Exactly Learning Automata of Small Cover Time

Dana Ron *
Laboratory of Computer Science
MIT
Cambridge, MA 02139
danar@theory.lcs.mit.edu

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. The learner has no means of a reset, and does not have access to a teacher that answers equivalence queries and gives the learner counterexamples to its hypotheses. We present two algorithms: The first is for the case in which the outputs observed by the learner are always correct, and the second is for the case in which the outputs might be corrupted by random noise. The running times of both algorithms are polynomial in the cover time of the underlying graph of the target automaton.

*Supported by a National Science Foundation Postdoctoral Research Fellowship, Grant No. DMS-9508963

†Supported by ONR Young Investigator Award N00014-93-1-0590 and grant No. 92-00226 from the United States - Israel Binational Science Foundation (BSF), Jerusalem, Israel.

1 Introduction

In this paper we study the problem of actively learning an environment which is described by a deterministic finite state automaton (DFA). 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 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 answer equivalence queries and give the learner 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.

Angluin [2] has shown that the general problem of exactly learning finite automata by performing a walk on the target automaton, but without access to an equivalence oracle, is hard in the information theoretic sense (even when the learner has means of a reset). This is due to the existence of a subclass of automata, which are often referred to as *combination-lock automata*[1]. The central property of combination lock automata which is used in Angluin's hardness result is that they have hard-to-reach states: In particular, there is a single accepting state which is reachable only when the learner performs a particular walk of length n (called the "combination"), where n is the number of states in the automaton. All other walks result in an all zero sequence of outputs. Therefore, for every exact learning algorithm, there will be some combination lock automaton on which the algorithm requires exponential time (if the algorithm is randomized then it will require exponential expected time).

Thus, a natural question that arises is whether exact learning of automata remains hard when we assume the underlying graph of the target automaton has certain combinatorial properties such as small *cover time*. 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 $1/2$. An automaton with low cover time cannot have hard-to-reach states since a random walk whose length is of the order of the cover time is likely to reach all states.

It is known that a graph has polynomial cover time exactly when the probability assigned by the stationary distribution to each edge[2] is at least an inverse polynomial in the size of the graph (this can be inferred from the results described in [21]). Several natural classes of directed graphs are known to have this property. One important such class is the class of graphs in which the indegree of each node is equal to the outdegree [1]. This class includes the underlying graphs of permutation automata and automata that simulate undirected environments by replacing each undirected edge between states q, q', by two oppositely directed edges from q to q' and from q' to q (the labels of the directed edges can be arbitrary).

It is necessary that the learning algorithm be given an upper bound on the cover time of the target automaton (which is in turn a bound on the number of states): It is impossible for the learning algorithm to even approximate the cover time of an unknown automaton in an efficient

[1] Combination-lock automata have n states, $q_1, \ldots, q_n$, for which the output label of $q_i, 1 \leq i \leq n-1$ is 0 and the output label of q_n is 1. The start state is q_1. For each q_i, $1 \leq i \leq n-1$, there is one outgoing edge labeled by 0, and one labeled by 1, where one of these outgoing edges goes to q_1 and the other goes to q_{i+1}. The state q_n has an outgoing edge directed to q_1 and another directed to itself. The sequence of inputs that cause the automaton to traverse the state sequence $q_1, \ldots, q_n$ in order is called the "combination". The single accepting state is reachable only when the learner performs a walk which corresponds to the combination. All other walks result in an all zero sequence of outputs.

[2] We refer here to the stationary distribution as determined by the Markov chain corresponding to the graph in which for each state, all outgoing edges are assigned equal probability.

manner, due to the difficulty of distinguishing a combination lock automaton (which has exponential cover time) from the one state automaton with self-loops (which has cover time 1).

In this paper, we show that automata that have polynomial cover time can be exactly learned in polynomial time. For both the noise-free and the noisy settings described previously we present probabilistic learning algorithms for which the following holds. With high probability, after performing a single walk on the target automaton, the algorithm constructs 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 completed. Both algorithms run in time polynomial in the cover time of M. 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 by 1, and the output of each state is either 0 or 1. Our results are easily extendible to larger alphabets.

In our algorithms we apply ideas from the no-reset learning algorithm of Rivest and Schapire [22], which in turn uses Angluin's algorithm [3] as a subroutine. Angluin's algorithm is an algorithm for exactly learning automata from a teacher that can answer both membership queries and equivalence queries. Note that having a teacher which answers membership queries is equivalent to having the means of a reset. We use as a subroutine of our algorithm a variant of Angluin's algorithm which is similar to the one described in [2]. In this procedure (for learning with means of a reset) lies the first key to overcoming the need for a teacher which answers equivalence queries. At the start of the procedure, the learner performs a single random walk whose length is of the order of the cover time of the target automaton. It then proceeds by performing additional walks (starting from the start state) which are determined by the initial random walk. Using a simple argument (similar to an argument used in [2]), we show that all that is needed for the procedure to terminate (in polynomial time) with a hypothesis automaton which is equivalent to the target automaton, is that each state of the target automaton is passed in the initial walk.

As in [22], we use a *homing sequence* to overcome the absence of a reset. Informally, a homing sequence is a sequence such that whenever it is executed, the corresponding output sequence observed uniquely determines the final state reached. As was shown in [22], if a homing sequence is known, learning algorithms that use a reset can be easily converted into learning algorithms that do not use a reset. The idea is that if a homing sequence is executed at two different stages in an algorithm that does not use a reset (from two, possibly different, unknown states) and the output sequence observed is the same, then we know that at both stages we have reached the same state. Thus executing a homing sequence essentially plays the role of performing a reset. The problem that remains is how to construct a homing sequence when such a sequence is not known. Here we are able to construct a homing sequence without the aid of a teacher, while Rivest and Schapire's learner needs a teacher to answer its equivalence queries in order to construct a homing sequence. The rough idea is that by performing a random walk (whose length is bounded by the cover time) prior to each execution of the current "candidate" homing sequence, and by repeating each step in the subroutine that learns with a reset enough times, we can discover if the current candidate is not a true homing sequence and improve it. We thus "pay" for the absence of a teacher by giving an algorithm whose running time depends on the cover time of M, and hence the algorithm is efficient only if the cover time is polynomial in the number of states in M.

In the noisy setting the learning problem becomes harder since the outputs observed may be erroneous. If the learner has means of a reset then the problem can easily be solved [26] by running the noise-free algorithm and repeating each walk a large enough number of times so that the majority output observed is the correct output. However, when the learner does not have means

of a reset then we encounter several difficulties. One major difficulty is that it is not clear how the learner can orient itself since when executing a homing sequence, with high probability it does not observe the correct output sequence. In order to overcome this difficulty, we adapt a "looping" idea presented by Dean et al. [9]. 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[3]. It is known (and there are simple examples illustrating it) that some automata do *not* have a distinguishing sequence, and this remains true if we restrict our attention to automata with small cover time.

A natural question that arises is whether our results can be improved if we only require that the learner learn the target automaton *approximately*. When the learner has means of a reset it may be natural to assume that while we allow the learner to actively explore its environment, its goal is to perform well with respect to some underlying distribution on walks (each starting from the starting state). This model is equivalent to PAC learning with membership queries. Since Angluin's algorithm [3] can be modified to a PAC learning algorithm with membership queries, DFAs are efficiently learnable in this model. However, when the learner does not have means of a reset, and thus performs a single walk on M, we know of no natural notion of approximately correct learning.

In recent work of Freund et al. [13] our results have been improved as follows. Freund et al. consider the problem of learning *probabilistic output automata*. These are finite automata whose transition function is deterministic, but whose output function is probabilistic. Namely, for any given string, whenever performing the walk corresponding to the string from a certain state, we reach the same state. However, similarly to the model studied by Dean et al. [9], the output observed each time is determined by the probabilistic process of flipping a coin with a bias that depends on the state reached. In the case when the biases at each state are either η or $1-\eta$ for some $0 \leq \eta < 1/2$, this is essentially the problem of learning deterministic automata in the presence of noise, for which we give an algorithm in this paper. In [13], a learning algorithm is given that runs in time polynomial in the cover time of the target automaton, with no restrictions on the biases at each state.

Repeated games against computationally bounded opponents Another motivation for this work is the *game theoretical* problem of finding an optimal strategy when playing repeated games against a computationally bounded opponent. In this scenario there are two players. We refer to one as the *player*, and to the second as the *opponent*. At each step the player and the opponent each choose an action from a predefined set of actions according to some strategy. A strategy is a (possibly probabilistic) mapping from the history of play to the next action. The player then receives a payoff which is determined by the pair of actions played, using a fixed game matrix. The goal of the player is to maximize its average (expected) payoff. In particular, we are interested in finding good strategies of play for the player when the opponent's strategy can be computed by a computationally bounded machine such as a DFA. Namely, starting from the starting state, the opponent outputs the action labeling the state it is at, and the action played by the player determines the opponent's next state[4].

[3] A distinguishing sequence is a sequence of input symbols with the following property. If the automaton is at some unknown starting state and is given the sequence as input, then the output sequence observed determines this unknown starting state.

[4] There is a slight difference between the learning scenario and the game playing scenario since in the latter, the player sees the action chosen by the opponent only after choosing its action. However, our algorithms can easily be

It is known [15] that there exist optimal strategies in which the player simply forces the opponent DFA M to follow a cycle along the nodes of M's underlying graph. If M is known to the player, then it is not hard to prove that the player can find an optimal cycle strategy efficiently using dynamic programming. However, if M is not known to the player, then Fortnow and Whang [11] show, using the same combination-lock automata argument of Angluin [2], that it is hard to find an optimal strategy in the case of a general game[5]. Clearly, if a class of automata can be learned exactly and efficiently without reset, then an optimal cycle strategy can be found efficiently. However, it is important that the learning algorithm not use any additional source of information regarding the target automaton (such as counterexamples to its hypotheses), otherwise the learning algorithm cannot be used in the game playing scenario.

Other Related Work

Several researchers have considered the problem of learning DFAs *in the limit*. In this setting the learner is presented with an infinite sequence of examples labeled according to an unknown DFA and is required to output hypotheses that converge in the limit (of the number of examples) to the target DFA. We refer the reader to a survey by Angluin and Smith [6]. Here we briefly survey the known efficient learning algorithms for DFAs.

We start with the problem of exactly learning DFAs. In addition to the work of Angluin [2, 3] and Rivest and Schapire [22] that were discussed previously, the following is also known: Rivest and Schapire [23] show how permutation automata can be exactly learned efficiently without means of a reset and without making equivalence queries. Since permutation automata have the property that the indegree and outdegree of each node is equal, the underlying automata has small cover time and thus our result can be viewed as a generalization. Angluin [4] proves that the problem of exactly learning DFAs from equivalence queries alone is hard. Ibarra and Jiang [17] show that the subclass of k-bounded regular languages can be exactly learned from a polynomial number of equivalence queries.

Bender and Slonim [7] study the related problem of exactly learning directed graphs (which do no have any outputs associated with their nodes). They show that this task can be performed efficiently by two cooperating robots where each robot performs a single walk on the target graph. In contrast they show that this task cannot be performed efficiently by one robot which perform a single walk even if the robot may use a constant number of pebbles to mark states it passes. They also show how their algorithm can be modified and made more efficient if the graph has high conductance [28], where conductance is a measure of the expansion properties of the graph.

Bergando and Varricchio [8] show that automata with multiplicity can be exactly learned from multiplicity and equivalence queries. In particular this implies the learnability of probabilistic automata, in which each input string may correspond to many paths, each assigned a probability which is the product of the probabilities on the edges in the path. These automata can be exactly learned when given access to an equivalence oracle and an oracle which for any given string returns the probability that this string reaches an accepting state.

As for non-exact (approximate) learning, without the aid of queries, Kearns and Valiant [18] show that under certain number theoretical assumptions, the problem of PAC learning DFAs is

modified to adapt to this difference.

[5]For certain games, such as *penny matching* (where the player gets positive payoff if and only if it matches the opponent's action), the combination-lock argument cannot be applied. When the underlying game is penny matching, Fortnow and Whang [11] describe an algorithm that finds an optimal strategy efficiently, using ideas from Rivest and Schapire's [22] learning algorithm (but without actually learning the automaton).

hard when only given access to random examples. Learning algorithms for several special classes of automata have been studied in this setting: Li and Vazirani [20] give several examples of regular languages that can be learned efficiently, including 1-letter languages. In [10] a learning algorithm is given for languages accepted by width-2 branching programs that are read-once and leveled (a special case of DFAs). Schapire and Warmuth [27] have shown (see also [10]) that the problem of learning width-3 (read-once and leveled) branching programs is as hard as learning DNF, and they also observe that learning width-5 (read-once and leveled) branching programs is hard under certain number theoretical assumptions. In [14] 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 a type of mistake bound model.

In addition to the work of Dean et al. [9] which was previously mentioned, the following works consider the case when the labels of the examples are assumed to be noisy. In [25], an algorithm is given for PAC-learning DFAs with membership queries in the presence of persistent noise. In [12], an algorithm is given for learning DFAs by blurry concepts.

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 $M(u)$. Unless stated otherwise, all strings referred to are over the alphabet $\{0,1\}$.

The *cover time* of M, denoted by $C(M)$ is defined as follows. For every state $q \in Q$, with probability at least $1/2$, a random walk of length $C(M)$ on the underlying graph of M, starting at q, passes through *every* state in M.

For two strings s_1 and s_2, let $s_1 \cdot s_2$ denote the concatenation of s_1 with s_2. For a string s and an integer m let s^m denote m concatenations of s. 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 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. An exact learning algorithm is *efficient* if it runs 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 assume, without loss of generality, that M is irreducible. Namely, every pair of states q and q' in Q are distinguished by some string s, so that the output of the state reaches when executing s starting from q differs from the output of the state reached when performing the same walk starting from q'.

We also consider the easier setting in which the learning algorithm has a means of resetting the machine and performing a new walk starting from the start state. We require that for any given $\delta > 0$, after performing a polynomial (in n and $\log(1/\delta)$) number of walks, each of polynomial length, it output a hypothesis $\widehat{M}$, which is equivalent to M, i.e., for every string s, $\widehat{M}(s) = M(s)$.

2.2.2 The noisy model

Our assumptions on the noise follow the *classification* noise model introduced by Angluin and Laird [5]. 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 [3] for learning deterministic finite automata. The algorithm works in the setting where the learner has means of a reset. The analysis is similar to that in [2] and shows that if the target automaton M has cover time $C(M)$ then with high probability, the algorithm exactly learns the target automaton by performing $O(nC(M))$ walks, each of length $O(C(M))$. 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.

In this algorithm and in those described in the following sections we assume for simplicity that the algorithms actually know $C(M)$ and n. If only upper bounds $C_b(M) \geq C(M)$ and $n_b \geq n$ are known, then the algorithms can simply use these bounds instead of the exact values. The running times of the resulting algorithms are bounded by the running times of the original algorithms (which know $C(M)$ and n) where each occurrence of $C(M)$ should be replaced by $C_b(M)$ and each occurrence of n by n_b.

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

Algorithm Exact-Learn-with-Reset(δ)

1. let r be random string of length $m = C(M)\log(1/\delta)$;
2. let R_1 be the set of all prefixes of r; $R_2 \leftarrow R_1 \circ \{0,1\}$;
3. initialize the table 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 by performing corresponding walks on M;
   - /* 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**

r_i, and a column labeled by the string s_j, is $M(r_i \cdot s_j)$. We also refer to $M(r_i \cdot s_j)$ 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 are 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 if the set R has a certain property then 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 $row_T(r_i)$ denote the row in T labeled by r_i. Namely, if $S = \{s_1, \ldots, s_t\}$, then $row_T(r_i) = (T(r_i, s_1), \ldots, T(r_i, S_t))$. 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 an observation 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 $\sigma \in \{0,1\}$, $r_j \cdot \sigma \in R$.

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 (see [3]) that M^T is consistent with T in the sense that for every $r_i \in R$, and for every $s_j \in S$, $M^T(r_i \cdot s_j) = T(r_i, s_j)$.

The idea of the algorithm is as follows — first we use a random walk to construct a set R_1 of strings that with high probability reach every state in M. Namely, R_1 is such that for every state q in M, there exists a string s in R_1 such that if we take a walk on M corresponding to s and starting from q_0, then we end up at state q. Given R_1 we extend it to a set R of strings that traverse every edge in M. We then show how to use R 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 $C(M)\log(1/\delta)$ (where δ is the confidence parameter given to the algorithm). 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 the walks corresponding to the strings in R. Let us first observe that from the definition of $C(M)$, the probability that a random walk of length $C(M)\log(1/\delta)$ does not pass through some state in Q, is at most $(1/2)^{\log(1/\delta)} = \delta$. Therefore, 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 $T(r_i \cdot \sigma, s_k) \neq T(r_j \cdot \sigma, s_k)$, 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 $n-1$ 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 $row_T(r_i) \neq row_T(r_j)$, r_i and r_j reach two different states in M. Hence, after adding $O(nC(M)\log(1/\delta))$ entries to the table, each corresponding to a string of length $O(C(M)\log(1/\delta)+n)$, the algorithm has constructed a consistent table. We further make the following claim:

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 $M^T \equiv M$.*

Proof: In order to prove that $M^T \equiv M$, we show that there exists a mapping $\phi : Q \to Q^T$, which has 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. $\forall q \in Q$, $\gamma(q) = \gamma^T(\phi(q))$

Since we have assumed (without loss of generality) that M is irreducible, ϕ is an (output preserving) isomorphism between M and M^T. Clearly, the existence of such a function suffices to prove equivalence between M^T and M since by the above properties, for every $s \in \{0,1\}^*$,

$$\begin{aligned} \gamma(\tau(q_0, s)) &= \gamma^T(\phi(\tau(q_0, s)) \\ &= \gamma^T(\tau^T(\phi(q_0), s)) \\ &= \gamma^T(\tau^T(q_0^T, s)) \,. \end{aligned} \tag{1}$$

Let ϕ 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 three properties defined above.

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 $\gamma^T(T(r_i)) \stackrel{\text{def}}{=} T(r_i, \lambda) = M(r_i) = \gamma(\tau(q_0, r_i))$. 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, $\phi(\tau(q, \sigma)) = T(r_i \cdot \sigma) = \tau^T(T(r_i), \sigma)$. ■

We thus have the following theorem.

Theorem 1 *For every target automaton M, with probability at least $1 - \delta$, Algorithm* Exact-Learn-with-Reset *outputs a hypothesis DFA which is equivalent to M. Furthermore, the running time of the algorithm is*

$$O\left(n\left(C(M)\right)^2 \log^2\left(1/\delta\right)\right) .$$

4 Exact Learning without Reset

In this section we describe an efficient 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 [22]. 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. We name the algorithm Exact-Learn, and its pseudo-code appears in Figure 3.

The main problem encountered when the learner does not have means of a reset is that it cannot simply orient itself whenever needed by returning to the starting state. We thus need an alternative way by which the learner can orient itself. As in [22], we overcome the absence of a reset by the use of a *homing sequence*. A homing sequence is a sequence such that whenever it is executed, the corresponding output sequence observed uniquely determines the final state reached. More formally:

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.* [19]) 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.

4.1 Learning When a Homing Sequence is Known

Assume we had a homing sequence h of length at most n^2 for our target DFA M (we remove this assumption shortly). Then we could run the algorithm Exact-Learn-Given-Homing-Sequence whose pseudo-code appear is Figure 2. This algorithm creates 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

which may be observed when h is executed. At each stage, the algorithm walks according to h, observes the output sequence π, and then performs the next walk ELR_π would have performed (starting from q_0), from the current state reached. Since h is a homing sequence, for any given output sequence π, whenever h is executed and π is observed, we have reached the same state. We refer to this state as the *effective* starting state of ELR_π. Thus, each copy ELR_π constructs its own observation table, T_π, where the entries are filled by performing walks which all start from the effective starting state of ELR_π. The algorithm terminates when one of these copies completes, The completed copy's hypothesis automaton can then be used to predict correctly the outcome of any walk. If, as described in the pseudo-code of Exact-Learn-Given-Homing-Sequence (see Figure 2), we run each copy of Exact-Learn-with-Reset with the confidence parameter δ/n, then by Theorem 1 and the fact that there are at most n copies of Exact-Learn-with-Reset, with probability at least $1-\delta$ the hypothesis of the completed copy is correct. The running time of the algorithm Exact-Learn-Given-Homing-Sequence is bounded by the running time of each copy, multiplied by the number of copies executed and the length of the homing sequence, and is thus $O\left(n^4 \left(C(M)\right)^2 \log^2\left(n/\delta\right)\right)$.

Algorithm Exact-Learn-Given-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 step of ELR_π to fill in any entry in T_π by performing the corresponding walk starting at the current state;
  4. if the observation table T_π of ELR_π is consistent and closed then ELR_π has completed;
  5. if T_π is consistent but not closed, then discard ELR_π;

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

4.2 Learning When a Homing Sequence is Unknown

If a homing sequence is unknown, consider the case in which we guess a sequence h which is *not* a homing sequence and run the algorithm Exact-Learn-Given-Homing-Sequence with h instead of a true homing sequence. Since h is not a homing sequence, there exist (at least) 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*. Let this output sequence be π. Hence, ELR_π has more than one effective starting state and when we simulate ELR_π, some of the walks performed to fill in entries in T_π might be performed starting from q_1, and some might be performed starting from q_2.

The first of two possible consequences of such an event is that the observation table T_π becomes consistent and closed, but the hypothesis M^{T_π} is *incorrect*. Namely, there exists some walk starting from the current state whose outcome is predicted incorrectly by M^{T_π}. The second possible consequence is that T_π just grows without becoming consistent, and the number of equivalence classes in the partition induced by T_π become larger than n. In what follows we describe how to modify Exact-Learn-Given-Homing-Sequence when we do not have a homing sequence so as to detect that a copy has more than one effective starting state and thus avoid the above two consequences.

Furthermore, the procedure for detection helps us "improve" h by extending it so that after at most $n-1$ such extensions it becomes a homing sequence, where initially $h = \lambda$.

Let Q_π be the set of effective starting states of T_π. Namely

$$Q_\pi \stackrel{\text{def}}{=} \{q \ : \ q \in Q, \ \exists q' \text{ s.t. } \tau(q', h) = q \text{ and } q'\langle h \rangle = \pi\} \ .$$

If for each $q \in Q_\pi$ it holds that for every state q' in Q there exists a row in T_π labeled by a string that reaches q' when starting from q, then the following is true. By the time we add at most $n-1$ columns to T_π, for each pair of states q_1 and q_2 in Q_π, there must exist at least one entry in T_π which distinguishes between the two states. This is true since otherwise, following Lemma 3.1, q_1 and q_2 would be equivalent, in contradiction to our assumption that M is irreducible. If we discover one such entry, then we have evidence that ELR_π has more than one effective starting state and therefore h is not a homing sequence. Moreover, we can concatenate the string corresponding to this entry to h, and restart the algorithm with the extended h.[6] After at most $n-1$ such extensions, h must become a homing sequence.

Algorithm Exact-Learn(δ)

1. $N \leftarrow 2C(M)\ln(4n/\delta)$;
2. $h \leftarrow \lambda$;
3. while no copy of **Exact-Learn-with-Reset-R** is completed do:
   (a) choose uniformly a length $\ell \in [0, \ldots, C(M)]$, and 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**(N,$\delta/(2n^2)$), then create such a new copy;
   (d) simulate the next step of $ELRR_\pi$ to fill in any entry in T_π by performing the corresponding walk w starting at the current state;
   (e) if it is the first execution of w, then fill in the corresponding entry in T_π with the (final) output observed;
   (f) else if the output of the state reached is different from the output of the previous state reached when performing w then do:
      i. $h \leftarrow h \cdot w$;
      ii. discard all existing copies of **Exact-Learn-with-Reset-R**, and go to 3; /* restart algorithm with extended h */
   (g) if the observation table T_π of $ELRR_\pi$ is consistent and closed then $ELRR_\pi$ has completed;
   (h) if T_π is consistent but not closed, then discard $ELRR_\pi$;

Figure 3: Algorithm **Exact-Learn**. Algorithm **Exact-Learn-with-Reset-R** is a variant of **Exact-Learn-with-Reset** in which given an integer N, each walk to fill in an entry in the table is repeated N times and only if a single output is observed, then this output is entered.

[6] As in [22], 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 *preset* homing sequence.

4.2.1 Detecting Distinguishing Entries

We next show how to detect entries which distinguish between two effective starting states. Let Exact-Learn-with-Reset-R be a variant of Exact-Learn-with-Reset, in which each walk to fill in an entry in the table is *repeated* N consecutive times for a given N. If all N walks give the same output then the entry is filled with that output. Otherwise, we have found a distinguishing entry. Thus, in the algorithm Exact-Learn, instead of simulating copies ELR_π of Exact-Learn-with-Reset, as in Exact-Learn-Given-Homing-Sequence, we simulate copies $ELRR_\pi$ of Exact-Learn-with-Reset-R with a parameter N that is set subsequently and with confidence $\frac{\delta}{2n^2}$. If for some copy we find that its observation table includes a distinguishing entry, then as described previously, we extend h by the string corresponding to this entry and restart the algorithm with the new h. We continue in this way until one copy terminates. In order to ensure that we never fill in a distinguishing entry without identifying it as one, we need to ensure that for every entry (r_i, s_j) we need to fill, if (r_i, s_j) is a distinguishing entry in T_π then the following holds: For some pair of effective starting states, q_1 and q_2, which are distinguished by (r_i, s_j), at least one of the N executions of $r_i \cdot s_j$ starts from q_1 and at least one starts from q_2.

To this end we do the following. Each time before executing h, we randomly choose a length $0 \leq \ell \leq C(M)$, and perform a random walk of length ℓ. The idea behind this random walk is that for every state there is some non-negligible probability of reaching it upon performing the random walk. More precisely: For a distinguishing entry (r_i, s_j) in T_π, consider the N executions of h whose outcome was π and which were followed by performing the walk corresponding to $r_i \cdot s_j$. 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.,

$$B(q) \stackrel{\text{def}}{=} \{q' \ : \ q' \in Q, \ \tau(q', h) = q\} \ .$$

For a given $q \in Q_\pi$, the probability that we did not reach q after any one of the N executions of h in which π was observed, equals the probability that following all preceding random walks, we did not reach a state in $B(q)$. This probability is bounded 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 $C(M)$, 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. For each of the random strings t, the probability that it passes a state in $B(q)$ is at least $1/2$. Given that it passes a state in $B(q)$, the probability that the randomly chosen prefix ends on that state is at least $1/C(M)$. Together, the probability that we reach a given state in $B(q)$ is at least $1/2C(M)$. Thus, for a given state $q \in Q_\pi$, the probability that q is not reached in any of the corresponding N executions of h is bounded from above by

$$\left(1 - \frac{1}{2C(M)}\right)^N \leq \exp\left(-\frac{N}{2C(M)}\right) \ . \tag{2}$$

4.2.2 Bounding the Error and the Running Time of the Algorithm

It remains to set N so that the total error probability of Exact-Learn is at most δ, and then to bound the algorithm's running time. We have two types of events we want to avoid so as to ensure that the algorithm constructs a correct hypothesis. We shall bound the probability that each type of event occurs by $\delta/2$. The first type of event is that for some copy $ELRR_\pi$ and for one of its effective starting states q, there exists a state q' in Q such that no row in T_π is labeled by a string

which reaches q' when starting from q. In the course of the algorithm, h takes on at most n values. For each value there are at most n effective starting states for all existing copies $ELRR_\pi$ (even though a single state may be an effective starting state of more than on copy). Since we simulate each copy with error parameter $\delta/(2n^2)$, then with probability at least $1-\delta/2$, the above type of event does not occur. In such a case, it follows from 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.

The second type of bad event is that when filling an entry in some table T_π, we do not detect that it is a distinguishing entry. For each value of h consider the first entry to be filled (in some table T_π) that is a distinguishing entry. Since h takes at most n values, there are at most n such first entries[7].

For each such entry, there exists at least one pair of effective starting states which it distinguishes. Let $N = 2C(M)\ln(4n/\delta)$. Then by Equation (2), for a given distinguishing entry, the probability that we did not reach both of the states in the pair of effective starting states it distinguishes is at most $\delta/2n$. It follows that with probability at least $1-\delta/2$, for each first distinguishing entry, we perform the walk corresponding to that entry starting from each of the two effective starting states it distinguishes. Therefore, with probability $1-\delta/2$ we always detect the first distinguishing entry for every value of h, and thus do not output a hypothesis of a copy $ELRR_\pi$ which corresponds to more than one effective starting state.

The running time of the algorithm is bounded by the product of the number of phases of the algorithm (one for each value of h) which is n, and the running time of each phase. The running time of each phase is the product of:

- the number of copies of Exact-Learn-with-Reset-R in each phase (which is at most n),
- the number of entries added to each table (which is $O(nC(M)\log(2n^2/\delta))$),
- the number of times the walk corresponding to each entry is repeated (which is $N = O(C(M)\log(n/\delta))$,
- the sum of:
  - the maximum length of each walk to fill in an entry (which is $O(C(M)\log(2n^2/\delta))$,
  - the maximum length of h (which is $O(n^2C(M)\log(2n^2/\delta))$),
  - and the maximum length of the random walk performed prior to the execution of h (which is $C(M)$).

The total running time is hence $O\left(n^5\left(C(M)\right)^3\log^3\left(n/\delta\right)\right)$.

We have thus proven that:

Theorem 2 *Algorithm* Exact-Learn *is an exact learning algorithm for DFAs, and its running time is* $O\left(n^5\left(C(M)\right)^4\log^4\left(n/\delta\right)\right)$.

As mentioned previously, Rivest and Schapire [22] give an exact learning algorithm that runs in time polynomial in n and $\log(1/\delta)$ and does not depend on any other parameter related to

[7] Note that each such entry is uniquely determined by the current h, the initial random walks which label the rows of the corresponding tables, and the random walks executed prior to the executions of h.

the target automaton. However, they rely on a teacher that gives the learner counterexamples to the incorrect hypotheses output by the learner. It is interesting to note that the (tempting) idea to simply run Rivest and Schapire's algorithm but instead of making equivalence queries try and randomly guess a counterexample whenever the learner has a hypothesis, does not work even in the case of automata that have small cover time. Rivest and Zuckerman [24] construct a pair of automata which both have small cover time, but for which the probability of randomly guessing a sequence which distinguishes between the automata is exponentially small. These automata are described in Appendix A.

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 6. We start by showing how to compute a good estimate of the noise rate. We then show how to use this estimate to learn the target DFA when a homing sequence is known, and finally describe a learning algorithm which is not given a homing sequence.

5.1 Estimating the Noise Rate

According to our learning model, the algorithm is given only an upper bound $1/2 - \alpha$ on the noise rate, η. Since we need a good approximation $\hat{\eta}$ of η, we first show that η can be efficiently approximated (with high probability) within a small additive error. This is done by running Procedure Estimate-Noise-Rate whose pseudo-code appears in Figure 4, and which is analyzed in the following lemma. A very similar procedure was described in [25].

Lemma 5.1 *For any given $\delta' > 0$, and $\mu > 0$, after time polynomial in $\log(1/\delta')$, $1/\mu$, n, and $1/\alpha$, Procedure* Estimate-Noise-Rate *outputs an approximation $\hat{\eta}$ of η, such that with probability at least $1 - \delta'$, $|\hat{\eta} - \eta| < \mu$.*

Proof: Before going into the details of the procedure we describe the idea it is based on. Consider a pair of states q_1 and q_2. For a string z, and $i \in \{0, 1\}$, let the *observed* behavior of q_i on z be the output observed by the learner after executing the walk corresponding to z starting from q_i, and let the *actual* behavior of q_i on z be the (correct) output of the state reached. If $q_1 = q_2$ then for *every* string z, $\tau(q_1, z) = \tau(q_2, z)$. Thus, the observed difference in the behavior of q_1 and q_2 on any set of strings is entirely due to the noise process. If $q_1 \neq q_2$, then the difference in their observed behavior on a set of strings Z is due to the difference in their actual behavior on Z as well as the noise. Thus in order to estimate the noise rate, we look for strings that seem to reach the same state and deduce the noise rate from the difference in their observed behavior. More precisely, this is done as follows.

Let t be an arbitrary string of length L, where L is set subsequently. Suppose t is executed $n+1$ times. For $1 \leq i \leq n+1$, let $q^{(i)}$ be the state reached after performing t exactly $i-1$ times and let $o^{(i)} = o_1^{(i)} \ldots o_L^{(i)}$ be the sequence of outputs corresponding to the i^{th} 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 $d^{ij} = \frac{1}{L} \sum_{k=1}^{L} o_k^{(i)} \oplus o_k^{(j)}$. Thus, d^{ij} is the fraction of indices in which the sequences $o^{(i)}$ and $o^{(j)}$ differ, or equivalently, it is the fraction of strings among all prefixes of t on which there is an observed difference in behavior between $q^{(i)}$ and $q^{(j)}$. The key observation is that if $q^{(i)} = q^{(j)}$ then the expected value of d^{ij} is $2\eta(1-\eta)$, while if $q^{(i)} \neq q^{(j)}$ it is at least as large. More precisely, if the fraction of prefixes of t on

Procedure Estimate-Noise-Rate(δ', μ)

1. $L \leftarrow (1/\mu)^2(1/\alpha)^2 \log(n/\delta')$;
2. let t be an arbitrary string of length L;
3. perform the walk corresponding to t^{n+1};
4. let $o^{(i)} = o_1^{(i)} \ldots o_L^{(i)}$ be the sequence of outputs corresponding to the i^{th} execution of t; (i.e. the complete output sequence corresponding to to t^{n+1} is $o_1^{(1)} \ldots o_L^{(1)}, \ldots, o_1^{(n+1)} \ldots o_L^{(n+1)}$)
5. $\forall i, j,\ 1 \leq i < j \leq n+1$, let $d^{ij} \leftarrow \frac{1}{L} \sum_{k=1}^{L} o_k^{(i)} \oplus o_k^{(j)}$;
6. let $d_{min} \leftarrow \min_{i,j} d^{ij}$;
7. if $d_{min} > 1/2$ then *goto 1*;
8. let $\hat{\eta}$ be the solution to $d_{min} = 2\hat{\eta}(1-\hat{\eta})$ s.t. $\hat{\eta} < 1/2$;
9. return $\hat{\eta}$;

Figure 4: Procedure **Estimate-Noise-Rate**

which $q^{(i)}$ and $q^{(j)}$ actually differ is ϕ, then the expected observed difference in behavior between the states is

$$(1-\phi) \cdot 2\eta(1-\eta) + \phi \cdot ((1-\eta)^2 + \eta^2) \;\; = \;\; 2\eta(1-\eta) + \phi(1-2\eta)^2 \; . \tag{3}$$

We therefore define d_{min} to be the minimum value over all d^{ij}'s, and let $\hat{\eta} < 1/2$ be the solution of the quadratic equation $2\hat{\eta}(1-\hat{\eta}) = d_{min}$. Since we have less than n^2 pairs, if $L = \Omega((1/\mu)^2(1/\alpha)^2 \log(n/\delta'))$, then by Hoeffding's inequality [16], with probability at least $1-\delta'$, for every pair i, j, $|d^{ij} - E[d^{ij}]| < \alpha\mu$, and hence $|d_{min} - 2\eta(1-\eta)| < 2\alpha\mu$. It directly follows (see [25]) that $|\hat{\eta} - \eta| < \mu$. ■

We thus assume from here on that we have a good approximation, $\hat{\eta}$, of η. In particular we assume that $\hat{\eta}$ is at most $\alpha/8C(M)$ away from η.

5.2 Learning When a Homing Sequence is Known

As in the noise free case, we first assume that the algorithm has means of a reset. With this assumption, we define a slight modification of Exact-Learn-with-Reset, named Exact-Noisy-Learn-with-Reset. Given a large enough integer N this procedure simply repeats each walk to fill in an entry in the table N times, and fills the corresponding entry 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 state reached.

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 [9]. The idea is to construct a new "robust" homing sequence out of h, such that we see many samples of each bit of the output of h, and can thus infer the correct output of h by majority vote: Assume we execute h for m consecutive times where $m >> n$ and is set subsequently. In order to gain some intuition, consider first a directed graph H whose set of vertices is Q, and in which there is an edge from q_1 to q_2 if and only if q_2 is reached

from q_1 upon the execution of h. Then, m executions of h on M correspond to a walk of length m on H. Clearly, if $m > n$, then after at most n steps this walk will start following a (simple) cycle on H. If we now return to M, the m executions of h pass the same states which are on the cycle in H, and hence follow a cycle on the underlying graph of M. It should be noted that if $|h| > 1$, the cycle in M may not be simple.

We now show how to use the existence of this cycle in order to estimate the output sequence corresponding to the last (m^{th}) execution of h. Let the state from which we start the m^{th} execution of h be denoted by $q^{(m)}$. The idea is that since the m executions of h follow a cycle, then in particular (assuming m is large enough), h is executed starting from $q^{(m)}$ many times. Assume we were able to identify each occurrence of $q^{(m)}$ (or equivalently, to find the length of the simple cycle on H). Then we could use all these executions (whose outputs are noisy) to infer by majority vote (with high probability) the correct output sequence which corresponds to the execution of h starting from $q^{(m)}$, and which reaches the current state.

More formally: For $1 \leq i \leq m$, let $q^{(i)}$ be the state reached after the i^{th} execution of h, and let $o^{(i)} = o_1^{(i)} \ldots o_{|h|}^{(i)}$ be the (noisy) output sequence corresponding to this execution. For each possible length $1 \leq v \leq n$, let

$$b_v \stackrel{\text{def}}{=} \lfloor (m-n)/v \rfloor \ . \tag{4}$$

Then there exists some (minimal) *period* p, where $1 \leq p \leq n$, such that for every $1 \leq k \leq b_p$. $q^{(m)} = q^{(m-kp)}$. In other words, every p executions of h it was executed starting from $q^{(m)}$ (and p is simply the length of the simple cycle in H). Thus, if we know p, then we can compute with high probability the correct output sequence corresponding to the last execution of h (which started from $q^{(m)}$) by considering all previous executions which started from $q^{(m)}$: For every $1 \leq j \leq |h|$ we let $\pi_j = 1$ if $1/b_p \sum_{k=1}^{b_p} o_j^{(m-kv)} > 1/2$, 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-Given-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? Let $q_j^{(i)}$ be the state reached after i executions of h, followed by the length j prefix of h. By definition of p, for every k, k', and for every j, $q_j^{(m-kp)} = q_j^{(m-k'p)}$ (the states reached at step numbers that differ by multiples of p are the same). Let $\vec{\psi}^{(v)}$ be an $|h|$ dimensional vector which is defined as follows. For $1 \leq j \leq |h|$,

$$\psi_j^{(v)} = 1/b_v \sum_{k=1}^{b_v} o_j^{(m-kv)} \ , \tag{5}$$

where b_v was defined in Equation (4). When $v = p$, then for each $1 \leq j \leq |h|$, it holds that for every k, the outputs $o_j^{(m-kp)}$ are generated from the same state. Since the noise is added independently, we have that for an appropriate choice of N, with high probability, for every j, $\psi_j^{(p)}$ is either within ϵ of $1-\eta$, or within ϵ of η, for some small additive error ϵ. In particular we shall choose m to ensure that $\epsilon \leq \alpha/2n$. Under our assumption that $|\hat{\eta} - \eta| < \alpha/8C(M) < \alpha/2n$, we have that either

$$\hat{\eta} - \alpha/n \ < \ \psi_j^{(p)} \ < \ \hat{\eta} + \alpha/n \tag{6}$$

or

$$(1-\hat{\eta}) - \alpha/n \ < \ \psi_j^{(p)} \ < \ (1-\hat{\eta}) - \alpha/n \tag{7}$$

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

$\pi_j^{(v)}$ to be 1 if $\psi_j^{(v)}$ is greater than 1/2, and 0 if it is at most 1/2. Then, with high probability, $\pi^{(v)} = \pi_1^{(v)} \ldots \pi_{|h|}^{(v)}$ is the correct output sequence corresponding to the last execution of h. In such a case, v effectively behaves as a period. 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/b_v and K_1/b_v are at least $1/p$ which is at least $1/n$. This is true since $v \cdot p$ must be a period as well, and hence for every k and k' which are multiples of p, $q_j^{(m-kv)} = q_j^{(m-k'v)}$. Let $\beta = K_1/b_v$. Then

$$E[\psi_j^{(v)}] = \beta(1-\eta) + (1-\beta)\eta \ . \tag{8}$$

$E[\psi_j^{(v)}]$ can be written in two equivalent forms:

$$\begin{aligned} E[\psi_j^{(v)}] &= \beta(1-\eta) + \eta - \beta \cdot \eta \\ &= \eta + (1-2\eta)\beta \end{aligned} \tag{9}$$

and

$$\begin{aligned} E[\psi_j^{(v)}] &= (1-(1-\beta))(1-\eta) + (1-\beta)\eta \\ &= (1-\eta) - (1-2\eta)(1-\beta) \end{aligned} \tag{10}$$

Since $\beta \geq 1/n$, then by Equation (9),

$$E[\psi_j^{(v)}] \geq \eta + (1-2\eta)\frac{1}{n} \ . \tag{11}$$

On the other hand, since $\beta \leq 1 - 1/n$ (which implies that $1 - \beta \geq 1/n$), then by Equation (10),

$$E[\psi_j^{(v)}] \leq (1-\eta) - (1-2\eta)\frac{1}{n} \ . \tag{12}$$

Since $1 - 2\eta \geq 2\alpha$, and since we are assuming that $|\hat{\eta} - \eta| < \alpha/8C(M) < \alpha/2n$

$$\hat{\eta} + \frac{3\alpha}{2n} \ < \ E[\psi_j^{(v)}] \ < \ (1-\hat{\eta}) - \frac{3\alpha}{2n} \ . \tag{13}$$

Thus, if $\psi_j^{(v)}$ is at most $\alpha/2n$ away from its expected value for every j, then

$$\hat{\eta} + \alpha/n \ < \ \psi_j^{(v)} \ < \ (1-\hat{\eta}) - \alpha/n \ . \tag{14}$$

Since we have shown (see Equations (6) and (7)) that with high probability, $\psi_j^{(p)}$ is within α/n either from $\hat{\eta}$ or from $1 - \hat{\eta}$, we are able to detect whether or not v is the minimal period p (or at least effectively behaves as such). Consequently we can compute the correct output sequence corresponding to the homing sequence h. The pseudo-code for the procedure described above appears in Figure 5. Note that we did not actually use the fact that h is a homing sequence and hence this procedure can be used to compute the correct output sequence corresponding to any given sequence.

5.3 Learning When a Homing Sequence is Unknown

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 $q \in Q$ which can be reached from some state following an execution of h, and where

Procedure Execute-Homing-Sequence(h)

1. $m \leftarrow 100\,(n/\alpha)^2 \log(nC(M)/\delta)$;
2. choose uniformly a length $\ell \in [0, \ldots, C(M)]$, and then perform a random walk of length ℓ.
3. perform the walk corresponding to h^m, and for $1 \leq i \leq m$, let $o^{(i)}$ be the output sequence corresponding to the i^{th} execution of h;
4. 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 $m_v = \lfloor m/v \rfloor$;
5. let v be such that for every j either $|\psi_j^{(v)} - \hat{\eta}| < \alpha/n$, or $|\psi_j^{(v)} - (1-\hat{\eta})| < \alpha/n$; if there is no such v, then return to (1);
6. for $1 \leq j \leq |h|$, let $\pi_j = 1$ if $\psi_j^{(v)} > 1/2$, and 0 otherwise;
7. return π;

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

the corresponding output is π. That is, there exists a state $q' \in Q$ such that $\tau(q', h) = q$ and $q'\langle h \rangle = \pi$. 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 corresponding to π for which there exist $q_1, q_2 \in Q_\pi$, such that $\gamma(\tau(q_1, r_i \cdot s_j)) \neq \gamma(\tau(q_2, r_i \cdot s_j))$. As we have argued in the noise free case, if there is no such entry for any of the possible output sequences π, then h is a homing sequence. 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 analogously. As in the noise free case, the walk corresponding to a given entry is repeated N times, and a random walk of a length ℓ chosen uniformly in the range $[0, \ldots, C(M)]$ is performed prior to the m executions of h. Let β_1 be the fraction of times that a state $q \in Q_\pi^1$ is reached and let β_0 $(= 1 - \beta_1)$ be defined analogously. By the same argument used in the noise free case (in the discussion preceding Equation (2)), $E[\beta_1] \geq 1/(2C(M))$, and $E[\beta_0] \geq 1/(2C(M))$. By applying a Chernoff bound we have that for each $i \in \{0, 1\}$,

$$Pr\left[\beta_i < \frac{1}{4C(M)}\right] \leq \exp\left(-\frac{N}{32C(M)}\right)\ . \tag{15}$$

Let $w = r_i \cdot s_j$, and let $f(w)$ be as defined in Exact-Noisy-Learn. That is, $f(w)$ is the fraction of 1's observed, among all N repetitions of the walk executed to fill in the (distinguishing) entry (r_i, s_j). Then, similarly to the calculations performed in Equations (8) through (14),

$$\begin{aligned} E[f(w)] &\geq \eta + (1-2\eta)\frac{1}{4C(M)} && (16)\\ &> \hat{\eta} + \frac{3\alpha}{8C(M)} && (17) \end{aligned}$$

and

$$\begin{aligned} E[f(w)] &\leq (1-\eta) - (1-2\eta)\frac{1}{4C(M)} && (18)\\ &< (1-\hat{\eta}) - \frac{3\alpha}{8C(M)} && (19) \end{aligned}$$

On the other hand, if (r_i, s_j) is not a distinguishing entry then $E[f(w)]$ equals either η or $1-\eta$, and is hence within $\alpha/8C(M)$ either from $\hat{\eta}$ or from $1-\hat{\eta}$. If we choose N so as to ensure (with high probability) that $|f(w) - E[f(w)]| < \alpha/8C(M)$, then we can determine when an entry is a distinguishing entry and extend h by the string corresponding to this entry.

Algorithm Exact-Noisy-Learn(δ)

1. $N \leftarrow 100\left(\left(\frac{C(M)}{\alpha}\right)^2 \log^2(nC(M)/\delta)\right)$;
2. $\hat{\eta} \leftarrow$ **Estimate-Noise-Rate**$(\delta/5, \alpha/8C(M))$;
3. $h \leftarrow \lambda$;
4. while no copy of **Exact-Noisy-Learn-with-Reset** is completed do:
   (a) $\pi \leftarrow$ **Execute-Homing-Sequence**(h);
   (b) if a copy $ENLR_\pi$ of **Exact-Noisy-Learn-with-Reset**$(N, \delta/(5n^2))$ does not exist, then create such a new copy;
   (c) simulate the next step of $ENLR_\pi$ by performing the corresponding walk w; let $\theta_i(w)$ be the output of the state reached, where i is the number of times w has been executed.
   (d) if $i = N$ then let $f(w) = (1/N)\sum_{i=1}^{N} \theta_i(w)$. If
   $$\hat{\eta} + \alpha/4C(M) \;<\; f(w) \;<\; (1-\hat{\eta}) - \alpha/4C(M)$$
   then do:
      i. $h \leftarrow h \cdot w$;
      ii. discard all existing copies of **Exact-Noisy-Learn-with-Reset**, and go to 4; /* restart algorithm with extended h */
   (otherwise, the value of the entry is set to be the majority observed label by $ENLR_\pi$);
   (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 6: Algorithm **Exact-Noisy-Learn**. Algorithm Exact-Noisy-Learn-with-Reset is a variant of Exact-Learn-with-Reset in which given an integer N, each walk to fill in an entry in the table is repeated N times and the majority valued is entered.

5.3.1 Bounding the Error and Running Time of the Algorithm

We start by bounding the error of the algorithm. We have the following 5 types of events we need to prevent from occurring, and we shall bound the probability that each type occurs by $\delta/5$. Whenever bounding the probability that a bad event occurs, we assume that no other bad event has occurred previously.

1. Our estimation $\hat{\eta}$ of η, is not good enough. If we call the procedure Estimate-Noise-Rate with the confidence parameter $\delta' = \delta/5$ and with the estimation parameter $\mu = \alpha/8C(M)$, we know by Lemma 4, that with probability at least $1 - \delta/5$, $|\hat{\eta} - \eta| < \alpha/8C(M)$.

2. For some copy $ENLR_\pi$ and for one of its effective starting states q, there exists a state q' in Q such that no row in T_π is labeled by a string which reaches q' when starting from q. As in the noise-free case, in the course of the algorithm, h takes on at most n values. For each value there are at most n effective starting states for all existing copies $ENLR_\pi$. Since we simulate each copy with the parameter $\delta/(5n^2)$, then with probability at least $1-\delta/5$, the above type of event does not occur.

3. For some candidate homing sequence h, the first distinguishing entry to be filled is not detected. In order to ensure that this event does not occur with probability larger than $\delta/5$, we do the following. We first ensure that with probability at least $1-\delta/10$, for each such entry, and for some pair of effective starting states which are distinguished by this entry, the fraction of times we execute the corresponding walk starting from each of these states is at least $1/4C(M)$. We then ensure that with probability at least $1-\delta/10$ the fraction of 1's observed does not differ from its expectation by more than $\alpha/8C(M)$. As we have argued at the opening of this subsection, in such a case, distinguishing entries are always detected.

   We start with the former requirement. By Equation (15), if $N = \Omega\left(C(M)\log(n/\delta)\right)$, then the probability that a given distinguishing entry is not detected is at most $\delta/10n$. The probability that this event occurs for any h is at most $\delta/10$. As for the second requirement, by Hoeffding's inequality, it suffices that

$$N = \Omega\left(\left(\frac{C(M)}{\alpha}\right)^2 \log(n/\delta)\right)$$

4. For some table and some non-distinguishing entry in the table, the majority observed output is incorrect, or the entry is thought to be distinguishing. To avoid the latter type of error (which also means that we avoid the former) we need to ensure that for all entries (in all tables) the fraction of 1's observed when filling each entry does not differ by more than $\alpha/8C(M)$ from its expected value (which is either η or $(1-\eta)$). We construct at most n^2 tables, each of size $O(nC(M)\log(5n^2/\delta))$. Thus we simply need to set N to be larger than our previous bound by a factor of $\Omega(\log(nC(M)/\delta))$ in order to ensure that this type of event does not occur with probability greater than $\delta/5$. We thus require that

$$N = \Omega\left(\left(\frac{C(M)}{\alpha}\right)^2 \log^2(nC(M)/\delta)\right)$$

5. For some execution of a candidate sequence h (where execution here will actually denote the m consecutive executions of h), the output computed for h is incorrect. The maximum length of h is $O(n^2C(M)\log(5n^2/\delta))$, and the number of values taken by v when computing $\psi_j^{(v)}$ is n. h takes on at most n values, and for each value, h is executed at most $n|T|N$ times where $|T|$ denotes the maximum size of each table and is $O(nC(M)\log(5n^2/\delta))$. By Hoeffding's inequality, if

$$m = \Omega\left(\left(\frac{n}{\alpha}\right)^2 \log\frac{nC(M)N\log(n/\delta)}{\delta}\right) , \tag{20}$$

   then with probability at least $1-\delta/5$, every $\psi_j^{(v)}$ is at most $\alpha/2n$ away from its expected value. From the discussion following Equation (13) this suffices for the correct computation of the output sequence corresponding to h.

The running time of the algorithm is bounded by the sum of:

1. the running time of Procedure Estimate-Noise-Rate
2. the number of phases of the algorithm (one for each value of h) which is n, multiplied by the running time of each phase.

The running time of Procedure Estimate-Noise-Rate is $O(Ln^2) = O\left((C(M))^2 n^2 \alpha^{-4} \log(n/\delta)\right)$ (where L is defined in Figure 4). The running time of each phase is the product of:

- the number of copies of Exact-Noisy-Learn-with-Reset in each phase (which is at most n),
- the number of entries added to each table (which is $O\left(nC(M)\log(5n^2/\delta)\right)$),
- the number of times the walk corresponding to each entry is repeated (which is $N = O\left(\left(\frac{C(M)}{\alpha}\right)^2 \log^2(nC(M)/\delta)\right)$),
- the sum of:
  - the maximum length of each walk to fill in an entry (which is $O\left(C(M)\log(5n^2/\delta)\right)$),
  - the maximum length of h (which is $O\left(n^2 C(M)\log(5n^2/\delta)\right)$) times m,
  - and the maximum length of the random walk performed prior to the execution of h^m (which is $C(M)$).

We have thus proven the following theorem:

Theorem 3 *Algorithm* Exact-Noisy-Learn *is an exact learning algorithm in the presence of noise for DFAs, and its running time is:* $O\left(n^2(C(M))^2\alpha^{-4}\log(n/\delta) + n^7\left(C(M)\right)^4\alpha^{-2}\log^5(C(M)/\alpha\delta)\right)$.

Acknowledgments We wish to thank Michael Kearns and Rob Schapire for conversations that stimulated this line of research. Dana Ron would like to thank the Eshkol Fellowship for its support. Part of this work was done while the authors were visiting AT&T Bell Laboratories.

References

[1] Romas Aleliunas, Richard M. Karp, Richard J. Lipton, Laszlo Lovász, and Charles Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *Proceedings of the Twentieth Annual Symposium on Foundations of Computer Science*, pages 218–223, October 1979.

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

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

[4] D. Angluin. Negative results for equivalence queries. *Machine Learning*, 5(2):121–150, 1990.

[5] D. Angluin and P. Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

[6] D. Angluin and C. H. Smith. Inductive inference: Theory and methods. *Computing Surveys*, 15(3):237–269, September 1983.

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

[8] F. Bergando and S. Varricchio. Learning behaviors of automata from multiplicity and equivalnece queries. In *Algorithms and complexity, Proceedings of the 2nd Italian conference*, pages 54–62, 1994. To appear in Siam J. of Computing.

[9] T. Dean, D. Angluin, K. Basye, S. Engelson, L. Kaelbling, E. Kokkevis, and O. Maron. Inferring finite automata with stochastic output functions and an application to map learning. *Machine Learning*, 18(1):81–108, January 1995.

[10] F. Ergün, S. Ravikumar, and R. Rubinfeld. On learning bounded-width branching programs. In *Proceedings of the Eighth Annual ACM Conference on Computational Learning Theory*, pages 361–368, 1995.

[11] L. Fortnow and D. Whang. Optimality and domination in repeated games with bounded players. In *The 25th Annual ACM Symposium on Theory of Computing*, pages 741–749, 1994.

[12] M. Frazier, S. Goldman, N. Mishra, and L. Pitt. Learning from a consistently ignorant teacher. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, pages 328–339, 1994. To appear in Journal of Computer Systems Science.

[13] Y. Freund, M. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, and R. Schapire. Efficient algorithms for learning to play repeated games against computationally bounded adversaries. In *Proceedings of the Thirty Seventh Annual Symposium on Foundations of Computer Science*, pages 332–341, 1996.

[14] Y. Freund, M. Kearns, D. Ron, R. Rubinfeld, R. Schapire, and L. Sellie. Efficient learning of typical finite automata from random walks. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, 1993.

[15] I. Gilboa and D. Samet. Bounded versus unbounded rationality: The tyranny of the weak. *Games and Economic Behavior*, 1(3):213–221, 1989.

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

[17] O. Ibarra and T. Jiang. Learning regular languaages from counterexamples. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 371–385, 1988.

[18] M. Kearns and L. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the Association for Computing Machinery*, 41:67–95, 1994. An extended abstract of this paper appeared in STOC89.

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

[20] M. Li and U. Vazirani. On the learnability of finite automata. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 359–370, 1988.

[21] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, first edition, 1995.

[22] R. Rivest and R. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, 1993.

[23] R. Rivest and R. Schapire. Diversity-based inference of finite automata. *Journal of the Association for Computing Machinery*, 43(3):555–589, 1994.

[24] R. Rivest and D. Zuckermann. Private communication. 1992.

[25] D. Ron and R. Rubinfeld. Learning fallible finite state automata. *Machine Learning*, 18:149–185, 1995.

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

[27] R. Schapire and M. Warmuth. Presented at COLT90 rump session, 1990.

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

A Rivest and Zuckerman's example

We describe below a pair of automata, constructed by Rivest and Zuckerman [24], which have the following properties. Both automata have small cover time (order of $n \log n$), but the probability that a random string distinguishes between the two is exponentially small. The automata are depicted in Figure 7.

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 $0 \leq i < k$,

$$\tau\left(q[i,j],0\right) = q[i+1, max(1,i-1)]\ ,$$

and

$$\tau\left(q[i,j],1\right) = q[i+1, min(3,i+1)]\ .$$

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.

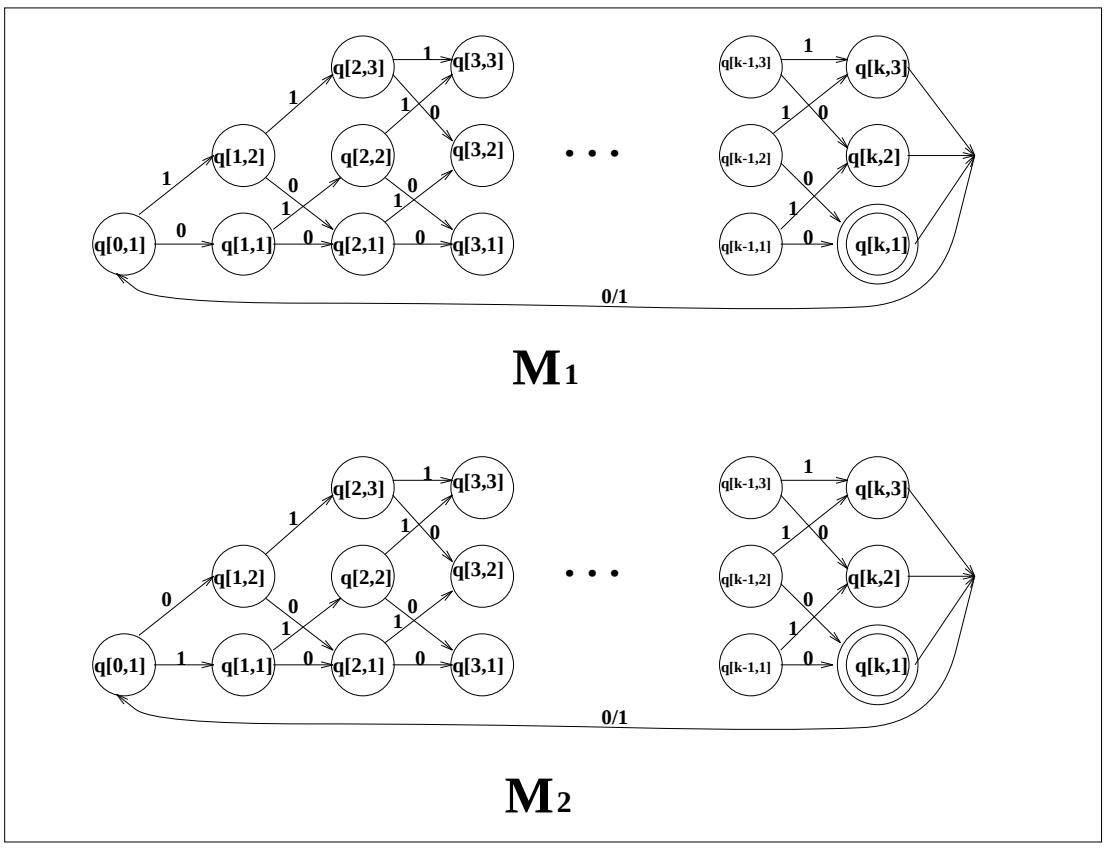

Figure 7: Automata M_1 and M_2 described in the Appendix