Inference of Finite Automata Using Homing Sequences
(Extended Abstract)

Ronald L. Rivest Robert E. Schapire

MIT Laboratory for Computer Science
Cambridge, MA 02139

Abstract

We present new algorithms for inferring an unknown finite-state automaton from its input/output behavior *in the absence of a means of resetting the machine to a start state.* A key technique used is inference of a *homing sequence* for the unknown automaton.

Our inference procedures experiment with the unknown machine, and from time to time require a teacher to supply counterexamples to incorrect conjectures about the structure of the unknown automaton. In this setting, we describe a learning algorithm which, with probability $1-\delta$, outputs a correct description of the unknown machine in time polynomial in the automaton's size, the length of the longest counterexample, and $\log(1/\delta)$. We present an analogous algorithm which makes use of a diversity-based representation of the finite-state system. *Our algorithms are the first which are provably effective for these problems, in the absence of a "reset."*

We also present probabilistic algorithms for permutation automata which do not require a teacher to supply counterexamples. For inferring a permutation automaton of diversity D, we improve the best previous time bound by roughly a factor of $D^3/\log D$.

1 Introduction

We address the problem of inferring a finite automaton from its input/output behavior. This well-studied problem continues to generate new interest. In the past, a number of learning protocols have been considered. It is now known that inferring finite automata in many of these situations is computationally hard, while feasible in a few others.

Angluin [3] and Gold [4] show that it is NP-complete to find the smallest automaton consistent with a given sample of input/output pairs. Pitt and Warmuth [9] show that merely finding an approximate solution is intractable (assuming $P \neq NP$).

These NP-hardness results depend on a restriction that the learner use a particular representation of the automaton. In fact, learning is even hard when no such restriction is made. Kearns and Valiant [6, 7] consider the "representation-free" problem of predicting the output of the automaton on a randomly chosen input, based on a sample of the machine's behavior. Extending the work of Pitt and Warmuth [10], they show that this problem is as hard as factoring Blum integers, inverting RSA, or deciding quadratic residuosity.

Thus, learning by passively observing the behavior of the unknown machine is apparently infeasible. But what about learning by actively *experimenting* with it?

Angluin [2] shows that this problem is also hard. She describes a family of automata which cannot be identified in less than exponential time when the learner can only observe the behavior of the machine on inputs of the learner's own choosing. The difficulty here is in accessing certain hard to reach states.

In spite of these negative results, Angluin [1], elaborating on Gold's results [5], shows that a *combination* of active and passive learning is feasible. Her inference procedure is able to experiment with the unknown automaton, and is given, in response to each incorrect conjecture of the automaton's identity, a counterexample, a string accepted by either the unknown automaton or the conjectured automaton, but not by the other. Her algorithm runs in time polynomial in the automaton's size and the length of the longest counterexample.

A serious limitation of Angluin's procedure is its critical dependence on a means of *resetting* the automaton to a fixed start state. Thus, the learner can never really "get lost" or lose track of its current state since it can always reset the machine to its start state. In this paper, we extend Angluin's algorithm, demonstrating that an unknown automaton can be inferred even when the learner is not provided with a reset.

This problem of inferring an automaton from its input/output behavior in the absence of a reset is relevant to the problem of identifying an environment by experimentation. We imagine a robot placed in an un-

This paper prepared with support from NSF grant DCR-8607494, ARO Grant DAAL03-86-K-0171, and a grant from the Siemens Corporation. Part of this research was done while R. Schapire was visiting GTE Laboratories in Waltham, Massachusetts. Authors' net addresses: rivest@theory.lcs.mit.edu, rs@theory.lcs.mit.edu.

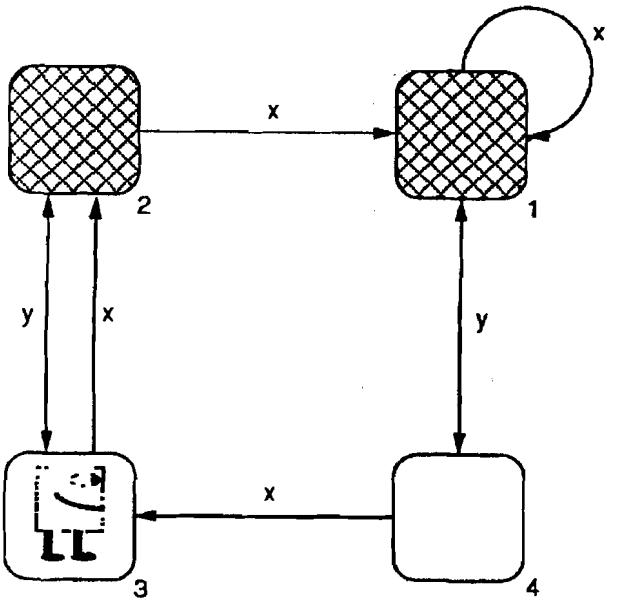

Figure 1: An Example Robot Environment

familiar environment who must learn the structure of its world to function effectively in it. For example, the robot might find itself on a directed graph, such as the one in Figure 1. In this environment, the robot can sense his local environment (e.g., can see whether the node it's on is shaded or not), and can select one of the out-going labeled edges to traverse. It is natural to assume that the robot does *not* have a means of "resetting" the environment to some start state, or of "backing up" to a previous state. As in real life, the robot must gather data in one continuous experiment.

The generality of our results allows us to handle any directed graph environment. This means that we can handle many special cases as well, such as undirected graphs, planar graphs, and environments with special spatial relations. However, our procedures do not take advantage of such special properties of these environments, some of which could probably be handled more effectively. For example, we have found that permutation automata are generally easier to handle than non-permutation automata.

In our previous papers [11, 12], we introduced the "diversity-based" representation of finite automata and described an algorithm which we proved to be effective for permutation automata. We also described some general techniques for handling non-permutation automata which, although not provably effective, seemed to work well in practice for a variety of simple environments.

In this paper, we generalize these results, demonstrating probabilistic inference procedures which are provably effective for both permutation and non-permutation automata. More generally, we present new inference procedures for the usual global state representation, as well as for the diversity-based representation.

Like Angluin, we assume that the inference procedures have an unspecified source of counterexamples to incorrectly conjectured models of the automaton. This differs from our previous work where the learning model incorporated no such source of counterexamples; as already mentioned, this limitation makes learning of finite automata infeasible in the general case. For a robot trying to infer the structure of its environment, a counterexample is discovered whenever the robot's current model makes an incorrect prediction. For the special class of permutation automata, we show that an artificial source of counterexamples is unnecessary.

Our algorithms use powerful new techniques based on the inference of *homing sequences*. Informally, a homing sequence is a sequence of inputs that, when fed to the machine, is guaranteed to "orient" the learner: the outputs produced for the homing sequence completely determine the state reached by the automaton at the end of the homing sequence. Every finite state machine has a homing sequence. For each inference problem, we show how a homing sequence can be used to infer the unknown machine, and how a homing sequence can be inferred as part of the overall inference procedure.

2 Two Representations of Finite Automata

2.1 Global State-Space or Standard Representation

Definition 1 *The standard representation of an* environment *or* finite state automaton $\mathcal{E}$ *is a tuple* $(Q, B, \delta, q_0, \gamma)$ *where:*

- Q *is a finite nonempty set of* states,
- B *is a finite nonempty set of* input symbols *or* basic actions,
- δ *is the* next-state function, *which maps* $Q \times B$ *into* Q,
- q_0, *a member of* Q, *is the* initial state, *and*
- γ *is the* output function, *which maps* Q *to* $\{0, 1\}$.

For example, the graph of Figure 1 depicts the global state representation of an automaton whose states are the vertices of the graph, whose transition function is given by the edges, and whose output function is given by the shading of the vertices.

We denote the set of all finitely long action sequences by $A = B^*$, and we extend the domain of the function $\delta(q, \cdot)$ to A in the usual way: $\delta(q, \lambda) = q$, and $\delta(q, ab) = \delta(\delta(q, a), b)$ for all $q \in Q, a \in A, b \in B$. Here, λ denotes

the empty or null string. For shorthand, we write qa to mean $\delta(q, a)$, the state reached by executing a from q. We say that $\mathcal{E}$ is a *permutation automaton* if for every action b, the function $\delta(\cdot, b)$ is a permutation of Q.

We refer to the sequence of outputs produced by executing a sequence of actions $a = b_1 b_2 \ldots b_r$ from a state q as the *output of a at q*, denoted $q\langle a \rangle$:

$$q\langle a \rangle = (\gamma(q), \gamma(qb_1), \gamma(qb_1b_2), \ldots, \gamma(qb_1b_2 \ldots b_r)).$$

(Don't confuse $\gamma(qa)$ and $q\langle a \rangle$. The former is a single value, the output of the state reached by executing a from q. In contrast, $q\langle a \rangle$ is a $(|a|+1)$-tuple consisting of the sequence of outputs produced by executing a from state q.) Finally, for $a \in A$, we denote by $Q\langle a \rangle$ the set of possible outputs on input a:

$$Q\langle a \rangle = \{q\langle a \rangle : q \in Q\}.$$

Clearly, $|Q\langle a \rangle| \leq |Q|$ for any a.

Action sequence a is said to *distinguish* two states q_1 and q_2 if $q_1\langle a \rangle \neq q_2\langle a \rangle$. We assume that $\mathcal{E}$ is *reduced*: for every pair of unequal states there is some action sequence which distinguishes them.

2.2 The Diversity-Based Representation

In this section, we describe the second of our representations. See [11, 12] for further background and detail. The representation is based on the notion of *tests* and *test equivalence.*

A *test* is an action sequence. (This definition differs slightly from that given in previous papers where we consider automata with multiple output ("sensations") at each state.) The *value* of a test t at state q is $\gamma(qt)$, the output of the state reached by executing t from q.

Two tests t_1 and t_2 are *equivalent*, written $t_1 \equiv t_2$, if the tests have the same value at every state. It's easy to verify that this defines an equivalence relation on the set of tests. We write $[t]$ to denote the *equivalence class* of t, the set of tests equivalent to t. The value of $[t]$ at q is well defined as $\gamma(qt)$. The *diversity* of the environment, $D(\mathcal{E})$, is the number of equivalence classes of the automaton: $D(\mathcal{E}) = |\{[t] : t \in A\}|$. In [11, 12], it is shown that $\lg(|Q|) \leq D(\mathcal{E}) \leq 2^{|Q|}$, so the diversity of a finite automaton is always finite.

The equivalence classes can be viewed as *state variables* whose values entirely describe the state of the environment. This is true because two states are equal (in a reduced sense) if and only if every test has the same value in both states.

It is often convenient to arrange the equivalence classes in an *update graph* such as the one in Figure 2 for the environment of Figure 1. Each vertex in the graph is an equivalence class so the size of the graph is $D(\mathcal{E})$. An edge labeled $b \in B$ is directed from vertex $[t_1]$ to $[t_2]$ iff $t_1 \equiv bt_2$. Note that each vertex has exactly one in-going edge labeled with each of the basic actions. This is because if $t_1 \equiv t_2$ then $bt_1 \equiv bt_2$.

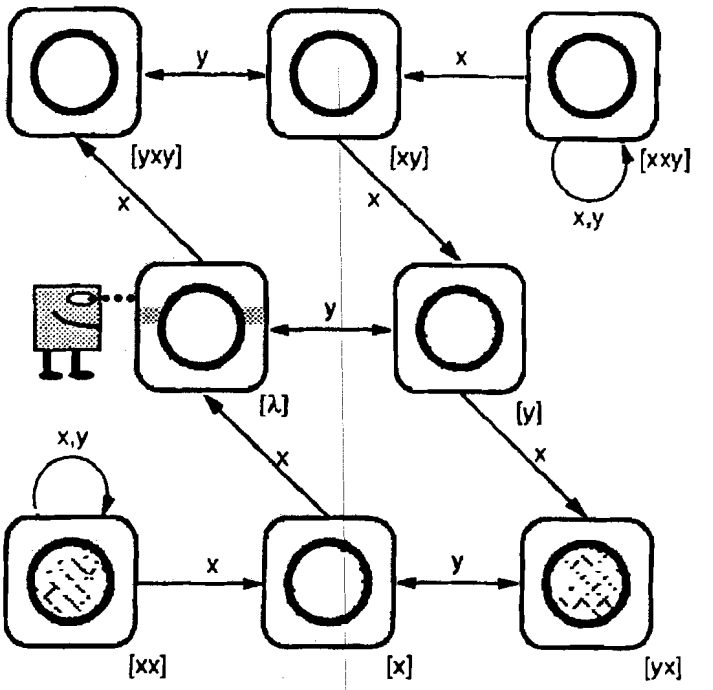

Figure 2: The Update Graph for the Environment of Figure 1

We associate with each vertex $[t]$ the value of t in the current state q. In the figure, we have used shading to indicate the value of each vertex in the robot's current state. The output of the current state is given by vertex $[\lambda]$, so this is the only vertex whose value can be observed by the robot. When an action b is executed from q, each vertex $[t]$ is replaced by the old value of $[bt]$, the vertex at the tail of $[t]$'s (unique) in-going b-edge. That is, in the new state qb, equivalence class $[t]$ takes on the old value of $[bt]$ in the starting state q. This follows from the fact that $\gamma((qb)t) = \gamma(q(bt))$. Thus, the update graph can be used to simulate the environment.

On first blush, the structures of the update graph of Figure 2 and the transition diagram of Figure 1 appear to be quite similar. In fact, their interpretations are very different. In the global state representation, the robot moves from state to state while the output (shading) of the states remains unchanged. On the other hand, in the diversity-based representation, the robot remains stationary, only observing the output of one vertex ($[\lambda]$), and allowing the values of the other vertices to come to him. Thus, the diversity-based representation is more egocentric — the world is represented relative to the robot, while in the state-based representation, the world is represented by its global structure which the robot moves within.

Input: $\mathcal{E}$ - a finite state automaton
Output: h - a homing sequence
Procedure:
  $h \leftarrow \lambda$
  while $(\exists q_1, q_2 \in Q) q_1\langle h\rangle = q_2\langle h\rangle$ but $q_1 h \neq q_2 h$ **do**
    let $x \in A$ distinguish $q_1 h$ and $q_2 h$
    $h \leftarrow hx$
  end

Figure 3: A State-Based Algorithm for Constructing a Homing Sequence

3 Homing Sequences

Henceforth, we set $D = D(\mathcal{E})$, $n = |Q|$, $k = |B|$.

Definition 2 *A* homing sequence *is an action sequence h for which the state reached by executing h is uniquely determined by the output produced: h is a homing sequence iff*

$$(\forall q_1 \in Q)(\forall q_2 \in Q) q_1\langle h\rangle = q_2\langle h\rangle \Rightarrow q_1 h = q_2 h.$$

As a quick example, the string consisting of the single action "**x**" is a homing sequence for the environment of Figure 1. If $q\langle \mathbf{x}\rangle = \square\,\square$, then $q\mathbf{x} = 3$; if $q\langle \mathbf{x}\rangle = \square\,\blacksquare$, then $q\mathbf{x} = 2$; and, if $q\langle \mathbf{x}\rangle = \blacksquare\,\blacksquare$ then $q\mathbf{x} = 1$.

Kohavi [8] gives a complete discussion of homing sequences. He distinguishes between "preset" and "adaptive" homing sequences. We make use only of the former in this abstract because they are simpler; we show in the full paper that our inference procedures can be improved using adaptive homing sequences.

Given full knowledge of the structure of $\mathcal{E}$, it is easy to construct a homing sequence h, as shown in Figure 3. Initially, $h = \lambda$. On each iteration of the loop, a new extension x is appended to the end of h so that h now distinguishes two states not previously distinguished. Thus, $|Q\langle h\rangle| < |Q\langle hx\rangle| \leq n$, and therefore the program will terminate after at most n iterations. Further, since each extension need only have length n, we have shown how to construct a homing sequence of length at most n^2.

A *diversity-based homing sequence* is an action sequence h which has the property that for every test t, there exists a prefix p of h such that $p \equiv ht$. Every diversity-based homing sequence h is a homing sequence. For if $q_1 h \neq q_2 h$ then there is some t for which $\gamma(q_1 ht) \neq \gamma(q_2 ht)$. Since ht is equivalent to some prefix p of h, we have $\gamma(q_1 p) \neq \gamma(q_2 p)$. Thus, $q_1\langle h\rangle \neq q_2\langle h\rangle$.

Figure 4 shows an algorithm for constructing a diversity-based homing sequence h. Again, h is built up from λ by appending extensions x. On each iteration, $|\{[p] : p \text{ prefix of } h\}|$ increases by at least one, so there are at most D iterations. Each extension need be no longer than D, so we can find a diversity-based homing sequence of length at most D^2.

Input: $\mathcal{E}$ - a finite state automaton
Output: h - a diversity-based homing sequence
Procedure:
  $h \leftarrow \lambda$
  while $(\exists x \in A)(\forall p \text{ prefix of } h) p \not\equiv hx$ **do**
    $h \leftarrow hx$
  end

Figure 4: A Diversity-Based Algorithm for Constructing a Homing Sequence

Some remarks about the length of homing sequences: First, the homing sequences constructed by the preceding algorithms are the best possible in the sense that there exist environments whose shortest homing sequence has length $\Omega(n^2)$ (or $\Omega(D^2)$). However, given a state-based (or a diversity-based) description of a finite-state machine, it is NP-complete to find the shortest homing sequence for the automaton. (The reduction is from exact 3-set cover.)

4 Inference of Finite Automata – The General Case

In this section, we describe algorithms for inferring the structure of an unknown environment $\mathcal{E}$.

We say that the learner has a *perfect model* of his environment if he can predict perfectly the output of the environment given any sequence of actions. The goal of our inference procedures is to construct a perfect model.

We assume that the learner is given access to $\mathcal{E}$, that the learner can observe the output of the environment when actions of his choosing are executed. We also assume that there is a "teacher" who provides the learner with counterexamples to incorrectly conjectured models of the environment. A counterexample is a sequence of actions whose true output from the current state differs from that predicted by the learner's model. Typically, there will be many sequences of actions which are counterexamples to a given conjecture, and by choosing an especially long or short counterexample, the teacher can significantly affect the running time of the procedure. This fact is reflected in our running times which depend on the length of the counterexamples provided.

In the framework of a robot learning about its environment, we might imagine the robot, upon completion of a model of the environment which it believes to be correct, using that model to make predictions of the output of the environment's next state until an incorrect prediction is made. In this situation, the sequence of actions leading up to the error is the needed counterexample.

We generally assume that the unknown automaton is *strongly connected*, that is, every state can be reached

from every other state:

$$(\forall q_1 \in Q)(\forall q_2 \in Q)(\exists a \in A)(q_1 a = q_2).$$

We make this assumption with little loss of generality: if $\mathcal{E}$ is not strongly connected, then an experimenting inference procedure, having no reset operation, will sooner or later fall into a strongly connected component of the state space from which it cannot escape, and so will have to be content thereafter learning only about that component.

4.1 A Global State-Based Algorithm

In this section, we describe an algorithm based on the global state representation for inferring an arbitrary unknown automaton.

Our procedure is based closely on Angluin's L^* algorithm for learning regular sets [1]. Angluin shows how to efficiently infer the structure of any finite-state machine in the presence of what she calls a *minimally adequate teacher*. Such a teacher can answer two kinds of queries: On a *membership query*, the learner asks whether a given input string w is in the unknown language U, that is, whether the string is accepted by the unknown machine. On an *equivalence query*, the learner conjectures that the unknown machine is isomorphic to one it has constructed. The teacher replies that the conjecture is either correct or incorrect, and in the latter case provides a counterexample w, a string accepted by one machine but not the other.

The idea of Angluin's algorithm is to maintain an *observation table* (S, E, T). Here, S and E are prefix-closed sets of strings. We can think of S as a set of strings that lead from the start state to the states of the automaton, and E as experiments which are executed from these states. The last variable T is a two-dimensional table whose rows are given by $S \cup SB$, and whose columns are given by E. Each entry $T(se)$, where $s \in S \cup SB$ and $e \in E$, records whether the string se is in the unknown language. For fixed s, Angluin denotes by $row(s)$ the vector of entries $T(se)$ for varying $e \in E$. Her algorithm extends S and E based on the results of queries, and ultimately outputs the correct automaton based on an equivalence between the states of the unknown machine and the distinct rows of the table T. We denote by N_M and N_E the number of membership and equivalence queries made by L^*. These variables are implicit functions of n, k and m, where m is the length of the longest counterexample received. For Angluin's procedure L^*, we have $N_M = O(kmn^2), N_E = n - 1$. However, an unpublished result due to Schapire improves N_M to $O(kn^2 + n \log m)$.

In our framework, the learner could easily simulate Angluin's algorithm L^* if it were given a reset: to perform a membership query on w, the learner resets the environment, and executes the actions of w, observing the output of the last state reached. To perform an equivalence query on $\mathcal{E}'$, the learner resets the automaton and conjectures that $\mathcal{E}'$ is a perfect model of the environment. The teacher returns an action sequence w on which the conjectured model fails; this is the counterexample needed by L^*.

```
Input:  access to E, a finite state automaton
        h - a homing sequence for E
Output: a perfect model of E
Procedure:
  repeat
    execute h, producing output σ
    if it doesn't already exist, create L*_σ, a new copy of L*
    simulate the next query of L*_σ:
      if L*_σ queries the membership of action sequence a
        then execute a and supply L*_σ with the output
          of the final state reached
      if L*_σ makes an equivalence query then
        if the conjectured model E' is correct then
          stop and output E'
        else
          obtain a counterexample and supply it to L*_σ
  end
```

Figure 5: A State-Based Algorithm for Inferring $\mathcal{E}$ Given a Correct Homing Sequence

However, in our model the learner is not provided with a reset. *The main idea of our algorithm is to replace the reset with a homing sequence.* In many respects, a homing sequence behaves like a reset: by executing the homing sequence, the learner discovers "where it is," what state it is at in the environment. However, unlike a reset, the final state is not fixed, and the learner does not know beforehand what state it will end up in. (Note that an automaton need not possess a *synchronizing sequence*, a sequence that forces the automaton into a given state independent of its starting state. So we use homing sequences instead.)

We begin by supposing that the learner has been provided with a correct homing sequence h. Later, we will show how to remove this assumption.

Suppose we execute h from the current state q, producing output $\sigma = q\langle h \rangle$. If we ever repeat this experiment from state q' and find $q'\langle h \rangle = \sigma$, then, because h is a homing sequence, the states where we finished must have been the same in both cases: $qh = q'h$. If we could guarantee that the output of h would continue to come up σ with good regularity, then we could simply infer $\mathcal{E}$ by simulating Angluin's algorithm, treating qh as the initial state. When L^* demands a reset, we execute h: if the output comes up σ, then we must be at qh, and our "reset" has succeeded; otherwise, try again. Unfortunately, in the general case, it may be very difficult to make h produce σ regularly.

Instead, *we simulate an independent copy L^*_σ of L^**

for each possible output σ of executing h, as shown in Figure 5. Since $|Q(h)| \leq n$, no more than n copies of L^* will be created and simulated. Furthermore, on each iteration of the loop, at least one copy makes one query and so makes progress towards inference of $\mathcal{E}$. Thus, this algorithm will succeed in inferring $\mathcal{E}$ after no more than $n(N_M + N_E)$ iterations.

We now describe how to combine construction of the homing sequence h with the inference of $\mathcal{E}$. We maintain throughout the algorithm a sequence h which we presume is a true homing sequence. When evidence arises indicating that this is not the case, we will see how h can be extended and improved, eventually leading to the construction of a correct homing sequence. Initially, we take $h = \lambda$.

We use our presumably correct homing sequence h as described above and in Figure 5. If h is indeed a true homing sequence, we will of course succeed in inferring $\mathcal{E}$.

On the other hand, if h is incorrect, we may discover *inconsistent behavior* in the course of simulating some copy of L^*: suppose on two different iterations of the loop in Figure 5, we begin in states q_1 and q_2, execute h, produce output $q_1\langle h\rangle = q_2\langle h\rangle = \sigma$, and, as part of the simulation of L^*_σ, execute action sequence x. If h were a homing sequence, then x's output would have to be the same on both iterations since q_1h and q_2h must be equal.

However, if h is not a homing sequence, then it may happen that $q_1h\langle x\rangle \neq q_2h\langle x\rangle$. That is, we have discovered that x distinguishes q_1h and q_2h, and so, just as was done in the algorithm of Figure 3, we replace h with hx, producing in a sense a "better" approximation to a homing sequence. At this point, the existing copies of L^* are discarded, and the algorithm begins from scratch (except for resetting h, of course). Since h can only be extended in this fashion n times, this only means a slowdown by at most a factor of n, compared to the algorithm of Figure 5.

Figure 6 shows how we have implemented these ideas. Here we have assumed n, the number of global states, has been provided to the learner. In fact, this assumption is entirely unnecessary. Although we omit the details, we can show that the stated bounds below hold (up to a constant) for a slightly modified algorithm which does not require that the learner be explicitly provided with the value of n. The trick is the usual one of repeatedly doubling our estimate of n.

Recall that L^* requires maintenance of an observation table (S, E, T). Let $(S_\sigma, E_\sigma, T_\sigma)$ denote the observation table of L^*_σ. Of course, T_σ can only record output produced when executing an action sequence from what is only *presumed* to be a fixed initial state.

Angluin's analysis implies that if L^*_σ makes more than $N_M + N_E$ queries, then the number of distinct rows will exceed n. This can only happen if h is not a homing sequence, but how do we know how to correctly

Input: access to $\mathcal{E}$, a finite state automaton
n - the number of states of $\mathcal{E}$
Output: a perfect model of $\mathcal{E}$
Procedure:
$h \leftarrow \lambda$
repeat
  execute h, producing output σ
  if it doesn't already exist, create L^*_σ, a new copy of L^*
  if $|\{row(s) : s \in S_\sigma\}| \leq n$ **then**
    simulate the next query of L^*_σ as in Figure 5
      (and check for inconsistency)
  else
    let $\{s_1, \ldots, s_{n+1}\} \subset S_\sigma$ be such that
      $row(s_i) \neq row(s_j)$
    randomly choose a pair s_i, s_j from this set
    let $e \in E_\sigma$ be such that $T_\sigma(s_ie) \neq T_\sigma(s_je)$
    with equal probability, re-execute either s_ie or s_je
      (and check for inconsistency)
  if inconsistency found executing some string x **then**
    $h \leftarrow hx$
    discard all existing copies of L^*
until a correct conjecture is made

Figure 6: A State-Based Algorithm for Inferring $\mathcal{E}$

extend h if we have not actually seen an inconsistency? We show that if an inconsistency has not been found by the time the number of rows exceeds n, then we can use a probabilistic strategy to find one quickly with high probability.

Suppose we execute h from state q, with output σ, and we find that for L^*_σ, there are more than n distinct rows. Then let $s_1, \ldots, s_{n+1}$ be as in Figure 6. By the pigeon-hole principle, there is at least one pair of distinct rows s_i, s_j such that $qhs_i = qhs_j$. Further, since $row(s_i) \neq row(s_j)$, there is some $e \in E_\sigma$ for which $T_\sigma(s_ie) \neq T_\sigma(s_je)$. However, $\gamma(qhs_ie) = \gamma(qhs_je)$. Therefore, either $\gamma(qhs_ie) \neq T_\sigma(s_ie)$ or $\gamma(qhs_je) \neq T_\sigma(s_je)$, and so re-executing s_ie (or s_je, respectively) from the current state qh will produce the desired inconsistency.

So the chance of choosing the correct pair s_i, s_j as above is at least $\binom{n+1}{2}^{-1}$, and the chance of then choosing the correct experiment to re-run of s_ie or s_je is at least $1/2$. Thus, it can be verified that the probability of finding an inconsistency using the technique of Figure 6 in this situation is at least $1/n(n+1)$. Repeating this technique $n(n+1)\ln(1/\delta)$ times gives a probability of at least $1 - \delta$ of finding an inconsistency. Also, no more than n^2 copies of L^* are ever created, and $|h|$ does not exceed $O(n^2 + nm)$.

Putting these facts together, we can show:

Theorem 1 *Given $\delta > 0$, the algorithm described in Figure 6 will correctly infer the structure of $\mathcal{E}$ with probability at least $1 - \delta$ after executing*

$$O(n^3(n+m)(n^2\log(n/\delta) + N_M + N_E))$$

actions, and in time polynomial in n,m,k and $1/\delta$.

If we assume $m = O(n)$ and $k = O(1)$ and use the previously given bounds on N_M and N_E, then the number of actions executed by the procedure (and the running time as well) simplifies to $O(n^6 \log(n/\delta))$.

Finally, the procedure can be modified, replacing the preset homing sequence which we have been using with an adaptive one (see [8]) whose input at each step depends on the output seen up that point. This modification shaves a factor of n off the bounds described above. (Details omitted.) Again assuming $m = O(n)$ and $k = O(1)$, this gives a time bound of $O(n^5 \log(n/\delta))$.

It is an open question whether this bound can be significantly tightened. It seems likely that an algorithm which combines the many copies of L^* into one would have a superior running time, although we have not been successful in implementing this intuition.

4.2 A Diversity-Based Algorithm

We only sketch some of the main ideas of our diversity-based algorithm for inferring finite automata in the general case.

In [11, 12], we show how the update graph can be constructed given access to an oracle for deciding the equivalence of any two tests. We therefore focus on the problem of deciding if any two tests are equivalent since with this capability, we can use previous results to fully construct the update graph.

Suppose we have been given a diversity-based homing sequence h. Let t be any test of interest, say one of a pair of tests which we are testing for equivalence. Then ht is equivalent to some prefix of h. We maintain for each such test t a *candidate set* $C(t)$ of the prefixes of h which could plausibly be equivalent to ht.

Initially, we let $C(t) = \{p : p \text{ prefix of } h\}$. Suppose we execute ht from some state q, and let $p \in C(t)$. Since p is a prefix of h, in executing ht we have observed both the outputs $\gamma(qp)$ and $\gamma(qht)$. If we find these outputs differ, then clearly $p \not\equiv ht$ so we eliminate p from $C(t)$.

If we find for tests t_1 and t_2 that $C(t_1)$ and $C(t_2)$ are disjoint, then t_1 and t_2 cannot possibly belong to the same equivalence class. Moreover, if for any $a \in A$ we find that $C(at_1)$ and $C(at_2)$ are disjoint, then $at_1 \not\equiv at_2$ and therefore $t_1 \not\equiv t_2$. These are the basic techniques for determining inequivalence between tests, given a diversity-based homing sequence.

When such a sequence is not provided, we again presume that h is a true homing sequence until it becomes necessary to extend and improve h. Initially, $h = \lambda$. If for some test x, $C(x)$ is reduced to the empty set, then clearly h cannot be a diversity-based homing sequence since hx is inequivalent to every prefix of h. Thus, we start again from scratch, replacing h with hx as is done in the algorithm of the preceding section. Extending h in this manner at most D times, we converge to a correct homing sequence.

Theorem 2 *There exists an algorithm which, given $\delta > 0$, access to an unknown environment $\mathcal{E}$, and a source of counterexamples, outputs a correct description of $\mathcal{E}$ with probability at least $1 - \delta$ in time*

$$O(kD^4(D+m)(mD + \log(kD/\delta))).$$

5 Inference of Permutation Automata

In this section, we sketch algorithms for inferring permutation automata. Unlike the procedures described up to this point, these procedures do *not* rely on a means of discovering counterexamples; the procedures actively experiment with the unknown environment, and output a perfect model with arbitrarily high probability.

As before, we describe both a state-based and a diversity-based procedure. In both cases, we describe deterministic procedures that, given a (diversity-based) homing sequence h, will output a perfect model of the environment in time polynomial in n (or D) and $|h|$. To construct the needed homing sequence, we show that any sufficiently long random sequence of actions is likely to be a homing sequence.

5.1 A Global State-Based Algorithm

Imagine a simpler situation in which the identity of each state is readily observable, i.e., the automaton is *visible*. For instance, suppose each state, instead of outputting 0 or 1, outputs its own name. In this situation, inference of the automaton is almost trivial. From the current state q, we can immediately learn the value of $\delta(q, b)$ by simply executing b and observing the state reached. If $\delta(q, b)$ is already known for all the basic actions, then either we can find a path based on what is already known about δ to a state for which this is not the case, or we have finished exploring the automaton. It is not hard to see that $O(kn^2)$ actions are executed in total by this procedure.

Now suppose that the unknown environment $\mathcal{E}$ is a permutation automaton and that a homing sequence h has been provided. Because $\mathcal{E}$ is a permutation environment, we can easily show that h is also a *distinguishing sequence*, that is, h distinguishes every pair of unequal states of $\mathcal{E}$. Put another way, $q_1\langle h\rangle = q_2\langle h\rangle$ iff $q_1 = q_2$, and thus the identity of any state is uniquely given by the output of h at that state. The identity of each state is almost directly observable.

To infer the environment, we therefore use the inference procedure described above for visible automata.

Each state q is named or represented by $q\langle h\rangle$, the output of h at that state. To identify the current state, simply execute h and observe the output produced.

Although executing h is helpful in identifying the state at the start of the sequence, doing so is also likely to leave us in a state at the end of the sequence whose identity is unknown. This is a problem because the visible automaton inference procedure requires that we be able to find a state whose identity is known even without executing h. We can overcome this problem, however, by maintaining a table u which records the fact that if $\sigma = q\langle h\rangle$ was just observed as the output of executing h, then the output of h if executed from the current state qh is given by $u(\sigma)$.

Thus, we can reach a state whose identity is known (without executing h from it), we can execute an experiment as dictated by the visible automaton inference procedure, and we can identify the last state reached by executing h. This can of course be repeated as many times as necessary. Thus, we can show:

Theorem 3 *There exists an algorithm which, given access to a permutation environment $\mathcal{E}$, and a homing sequence h for $\mathcal{E}$, outputs a perfect model of $\mathcal{E}$ in time $O(kn(|h| + kn))$. Furthermore, the total number of actions executed by this algorithm is at most $n|h| + kn(|h| + n)$.*

Finally, we must consider how to construct h. In fact, any sufficiently long random sequence of actions is almost certain to be a homing sequence:

Theorem 4 *Let $\delta > 0$, and let h be a random action sequence of length $4kn^6 \cdot \ln(n) \cdot \ln(n/\delta)$. Then h is a homing sequence with probability at least $1 - \delta$.*

Proof: (sketch) The idea is to construct the homing sequence in the manner described in Figure 3. On each iteration, an appropriate extension x which distinguishes some pair of states as needed by the algorithm is likely to be given by any sufficiently long random walk. This follows from the results on random walks in [12]. ■

These theorems give our inference procedure a running time of $O(k^2n^7 \log(n) \cdot \log(n/\delta))$.

5.2 A Diversity-Based Algorithm

We can show in a similar manner how a permutation environment can be inferred using a diversity-based representation. As before, we reduce the problem to that of inferring a visible automaton — in this case, one for which all of the test equivalence classes are known, and for which the value of each test class is observable in every state. The problem of inferring such automata is solved in Chapter 4 of [12]; the solution is based on the careful planning of experiments, and on the maintenance of candidate sets similar to those described in Section 4.2.

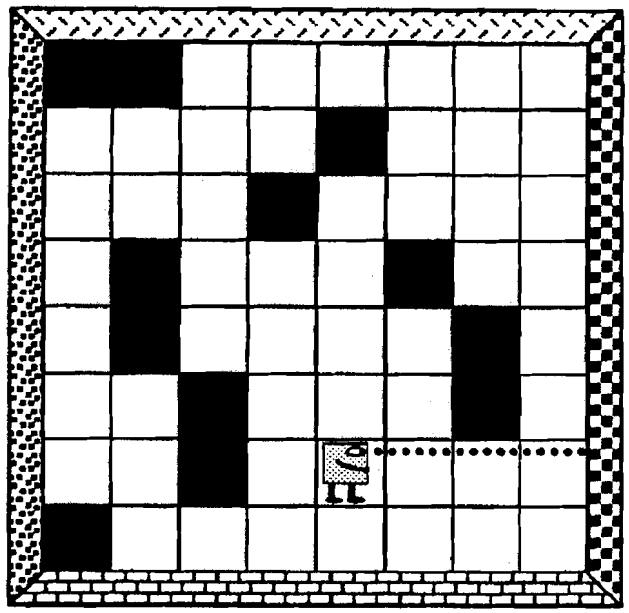

Figure 7: A Crossword Puzzle Environment

Let h be a given diversity-based homing sequence for the unknown permutation environment $\mathcal{E}$. As before, to simulate the inference algorithm for visible automata, it suffices to show that the state of the automaton (i.e. the values of the test classes) can be observed by executing h, and further that it is possible to reach a state whose identity is known even without executing h. Since $\mathcal{E}$ is a permutation environment, we can show that every test class is represented by some prefix of h. Therefore, at the current state q, the values of all the test classes can be observed simply by executing h.

To find a state in which the output of h is known (and thus the values of all the test classes as well) without actually executing the sequence, we maintain for each prefix p of h a candidate set $C(p)$ as in Section 4.2. Suppose $\sigma = q\langle h\rangle$ is the output just produced by executing h, and consider the set $X = \{\gamma(qp') : p' \in C(p)\}$ which is easily computed from σ. At all times, there is some prefix $p' \in C(p)$ for which $p' \equiv hp$. Therefore, the output of p from the current state qh is the same as that of p' from q for some $p' \in C(p)$. Thus, if $C(p)$ is *coherent*, that is, if X is a singleton, then $\gamma(qhp)$, the value of p at the current state qh, is known. If the candidate sets for all the prefixes are coherent, then $qh\langle h\rangle$, the output of the entire sequence, is known in the current state. On the other hand, if one of the candidate sets is incoherent, then by re-executing h we are guaranteed to eliminate at least one prefix from one of the candidate sets. Thus, we can quickly reach a state in which the output of h is known without actually executing it.

Combining these ideas, we can show:

Theorem 5 *There exists an algorithm which, given access to a permutation environment $\mathcal{E}$, and a hom-*

ing sequence h for $\mathcal{E}$, outputs a perfect model of $\mathcal{E}$ in time $O(kD(|h| + kD^2))$. Furthermore, the total number of actions executed by this algorithm is at most $D|h| + kD(|h| + D)$.

Again, we can construct h by randomly choosing a sequence of actions:

Theorem 6 *Let $\delta > 0$, and let h be a random sequence of length $2kD^3H_D \cdot \ln(D) \cdot \ln(D/\delta)$. Then h is a diversity-based homing sequence with probability at least $1 - \delta$.*

Here, H_n is the nth harmonic number. Thus, our inference procedure runs in time $O(k^2D^4\log^2(D) \cdot \log(D/\delta))$. This improves the previously best-known bound of $O(k^2D^7\log(D) \cdot \log(kD/\delta))$ given in [12] by roughly a factor of $D^3/\log(D)$.

6 Experimental Results

The algorithm described in Section 4.1 has been implemented and tested on several simple robot environments.

In the "Random Graph" environment, the robot is placed on a randomly generated directed graph. The graph has n vertices, and each vertex has one out-going edge labeled with each of the k basic actions. For each vertex i, one edge (chosen at random) is directed to vertex $i + 1 \bmod n$; this ensures that the graph contains a Hamiltonian cycle, and so is strongly connected. The other edges point to randomly chosen vertices, and the output of each vertex is also chosen at random.

In the "Knight Moves" environment, the robot is placed on a square checker-board, and can make any of the legal moves of a chess knight. However, if the robot attempts to move off the board, its action fails and no movement occurs. The robot can only sense the color of the square it occupies. Thus, when away from the walls, every action simply inverts the robot's current sensation: any move from a white square takes the robot to a black square, and *vice versa*. This makes it difficult for the robot to orient itself in this environment.

Finally, in the "Crossword Puzzle" environment, the robot is on a crossword puzzle grid such as the one in Figure 7. The robot has three actions available to it: it can step ahead one square, or turn left by 90 degrees, or turn right. The robot can only occupy the white squares of the crossword puzzle; an attempt to move onto a black square is a "no-op." Attempting to step beyond the boundaries of the puzzle is also forbidden. Each of the four "walls" of the puzzle has been painted a different color. The robot looks as far ahead as possible in the direction it faces: if its view is obstructed by a black square, then it sees "black"; otherwise, it sees the color of the wall it is facing. Thus, the robot has five possible sensations. Since this environment is essentially a maze, it may contain regions which are difficult to reach or difficult to get out of.

In the current implementation, we have used an adaptive homing sequence or homing tree. We have also used a modified version of L^* that is guaranteed to require fewer membership queries. Finally, we have implemented a heuristic that attempts to focus effort on copies of L^* that have already made the most progress: if the homing sequence is executed and the L^* copy reached is not very far along, then the procedure is likely to re-execute the homing sequence to find one that is closer to completion. The idea of the heuristic is not to waste time on copies that have a long way to go. The heuristic seems to improve the running time for these three environments by as much as a factor of six.

For the "Random Graph" and "Crossword Puzzle" environments, the inference procedure was provided in some experiments with an oracle which would return the shortest counterexample to an incorrect conjecture. All three environments were also tested with no external source of counterexamples; to find a counterexample, the robot would instead execute random actions until its model of the environment made an incorrect prediction of the output of some state.

Table 1 summarizes how our procedure handled each environment. In the table, "Source" refers to the robot's source of counterexamples: "S" indicates that the robot had access to the shortest counterexample, and "R" indicates that it had to rely on random walks. The column labeled "$|\text{range}(\gamma)|$" gives the number of possible sensations which might be experienced by the robot. (Extending our algorithms to the case that the range of γ consists of more than two elements is trivial.) "Copies" is the number of copies of L^* which were active when a correct conjecture was made, "Queries" is the total number of membership and equivalence queries which were simulated, "Actions" is the total number of actions executed by the robot, and "Time" is elapsed cpu time in minutes and seconds. The procedure was implemented in C on a DEC MicroVax III. For example, inferring the 8×8 "Knight Moves" environment using randomly generated counterexamples required about 400,000 moves and 19 seconds of cpu time.

Note that for the "Random Graph" environment, the learning procedure sometimes did better with randomly generated counterexamples than with an oracle providing the shortest counterexample. It is not clear why this is so, although it seems plausible that in some way the random walk sequences give more information about the environment. For example, the counterexamples often become subsequences of the homing sequence, and it may be that random walk counterexamples make for better, more distinguishing homing sequences.

Environment	size	n	k	$\lvert\text{range}(\gamma)\rvert$	Source	Copies	Queries	Actions	Time (min:sec)
Random Graph	25	25	3	2	S	20	1,108	10,504	:01.0
					R	21	1,670	17,901	:01.2
	50	50	3	2	S	37	5,251	69,861	:06.0
					R	33	4,581	61,325	:03.6
	100	100	3	2	S	68	14,788	279,276	:24.1
					R	64	17,221	342,450	:18.1
	200	200	3	2	S	137	34,182	1,100,244	1:31.9
					R	136	29,796	1,012,279	:47.5
	400	400	3	2	S	275	72,027	3,010,377	4:52.0
					R	258	33,388	1,757,720	1:19.5
Knight Moves	4	16	8	2	R	10	2,082	19,621	:01.4
	8	64	8	2	R	50	17,818	385,678	:19.4
	12	144	8	2	R	88	22,208	780,595	:36.3
	16	256	8	2	R	124	63,476	3,855,520	2:41.9
	20	400	8	2	R	157	129,407	8,329,257	5:58.9
Crossword Puzzle	4	48	3	5	S	41	2,424	30,285	:02.5
					R	41	2,817	55,749	:04.1
	8	208	3	5	S	97	18,523	839,087	:52.9
					R	104	16,643	1,049,466	:51.0
	12	416	3	5	S	188	68,793	5,564,299	5:15.6
					R	193	58,222	8,850,079	7:12.5

Table 1: Experimental Results

In sum, the running times given are quite fast, and the number of moves taken far less than allowed for by the theoretical worst-case bounds. Nevertheless, it is also true that the number of actions executed is still somewhat large, much too great to be practical for a real robot. There are probably many ways in which our algorithm might be improved — both in a theoretical sense, and in terms of heuristics which might improve the performance in practice. We leave these questions as open problems.

7 Conclusions

We have shown how to infer an unknown automaton, in the absence of a reset, by experimentation and with counterexamples. For the class of permutation automata, we have shown that the source of counterexamples is unnecessary. We have described polynomial time algorithms which are both state-based and diversity-based.

References

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

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

[3] Dana Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39:337–350, 1978.

[4] E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[5] E. Mark Gold. System identification via state characterization. *Automatica*, 8:621–636, 1972.

[6] Michael Kearns and Leslie G. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, Seattle, Washington, May 1989.

[7] Michael Kearns and Leslie G. Valiant. *Learning Boolean Formulae or Finite Automata is as Hard as Factoring.* Technical Report TR 14-88, Harvard University Aiken Computation Laboratory, 1988.

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

[9] Leonard Pitt and Manfred K. Warmuth. The minimum DFA consistency problem cannot be approximated within any polynomial. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, Seattle, Washington, May 1989.

[10] Leonard Pitt and Manfred K. Warmuth. Reductions among prediction problems: on the difficulty of predicting automata (extended abstract). In *3rd IEEE Conference on Structure in Commplexity Theory*, pages 60–69, Washington, DC, June 1988.

[11] Ronald L. Rivest and Robert E. Schapire. Diversity-based inference of finite automata. In *Proceeding of the Twenty-Eighth Annual Symposium on Foundations of Computer Science*, pages 78–87, Los Angeles, California, October 1987.

[12] Robert E. Schapire. *Diversity-Based Inference of Finite Automata.* Master's thesis, MIT Lab. for Computer Science, May 1988. Technical Report MIT/LCS/TR-413.