On the Exact Learnability of Automata with Small Cover Time

Dana Ron
Computer Science Institute
Hebrew University
Jerusalem 91904, Israel
danar@cs.huji.ac.il

Ronitt Rubinfeld
Computer Science Department
Cornell University
Ithaca, NY 14853
ronitt@cs.cornell.edu

Abstract

We present algorithms for exactly learning unknown environments that can be described by deterministic finite automata. The learner performs a walk on the target automaton, where at each step it observes the output of the state it is at, and chooses a labeled edge to traverse to the next state. We assume that the learner has no means of a reset, and we also assume that the learner does not have access to a teacher that gives it counterexamples to its hypotheses. We present two algorithms, one assumes that the outputs observed by the learner are always correct and the other assumes that the outputs might be erroneous. The running times of both algorithms are polynomial in the cover time of the underlying graph of the target automaton.

1 Introduction

In this paper we study the problem of actively learning an environment which is described by a deterministic finite state automaton. The learner can be viewed as a robot performing a walk on the target automaton M, beginning at the start state of M. At each time step it observes the output of the state it is at, and chooses a labeled edge to traverse to the next state. The learner does not have a means of a reset (returning to the start state of M). In particular, we investigate *exact* learning algorithms which *do not have access to a teacher that can give it counterexamples* to its hypotheses. We also study the case in which the environment is noisy, in the sense that there is some fixed probability η that the learner observes an incorrect output of the state it is at.

For both the noise-free and the noisy settings we present probabilistic learning algorithms for which the following holds. With high probability, the algorithm outputs a hypothesis automaton which can be used to correctly predict the outputs of the states on any path starting from the state at which the hypothesis was output. Both algorithms run in time polynomial in the *cover time* of M. The cover time of M is defined to be the smallest integer t such that for every state q in M, a random walk of length t starting from q visits every state in M with probability at least $1/2$. In the noisy setting we allow the running time of the algorithm to depend polynomially in $1/\alpha$, where α is a lower bound on $1/2 - \eta$. We restrict our attention to the case in which each edge is labeled either by 0 or 1, and the output of each state is either 0 or 1. Our results are easily extendible to larger alphabets.

Our results follow the no-reset learning algorithm of Rivest and Schapire [10], which in turn uses Angluin's algorithm [2] (for learning automata with reset) as a subroutine. We use a variant of Angluin's algorithm which is similar to the one described in [1]. As in [10], we use a *homing sequence* to overcome the absence of a reset, only we are able to construct such a sequence without the aid of a teacher, while Rivest and Schapire need a teacher to answer their equivalence queries and supply them with counterexamples for their incorrect hypotheses. We "pay" for the absence of a teacher by having the running time of the algorithm depend on the cover time of M, and thus the algorithm is efficient only if the cover time is polynomial in the number of states in M.

In the noisy setting we use a "looping" idea presented by Dean *et. al.* [5]. Dean *et. al.* study a similar setting in which the noise rate is not fixed but is a function of the current state, and present a learning algorithm for this problem. However, they assume that the algorithm is either given a distinguishing sequence for the target automaton, or can generate one efficiently with high probability. It is known that not every automaton has a distinguishing sequence. Moreover, even if the target automaton is known to have a distinguishing sequence, then there is not necessarily an efficient procedure for finding one such sequence, and this remains true when the automaton has small cover time.

Other Related Work Angluin [1] proves that it is hard to exactly learn automata (even with reset) when the learner has access to a membership oracle but not to an equivalence oracle. However, this does not contradict our results, as the adversary argument made by Angluin uses automata which have hard to reach states and hence exponential cover time. In fact, Angluin shows that if you have a method of efficiently reaching all states of the automaton (which is true for graphs with polynomial cover time), then you can exactly learn the automaton using reset.

In [7], it is shown how to learn typical automata (automata in which the underlying graph is arbitrary, but the accept/reject labels on the states are chosen randomly) by passive learning (the

edge traversed by the robot is chosen randomly). In [12], an algorithm is given for learning DFAs with reset in the presence of persistent noise. In [6], an algorithm is given for learning DFAs by blurry concepts. Bender and Slonim [4] show how two robots can exactly learn directed graphs with no outputs on the states, while this task can not be performed by one robot and a constant number of pebbles. They generalize the notion of a homing sequence to this setting, and show how such a *two-robot* homing sequence can be constructed efficiently (without a teacher) using the power of two robots. They also show how their algorithm can be modified and made more efficient if the graph has high conductance [14], where conductance is a measure of the expansion properties of the graph (not necessarily related to its cover time).

2 Preliminaries

2.1 Basic Definitions

Let M be the deterministic finite state automaton (DFA) we would like to learn. M is a 4-tuple (Q, τ, q_0, γ) where Q is a finite set of n *states*, $\tau : Q \times \{0,1\} \to Q$ is the *transition* function, $q_0 \in Q$ is the *starting* state, and $\gamma : Q \to \{0,1\}$, is the *output* function. The transition function, τ, can be extended to be defined on $Q \times \{0,1\}^*$ in the usual manner. The *output* of a state q is $\gamma(q)$. The *output* associated with string $u \in \{0,1\}^*$ is defined as the output of the state reached by u, i.e., the output of $\tau(q_0, u)$, and is denoted by $\overline{M}(u)$. Unless stated otherwise, all strings referred to are over the alphabet $\{0,1\}$.

For $0 < \Delta < 1$, let the Δ-*cover time* of M, denoted by $C_\Delta(M)$ be defined as follows. For every state $q \in Q$, with probability at least $1 - \Delta$, a random walk of length $C_\Delta(M)$ on the underlying graph of M, starting at q, passes through *every* state in M. The *cover time* of M, denoted by $C(M)$ is defined to be the $1/2$-cover time of M. Clearly, for every $0 < \Delta < 1/2$, $C_\Delta(M) \leq C(M)\log(1/\Delta)$.

For two strings s_1 and s_2, let $s_1 \cdot s_2$ denote the concatenation of s_1 with s_2. For two sets of strings S_1 and S_2 let $S_1 \circ S_2 \stackrel{\text{def}}{=} \{s_1 \cdot s_2 \mid s_1 \in S_1, s_2 \in S_2\}$. Let the empty string be denoted by λ. A set of strings S is said to be *prefix closed* if for every string $s \in S$, all prefixes of s (including λ and s itself), are in S. A *suffix closed* set of strings is defined similarly. For a string $s = s_1 \ldots s_t$, and for $0 \leq \ell \leq t$, the *length* ℓ *prefix* of s is $s_1 \ldots s_\ell$, (where the length 0 prefix is defined to be λ).

2.2 The Learning Models

2.2.1 The noise free model

The problem we study is that of exactly learning a deterministic finite state automaton when the learning algorithm has no means of resetting the automaton. The learning algorithm can be viewed as performing a "walk" on the automaton starting at q_0. At each time step, the algorithm is at some state q, and can observe q's output. The algorithm then chooses a symbol $\sigma \in \{0,1\}$, upon which it moves to the state $\tau(q, \sigma)$. In the course of this walk it constructs a hypothesis DFA. The algorithm has *exactly learned* the target DFA if its hypothesis can be used to correctly predict the sequence of outputs corresponding to *any* given walk on the target DFA starting from the current state that it is at. The learning algorithm is an *exact* learning algorithm, if for every given $\delta > 0$, with probability at least $1 - \delta$, it exactly learns the target DFA. We also require that it be efficient,

i.e., that it run in time polynomial in n and $\log(1/\delta)$. We assume that the algorithm is given an upper bound on the cover time of M.

We also consider the easier setting in which the learning algorithm has a means of resetting the machine and may ask membership queries. Namely, for any string $s \in \{0,1\}^*$ it can query a *teacher* on the output of s according to M. We require that for any given $\delta > 0$, after asking a polynomial (in n and $\log 1/\delta$) number of queries, each of polynomial length, it output a hypothesis $\widehat{M}$, which equivalent to M, i.e., for every string s, $\overline{\widehat{M}}(s) = \overline{M}(s)$.

2.2.2 The noisy model

Our assumptions on the noise follow the *classification* noise model introduced by Angluin and Laird [3]. We assume that for some fixed noise rate $\eta < 1/2$, at each step, with probability $1-\eta$ the algorithm observes the (correct) output of the state it has reached, and with probability η it observes an incorrect output. The observed output of a state q reached by the algorithm is thus an independent random variable which is $\gamma(q)$ with probability $1-\eta$, and $\overline{\gamma(q)}$ with probability η. We do not assume that η is known, but we assume that some lower bound, α, on $1/2-\eta$, is known to the algorithm.

As in the noise free model, the algorithm performs a single walk on the target DFA M, and is required to exactly learn M as defined above, where the predictions based on its final hypothesis must all agree with the correct outputs of M. Since the task of learning becomes harder as η approaches $1/2$, and α approaches 0, we allow the running algorithm to depend polynomially on $1/\alpha$, as well as on n and $\log 1/\delta$.

3 Exact Learning with Reset

In this section we describe a simple variant of Angluin's algorithm [2] for PAC learning deterministic finite automata that was used in [12] and is similar to that in [1]. The algorithm works in the setting where the learner is given access to membership queries. The analysis is similar to that in [1] and shows that if the target automaton M has cover time $C(M)$ then the algorithm exactly learns the target automaton in time linear in $nC(M)$. We name the algorithm Exact-Learn-with-Reset, and it is used as a subroutine in the learning algorithm that has no means of a reset, which is described in Section 4.

Following Angluin, the algorithm constructs an *Observation Table*. An observation table is a table whose rows are labeled by a prefix closed set of strings, R, and whose columns are labeled by a suffix closed set of strings, S. An entry in the table corresponding to a row labeled by the string r_i, and a column labeled by the string s_j, is $\overline{M}(r_i \cdot s_j)$. We also refer to $\overline{M}(r_i \cdot s_j)$ as the *behavior* of r_i on s_j. An observation table T induces a *partition* of the strings in R, according to their behavior on suffixes in S. Strings that reach the same state will be in the same equivalence class of the partition. The aim is to refine the partition such that *only* strings reaching the same state will be in the same equivalence class, in which case we show that we can construct an automaton based on the partition which is equivalent to the target automaton.

More formally, for an observation table T and a string $r_i \in R$, let $T(r_i)$ denote the row in T labeled by r_i. Namely, if $S = \{s_1, \ldots, s_t\}$, then $row_T(r_i) = (T(r_i \cdot s_1), \ldots, T(r_i \cdot s_t))$. We say that two strings, $r_i, r_j \in R$ belong to the same *equivalence class* according to T, if $row_T(r_i) = row_T(r_j)$.

Given table T, we say that T is *consistent* if the following condition holds. For every pair of strings $r_i, r_j \in R$ such that r_i and r_j are in the same equivalence class, if $r_i{\cdot}\sigma, r_j{\cdot}\sigma \in R$ for $\sigma \in \{0,1\}$, then $r_i{\cdot}\sigma$ and $r_j{\cdot}\sigma$ belong to the same equivalence class as well. We say that T is *closed* if for every string $r_i \in R$ such that for some $\sigma \in \{0,1\}$, $r_i{\cdot}\sigma \notin R$, there exists a string $r_j \in R$ such that r_i and r_j belong to the same equivalence class according to T, and for every $\sigma \in \{0,1\}$, $r_i{\cdot}\sigma \in R$.

Given a closed and consistent table T, we define the following automaton, $M^T = \{Q^T, \tau^T, q_0^T, \gamma^T\}$, where each equivalence class corresponds to a state in M^T:

- $Q^T \stackrel{\text{def}}{=} \{row_T(r_i) \mid r_i \in R, \forall \sigma \in \{0,1\}, r_i{\cdot}\sigma \in R\}$;
- $\tau^T(row_T(r_i), \sigma) \stackrel{\text{def}}{=} row_T(r_i{\cdot}\sigma)$;
- $q_0^T \stackrel{\text{def}}{=} row_T(\lambda)$;
- $\gamma^T(row_T(r_i)) \stackrel{\text{def}}{=} T(r_i, \lambda)$;

It is not hard to verify that M^T is consistent with T in the sense that for every $r_i \in R$, and for every $s_j \in S$, $\overline{M^T}(r_i{\cdot}s_j) = T(r_i, s_j)$.

The idea of the algorithm is as follows — first use a random walk to construct a set of strings that with high probability reach every state in the automaton. Then we show how to use these strings to construct an observation table that has an equivalence class for each state. Let $r \in \{0,1\}^m$ be a random string of length $m = C_\delta(M)$ (where δ is the confidence parameter). Let $R_1 = \{r_i \mid r_i \text{ is a prefix of } r\}$, $R_2 = R_1 \circ \{\sigma\}$ for $\sigma \in \{0,1\}$, and $R = R_1 \bigcup R_2$. The learning algorithm initializes S to include only the empty string, λ, and fills in the (single columned) table by performing membership queries. Let us first observe that from the definition of $C_\delta(M)$, we have that with probability at least $1-\delta$, for every state $q \in Q$, there exists a string $r_i \in R_1$, such that $\tau(q_0, r_i) = q$. Assume that this is in fact the case. It directly follows that T is always closed. Hence, the learning algorithm must only ensure that T be consistent. This is done as follows. If there exists a pair of strings $r_i, r_j \in R$ such that $row_T(r_i) = row_T(r_j)$, but for some $\sigma \in \{0,1\}$, $row_T(r_i{\cdot}\sigma) \neq row_T(r_j{\cdot}\sigma)$, then a string $\sigma{\cdot}s_k$ is added to S, where $s_k \in S$ is such that $T(r_i{\cdot}\sigma, s_k) \neq T(r_j{\cdot}\sigma, k)$, and the new entries in T are filled in. The pseudo-code for the algorithm appears in Figure 1.

It is clear that the *inconsistency resolving* process (stage 4 in the algorithm given in Figure 1) ends after at most n steps. This is true since every string added to S refines the partition induced by T. On the other hand, the number of equivalence classes cannot exceed n, since for every pair of strings $r_i, r_j \in R$ such that $row_T(r_i) \neq row_T(r_j)$, $\tau(q_0, r_i) \neq \tau(q_0, r_j)$. The running time of the algorithm is hence at most $3nC(M)\log(1/\delta)$. We further make the following claim, whose proof is given in Appendix A.

Lemma 3.1 *If for every state $q \in Q$, there exists a string $r_i \in R_1$ such that $\tau(q_0, r_i) = q$, then $M^T \equiv M$.*

We thus have the following theorem.

Theorem 1 *For every target automaton M, if $C(M) = poly(n)$, then the running time of* Exact-Learn-with-Reset *is $poly(n, \log(1/\delta))$, and with probability at least $1-\delta$ it outputs a hypothesis DFA which is equivalent to M.*

Algorithm Exact-Learn-with-Reset(δ)

1. let r be random string of length $m = C_\delta(M)$;
2. let R_1 be the set of all prefixes of r; $R_2 \leftarrow R_1 \circ \{0,1\}$;
3. initialize T: $R \leftarrow R_1 \bigcup R_2$, $S \leftarrow \{\lambda\}$, query all strings in $R \circ S$ to fill in T;
4. while T is not consistent do:
   - if exist $r_i, r_j \in R_1$, s.t. $row_T(r_i) = row_T(r_j)$ but for some $\sigma \in \{0,1\}$, $row_T(r_i \cdot \sigma) \neq row_T(r_j \cdot \sigma)$ then:
     (a) let $s_k \in S$ be such that $T(r_i \cdot \sigma, s_k) \neq T(r_j \cdot \sigma, s_k)$;
     (b) update T: $S \leftarrow S \bigcup \{\sigma \cdot s_k\}$, fill new entries in table;
   - /* else table is consistent */
5. if exists $r_i \in R_2$ for which there is no $r_j \in R_1$ such that $row_T(r_i) = row_T(r_j)$ (T is not closed), then return to 1 (rerun algorithm);[a]

[a] Though we assume that with high probability the event that the table is not closed does not occur, we add this last statement for completeness. We could of course solve this situation as in Angluin's algorithm, but we choose this solution for the sake of brevity.

Figure 1: Algorithm **Exact-Learn-with-reset**

4 Exact Learning without Reset

In this section we describe an exact learning algorithm (as defined in Subsection 2.2) for automata whose cover time is polynomial in their size. This algorithm closely follows Rivest and Schapire's learning algorithm [10]. However, we use new techniques that exploit the small cover time of the automaton in place of relying on a teacher who supplies us with counterexamples to incorrect hypotheses.

As in [10], we overcome the absence of a reset, by the use of a *homing sequence*, defined below.

DEFINITION 4.1 *For a state q and sequence $s = s_1 \ldots s_t \in \{0,1\}^t$, let*

$$q\langle s \rangle \stackrel{\text{def}}{=} \gamma(q)\gamma(\tau(q,s_1)) \ldots \gamma(\tau(q,s)) \ .$$

A homing sequence *$h \in \{0,1\}^*$, is a sequence of symbols such that for every pair of states $q_1, q_2 \in Q$, if $q_1\langle h \rangle = q_2\langle h \rangle$, then $\tau(q_1,h) = \tau(q_2,h)$.*

It is not hard to verify (*cf.* [9]) that *every* DFA has a homing sequence of length at most quadratic in its size. Moreover, given the DFA, such a homing sequence can be found efficiently. Assume we had a homing sequence h for our target DFA M (we shall of course remove this assumption shortly). Then we would create at most n copies of the algorithm *Exact-Learn-with-Reset*, $ELR_{\pi^1}, \ldots, ELR_{\pi^n}$, each corresponding to a different output sequence π^i of the homing sequence, and hence to a different effective starting state. At each stage, the algorithm walks according to the homing sequence, observing the output sequence π, and then simulates the next step of ELR_π. The algorithm terminates when one of these copies completes. If we run each copy of *Exact-Learn-with-Reset* with the confidence parameter δ/n, then using Theorem 1, with

probability at least $1-\delta$ the hypothesis of the completed copy is correct, and the running time of the algorithm is at most $n^4 C(M)\log(n/\delta)$. Details of the algorithm are given in Figure 2.

Algorithm Exact-Learn-with-Homing-Sequence(δ)

- while no copy of **Exact-Learn-with-Reset** has completed do:
  1. perform the walk corresponding to h, and let π be the corresponding output sequence;
  2. if there does not exist a copy ELR_π of **Exact-Learn-with-Reset**(δ/n), then create such a new copy;
  3. simulate the next query of ELR_π by performing the corresponding walk;
  4. if the observation table T_π of ELR_π is consistent and closed then output M^{T_π}; /* ELR_π has completed */
  5. if T_π is consistent but not closed, then discard ELR_π;

Figure 2: Algorithm **Exact-Learn-with-Homing-Sequence**

If h is unknown, consider the case in which we guess a sequence h' which is *not* a homing sequence and run the algorithm *Exact-Learn-with-Homing-Sequence* with h' instead of h. Since h' is not a homing sequence, there exist two states $q_1 \neq q_2$, such that for some pair of states q_1', q_2', a walk starting at q_1' reaches q_1 upon executing h' and a walk starting at q_2' reaches q_2 upon executing h', but the output sequence in both cases is the same. Therefore, when we simulate the copy ELR_π, some of the queries (walks) might be performed starting from q_1, and some might be performed starting from q_2.

There are two possible consequences to such an event. The first is that the algorithm *discovers* that there is no single starting state corresponding to the observation table T_π. This can be discovered either by observing two different outputs when performing the same walk (which might happen when the algorithm asks two queries such that one is a prefix of the other), or if the number of columns in T_π is greater than n. In the first case we have found a distinguishing sequence between q_1 and q_2, and in the second case, we can use a probabilistic procedure proposed by Rivest and Schapire [10] to find such a distinguishing sequence. In both cases we can simply extend h' by the distinguishing sequence found and restart the algorithm with the "improved" h'. Clearly, h' can be extended in this manner at most $n-1$ times before it becomes a true homing sequence, whose total length is at most $(n-1)(C(M)\log(n/\delta)+n)$ plus the initial length of h'.

The second possible consequence is that the observation table T_π becomes consistent and closed, but the hypothesis M^{T_π} is *incorrect*. This may occur since different entries correspond to queries answered from different effective starting states. If the learning algorithm had access to a teacher who would supply it with a counterexample to its incorrect hypothesis, then it could extend h' by the counterexample and proceed as described above. In what follows we describe how to modify the algorithm so that it constructs a (true) homing sequence without the aid of a teacher. The pseudocode for the algorithm appears in Figure 3. It is interesting to note that the (tempting) idea to try and randomly guess a counterexample does not work even in the case of automata that have small cover time. Rivest and Zuckerman [11] construct a pair of automata which both have small cover time, but for which the probability of randomly guessing a distinguishing sequence is exponentially small. These automata are described in Appendix B.

We now show how we can avoid the second possible consequence and discover whether a copy of

ELR_π is simulated from several effective starting states. We define the algorithm Exact-Learn-with-Reset-R to be a variant of Exact-Learn-with-Reset, in which each query is *repeated* N consecutive times, where N is set subsequently. Suppose that instead of simulating copies ELR_π, of Exact-Learn-with-Reset, as in Exact-Learn-with-Homing-Sequence, we simulate copies $ELRR_\pi$ of Exact-Learn-with-Reset-R. Then, if we find that for the same query w, we get two different answers (in two different simulations of some $ELRR_\pi$), then we know that h is not a homing sequence, and thus we extend h by w, and restart the algorithm.[1] We would like to ensure, that with high probability, if a copy $ELRR_\pi$ corresponds to more than one starting state, and there are two of these starting states that disagree on some entry in T_π, then we shall discover it.

For a candidate homing sequence h, let Q_π be the set of states reached upon executing h and observing π. Namely, $Q_\pi \stackrel{\text{def}}{=} \{q \ : \ q \in Q, \ \exists q' \text{ s.t. } q'\langle h \rangle = \pi\}$. For a state $q \in Q_\pi$, let $B(q)$ be the set of states from which q is reached upon executing h, i.e., $B(q) \stackrel{\text{def}}{=} \{q' \ : \ q' \in Q, \ \tau(q', h) = q\}$. We would like to guarantee, that with high probability, for every query to fill an entry in T_π, and for every $q \in Q_\pi$, one of the N repetitions of the query is executed starting from q. To this end we do the following. Each time before we execute h, we randomly choose a length $0 \leq \ell \leq C_\Delta(M)$, and perform a random walk of length ℓ. The idea behind this random walk is that for every state, there is some non-negligble probability of reaching it upon performing the random walk.

For some entry (r_i, s_j) in T_π, consider the N executions of h whose outcome was π and which were followed by the simulation of the query $r_i \cdot s_j$. For a given $q \in Q_\pi$, we bound the probability that we did not reach q after any one of the N executions of h as follows. Assume that instead of choosing a random length and performing a random walk of that length, we first randomly choose a string t of length $C_\Delta(M)$, then choose a random length ℓ, and finally perform a walk corresponding to the length ℓ prefix of t. Clearly the distribution on the states reached at the end of this walk is equivalent to the distribution on the states reached by the original randomized procedure. Thus, the probability that we did not reach some $q \in Q_\pi$ is at most $N\Delta + (1 - 1/C_\Delta(M))^N$. The first term is a bound on the probability that for one of the random strings t, none of the states in $B(q)$ were passed on the walk corresponding to t. Given that such an event did not occur, the second term is a bound on the probability that none of the prefixes chosen reached any of these states.

It remains to set Δ and N. Since we simulate at most n^2 copies $ELRR_\pi$ during the complete run of the algorithm, each with the parameter $\delta/(2n^2)$, then with probability at least $1 - \delta/2$, each copy has a row in its observation table which corresponds to each state in Q. It follows (Lemma 3.1) that when h finally turns into a homing sequence (after at most $n - 1$ extensions), and some table T_π becomes consistent, then M^{T_π} is a correct hypothesis.

For each of the possible n^2 copies $ELRR_\pi$, the number of entries in the observation table T_π is at most $nC(M)\log(2n/\delta)$. If we choose N and Δ so that

$$N\Delta + (1 - 1/(C(M)\log(1/\Delta)))^N < \delta/(2n^3 C(M)\log(2n/\delta)) \ ,$$

then with probability at least $1 - \delta/2$, we do not output a hypothesis of a copy $ELRR_\pi$ which corresponds to more than one effective starting state (unless the different starting states are equivalent).

[1] As in [10], we actually need not discard all copies and restart the algorithm, but we may only discard the copy in which the disagreement was found, and construct an *adaptive* homing sequence which results in a more efficient algorithm. For sake of simplicity of this presentation, we continue with the use of the less efficient, *preset* homing sequence.

The following choice gives us the required bound:

$$N = C(M)\log\left(\frac{4n^3C(M)\log\frac{2n}{\delta}}{\delta}\right)\log^2\left(C(M)\log\left(\frac{4n^3C(M)\log\frac{2n}{\delta}}{\delta}\right)\right) \quad ,$$

$$\Delta = \frac{\delta}{4Nn^3C(M)\log\frac{2n}{\delta}} \quad .$$

We have thus proven that:

Theorem 2 *Algorithm* Exact-Learn *is an exact learning algorithm for automata whose cover time is polynomial in* n.

Algorithm Exact-Learn(δ)

1. $h \leftarrow \lambda$;
2. while no copy of **Exact-Learn-with-Reset-R** is completed do:
   (a) choose uniformly and at random a length $\ell \in [0, \ldots, C_\Delta(M)]$, and then perform a random walk of length ℓ.
   (b) perform the walk corresponding to h, and let π be the corresponding output sequence;
   (c) if there does not exist a copy $ELRR_\pi$ of **Exact-Learn-with-Reset-R**($\delta/(2n^2)$), then create such a new copy;
   (d) simulate the next query w of $ELRR_\pi$ by performing the corresponding walk;
   (e) if the answer to w is different from a previous answer to w, then do:
      i. $h \leftarrow h \cdot w$;
      ii. discard all existing copies of **Exact-Learn-with-Reset-R**, and go to 2; /* restart algorithm with extended h */
   (f) if the observation table T_π of $ELRR_\pi$ is consistent and closed then output M^{T_π}; /* $ELRR_\pi$ has completed */
   (g) if T_π is consistent but not closed, then discard $ELRR_\pi$;

Figure 3: Algorithm **Exact-Learn**

5 Exact Learning in the Presence of Noise

In this section we describe how to modify the learning algorithm described in Section 4 in order to overcome a noisy environment. We name the new algorithm Exact-Noisy-Learn, and its pseudo-code appears in Figure 4 in Appendix C. Though the correctness of the algorithm described below relies on the fact the noise rate, η, is fixed, we hope that it can be modified to work in the case where η is not fixed, but rather that for each state q there may be a different error probability $\eta(q)$.

We first observe that η can easily be approximated within a small additive error μ, and with probability at least $1 - \delta/2$, in time polynomial in n, $1/\alpha$, $\log 1/\delta$, and $1/\mu$ by methods similar to those in [12]. The idea used is as follows. If strings w_1 and w_2 reach the same state, then for every

suffix z, $w_1 z$ and $w_2 z$ also reach the same state. Thus, the difference in the behavior of w_1 and w_2 on a set of suffixes is entirely due to the noise. If strings w_1 and w_2 reach different states, then their difference in behavior on a set of suffixes Z is due to the different actual behavior of the states on Z as well as the noise. Thus in order to estimate the noise rate, we find two strings that reach the same state and estimate the difference in their behavior. This can be done as follows. Let t be an arbitrary string of length L. Suppose t is executed $n+1$ times, and for $1 \leq i \leq n+1$, let q^i be the state reached after performing t exactly $i-1$ times and let $o^i = o^i_1 \ldots o^i_L$ be the sequence of outputs corresponding to the ith execution of t. Clearly, for some pair of indices $i \neq j$, $q^i = q^j$. For every pair $1 \leq i < j \leq n+1$, let $d^{ij} = \frac{1}{L}\sum_{k=1}^{L} o^i_k \oplus o^j_k$. Define d_{min} to be the minimum value over all such pairs, and let $\hat{\eta} < 1/2$ be the solution of the quadratic equation $2\hat{\eta}(1-\hat{\eta}) = d_{min}$. If $q^i = q^j$, then $E[d^{ij}] = 2\eta(1-\eta)$. It can easily be verified that if $q^i \neq q^j$, then $E[d^{ij}] \geq 2\eta(1-\eta)$. We can thus apply Hoeffding's inequality [8] to derive an upper bound on $|d_{min} - 2\eta(1-\eta)|$ as a function of L, which implies a lower bound on $|\hat{\eta} - \eta|$.[2] We thus assume from here on, that we have a good approximation, $\hat{\eta}$, of η.

As in the noise free case, assume first that the algorithm has means of a reset. Then we can use the technique of [13] and define a slight modification of Exact-Learn-with-Reset, named Exact-Noisy-Learn-with-Reset, which simply repeats each query N times, and fills the corresponding entry in the table with the *majority* observed label. Thus, with high probability, for an appropriate choice of N, the majority observed label is in fact the correct label of the queried string. Next we assume that the algorithm has no means of a reset, but instead, has a homing sequence, h. Clearly, in a single execution of h, with high probability, the output sequence will be erroneous. We thus adapt a technique that was used in [5]

Assume we execute the homing sequence m consecutive times, where $m >> n$. The last $m-n$ executions of the homing sequence must be following a cycle. We use this fact to estimate the output sequence corresponding to the last homing sequence executed. For $1 \leq i \leq m$, let q^i be the state reached after the ith execution of h, and let $o^i = o^i_1 \ldots o^i_{|h|}$ be the output sequence corresponding to this execution. Then there exists some (minimal) *period* p, where $1 \leq p \leq n$, such that for every $1 \leq k \leq \lfloor (m-n)/p \rfloor$, $q^m = q^{m-kp}$. For every $1 \leq j \leq |h|$ we let $\pi_j = 1$ if $(1/\lfloor (m-n)/p \rfloor)\sum_{k=1}^{\lfloor (m-n)/p \rfloor} o^{m-kv}_j > 1/2$, and 0 otherwise. It follows that with high probability, for an appropriate choice of m, the sequence $\pi = \pi_1 \ldots \pi_{|h|}$ is the *correct* output sequence corresponding to the last execution of h. In this case we could proceed as in Exact-Learn-with-Homing-Sequence, simulating copies of Exact-Noisy-Learn-with-Reset, instead of copies of Exact-Learn-with-Reset.

How do we find the period p? For each possible length $1 \leq v \leq n$, let $m_v = \lfloor (m-n)/v \rfloor$, and let $\vec{\psi}^v$ be an $|h|$ dimensional vector which is defined as follows. For $1 \leq j \leq |h|$,

$$\psi^v_j = 1/m_v \sum_{k=1}^{m_v} o^{m-kv}_j \quad .$$

Let q^i_j be the state reached after i executions of h, followed by the length j prefix of h. When $v = p$, then for every k, k', and for every j, $q^{m-kv}_j = q^{m-k'v}_j$. Therefore, with high probability, for every j, ψ^v_j is either $1-\hat{\eta} \pm \epsilon$, or $\hat{\eta} \pm \epsilon$, for some small additive error ϵ.

When $v \neq p$, then there are two possibilities. If for every j and for every k, k', $\gamma(q^{m-kv}_j) = \gamma(q^{m-k'v}_j)$ (even though q^{m-kv}_j might differ from $q^{m-k'v}_j$), then the following is still true. Define

[2] For a more detailed analysis, see [12].

π_j^v to be 1 if ψ_j^v is greater than $1/2$, and 0 if it is at most $1/2$. Then π^v is the correct output sequence corresponding to the last execution of h. Otherwise, let j be an index for which the above does not hold, and let $K_0 = \{k \mid \gamma(q_j^{m-kv}) = 0\}$, and $K_1 = \{k \mid \gamma(q_j^{m-kv}) = 1\}$. We claim that both K_0/m_v and K_1/m_v are at least $1/p \geq 1/n$. This is true since $v \cdot p$ must be a period as well, and hence for every k and k' which are multiples of $v \cdot p$, $q_j^{m-kv} = q_j^{m-k'v}$. It follows that $E[\psi_j^v] = \beta(1-\eta) + (1-\beta)\eta$ where $\beta = K_1/m_v$. Since $1/n \leq \beta \leq 1 - 1/n$,

$$\eta + (1-2\eta)\frac{1}{n} \;\leq\; E[\psi_j^v] \;\leq\; (1-\eta) - (1-2\eta)\frac{1}{n} \;.$$

For an appropriate choice of m we can thus infer the correct output sequence corresponding to the homing sequence h (or any other given sequence). The pseudo-code for the procedure described above appears in Figure 5 in Appendix C.

It remains to treat the case in which a homing sequence is not known. Similarly to the noise free case, for a (correct) output sequence π corresponding to a candidate homing sequence h, let Q_π be all states $q \in Q$ for which there exists a state $q' \in Q$ such that $q'\langle h \rangle = \pi$. For a state $q \in Q_\pi$, let $B(q)$ be the set of states q' such that $\tau(q', h^m) = q$. Let (r_i, s_j) be an entry in the table for which there exist $q_1, q_2 \in Q_\pi$, such that $\gamma(\tau(q_1, r_i \cdot s_j)) \neq \gamma(\tau(q_2, r_i \cdot s_j))$. Let $Q_\pi^1 = \{q \mid q \in Q_\pi,\ \gamma(\tau(q_1, r_i \cdot s_j)) = 1\}$, and let Q_π^0 be defined similarly. If, as in the noise free case, the query $r_i \cdot s_j$ is repeated N times, and a random walk of some length $\ell \leq kC(M)$ is performed prior to the m executions of h, then with high probability the fraction of times a state $q \in Q_\pi^1(Q_\pi^0)$ is reached is at least $1/kC(M)$. As in the analysis above for identifying a length v which is not a period, we can identify that we have a mixture of more than one starting state, and extend h by $r_i \cdot s_j$.

The above discussion constitutes a proof sketch of the following theorem:

Theorem 3 *Algorithm* Exact-Noisy-Learn *is an exact learning algorithm in the presence of noise for automata whose cover time is polynomial in n.*

References

[1] Dana Angluin. A note on the number of queries needed to identify regular languages. *Information and Control*, 51:76–87, 1981.

[2] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, November 1987.

[3] Dana Angluin and Philip Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

[4] Michael Bender and Donna Slonim. The power of team exploration: Two robots can learn unlabeled directed graphs. In *Proceedings of the Thrity Fifth Annual Symposium on Foundations of Computer Science*, pages 75–85, 1994.

[5] Thomas Dean, Dana Angluin, Kenneth Basye, Sean Engelson, Leslie Kaelbling, Evangelos Kokkevis, and Oded Maron. Inferring finite automata with stochastic output functions and an application to map learning. In *Proceedings Tenth National Conference on Artificial Intelligence*, pages 208–214, July 1992.

[6] Michael Frazier, Sally Goldman, Nina Mishra, and Leonard Pitt. Learning from a consistently ignorant teacher. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, pages 328–339, 1994.

[7] Yoav Freund, Michael J. Kearns, Dana Ron, Ronitt Rubinfeld, Robert E. Schapire, and Linda Sellie. Efficient learning of typical finite automata from random walks. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, 1993.

[8] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963.

[9] Zvi Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill, second edition, 1978.

[10] Ronald. L. Rivest and Robert. E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, 1993.

[11] Ronald L. Rivest and David Zuckermann. Private communication. 1992.

[12] Dana Ron and Ronitt Rubinfeld. Learning fallible finite state automata. In *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory*, pages 218–227, 1993. To appear in Machine Learning, COLT '93 special issue.

[13] Yasubumi Sakakibara. On learning from queries and couterexamples in the presence of noise. *Information Processing Letters*, 37:279–284, 1991.

[14] Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation, and rapidly mixing Markov chains. *Information and Computation*, 82:93–13, 1989.

A Proof of Lemma 3.1

Proof of Lemma 3.1: Let $\phi : Q \to Q^T$ be defined as follows: for each $q \in Q$, $\phi(q) = T(r_i)$, where $r_i \in R$ is such that $\tau(q_0, r_i) = q$. From the assumption in the statement of the lemma we have that for every state $q \in Q$, there exists a string $r_i \in R_1$ such that $\tau(q_0, r_i) = q$. By definition of deterministic finite automata, if for $r_i \neq r_j$ in R, $\tau(q_0, r_i) = \tau(q_0, r_j)$, then necessarily $T(r_i) = T(r_j)$. It follows that ϕ is well defined. We next show that ϕ satisfies the following properties:

1. $\phi(q_0) = q_0^T$;
2. $\forall q \in Q, \forall \sigma \in \{0,1\}, \phi(\tau(q,\sigma)) = \tau^T(\phi(q),\sigma)$;
3. $\forall q \in Q,\ \gamma(q) = \gamma^T(\phi(q))$

If ϕ has the above properties, then for every string $s \in \{0,1\}^*$, $\gamma(\tau(q_0, s)) = \gamma^T(\tau^T(q_0^T, s))$, and the claim follows.

ϕ has the first property since $\tau(q_0, \lambda) = q_0$, and $q_0^T \stackrel{\text{def}}{=} T(\lambda)$. ϕ has the third property since $\gamma^T(T(r_i)) \stackrel{\text{def}}{=} T(r_i, \lambda) = \overline{M}(r_i) = \gamma(\tau(q_0, r_i))$. It remains to prove the second property. Let $r_i \in R_1$ be such that $\tau(q_0, r_i) = q$. From the assumption in the statement of the lemma, we know there exists such a string. Thus, $\phi(q) = T(r_i)$. By definition of M^T, $\tau^T(T(r_i), \sigma) = T(r_i \cdot \sigma)$. Since $\tau(q_0, r_i) = q$, we have that $\tau(q, \sigma) = \tau(q_0, r_i \cdot \sigma)$, and by definition of ϕ, $\phi(\tau(q,\sigma)) = T(r_i \cdot \sigma) = \tau^T(T(r_i), \sigma)$.

Note that if M is irreducible in the sense that for every pair of states $q, q' \in Q$, there exists a string s of length at most n, such that the label of $\tau(q, s)$ differs from the label of $\tau(q', s)$, then ϕ is an isomorphism. ■

B Rivest and Zuckerman's example

We describe below a pair of automata, constructed by Rivest and Zuckerman, which have the following properties. Both automata have small cover time ($O(n \log n)$), but the probability that a random string distinguishes between the two is exponentially small.

The first automaton, M_1, is defined as follows. It has $n = 3k$ states that are ordered in $k+1$ columns where k is odd. Each state is denoted by $q[i,j]$, where $0 \leq i \leq k$ is the column the state belongs to, and $1 \leq j \leq 3$ is its height in the column. The starting state, $q[0,1]$ is the only state in column 0. In column 1 there are two states, $q[1,1]$ and $q[1,2]$, and in all other columns there are three states. All states have output 0 except for the state $q[k,1]$ which has output 1. The transition function, $\tau(\cdot,\cdot)$, is defined as follows. For $0 \leq i < k$,

$$\tau\left(q[i,j],0\right) = q[i+1, max(1, i-1)] \qquad \text{and} \qquad \tau\left(q[i,j],1\right) = q[i+1, min(3, i+1)]\ . \tag{1}$$

All transition from the last column are to $q[0,1]$, i.e., for $\sigma \in \{0,1\}$, $\tau\left(q[k,j],\sigma\right) = q[0,1]$.

The second automaton, M_2, is defined the same as M_1, except for the outgoing edges of $q[0,1]$, which are switched. Namely, in M_2, $\tau\left(q[0,1],0\right) = q[1,2]$, and $\tau\left(q[0,1],1\right) = q[1,1]$.

The underlying graphs of M_1 and M_2, have a strong *synchronizing* property: any walk performed in parallel on the two graphs, in which there are either two consecutive 0's or two consecutive 1's (where the latter does not include the first two symbols), will end up in the same state on both

graphs. Therefore, the *only* way to distinguish between the automata is that after any outgoing edge of $q[0,1]$ is traversed, to perform a walk corresponding to the sequence $(10)^{\frac{k-1}{2}}$. The probability this sequence is chosen on a random walk of polynomial length is clearly exponentially small.

C Algorithm Exact-Noisy-Learn

Algorithm Exact-Noisy-Learn(δ)

1. $h \leftarrow \lambda$;
2. while no copy of **Exact-Noisy-Learn-with-Reset** is completed do:
   (a) $\pi \leftarrow$ **Execute-Homing-Sequence**(h);
   (b) if there does not exist a copy $ENLR_\pi$ of **Exact-Noisy-Learn-with-Reset**($\delta/(2n^2)$), then create such a new copy;
   (c) simulate the next query w of $ENLR_\pi$ by performing the corresponding walk; let $\theta_i(w)$ be the output of the state reached, where i is the number of times w has been queried.
   (d) if $i = N$ then let $f(w) = (1/N)\sum_{i=1}^{N} \theta_i(w)$. If $|f(w) - \hat{\eta}| > \epsilon$, and $|f(w) - (1-\hat{\eta})| > \epsilon$ then do:
      i. $h \leftarrow h \cdot w$;
      ii. discard all existing copies of **Exact-Noisy-Learn-with-Reset**, and go to (2); /* restart algorithm with extended h */
   (e) if the observation table T_π of $ENLR_\pi$ is consistent and closed then output M^{T_π}; /* $ENLR_\pi$ has completed */
   (f) if T_π is consistent but not closed, then discard $ENLR_\pi$;

Figure 4: Algorithm **Exact-Noisy-Learn**

Procedure Execute-Homing-Sequence(h)

1. choose uniformly and at random a length $\ell \in [0, \ldots, C_\Delta(M)]$, and then perform a random walk of length ℓ.
2. perform the walk corresponding to h for m consecutive times, and for $1 \leq i \leq m$, let o^i be the output sequence corresponding to the ith execution of h;
3. for each length $1 \leq v \leq n$, and for every $1 \leq j \leq |h|$, let $\psi_j^v = 1/m_v \sum_{k=1}^{m_v} o_j^{m-kv}$, where $m_v = \lfloor m/v \rfloor$;
4. let v be such that for every j either $|\psi_j^v - \hat{\eta}| \leq \epsilon$, or $|\psi_j^v - (1-\hat{\eta})| \leq \epsilon$; if there is no such v, then return to (1);
5. for $1 \leq j \leq |h|$, let $\pi_j = 1$ if $m_v^j > 1/2$, and 0 otherwise;
6. return π;

Figure 5: Procedure **Execute-Homing-Sequence**