have a good chance of executing the current state's local homing sequence h_q.

The second obstacle is that we are not given a local homing sequence for any state, nor its associated output sequence. However, if we assume that there exists a short local homing sequence for every state, then we can try all possible input/output sequences. As we will see, those that do not correspond to "true" local homing sequences can be quickly eliminated.

We will assume henceforth that the target automaton has the following properties: (1) every state has a local homing sequence of length $l = r + r^2/(\log(n/r) - 1)$, where $r = 2\log(2n^2/\delta)$ and (2) every pair of inequivalent states have a distinguishing string of length at most $d = 2\log(2n^2/\delta)$. By Lemma 7 and Theorem 2, these assumptions hold for uniformly almost all automata.

Our algorithm uses the ideas described above. Specifically, we create one copy $\mathbf{R}_{i,o}$ of algorithm **Reset** for each input/output pair $\langle i, o \rangle$ where $i \in \{0, 1\}^l$ and $o \in \{+, -\}^{l+1}$.

We call a copy $\mathbf{R}_{i,o}$ *good* if i is a local homing sequence for some state q, and if $q\langle i \rangle = o$; all other copies are *bad*. We call a copy $\mathbf{R}_{i,o}$ *live* if it has not yet been identified as a bad copy. Initially all copies are live, but a copy is killed if we determine that it is bad.

Here is a description of our algorithm, which we call **No-Reset**.

Repeat forever:

1. Observe a random input sequence i of length l producing output sequence o. If $\mathbf{R}_{i,o}$ is dead, repeat this step. Predict "?" throughout the execution of this step.

2. Execute the next step of the reset algorithm for $\mathbf{R}_{i,o}$. More precisely: simulate the copy $\mathbf{R}_{i,o}$ of the Reset-on-Default algorithm relying on $\mathbf{R}_{i,o}$'s predictions until $\mathbf{R}_{i,o}$ hits the reset button or until it makes a prediction mistake.

3. If $\mathbf{R}_{i,o}$ makes a prediction mistake, or if the number of signatures (that is, states) of $\mathbf{R}_{i,o}$ exceeds n, then kill copy $\mathbf{R}_{i,o}$.

Note that if $\mathbf{R}_{i,o}$ is good then it will never be killed because every simulation of this algorithm truly begins in the same state, and therefore $\mathbf{R}_{i,o}$ will make no prediction mistakes and will not create more than n states (as proved in Section 4.2).

LEMMA 8. *Algorithm* **No-Reset** *makes at most* $n2^l$ *prediction mistakes.*

Proof. If $\mathbf{R}_{i,o}$ makes a prediction mistake at step 2, then it is immediately killed at step 3. Thus each copy $\mathbf{R}_{i,o}$ makes at most one prediction mistake.

Although there are 2^{2l+1} copies $\mathbf{R}_{i,o}$, at most $n2^l$ will ever be activated. This follows from the observation that for every input sequence there are at most n output sequences, one for every state in the automaton. Thus at most $n2^l$ input/output pairs $\langle i, o \rangle$ will ever be observed. ▮(Lemma 8)

Let m_R be the expected number of default mistakes made by the reset algorithm of Section 4.2. The following lemmas prove that algorithm **No-Reset** expects to incur $n2^l(l2^l + 1)\, m_R$ default mistakes.

LEMMA 9. *On each iteration, the expected number of default mistakes incurred at step* 1 *is at most* $l2^l$.

Proof. Let q be the current state. By assumption, q has a local homing sequence h_q of length l. Since $\mathbf{R}_{h_q, q\langle h_q \rangle}$ is good, it must be live. Therefore, the probability that a live copy is reached is at least the probability of executing h_q, which is at least 2^{-l}.

Thus, we expect step 1 to be repeated at most 2^l times. Since each repetition causes l default mistakes, this proves the lemma. ▌(Lemma 9)

Thus for every default mistake of algorithm **Reset** we incur $l2^l$ additional default mistakes in our new algorithm. The following lemma can now be proved.

LEMMA 10. *The total number of default mistakes made by the algorithm is at most* $n2^l(l2^l+1)\, m_R$.

Proof. Note first that we expect that each copy $\mathbf{R}_{i,o}$ makes at most m_R default mistakes, even if $\mathbf{R}_{i,o}$ is bad. This follows essentially from the proof of Lemma 4, combined with the fact that the number of signatures of each copy is bounded by n (copies that exceed this bound are killed at Step 3).

As noted in the proof of Lemma 8, there are $n2^l$ copies of the algorithm that are ever activated. We expect each of these to make at most m_R mistakes, so we expect the outer loop of the algorithm to iterate at most $n2^l m_R$ times. Combined with Lemma 9, this gives the stated bound on the expected number of default mistakes. ▌(Lemma 10)

6. EXTENSIONS

6.1. *Replacing Randomness with Semi-Randomness*

Our results so far have assumed the uniform distribution in two different contexts. The label of each state of the automaton graph and each bit of the random walk observed by the learner were both assumed to be the outcome of independent and unbiased coin flips. While entirely removing this randomness in either place and replacing it with worst-case models would invalidate our results, the performance of our algorithms degrades gracefully if the state labels and walk bits are not truly random.

More precisely, suppose we think of the random bits for the state labels and the random walk as being obtained from a bit generator G. Then our algorithms still work even in the case where G does not generate independent, unbiased bits but is instead a *semi-random source* as defined by Santha and Vazirani [24]. Briefly, a semi-random source in our context is an omniscient adversary with complete knowledge of the current state of our algorithm and complete memory of all bits previously generated. Based on this information, the adversary is then allowed to choose the *bias* of the next output bit to be any real number ρ in the range $[\Delta, 1-\Delta]$ for a fixed constant $0<\Delta\leqslant 1/2$. The next output bit is then generated by flipping a coin of bias ρ. Thus, a semi-random source guarantees only a rather weak form of independence among its output bits.

Semi-randomness was introduced by Santha and Vazirani and subsequently investigated by several researchers [6, 24, 27, 28] for its abstract properties as a computational resource and its relationship to true randomness. However, we are not the first authors to use semi-randomness to investigate models between the worst case and the average (random) case. Blum [5] studied the complexity of coloring semi-random graphs, and Azar *et al.* have considered semi-random sources to model biased random walks on graphs [3].

Now assume that the adversary can choose the label of each state in G_M by flipping a coin whose bias is chosen by the adversary from the range $[\Delta_1, 1-\Delta_1]$. A simple alteration of Theorem 1 (details omitted) gives that the probability that two inequivalent states are not distinguished by their signature is at most $(1-\Delta_1)^{(d+1)/2}$ (instead of $2^{-(d+1)/2}$, which held for the uniform distribution). This implies that in order to achieve the same confidence, it suffices to increase the depth of the signature by a factor of $-1/\log(1-\Delta_1)$.

The relaxation of our assumption on the randomness of the observed input sequence is similar. Assume that the next step of the walk observed by the learner is also generated by an adversarially biased coin flip in the range $[\Delta_2, 1-\Delta_2]$. In algorithm **Reset** the expected number of default mistakes required to complete a partial signature is higher than in the uniform case. As the probability of a sequence of length d can decrease from $(1/2)^d$ to Δ_2^d, the expected number of default mistakes per signature increases from $O(d2^d)$ to at most $O(d(1/\Delta_2)^d)$.

These alterations imply that the expected number of default mistakes made by **Reset** increases from

$$O((n^5/\delta^2)\cdot\log(n/\delta))$$

to

$$O(n(n^2/\delta)^{(2\log(1/\Delta_2)/\log(1-\Delta_1))}\log(n/\delta)).$$

The deviations from uniformity increase the degree of the polynomial dependence on n and $1/\delta$. Notice that the sensitivity to nonuniformity in the labeling process is stronger than the sensitivity to nonuniformity in the random walks.

Similar implications follow for **No-Reset**. The length of the local homing sequences l has to be increased from l to $l' = l\log(1-\Delta_1)^{-2}$ which increases the number of prediction mistakes from $n2^l$ to $n2^{l'}$ and the expected number of default mistakes from $n2^l(l2^l+1)\,m_R$ to $n2^{l'}(l'\Delta_2^{-l'}+1)\,m'_R$.

6.2. *Learning Distributions on Strings*

An interesting extension of our results is to a model where automata are used to represent distributions over binary strings such as those discussed by Feder *et al.* [8], rather than as acceptors of languages. In this model no binary labels are associated with the states Q. Instead, there is a real-valued function φ: $Q \to [0, 1]$. The underlying state graph (defined by the transition function τ) together with φ defines a probabilistic generator of binary strings in the following manner: we now think of the bits on the edges of the machine as the output bits. Starting in the initial state, if at any point we have traversed the path of i edges labeled $b_1 \cdots b_i$ and are currently in state q, the next bit of the path is determined by flipping a coin of bias $\varphi(q)$. We then move to the indicated state and continue. The machine thus generates a distribution over finite binary strings of any fixed length: the probability of a binary sequence σ is the product of the values of φ at the nodes on the path defined by σ.

A common and natural learning goal in this context is to learn to correctly predict approximately the probability of a binary string approximately predict the probability of a binary string after observing a sample of strings from the distribution. In general, this problem is shown to be as hard as the problem of learning parity with noise in [13], which is related to a longstanding open problem in coding theory.

However, there is a simple variant of **Reset** that can perform an online version of this task efficiently with high probability if the function φ is chosen according to some natural classes of distributions. As usual, the structure of the automaton as defined by τ is unrestricted (worst-case).

The algorithm of Ron *et al.* [23] which learns acyclic probabilistic finite automata builds on the algorithm given here. Their algorithm has been successfully applied to the problem of handwriting recognition and to construct multiple-pronunciation models for spoken words.

We define two parameters $0 < \Delta, \mu < 1/2$ and associate with each state $q \in Q$ some distribution P_q over the interval $[\Delta, 1-\Delta]$. We assume that the value of $\varphi(q)$ is chosen independently for each state q according to P_q. The only requirement we make on the distributions P_q is that if q_1 and q_2 are two different states in the automaton, then the probability that $|\varphi(q_1) - \varphi(q_2)| \geqslant \mu$ is at least 1/2, and that otherwise $\varphi(q_1) = \varphi(q_2)$. More generally, we can require that the probability that $|\varphi(q_1) - \varphi(q_2)| < \mu$ be at most 1/2, but for sake of the exposition we consider this simpler case. This requirement is analogous to the random $\{+, -\}$ labeling of the states in the case of typical DFA. A simple legal example for $\Delta = 1/4$, $\mu = 1/8$ is when all P_q are equal to the uniform distribution over $\{i/8 : 2 \leqslant i \leqslant 6\}$. (If $|\varphi(q_1) - \varphi(q_2)|$ is allowed to be small, but non-zero, then a legal example for $\Delta = 1/4$, $\mu = 1/8$ is when all P_q are equal to the uniform distribution over $[1/4, 3/4]$.)

In this context we say that a string $x \in \{0, 1\}^*$ is a *distinguishing sequence* for q_1 and q_2, if there exists a prefix x' of x, such that $|\varphi(\tau(q_1, x')) - \varphi(\tau(q_2, x'))| \geqslant \mu$. It is easily verified that the proof of Theorem 2 also yields that for uniformly almost all distribution generating automata, every pair of inequivalent states have a distinguishing string of length at most $2 \log(n^2/\delta)$.

For the task of learning an unknown distribution in this setting, we use a slightly modified version of **Reset**:

1. Instead of collecting the labels of each node in the d-tree, it waits until each node is visited at least m times, and counts the number of times the symbol 1 each of the two input symbols is observed when the node is visited.

2. After all nodes in a d-tree have been visited m times, the signature tree is calculated. The signature tree associates with each node in the d-tree the empirical estimate of the value of φ that is associated with the node: that is, $\hat{\varphi}$ is the number of times a 1 has been observed in the node divided by the total number of visits to the node. It then compares the signature tree to all previously collected signature trees. Two trees are identified if for every corresponding pair of nodes v and v' we have $|\hat{\varphi}(v) - \hat{\varphi}(v')| \leqslant \mu/2$.

3. At each step, the learner must predict an approximation to the value of $\varphi(q)$, where q is the current state that the learner is at. It does so using the estimates $\hat{\varphi}(\cdot)$ unless it is at a node which has been visited less than m times in which case it outputs "?" (and makes a default mistake). If the learner predicts a value that is more than $\mu/2$ away from the correct value, then it is considered to be a prediction mistake. (note that the learner will not immediately know may not know that it has made a prediction mistake, but we show that with high probability it is very unlikely that our algorithm does not make any prediction mistakes).

Assume for a moment that the estimates of the values of φ were completely accurate. In such a case, as was observed above, for every pair of inequivalent states, there exists at least one node in their respective signature trees which will have a significantly different φ value, and hence their signature trees will not be identified. Thus the analysis of this case essentially reduces to the analysis of learning typical DFA. Since $\varphi(q) \in [\Delta, 1-\Delta]$ also determines the bias for the choice of the next state, we can apply the result stated in the previous subsection and get that the expected number of default mistakes made by the algorithm is

$$O(m \cdot n(n^2/\delta)^{2\log(1/\Delta)} \log(n/\delta)),$$

where m (the number of times each node is visited before the signature is complete) is set below.

As the estimates of φ are not exact, there might be mistakes in identifying signatures. However, using Chernoff bounds it is easy to show that the probability that the estimate differs from the true value by a multiplicative constant decreases exponentially with m. In particular, it suffices that $m = \Theta(\log(1/\delta')/\mu)$ so that this event will not occur for a particular node with probability at least $1-\delta'$ for any choice of confidence parameter δ'. Since the number of nodes in all possible signature trees constructed is at most $2n \cdot 2^{d+1}$, we should choose $m = \Theta(d\log(n/\delta')/\mu)$ in order to ensure that with probability at least $1-\delta'$, this event never occurs. As long as this event does not occur, the algorithm does not make any prediction mistakes and we get that the expected number of default mistakes is

$$O((1/\mu) \cdot n(n^2/\delta)^{2\log(1/\Delta)} \cdot \log^2(n/\delta) \cdot \log(n/\delta')).$$

Note that the algorithm can be slightly modified so that the accuracy of the algorithm's predictions improves as the number of trials increases. All that is needed is to keep updating the $\hat{\varphi}(\cdot)$ values associated with states whose signatures have been completed.

APPENDIX A

Technical Appendix: Proof of Theorem 2

THEOREM 2. *For uniformly almost all automata, every pair of inequivalent states have a distinguishing string of length at most* $2\log(n^2/\delta)$. *Thus for* $d \geq 2\log(n^2/\delta)$, *uniformly almost all automata have the property that the d-signature of every state is unique.*

Theorem 2 will be proved via a series of lemmas. The graph G_M is fixed throughout the proof. In the lemmas we prove that for any labeling of G_M, if two states in the automaton are inequivalent, then either their d-signatures are different or at least one of the d-trees includes many different states. Using this fact we shall later prove that for most of the labelings of G_M the d-signatures of the two states are different. As the lemmas are proven for any labeling, let us fix any automaton M for the duration of the lemmas.

We begin by giving a lemma saying that the shortest distinguishing string passes through many states on walks from q_1 and q_2. Our eventual goal is to find a much shorter string with this property.

LEMMA 11. *Let q_1 and q_2 be inequivalent states of M, and let $x \in \{0, 1\}^*$ be a shortest distinguishing string for q_1 and q_2. Let T_1 and T_2 be the sets of states of M passed through on taking an x-walk from q_1 and q_2 respectively. Then $|T_1 \cup T_2| \geqslant |x| + 2$.*

Proof. Let R be a set of states in M, and let y be a string over $\{0, 1\}$. We define the *partition of R induced by y* to be the partition of the states in R according to their behavior on the string y. More precisely, two states $r_1, r_2 \in R$ belong to the same block of the partition if and only if $r_1\langle y\rangle = r_2\langle y\rangle$.

Let $x_i \in \{0, 1\}$ denote the ith bit of x, and let $l = |x|$. For $1 \leqslant i \leqslant l+1$, let y_i be the string $x_i x_{i+1} \cdots x_l$.

We claim that for every $1 \leqslant i \leqslant l$, the partition of $T_1 \cup T_2$ induced by y_i is a strict refinement of the partition induced by y_{i+1}. Suppose to the contrary that there exists an index $1 \leqslant j \leqslant l$ for which the partition of $T_1 \cup T_2$ induced by y_j is the same as that induced by y_{j+1}. Let $r_1 = q_1 x^{(j-1)}$ and $r_2 = q_2 x^{(j-1)}$ (recall that $x^{(i)}$ denotes the length i prefix of x). Since we assume that x distinguishes q_1 and q_2, we have that $r_1\langle y_j\rangle \neq r_2\langle y_j\rangle$, and so r_1 and r_2 must already be in different classes according to the partition induced by y_{j+1}. Therefore $q_1\langle x^{(j-1)} y_{j+1}\rangle \neq q_2\langle x^{(j-1)} y_{j+1}\rangle$ and so $x^{(j-1)} y_{j+1}$ is a shorter distinguishing string for q_1 and q_2, contradicting our assumption on x.

Now since the number of classes in the partition induced by X_{l+1} (the set $\{\lambda\}$) is two, the number of classes in the partition induced by X_1 is at least $l+2$. On the other hand, the size of the partition of $T_1 \cup T_2$ induced by any set of strings is at most $|T_1 \cup T_2|$, and hence $l + 2 \leqslant |T_1 \cup T_2|$ as desired. ▮(Lemma 11)

Lemma 11 can be thought of as a statement about the density of unique states encountered by taking the x-walk from q_1 or q_2: either along the x-walk from q_1 or along the x-walk from q_2, we must pass through at least $(|x| + 2)/2$ different states. What we would like to develop now is a similar but stronger statement holding for any prefix x' of x, in which we claim that along the x'-walk from either q_1 or q_2 we must encounter $\Theta(|x'|)$ different states.

In order to do this we first present the following construction. Let x' be a proper prefix of the shortest distinguishing string x for q_1 and q_2, and let $y \in \{0, 1\}^*$ be such that $x = x'y$ (note that $|y| \geqslant 1$). Let z be a new defined symbol which is neither 0 nor 1. The symbol z will act as a kind of "alias" for the string y. We construct a new automaton $M^y = (Q^y, \tau^y, \gamma^y, q_0^y)$ over the input alphabet $\{0, 1, z\}$. The start state in this automaton is the start state of M, so $q_0^y = q_0$. We set $Q^y = Q \cup \{q_+, q_-\}$ where q_+ and q_- are new special states, and γ^y extends γ with $\gamma^y(q_-) = -$ and $\gamma^y(q_+) = +$. All the transitions in M^y from states in Q on input symbols from $\{0, 1\}$ remain the same as in M, and the z transitions are defined as follows: for $q \in Q$, $\tau^y(q, z) = q_+$ if $\gamma(qy) = +$, and $\tau^y(q, z) = q_-$ otherwise. Thus, the special input symbol z from any state in M^y results in the same final label as the input string y from the corresponding state in M. For $q \in \{q_+, q_-\}$, $\tau^y(q, b) = q_-$ for any $b \in \{0, 1, z\}$.

LEMMA 12. *$x'z$ is a shortest distinguishing string for q_1 and q_2 in M^y.*

Proof. Clearly, $x'z$ distinguishes q_1 and q_2 in M^y.

Suppose t is a shortest distinguishing string for q_1 and q_2 that is shorter than $x'z$. We claim that z cannot appear in the middle of t. Suppose to the contrary that $t = t'zt''$ where z does not appear in t' and $|t''| \geqslant 1$. By construction, every z transition takes the machine into either q_+ or q_-. Therefore, because t is a shortest distinguishing string, it must be that $q_1 t'z = q_2 t'z$. Thus, $q_1 t'zt'' = q_2 t'zt''$, and so $t'zt''$ cannot be a shortest distinguishing string.

If t is of the form $t'z$, then $t'y$ is a shorter distinguishing string than $x = x'y$ for q_1 and q_2 in M which contradicts our assumption on x. If z does not appear at all in t, then since $|y| \geqslant |z|$, t itself is a shorter distinguishing string than x in M, again contradicting our assumption. ▌(Lemma 12)

We are now prepared to generalize Lemma 11.

LEMMA 13. *Let x' be the length l' prefix of x for $l' < l$. Let $T'_1 \subseteq T_1$ and $T'_2 \subseteq T_2$ be the sets of states in M passed upon executing x' starting from q_1 and q_2 respectively. Then $|T'_1 \cup T'_2| \geqslant l' + 1$.*

Proof. According to Lemma 12, $x'z$ is a shortest distinguishing string for q_1 and q_2 in M^y. The set of states passed upon executing $x'z$ starting from q_1 is $T'_1 \cup \{q_+\}$, and those passed starting from q_2 is $T'_2 \cup \{q_-\}$. Applying Lemma 11, we get that $|T'_1 \cup T'_2| + 2 \geqslant |x'z| + 2 = l' + 3$ and hence $|T'_1 \cup T'_2| \geqslant l' + 1$. ▌(Lemma 13)

We now move to combine these statements that hold for all labelings of G_M to a proof of the statement about most of the labelings of G_M.

Proof of Theorem 2. Let G_M be an underlying automaton graph, and let q_1 and q_2 be two distinct states in G_M. Let M be the random variable representing the machine obtained from G_M by assigning random labels to every state. For fixed d, we first wish to bound the probability (over the random choice of M) that q_1 and q_2 are inequivalent but indistinguishable by any string of length d.

Suppose first that for every labeling of G_M, states q_1 and q_2 are either equivalent, or they can be distinguished by a string of length at most d. Then this will certainly be the case for a random labeling of G_M, and therefore, in this case, the probability that q_1 and q_2 are inequivalent in M but indistinguishable by any string of length d is zero.

Otherwise, there exists some labeling of G_M which yields a machine M_0 with respect to which q_1 and q_2 are inequivalent but whose shortest distinguishing string x has length l greater than d. Let us consider the $x^{(d)}$-walks from q_1 and q_2 in M_0, or equivalently, in the unlabeled graph G_M. Define the $d+1$ state pairs (r_1^i, r_2^i) of G_M by $(r_1^i, r_2^i) = (q_1 x^{(i)}, q_2 x^{(i)})$ for $0 \leqslant i \leqslant d$. Since x was a distinguishing string in M, $r_1^i \neq r_2^i$ for all i. Furthermore, Lemma 13 tells us that at least $d+1$ unique states appear in these state pairs. Consider the following process for randomly and independently labeling the states of G_M appearing in the state pairs: initially all states are unlabeled. At each step, we choose a state pair (r_1^i, r_2^i) in which one or both states are still unlabeled and choose a random label for the unlabeled state(s). Note that with probability 1/2, on the current step $x^{(d)}$ becomes a distinguishing string for q_1 and q_2 in the automaton under construction. Now after k steps of this process, at most $2k$ states can be labeled. As long as $2k < d+1$ there must still remain a pair with both states unlabeled. This method yields at least $(d+1)/2$ independent trials, each of which has probability 1/2 of making $x^{(d)}$ a distinguishing string for q_1 and q_2. Thus the probability that $x^{(d)}$ fails to be a distinguishing string for q_1 and q_2 in M is at most $2^{-(d+1)/2}$.

For any fixed pair of states q_1 and q_2 of G_M, the probability that q_1 and q_2 are inequivalent in M but indistinguishable by strings of length d is at most $2^{-(d+1)/2}$. Thus the probability of this occurring for any pair of states in M is bounded by $n^2 \cdot 2^{-(d+1)/2}$. If $d \geqslant 2 \log(n^2/\delta)$ this probability is smaller than δ. ▌(Theorem 2)

APPENDIX B

Learning Typical Automata in the PAC Model

In this appendix we show how algorithm **Reset** although derived in the Reset-on-Default model, can be modified to learn finite automata in an average-case, PAC-like model. Specifically, such a model assumes that each example is a random input sequence along with the output sequence that results from executing the input sequence on the target machine.[1] Each input sequence is generated by the following process: first, the length l of the sequence is chosen according to an arbitrary distribution; then an input sequence is chosen uniformly at random from $\{0, 1\}^l$. The goal is to learn to predict well the output sequences given a random input sequence.

Definition 5. Let $M = (Q, \tau, \gamma, q_0)$ be the target automaton, let $D: \mathcal{N} \rightarrow [0, 1]$ be an arbitrary distribution over the lengths of the input sequences, and let $D_U: \{0, 1\}^* \rightarrow [0, 1]$ be the distribution on sequences defined by choosing a length l according to D, and then choosing a sequence of length l uniformly. We say that $M' = (Q', \tau', \gamma', q_0')$ is an ε-good hypothesis with respect to M and D_U if

$$Pr_{D_U}[q_0\langle x\rangle \neq q_0'\langle x\rangle] \leqslant \varepsilon.$$

Theorem 14. *There exists an algorithm that takes n, the confidence parameter δ, an approximation parameter ε, and an additional confidence parameter δ_{PAC} as input,*

[1] The mere fact that we are providing the learner with the entire output of the machine on a complete walk does not in itself make (worst-case) PAC-learning of finite automata any easier; for instance, the negative results of Kearns and Valiant [14] hold even when the learner is provided with this extra information.

such that for uniformly almost all n-state automata, and for every distribution D on the length of the examples, with probability at least $1-\delta_{\mathrm{PAC}}$, *after seeing a sample of size polynomial in n,* $1/\delta$, $1/\varepsilon$, *and* $1/\delta_{\mathrm{PAC}}$, *and after time polynomial in the same parameters and the length of the longest sample sequence, the algorithm outputs a hypothesis* M' *which is an* ε*-good hypothesis with respect to* M *and* D_U.

Note that in the theorem above we have two sources of failure probabilities. The first stems from the random choice of the target automaton M, where we allow failure probability δ, and the second emanates from the random choice of the sample, where we allow failure probability δ_{PAC}.

Proof. The PAC learning algorithm uses **Reset** as a subroutine in the following straightforward manner. For every given sample sequence it simulates **Reset** on the random walk corresponding to this sequence until either **Reset** performs a default mistake or the walk ends. It continues in this manner, allowing **Reset** to build its hypothesis M', until either **Reset** has a complete hypothesis automaton (i.e., Q_{inc} is empty and τ' is completely defined), or, for $(1/\varepsilon)\log(2/\delta_{\mathrm{PAC}})$ consecutive sample sequences, **Reset** has not made any default mistake (and has not changed its hypothesis as well). This process can be viewed as *testing* the hypotheses constructed by **Reset** until one succeeds in predicting correctly the output sequences of a large enough number of randomly chosen sample sequences. In the former case we have a hypothesis automaton M' which is *equivalent* to M. In the latter case, we can extend τ' wherever it is undefined in an arbitrary manner and output the resulting M'. Since M' was *consistent* with M on a random sample of size $(1/\varepsilon)\log(2/\delta_{\mathrm{PAC}})$, with probability at least $1-\delta_{\mathrm{PAC}}/2$, it is an ε-good hypothesis with respect to M. It remains to show that with probability at least $1-\delta_{\mathrm{PAC}}/2$, the sample size needed to ensure that this event occurs, is polynomial in the relevant parameters.

From Theorem 1 we know that for uniformly almost all n-state automata, the expected number of default mistakes made by **Reset** in the Reset-on-Default model is $O((n^5/\delta^2)\log(n/\delta))$. In the PAC-like model considered here the algorithm receives a series of sequences, and hence, differently from the Reset-on-Default model, the target DFA is effectively reset at the end of each sample sequence even if **Reset** did not make a default mistake. However, it is easily verified that the analysis in Lemma 4 can be directly adapted to yield the same upper bound on the expected number of default mistakes made in the PAC-like model. By Markov's inequality we have that with probability at least $1-\delta_{\mathrm{PAC}}/2$, the *total* number of default mistakes **Reset** makes (on an infinite sample) is $O((n^5/(\delta^2\cdot\delta_{\mathrm{PAC}})\log(n/\delta))$. Therefore, with probability at least $1-\delta_{\mathrm{PAC}}/2$, after seeing $O((1/\varepsilon)\log(2/\delta_{\mathrm{PAC}})\cdot(n^5/(\delta^2\cdot\delta_{\mathrm{PAC}})\log(n/\delta))$ sample sequences, there must be $(1/\varepsilon)\log(2/\delta_{\mathrm{PAC}})$ consecutive sample sequences on which **Reset** has not made a single default mistake.

ACKNOWLEDGMENTS

This research was conducted while Y. Freund, D. Ron, R. Rubinfeld and L. Sellie were visiting AT&T Bell Labs.

Received May 31, 1996; final manuscript received May 2, 1997

REFERENCES

1. Angluin, D. (1987), Learning regular sets from queries and counterexamples, *Inform. and Comput.* **75**, 87–106.
2. Angluin, Dana (1978), On the complexity of minimum inference of regular sets, *Inform. and Control* **39**, 337–350.
3. Azar, Y., Broder, A. Z., Karlin, A. R., Linial, N., and Phillips, S. (1992), Biased random walks, *in* "Proceedings of the Twenty-Fourth Annual ACM Symposium on the Theory of Computing," pp. 1–9.
4. Barzdin', Y. M. (1970), Deciphering of sequential networks in the absence of an upper limit on the number of states, *Sov. Phys. Dokl.* **15**(2), 94–97.
5. Blum, A. (1994), Some tools for approximate 3-coloring, *J. Assoc. Comput. Mach.* **41**(3), 470–516.
6. Chor, B., and Goldreich, O. (1988), Unbiased bits from sources of weak randomness and probabilistic communication complexity, *SIAM J. Comput.* **17**, 230–261.
7. Dean, T., Angluin, D., Basye, K., Engelson, S., Kaelbling, L., Kokkevis, E., and Maron, O. (1995), Inferring finite automata with stochastic output functions and an application to map learning, *Mach. Learning* **18**(1), 81–108.
8. Feder, M., Merhav, N., and Gutman, M. (1992), Universal prediction of individual sequences, *IEEE Trans. Inform. Theory* **38**, 1258–1270.
9. Feller, W. (1968), "An Introduction to Probability and Its Applications," 3rd ed., Vol. 1, Wiley, New York.
10. Gold, M. E. (1972), System identification via state characterization, *Automat.* **8**, 621–636.
11. Gold, M. E. (1978), Complexity of automaton identification from given data, *Inform. and Control* **37**, 302–320.
12. Haussler, D., Littlestone, N., and Warmuth, M. K. (1994), Predicting $\{0, 1\}$-functions on randomly drawn points, *Inform. and Comput.* **115**, 248–292.
13. Kearns, M. J., Mansour, Y., Ron, D., Rubinfeld, R., Schapire, R. E., and Sellie, L. (1994), On the learnability of discrete distributions, *in* "The 25th Annual ACM Symposium on Theory of Computing," pp. 273–282.
14. Kearns, M. J., and Valiant, L. G. (1994), Cryptographic limitations on learning Boolean formulae and finite automata, *J. Assoc. Comput. Mach.* **41**, 67–95.
15. Lang, K. J. (1992), Random DFA's can be approximately learned from sparse uniform examples, *in* "Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory," pp. 45–52.
16. Littlestone, N. (1988), Learning when irrelevant attributes abound: A new linear-threshold algorithm, *Mach. Learning* **2**, 285–318.
17. Pitt, L., and Valiant, L. G. (1988), Computational limitations on learning from examples, *J. Assoc. Comput. Mach.* **35**(4), 965–984.
18. Pitt, L., and Warmuth, M. K. (1990), Prediction-preserving reducibility, *J. Comput. System Sci.* **41**(3), 430–467.
19. Pitt, L., and Warmuth, M. K. (1993), The minimum consistent DFA problem cannot be approximated within any polynomial, *J. Assoc. Comput. Mach.* **40**(1), 95–142.
20. Rivest, R. L., and Schapire, R. E. (1993), Inference of finite automata using homing sequences, *Inform. and Comput.* **103**(2), 299–347.
21. Rivest, R. L., and Schapire, R. E. (1994), Diversity-based inference of finite automata, *J. Assoc. Comput. Mach.* **43**(3), 555–589.
22. Rivest, R. L., and Sloan, R. (1988), Learning complicated concepts reliably and usefully, *in* "Proceedings AAAI-99," pp. 635–639.
23. Ron, D., Singer, Y., and Tishby, N. (1995), On the learnability and usage of acyclic probabilistic finite automata, *in* "Proceedings of the Eighth Annual ACM Workshop on Computational Learning Theory," pp. 31–40.

24. Santha, M., and Vazirani, U. V. (1986), Generating quasi-random sequences from semi-random sources, *J. Comput. System Sci.* **33**(1), 75–87.

25. Trakhtenbrot, B. A., and Barzdin', Ya. M. (1973), "Finite Automata: Behavior and Synthesis," North-Holland, Amsterdam.

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

27. Vazirani, U. V. (1987), Strong communication complexity or generating quasi-random sequences from two communicating semi-random sources, *Combinatorica* **7**, 375–392.

28. Vazirani, U. V., and Vazirani, V. V. (1985), Random polynomial time is equal to slightly-random polynomial time, *in* "26th Annual Symposium on Foundations of Computer Science," pp. 417–428.