

Learning Finite Automata Using Local Distinguishing Experiments

Wei-Min Shen

Microelectronics and Computer Technology Corporation
3500 West Balcones Center Drive
Austin, TX 78759, U.S.A.

Abstract

One of the open problems listed in [Rivest and Schapire, 1989] is whether and how that the copies of L^* in their algorithm can be combined into one for better performance. This paper describes an algorithm called D^* that does that combination. The idea is to represent the states of the learned model using observable symbols as well as hidden symbols that are constructed during learning. These hidden symbols are created to reflect the distinct behaviors of the model states. The distinct behaviors are represented as local distinguishing experiments (LDEs) (not to be confused with global distinguishing sequences), and these LDEs are created when the learner's prediction mismatches the actual observation from the unknown machine. To synchronize the model with the environment, these LDEs can also be concatenated to form a homing sequence. It can be shown that D^* can learn, with probability $1 - \xi$, a model that is an ϵ -approximation of the unknown machine, in a number of actions polynomial in the size of the environment and $\frac{1}{\epsilon}(\log \frac{1}{\xi})$.

1 Introduction

Learning deterministic finite automata is an interesting topic for many research areas. It relates to language learning, autonomous systems, constructive concept induction, and perception with hidden states. There are two basic problems that underlie the task: the *construction problem* and the *synchronization problem*. The former is a problem of how to construct a perfect model of the unknown machine based on the feedback of actions, and the latter is a problem of how to synchronize the model with the unknown machine. An existing solution to the construction problem is the L^* algorithm developed by [Angluin, 1987]. L^* can learn regular expressions using membership queries and equivalence queries but it requires a means to reset the unknown automaton to a fixed state for synchronization. The algorithm developed by [Rivest and Schapire, 1989] extends L^* to deal with the synchronization problem by learning a

homing sequence of the unknown machine. However, for each different output of the current homing sequence, they create an independent copy of L^* . Every time the homing sequence is updated, all (or some of) the copies of L^* are discarded. It is an open question whether and how that the copies of L^* can be combined into one.

This paper describes the D^* algorithm that solves both the construction problem and the synchronization problem using a single mechanism: local distinguishing experiments (LDE). Since LDEs are incrementally discovered through the interactions with the environment and can be used to construct both hidden states and homing sequences, there is no need for keeping separate copies of models or discarding the models when new LDEs are discovered.

2 The Basic Definitions

Learning deterministic finite automata can be viewed as a special type of learning from the environment [Shen, 1989; Shen, 1993a]. The task of the learner is to construct an exact model of the environment to predict the consequences of its actions. Here the *environment* E is defined as a variation of a Moore machine $(A, Z, Q, \delta, \rho, r)$, where:

- A is a finite set of input symbols;
- Z is a finite set of output symbols;
- Q is a finite set of *environmental states*;
- $\delta : Q \times A \rightarrow Q$ is the *transition function* of E ;
- $\rho : Q \rightarrow Z$ is the *appearance function* of E ;
- r is the *current state* of E . When an input symbol $a \in A$ is received by E , r is updated to be $\delta(r, a)$. Initially, r can be any state in Q .

We assume E is strongly connected. We say that E is *transparent* if its ρ is one-to-one, and E is *translucent* if its ρ is many-to-one.

A *model* M of the environment E is defined as (A, Z, S, ϕ, t) , where:

- A is a set of *actions* of the learner;
- Z is a set of *percepts* of the learner;
- S is a set of *model states* represented as $Z\{0, 1\}^i$, where $\{0, 1\}^i$ is a finite sequence of binary symbols constructed by the learner;

- $\phi: S \times A \rightarrow S$ is the *transition function* of M ;
- t is the *current model state* of M . When an action $a \in A$ is applied to M , t is updated to be $\phi(t, a)$.

Model states are described by expressions $s = z\{0,1\}^*$, where $z \in Z$. A description s represents a set of model states whose prefix is s . For example, $z1$ represents all the model states whose prefixes are $z1$, and $z10$ represents all the model states whose prefixes are $z10$. Clearly, $z10$ is a subset of $z1$. Notice that the symbol " s " is used to denote both a *set* of model states and a *single* model state (when the set is a singleton). From the context, this uniform representation of model state(s) should not cause any confusion.

Given a description $s = z\{0,1\}^*$, we use $\alpha(s)$ to denote the observed part of s , and $\beta(s)$ for the constructed part of s . For example, if $s = z0110$, then $\alpha(s) = z$ and $\beta(s) = 0110$.

We denote the set of all finitely long action sequences by $B = A^*$, and use $\delta(q, b)$, where $b \in B$, to denote the environmental state reached by executing sequence b from an environmental state q . This denotation is also applicable to M 's transition function ϕ .

In the process of learning, we allow the function $\phi(s, b)$ to be nondeterministic. That is, the value $s' = \phi(s, b)$ may represent a set of model states whose prefix is s' . In that case, $\phi(s, b)$ denotes the set of model states, s' , reachable by executing the action sequence b from the model state s . As a special case, if $\phi(s, b)$ does not exist, we write $\phi(s, b) = ?$. The symbol "?" represents the entire set of model states.

We say that a single model state s is *visibly equivalent* to an environmental state q if $\rho(q) = \alpha(s)$. Furthermore, we say that M is *perfect* and *synchronized* with respect to E if the current model state t is visibly equivalent to the current environmental state r , and for all sequences of actions $b \in B$, the model state $\phi(t, b)$ is also visibly equivalent to the environmental state $\delta(r, b)$. Our objective is to learn a perfect and synchronized model of the environment.

Let r be the current environmental state. We refer to the sequence of percepts observed in the environmental states visited by executing $b = a_0a_1\dots a_e$ from r as the *observation sequence of b from r* , denoted as:

$$OS(r, b) \equiv [\rho(r), \rho(\delta(r, a_0)), \dots, \rho(\delta(r, a_0a_1\dots a_e))].$$

We assume that the environment is reduced: for every pair of unequal environmental states q_1 and q_2 , there is an action sequence b such that $OS(q_1, b) \neq OS(q_2, b)$.

Somewhat parallel to an observation sequence of b from r , we define the *prediction sequence of b at t* to be the sequence of model states visited by executing actions $b = a_0a_1\dots a_e$ from the current model state t , denoted as:

$$PS(t, b) \equiv [t, \phi(t, a_0), \dots, \phi(t, a_0a_1\dots a_e)].$$

Before we define local distinguishing experiments, let us first consider an example. Suppose you are to predict (not actually doing it) whether you can lift some boxes. These boxes look identical from the outside. However, if you open the lid you can see that a box is either empty or containing some weight that is too heavy for you to

lift. Naturally, in order to make a correct prediction, all you need to do is open the lid and see inside. The sequence of "open-lid" and "see-inside" is a local distinguishing experiment. It uses your actions and percepts to reveal more information about a particular environmental state.

Formally, let z be the current observation $\rho(r)$ from the environment (seeing a box), and $s = zc$, where c is a binary string, be a set of model states (the box is either empty or too heavy) that contains the current model state t (the true status of the box). Notice that the necessary condition for s to contain t is that $\alpha(s) = \alpha(t)$. We refer to a set of relations that can reduce the set zc to either $zc0$ or $zc1$, whichever contains t , as a *local distinguishing experiment of zc* , denoted as $LDE(zc?)$:

$$LDE(zc?) \equiv (b; zc0, [x]; zc1, [y]),$$

where b is a sequence of actions, $[x] = OS(q_1, b)$ and $[y] = OS(q_2, b)$ are two different previous observation sequences of a from some environmental states q_1 and q_2 whose appearance are z (i.e., $\rho(q_1) = \rho(q_2) = z$), and $b, [x], [y]$, and the current environmental state r , satisfy the relation:

$$zc? = \begin{cases} zc0, & \text{if } OS(r, b) = [x] \\ zc1, & \text{if } OS(r, b) = [y]. \end{cases}$$

Thus, if it is known that the current model state t is in zc (e.g., a box is either empty or too heavy), then one can determine if t is in $zc0$ or $zc1$ by executing b (e.g., opening the lid) in the current environmental state r , comparing the actual observation sequence (e.g., what you see inside the box) with $[x]$ and $[y]$, and concluding either t was in $zc0$ or t was in $zc1$. We define the *length of $LDE(zc?)$* to be the length of its actions b .

3 The Construction and Synchronization Problems

It is interesting that when the environment is transparent, the construction and the synchronization problems can be solved easily. Since all the transparent environmental states are visibly different, we can treat $q \equiv z$. The synchronization problem is trivial because what we see is where we are. Thus, we can build $\phi(z, a)$ from a state z by simply executing a and observe the state reached. If $\phi(z, a)$ is already known for every action in A , then either we can find a path based on what is already known about ϕ to a state for which this not the case, or we have finished learning a perfect model of E . Figure 1 gives this simple algorithm in detail. For convenience, we will refer to this algorithm as D algorithm.

In general, however, environments are not transparent. Because the appearance function ρ is many-to-one, two or more environmental (and model) states may have the same output and the learner's current observation may not uniquely determine what the current environmental (and model) state is. For example, in the SHAPE environment (copied from [Rivest and Schapire, 1989]) shown in Figure 2, there are two actions, x and y , and four environmental states I, II, III, and IV. A learner, however, can only see the two shapes of the nodes: $\square$ or

0. Initializing $S = Z$ and $\phi = \{\}$.
1. Repeat
 2. Let the current model state $t = z_0$, where $z_0 = \rho(r)$ is the current observation from the environment;
 3. Select actions ba such that $\phi(\phi(t, b), a)$ does not exist;
 4. Execute ba ;
 5. Record $OS(r, ba) = [z_0, \dots, z_{|b|}, z_a]$;
 6. Insert $\phi(z_{|b|}, a) = z_a$ into ϕ ;
 7. Until $\phi(z, a)$ exists for all $z \in Z$ and $a \in A$.

Figure 1: The D algorithm for transparent environments

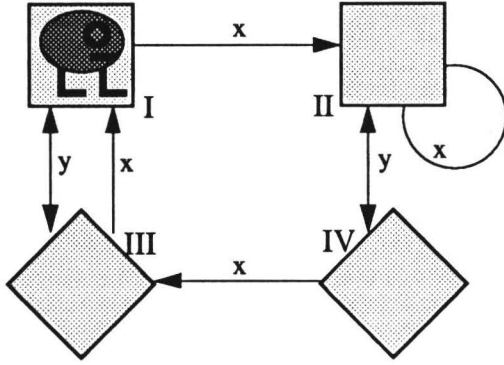


Figure 2: The SHAPE environment

◇. When □ is seen, the learner does not have enough information to determine whether the current environmental state is I or II.

To model a translucent environment, model states must consist of both observed symbols and constructed symbols in order to distinguish different environmental states that appear alike. For example, one way to model the states in the SHAPE environment is to use one hidden binary symbol for the model states. So, one can use □0 for state I, □1 for state II, ◇0 for state III, and ◇1 for state IV.

Because of the hidden symbols involved in the model states, both the construction problem and the synchronization problem become substantial. In particular, since the number of model states is greater than the number of percepts, the learner must determine when and how to construct new model states. Since there are more than one model states whose observed part matches the current observation, the learner must also determine which model state is current. Clearly, the simple D algorithm must be extended to deal with these problems.

4 Model Construction with LDEs

To solve the construction problem, we use a technique called *surprise-identify-split* [Shen, 1989; Shen, 1993a]. The main idea is that the learner always predicts the consequence of its action based on the model that is learned. Whenever the actual consequence does not match the predicted consequence (a surprise), the learner identifies a “faulty” state in the model that is responsible for the surprise and splits that state into two by appending to it

a new binary symbol. The action sequence that causes the surprise, along with the prediction sequence and the observation sequence at that time, is defined as a new LDE for distinguishing the two new states in the future.

4.1 Surprises

Let r be the current environmental state (which is unknown to the learner), $z_0 = \rho(r)$ be the current observation, and s_0 be the current model state, where $\alpha(s_0) = z_0$. Then, after executing a sequence of actions $b = a_0 a_1 \dots a_e$, we have a prediction sequence and observation sequence as follows:

$$PS(s_0, b) = [s_0, s_1, \dots, s_e, s_{e+1}], \quad (1)$$

$$OS(r, b) = [z_0, z_1, \dots, z_e, z_{e+1}]. \quad (2)$$

We say that there is a *surprise* if the latest prediction is visibly different from the latest observation: $z_{e+1} \neq \alpha(s_{e+1})$.

For example, suppose a learner in the SHAPE environment has learned, using the D algorithm, the following model:

$$M_0 = \begin{bmatrix} \phi(\diamond, x) = \diamond \\ \phi(\square, x) = \square \\ \phi(\diamond, y) = \square \\ \phi(\square, y) = \diamond \end{bmatrix},$$

and suppose that the current environmental state is III (which is unknown to the learner), and the learner’s current observation is ◇. Since the learner thinks that the current model state is ◇, the prediction sequence and the observation sequence of executing the action x are:

$$PS(\diamond, x) = [\diamond, \diamond],$$

$$OS(III, x) = [\diamond, \square].$$

There is a surprise in the above prediction/observation sequences because their last elements are visibly different: $\diamond \neq \square$.

4.2 Identify and Split

Since the learner’s predictions are based on previous observations, a surprise means that some model state is mistakenly representing more than one environmental state. The learner must identify such a “faulty” model state, and split it into two new model states: one represents the prediction, the other represents the surprising observation.

The process of identify and split is simple when the surprise is the first one (i.e., there is no previously defined LDEs). In this case, the state s_e in $PS(s_0, b)$ (see Equation 1) is the “faulty” state. This is because s_e is the state to which the same action, a_e , causes different consequences, s_{e+1} and z_{e+1} . Thus, s_e is split into s_{e0} and s_{e1} and a LDE for s_e is defined:

$$LDE(s_e?) = (a_e; s_{e0}, [\alpha(s_e), \alpha(s_{e+1})]; s_{e1}, [z_e, z_{e+1}]). \quad (3)$$

For example, in the above surprise between $PS(\diamond, x)$ and $OS(III, x)$, the faulty model state is ◇ and it is split into ◇0 and ◇1, and a new LDE is defined as:

$$LDE(\diamond?) = (x; \diamond 0, [\diamond, \diamond]; \diamond 1, [\diamond, \square]). \quad (4)$$

When the surprise is not the first one, then the process of identify and split is more complex. In this case, since there exist previously defined LDEs, the elements in $PS(s_0, b)$ (see Equation 1) and $OS(r, b)$ (see Equation 2) may represent sets of model (environmental) states. Before the faulty state can be identified, these elements must be "refined" (i.e., reducing the number of model states each element represents) using the existing LDEs. In fact, such a refinement is conducted every time the prediction/observation sequences are extended (i.e., a new action-observation pair becomes available).

The refinement of the prediction sequence is accomplished as follows. Let

$$\widehat{PS}(s_0, b) = [p_0, p_1, \dots, p_e, p_{e+1}] \quad (5)$$

be the refined sequence of $PS(s_0, b)$. By definition, the relation between $PS(s_0, b)$ and $\widehat{PS}(s_0, b)$ is that $p_i \subseteq s_i$ for all $0 \leq i \leq e+1$. Initially, each p_i is copied from the corresponding s_i in the prediction sequence $PS(s_0, b)$, except when $s_i = ?$ (i.e., no prediction for this particular step), p_i is copied from z_i in the observation sequence $OS(r, b)$. Then, for each p_i in $\widehat{PS}(s_0, b)$ from the end to the beginning, we repeatedly check if it can be "shrunk" by some existing LDEs. In other words, for each p_i , we check if there exists a $LDE(p_i?) = (l; p_i0, [x]; p_i1, [y])$ such that l is equal to a subsequence of b that begins with a_i , say $l = a_i \dots a_j$, and $[x]$ (or $[y]$) is equal to the subsequence $z_i, \dots, z_{j+1}$ in $OS(r, b)$. If so, p_i is set to be p_i0 (or p_i1). The same procedure is also used to refine the observation sequence $OS(r, b)$ to get:

$$\widehat{OS}(r, b) = [o_0, o_1, \dots, o_e, o_{e+1}], \quad (6)$$

where $o_i \subseteq z_i$ for all $0 \leq i \leq e+1$.

To illustrate the refinement procedure, suppose that the learner's current model of the SHAPE environment is M_1 (we will show how M_1 is built from M_0 shortly):

$$M_1 = \begin{bmatrix} \phi(\diamond 0, x) & = & \diamond? \\ \phi(\diamond 1, x) & = & \square \\ \phi(\square, y) & = & \diamond? \\ \phi(\square, x) & = & \square \end{bmatrix},$$

and the current prediction sequence and observation sequence are:

$$\begin{aligned} PS(\square, yx) &= [\square, \diamond?, ?], \\ OS(II, yx) &= [\square, \diamond, \diamond]. \end{aligned}$$

Initially, $\widehat{PS}(\square, yx) = [\square, \diamond?, \diamond]$ and $\widehat{OS}(II, yx) = [\square, \diamond, \diamond]$. Notice that the third element of $\widehat{PS}(\square, yx)$ is copied from $OS(II, yx)$ because the corresponding prediction in $PS(\square, yx)$ is a "?". Then, using the existing LDEs, we refine these sequences to get:

$$\begin{aligned} \widehat{PS}(\square, yx) &= [\square, \diamond 0, \diamond], \\ \widehat{OS}(II, yx) &= [\square, \diamond 1, \diamond]. \end{aligned}$$

Notice that the second elements in $\widehat{PS}(\square, yx)$ and $\widehat{OS}(II, yx)$ are inferred by the existing $LDE(\diamond?) = (x; \diamond 0, [\diamond \diamond]; \diamond 1, [\diamond \square])$, defined in Equation 4, respectively.

Once PS and OS are refined to be $\widehat{PS}$ and $\widehat{OS}$, we are ready to identify the faulty state. The faulty state is the state p_k in $\widehat{PS}(s_0, b)$, where k is the greatest index such that $p_k \subseteq o_k$ and $p_{k+1} \not\subseteq o_{k+1}$. The motivation is that p_k is the state to which applying the same action yields different next states. It can be shown that such p_k always exists when a surprise occurs. However, it is still an open question whether p_k is always the "optimal" faulty state to split. When p_k is not optimal, the states created by splitting p_k may become wasted: they become unreachable eventually.

After p_k is identified, it is split into p_k0 and p_k1 , and a new local distinguishing experiment for p_k is defined as follows:

$$\begin{aligned} LDE(p_k?) &= \\ (a_k \dots a_e; p_k0, [\alpha(p_k), \dots, \alpha(p_{e+1})]; p_k1, [z_k, \dots, z_{e+1}]). \end{aligned} \quad (7)$$

Notice that Equation 7 is a generalized version of Equation 3. Notice also that the length of the new LDE (because of the way p_k is identified) is at most one plus the maximum length of the existing LDEs.

In the SHAPE environment example, after $\widehat{PS}(\square, yx)$ and $\widehat{OS}(II, yx)$ are computed, the state $\square$ is identified to be faulty, and it is split into $\square 0$ and $\square 1$. A new local distinguishing experiment $LDE(\square)$ is defined as:

$$LDE(\square?) = (yx; \square 0, [\square, \diamond, \diamond]; \square 1, [\square, \diamond, \square]). \quad (8)$$

4.3 Model Refinement

Models are refined in two occasions: when the refined prediction sequence is more deterministic than the model, and when a new LDE is defined after a surprise.

In the first case, since a refined prediction sequence incorporates the most recent observations from the environment, it may contain information that is not in the current model. Such information provides opportunities to update the model accordingly. For example, from $\widehat{PS}(\square, yx)$ we can upgrade the nondeterministic transition $\phi(\square, y) = \diamond?$ in M_1 to $\phi(\square, y) = \diamond 0$, and the updated M_1 is:

$$M'_1 = \begin{bmatrix} \phi(\diamond 0, x) & = & \diamond? \\ \phi(\diamond 1, x) & = & \square \\ \phi(\square, y) & = & \diamond 0 \\ \phi(\square, x) & = & \square \end{bmatrix}.$$

In general, the transition function is updated according to $\widehat{PS}$ as follows. If there exists p_i, a_i and p_{i+1} , such that p_i is a single model state and p_{i+1} is a subset of $\phi(p_i, a_i)$, then $\phi(p_i, a_i)$ is set to be p_{i+1} .

In the second case, after the model state p_k is split, the transition function ϕ is modified as follows: (1) delete the obsolete transitions $\phi(p_k, \cdot)$ from ϕ since p_k is no longer a single model state; (2) insert two new transitions $\phi(p_k0, a_k) = p_{j+1}$ and $\phi(p_k1, a_k) = o_{j+1}$ into ϕ . Notice that all the transitions that go into p_i , i.e., $\phi(\cdot, \cdot) = p_i$, now become nondeterministic (since p_k now represents two model states p_i0 and p_i1). Such nondeterminism will be resolved in future explorations by model refinement in the first case.

An example of model refinement after a surprise is how M_1 is built from M_0 . When the state $\diamond$ is split in

Procedure $D^*(A, Z)$:

0. Initialize $S = Z$, $\phi = \{\}$, and $t = ?$;
1. Repeat
2. While t is a not single model state,
do $t \leftarrow \text{GOTO-A-M-STATE}()$;
3. Let r be the unknown current environmental state;
4. Select actions $b = b'a$ such that $\phi(\phi(t, b'), a) = p$
and p is not a single model state;
5. If b is empty, then $b \leftarrow$ random action sequence,
6. else $b = b \cdot l$ where l is the actions in $LDE(p?)$;
7. For each action a_j in $b = a_0, a_1, \dots, a_j, \dots, a_{|b|}$, do:
8. Execute a_j , construct $PS(t, a_0 \dots a_j)$, $OS(r, a_0 \dots a_j)$,
9. Compute $\widehat{PS}(t, a_0 \dots a_j) = [p_0, \dots, p_j, p_{j+1}]$
and $\widehat{OS}(r, a_0 \dots a_j) = [o_0, \dots, o_j, o_{j+1}]$,
10. Refine the model M (See case 1 in Section 4.3);
11. If $\alpha(s_{j+1}) \neq z_{j+1}$ (a surprise), then
12. call $\text{ID-SP-RE}(\widehat{PS}, \widehat{OS})$;
13. Goto 1;
14. Until no surprises for at least Ω actions.

Procedure $\text{ID-SP-RE}(\widehat{PS}(t, b), \widehat{OS}(r, b))$:

- Let k be the greatest index in $\widehat{PS}(t, b)$ such
that $p_k \subseteq o_k$, $p_{k+1} \not\subseteq o_{k+1}$;
- Replace p_k in S by $p_k 0$ and $p_k 1$;
- Define $LDE(p_k?)$ (See Equation 7);
- Refine the model M (See case 2 in Section 4.3).

Figure 3: The D^* algorithm

model M_0 after the first surprise (see Equation 4), we get model M_1 according to the above refinement steps. (The readers can verify that.) Similarly, when the state $\square$ is split after the second surprise (see Equation 8), the model M'_1 becomes a new model:

$$M_2 = \begin{bmatrix} \phi(\diamond 0, x) & = & \diamond 1 \\ \phi(\diamond 1, x) & = & \square? \\ \phi(\square 0, y) & = & \diamond 0 \\ \phi(\square 1, y) & = & \diamond 1 \end{bmatrix}.$$

5 The D^* Algorithm

With the techniques described in Section 4, we now present the D^* algorithm in Figure 3. For the moment, we assume that a solution for the synchronization problem is given as a procedure "GOTO-A-M-STATE" which tells the learner what the current model state is.

It is obvious that the D^* algorithm is an extension of the simple D algorithm. Line 2 determines the current model state t using the assumed procedure GOTO-A-M-STATE. Lines 4 to 6 select actions. Similar to the D algorithm where actions are selected for constructing nonexistent transitions, D^* selects actions to reduce the nondeterminism (nonexistence is a special case) of transitions. To reduce the nondeterminism of $\phi(\phi(t, b'), a) = p$, we use $LDE(p?)$ and run its actions l after $b'a$ (Line 6). If the current transition ϕ is already both complete (i.e., $\phi(s, b)$ exists for all $s \in S$ and $b \in B$) and deterministic, then a sequence of random actions is selected at Line 5. The purpose of random actions is to look for surprises (or counterexamples). Clearly, if an

oracle were available to provide such counterexamples, then line 5 will be the place to call the oracle.

For each action in the select action sequence, Lines 8 and 9 construct and refine the prediction and observation sequences (Section 4.2), and Line 10 tightens up some nondeterministic transitions in ϕ . When a surprise occurs (Line 11), Line 12 calls the procedure ID-SP-RE (See Sections 4.2 and 4.3) to identify and split the faulty state, define a new LDE, and refine the model.

The termination for D^* is different from that of the D algorithm. In a translucent environment, the learner cannot be absolutely certain that it has learned the perfect and synchronized model of the environment. (Of course, this is not a problem if an oracle exists.) Therefore, the learner must continue to wander in the environment even though its current model is already complete and deterministic. How long such wandering is needed depends on how confident the learner wants the model to be. For the moment, we assume that the wandering time before termination is a constant Ω .

6 Synchronization Using LDEs

We now describe the solution for the synchronization problem that is implemented in the procedure GOTO-A-M-STATE. The main idea is to use the concatenation of the actions in the existing LDEs as a homing sequence of the current model, and construct a homing function for identifying the model states that are reached at end of executions of the homing sequence.

A *homing sequence* [Rivest and Schapire, 1989] for an environment $E = (A, Z, Q, \delta, \rho, r)$ is a sequence of actions h such that the state reached by executing h from any state $q \in Q$ is uniquely determined by the observation sequence $OS(q, h)$. That is, h is a homing sequence iff for all $q_1, q_2 \in Q$:

$$[OS(q_1, h) = OS(q_2, h)] \implies [\delta(q_1, h) = \delta(q_2, h)].$$

It is known that every finite automaton has a homing sequence. So a model $M = (A, Z, S, \phi, t)$ must have a homing sequence too. We say that h is a homing sequence for M iff for all $s_1, s_2 \in S$:

$$[\alpha(PS(s_1, h)) = \alpha(PS(s_2, h))] \implies [\phi(s_1, h) = \phi(s_2, h)],$$

where $\alpha(PS(s, h))$ is the visible part of $PS(s, h)$.

Interestingly, the concatenation of the actions in the existing LDEs, in the order of their creations, is always a homing sequence for the current model. This can be proved by induction on the number of existing LDEs as follows.

Let i be the number of existing LDEs, M_i be the current model, and S_i be the set of model states in M_i . When $i = 0$, $S_0 = Z$ and all model states are visibly different so $h_0 = \{\}$ is a homing sequence of M_0 . Now assume that for M_i , $h_i = h_1 \dots h_i$ is a homing sequence, where h_i denotes the actions in LDE_i . To see that $h_{i+1} = h_i h_{i+1}$ is a homing sequence for M_{i+1} , we notice that the only difference between S_i and S_{i+1} is that a state $s_x \in S_i$ is replaced by $s_x 0$ and $s_x 1$ in S_{i+1} . Furthermore, $s_x 0$ and $s_x 1$ are the only two states in S_{i+1} that cannot be distinguished by h_i , that is, $s_x 0$ and $s_x 1$ can be reached from some states s_0 and

0. Let $s_0 = \rho(r)$ be the current observation;
1. Execute h_i and construct $\widehat{PS}(s_0, h_i)$ and $\widehat{OS}(r, h_i)$;
2. If a surprise occurs in step 1, then
 3. call ID-SP-RE($\widehat{PS}, \widehat{OS}$);
 4. set $H = \text{NULL}$; and return “?”;
5. Let $s \leftarrow H(OS(r, h_i))$, (if $h_i = \emptyset$, $H(s_0) = s_0$)
6. If s is a single model state, then returns s ,
7. If $s = ?$, then set both s and $H(OS(r, h_i))$ to be the last element in $\widehat{PS}(s_0, h_i)$;
8. Execute the actions h_s in $LDE(s)$ and extend $\widehat{PS}$ to $\widehat{PS}(s_0, h_i h_s)$ and $\widehat{OS}$ to $\widehat{OS}(r, h_i h_s)$;
9. If a surprise occurs in step 8, then
 10. call ID-SP-RE($\widehat{PS}, \widehat{OS}$);
 11. set $H = \text{NULL}$, and return “?”;
12. Set $H(OS(r, h_i))$ to be the l -th element in $\widehat{PS}(s_0, h_i h_s)$, where $l = |h_i|$;
13. Return the last element in $\widehat{PS}(s_0, h_i h_s)$.

Figure 4: The procedure GOTO-A-M-STATE

s_1 in S_{i+1} by h_i and $OS(s_0, h_i) = OS(s_1, h_i)$. However, LDE_{i+1} is defined to distinguish $s_x 0$ and $s_x 1$ and h_{i+1} has the property that $OS(s_x 0, h_{i+1}) \neq OS(s_x 1, h_{i+1})$. Therefore, $OS(s_0, h_{i+1}) \neq OS(s_1, h_{i+1})$ and h_{i+1} is a homing sequence for M_{i+1} . Notice that since the length of LDE_i is at most $i + 1$, the length of h_i is at most $\sum_{j=1}^i |h_j| = i(i+1)/2$.

Once we know the homing sequence h_i for the current model M_i , the synchronization problem can be solved by constructing a *homing function* H . The function H is a mapping from $Z^{|h_i|+1}$ to S_i , where $Z^{|h_i|+1}$ are observation sequences of length $|h_i| + 1$. Given an observation sequence $OS(q, h_i)$, $H(OS(q, h_i))$ is the model state that is reached at the end of h_i , regardless of what q is.

Thus, the task for GOTO-A-M-STATE is to construct the function H for the current model M_i until H is good enough to return a single model state immediately after executing a h_i . Figure 4 presents the details of the implementation. Notice that every time GOTO-A-M-STATE is called, either the current model will be updated due to a surprise, or the function H becomes more deterministic. In the first case, the procedure resets H and returns a “?” (A new H will be constructed for the new model when the procedure is called again.) In the second case, notice that the execution of $LDE(s)$ at Line 8 will gain more information for the model state s , and that information is used to make $H(OS(r, h_i))$ more certain at Line 12. For a model M_i , the number of actions to construct a complete H is at most $i(i + |Z|)(i^2 + i + 1)$: there are $i + |Z|$ possible entries in H , each of them takes at most i updates to complete, and each update takes at most $i^2 + i + 1$ actions (a homing sequence plus a LDE). Once H is complete for M_i , the procedure GOTO-A-M-STATE takes only h_i actions to execute for M_i and always returns a single model state. It is interesting to notice that at the end of a homing sequence, D^* goes to a model state while Rivest-Schapire’s algorithm goes to a copy of L^* .

7 Discussion

In the full paper, we show that D^* can learn a perfect and synchronized model in time $\max(O(n^5), O(\Omega(n - |P|)))$, assuming that no wasted model states are created. If the model is only required to be an ϵ -approximation of $\mathcal{E}$ with probability $1 - \xi$ [Angluin, 1987], then we can estimate $\Omega \geq \frac{1}{\epsilon}(\log \frac{1}{\xi})$. It is still an open problem, however, that why wasted states are created and what the upper bound of the number of such states is. For example, in learning randomly generated directed graphs with 25 states, 3 basic actions and 2 percepts, the best performance of the algorithm is about 1,500 actions. But there are graphs that cause D^* to generate over 50 states and uses more than 30,000 actions before a correct model is learned. We are currently investigating solutions for this problem using *adaptive* homing sequences [Rivest and Schapire, 1993].

Finally, the motivation for this algorithm is rather peculiar. I would like to combine the abilities of concept learning and learning from delayed feedback in the surprise-identify-revise framework. The version here is for learning from delayed feedback, but it links naturally to concept learning. When a surprise occurs, depending on whether there is any visible difference between the conditions of a failed prediction and a successful prediction, this algorithm can either learn a concept (i.e., a class of states that share some common properties) using complementary discrimination learning [Shen, 1993b], or create new state variables using the split technique here. In the latter case, this work can be viewed as a constructive approach to overcome inductive biases [Utgooff, 1986] and perceptual aliasing.

References

- [Angluin, 1987] D. Angluin. Learning regular sets from queries and counter-examples. *Information and Computation*, 75(2), November 1987.
- [Rivest and Schapire, 1989] R. Rivest and R. Schapire. Inference of finite automata using homing sequences. In *Proceedings of 21th Annual ACM Symposium on Theory of Computing*, 1989.
- [Rivest and Schapire, 1993] R. Rivest and R. Schapire. Inference of finite automata using homing sequences. *Information and Computation* (to appear), 1993.
- [Shen, 1989] W. Shen. *Learning from the Environment Based on Actions and Percepts*. PhD thesis, Carnegie Mellon University, 1989.
- [Shen, 1993a] W. Shen. *Autonomous Learning from the Environment*. W.H. Freeman, Computer Science Press, Forthcoming 1993.
- [Shen, 1993b] W. Shen. Discovery as autonomous learning from the environment. *Machine Learning*, 10, 1993.
- [Utgooff, 1986] P. Utgooff. *Machine Learning of Inductive Bias*. Kluwer Academic, 1986.