Chapter 3

Stochastic Learning Automata

An *automaton* is a machine or control mechanism designed to automatically follow a predetermined sequence of operations or respond to encoded instructions. The term *stochastic* emphasizes the adaptive nature of the automaton we describe here. The automaton described here do not follow predetermined rules, but adapts to changes in its environment. This adaptation is the result of the *learning* process described in this chapter.

> "The concept of learning automaton grew out of a fusion of the work of psychologists in modeling observed behavior, the efforts of statisticians to model the choice of experiments based on past observations, the attempts of operation researchers to implement optimal strategies in the context of the two-armed bandit problem, and the endeavors of system theorists to make rational decisions in random environments" [Narendra89].

In classical control theory, the control of a process is based on complete knowledge of the process/system. The mathematical model is assumed to be known, and the inputs to the process are deterministic functions of time. Later developments in control theory considered the uncertainties present in the system. Stochastic control theory assumes that some of the characteristics of the uncertainties are known. However, all those assumptions on uncertainties and/or input functions may be insufficient to successfully control the system if changes. It is then necessary to observe the process in operation and obtain further knowledge of the system, *i.e.*, additional information must be acquired on-line since a priori assumptions are not sufficient. One approach is to view these as problems in learning.

Rule-based systems, although performing well on many control problems, have the disadvantage of requiring modifications, even for a minor change in the problem space. Furthermore, rule-based approach, especially expert systems, cannot handle unanticipated situations. The idea behind designing a learning system is to guarantee robust behavior without the complete knowledge, if any, of the system/environment to be controlled. A crucial advantage of reinforcement learning compared to other learning approaches is that it requires no information about the environment except for the reinforcement signal [Narendra89, Marsh93].

A reinforcement learning system is slower than other approaches for most applications since every action needs to be tested a number of times for a satisfactory performance. Either the learning process must be much faster than the environment changes (as is the case in our

application), or the reinforcement learning must be combined with an adaptive forward model that anticipates the changes in the environment [Peng93].

Learning[1] is defined as any permanent change in behavior as a result of past experience, and a learning system should therefore have the ability to improve its behavior with time, toward a final goal. In a purely mathematical context, the goal of a learning system is the optimization of a functional not known explicitly [Narendra74].

In the 1960's, Y. Z. Tsypkin [Tsypkin71] introduced a method to reduce the problem to the determination of an optimal set of parameters and then apply stochastic hill climbing techniques. M.L. Tsetlin and colleagues [Tsetlin73] started the work on learning automata during the same period. An alternative approach to applying stochastic hill-climbing techniques, introduced by Narendra and Viswanathan [Narendra72], is to regard the problem as one of finding an optimal action out of a set of allowable actions and to achieve this using stochastic automata. The difference between the two approaches is that the former updates the parameter space at each iteration while the later updates the probability space.

The stochastic automaton attempts a solution of the problem without any information on the optimal action (initially, equal probabilities are attached to all the actions). One action is selected at random, the response from the environment is observed, action probabilities are updated based on that response, and the procedure is repeated. A stochastic automaton acting as described to improve its performance is called a *learning automaton.*

3.1 Earlier Works

The first learning automata models were developed in mathematical psychology. Early research in this area is surveyed by Bush and Mosteller [Bush58] and Atkinson *et al.* [Atkinson65]. Tsetlin [Tsetlin73] introduced deterministic automaton operating in random environments as a model of learning. Most of the works done on learning automata theory has followed the trend set by Tsetlin. Varshavski and Vorontsova [Varshavski63] described the use of stochastic automata with updating of action probabilities which results in reduction in the number of states in comparison with deterministic automata.

Fu and colleagues [Fu65a, Fu65b, Fu67, Fu69a, Fu69b, Fu71] were the first researchers to introduce stochastic automata into the control literature. Applications to parameter estimation, pattern recognition and game theory were initially considered by this school. Properties of linear updating schemes and the concept of a 'growing' automaton are defined by McLaren [McLaren66]. Chandrasekaran and Shen [Chand68,Chand69] studied nonlinear updating schemes, nonstationary environments and games of automata. Narendra and Thathachar have studied the theory and applications of learning autoamta and carried out simulation studies in the area. Their book *Learning Automata* [Narendra89] is an introduction to learning automata theory which surveys all the research done on the subject until the end of the 1980s.

[1] Webster defines 'learning' as 'to gain knowledge or understanding of a skill by study, instruction, or experience.'

The most recent (and second most comprehensive) book since the Narendra-Thathachar collaboration on learning automata theory and applications is published by Najim and Pznyak in 1994 [Najim94]. This book also includes several applications and examples of learning automata. Until recently, the applications of learning automata to control problems were rare. Consequently, successful updating algorithms and theorems for advanced automata applications were not readily available. Nario Baba's work [Baba85] on learning behaviors of stochastic automata under a nonstationary multi-teacher environment, and general linear reinforcement schemes [Bush58] are taken as a starting point for the research presented in this dissertation.

Recent applications of learning automata to real life problems include control of absorption columns [Najim91], bioreactors [Gilbert92], control of manufacturing plants [Sequeria91], pattern recognition [Oommen94a], graph partitioning [Oommen94b], active vehicle suspension [Marsh93, 95], path planning for manipulators [Naruse93], distributed fuzzy logic processor training [Ikonen97], and path planning [Tsoularis93] and action selection [Aoki95] for autonomous mobile robots.

Recent theoretical results on learning algorithms and techniques can be found in [Najim91b], [Najim94], [Sastry93], [Papadimitriou94], [Rajaraman96], [Najim96], and [Poznyak96].

3.2 The Environment and the Automaton

The learning paradigm the learning automaton presents may be stated as follows: a finite number of actions can be performed in a random environment. When a specific action is performed the environment provides a random response which is either favorable or unfavorable. The objective in the design of the automaton is to determine how the choice of the action at any stage should be guided by past actions and responses. The important point to note is that the decisions must be made with very little knowledge concerning the "nature" of the environment. It may have time-varying characteristics, or the decision maker may be a part of a hierarchical decision structure but unaware of its role in the hierarchy. Furthermore, the uncertainty may be due to the fact that the output of the environment is influenced by the actions of other agents unknown to the decision maker.

The environment in which the automaton "lives" responds to the action of the automaton by producing a response, belonging to a set of allowable responses, which is probabilistically related to the automaton action. The term *environment*[2] is not easy to define in the context of learning automata. The definition encompasses a large class of unknown random media in which an automaton can operate. Mathematically, an environment is represented by a triple $\{\underline{\alpha}, \underline{c}, \underline{\beta}\}$ where α represents a finite action/output set, β represents a (binary) input/response set, and $\underline{c}$ is a set of penalty probabilities, where each element c_i corresponds to one action α_i of the set $\underline{\alpha}$.

The output (action) $\alpha(n)$ of the automaton belongs to the set $\underline{\alpha}$, and is applied to the environment at time $t = n$. The input $\beta(n)$ from the environment is an element of the set $\underline{\beta}$ and can

[2] Commonly refers to the aggregate of all the external conditions and influences affecting the life and development of an organism.

take on one of the values β_1 and β_2. In the simplest case, the values β_i are chosen to be 0 and 1, where 1 is associated with failure/penalty response. The elements of $\underline{c}$ are defined as:

$$\text{Prob}\{\beta(n) = 1 \mid \alpha(n) = \alpha_i\} = c_i \qquad (i = 1,2,\ldots) \tag{3.1}$$

Therefore c_i is the probability that the action α_i will result in a penalty input from the environment. When the penalty probabilities c_i are constant, the environment is called a *stationary environment.*

There are several *models* defined by the response set of the environment. Models in which the input from the environment can take only one of two values, 0 or 1, are referred to as P-models[3]. In this simplest case, the response value of 1 corresponds to an "unfavorable" (failure, penalty) response, while output of 0 means the action is "favorable." A further generalization of the environment allows finite response sets with more than two elements that may take finite number of values in an interval [a, b]. Such models are called Q-models. When the input from the environment is a continuous random variable with possible values in an interval [a, b], the model is named S-model.

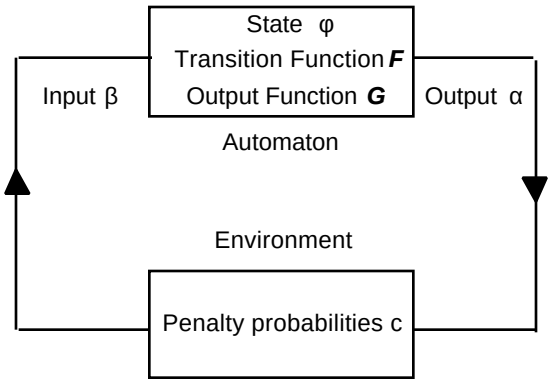


Figure 3.1. The automaton and the environment.

The automaton can be represented by a quintuple $\{\underline{\Phi}, \underline{\alpha}, \underline{\beta}, F(\bullet,\bullet), H(\bullet,\bullet)\}$ where:

- $\underline{\Phi}$ is a set of internal *state*s. At any instant *n*, the state $\varphi(n)$ is an element of the finite set $\underline{\Phi} = \{\varphi_1, \varphi_2, \ldots, \varphi_s\}$
- $\underline{\alpha}$ is a set of actions (or outputs of the automaton). The *output* or *action* of an automaton an the instant *n*, denoted by $\alpha(n)$, is an element of the finite set $\underline{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$
- $\underline{\beta}$ is a set of responses (or inputs from the environment). The input from the environment $\beta(n)$ is an element of the set $\underline{\beta}$ which could be either a finite set or an infinite set, such as an interval on the real line:

$$\underline{\beta} = \{\beta_1, \beta_2, \ldots, \beta_m\} \text{ or } \underline{\beta} = \{(a,b)\} \tag{3.2}$$

[3] We will follow the notation used in [Narendra89].

◆ $F(\bullet,\bullet)$: $\underline{\Phi} \times \underline{\beta} \rightarrow \underline{\Phi}$ is a function that maps the current state and input into the next state. F can be deterministic or stochastic:

$$\varphi(n+1) = F[\varphi(n), \beta(n)] \tag{3.3}$$

◆ $H(\bullet,\bullet)$: $\underline{\Phi} \times \underline{\beta} \rightarrow \underline{\alpha}$ is a function that maps the current state and input into the current output. If the current output depends on only the current state, the automaton is referred to as state-output automaton. In this case, the function $H(\bullet,\bullet)$ is replaced by an output function $G(\bullet)$: $\underline{\Phi} \rightarrow \underline{\alpha}$, which can be either deterministic or stochastic:

$$\alpha(n) = G[\varphi(n)] \tag{3.4}$$

For our applications, we choose the function $G(\bullet)$ as the identity function, *i.e.*, the states of the automata are also the actions.

3.3 The Stochastic Automaton

We now introduce the stochastic automaton in which at least one of the two mappings F and G is stochastic. If the transition function F is stochastic, the elements f_{ij}^{β} of F represent the probability that the automaton moves from state φ_i to state φ_j following an input β[4]:

$$f_{ij}^{\beta} = \Pr\{\varphi(n+1) = \varphi_j \,|\, \varphi(n) = \varphi_i, \beta(n) = \beta\} \qquad i, j = 1,2,\ldots, s \qquad \beta = \beta_1, \beta_2, \ldots, \beta_m \tag{3.5}$$

For the mapping G, the definition is similar:

$$g_{ij} = \Pr\{\alpha(n) = \alpha_j \,|\, \varphi(n) = \varphi_i\} \qquad i, j = 1,2,\ldots, r \tag{3.6}$$

Since f_{ij}^{β} are probabilities, they lie in the closed interval $[a, b]$[5]; and to conserve probability measure we must have:

$$\sum_{j=1}^{s} f_{ij}^{\beta} = 1 \qquad \text{for each } \beta \text{ and } i. \tag{3.7}$$

Example

States: φ_1, φ_2 Inputs: β_1, β_2

Transition matrices:

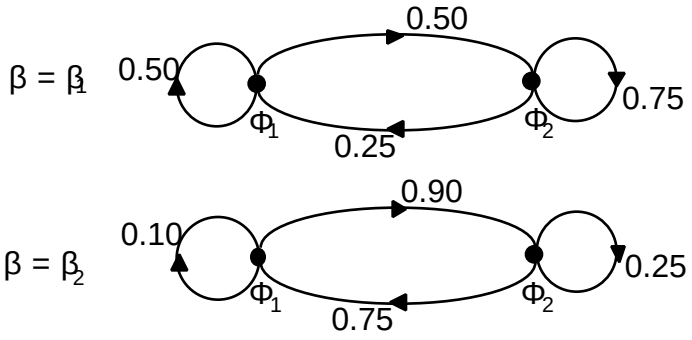
$$F(\beta_1) = \begin{bmatrix} 0.50 & 0.50 \\ 0.25 & 0.75 \end{bmatrix}$$

$$F(\beta_2) = \begin{bmatrix} 0.1 & 0.9 \\ 0.75 & 0.25 \end{bmatrix}$$

[4] In Chapter 4, where we define our initial algorithm, the probabilities f_{ij}^{β} will be the same for all inputs β and all states i given a state φ_j. Furthermore, the output mapping G is chosen as identity mapping.

[5] [a, b]=[0, 1] in most cases.

Transition Graphs:



In the previous example, the conditional probabilities f_{ij}^{β} were assumed to be constant, *i.e.*, independent of the time step n, and the input sequence. Such a stochastic automaton is referred to as a *fixed-structure automaton*. As we will discuss later, it is useful to update the transition probabilities at each step n on the basis of the environment response at that instant. Such an automaton is called *variable-structure automaton.*

Furthermore, in the case of variable-structure automaton, the above definitions of the transition functions F and G are not used explicitly. Instead of transition matrices, a vector of action probabilities $p(n)$ is defined to describe the reinforcement schemes — as we introduce in the next sections.

If the variable-structure automaton is a state-output automaton, and the probabilities of transition from one state to another f_{ij}^{β} do not depend on current state and environment input, then the relation between the action probability vector and the transition matrices is as follows:

· Since the automaton is a state-output automaton, we omit the transition matrix G, and only consider the transition matrix F.

· The transition does not depend on the environment response, therefore there is only one state transition matrix F where the elements are f_{ij}. For example, we must have:

$$F(\beta_1) = F(\beta_2) \equiv F = \begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{bmatrix} \tag{3.8}$$

· Furthermore, the probability of being in one state (or generating one action) does not depend on the initial/previous state of the automaton. Therefore, the transition matrix reduces to:

$$F = \left[f_{11} = f_{21} \equiv f_1 \quad f_{12} = f_{22} \equiv f_2 \right] \equiv p \tag{3.9}$$

where vector p consists of probabilities p_i of the automaton being in the i^{th} state (or choosing the i^{th} output/action).

3.4 Variable Structure Automaton and Its Performance Evaluation

Before introducing the variable structure automata, we will introduce the definitions necessary to evaluate the performance of a *learning* variable-structure automaton. A learning automaton generates a sequence of actions on the basis of its interaction with the environment. If the automaton is "learning" in the process, its performance must be superior to "intuitive" methods.

To judge the performance of the automaton, we need to set up quantitative norms of behavior. The quantitative basis for assessing the learning behavior is quite complex, even in the simplest P-model and stationary random environments. To introduce the definitions for "norms of behavior", we will consider this simplest case. Further definitions for other models and non-stationary environments will be given whenever necessary.

3.4.1 Norms of Behavior

If no prior information is available, there is no basis in which the different actions α_i can be distinguished. In such a case, all action probabilities would be equal — a "pure chance" situation. For an *r*-action automaton, the action probability vector $p(n) = Pr\{\alpha(n) = \alpha_i\}$ is given by:

$$p_i(n) = \frac{1}{r} \qquad i = 1,2,...,r \tag{3.10}$$

Such an automaton is called "pure chance automaton," and will be used as the standard for comparison. Any learning automaton must at least do better than a pure-chance automaton.

Consider a stationary random environment with penalty probabilities $\{c_1, c_2, ..., c_r\}$ where $c_i = \Pr\{\beta(n) = 1 \mid \alpha(n) = \alpha_i\}$. We define a quantity $M(n)$ as the average penalty for a given action probability vector:

$$\begin{aligned} M(n) &= E[\beta(n) \mid p(n)] = \Pr\{\beta(n) = 1 \mid p(n)\} \\ &= \sum_{i=1}^{r} \Pr\{\beta(n) = 1 \mid \alpha(n) = \alpha_i\} \ \Pr\{\alpha(n) = \alpha_i\} \\ &= \sum_{i=1}^{r} c_i p_i(n) \end{aligned} \tag{3.11}$$

For the pure-chance automaton, $M(n)$ is a constant denoted by M_o:

$$M_o = \frac{1}{r} \sum_{i=1}^{r} c_i \tag{3.12}$$

Also note that:

$$\begin{aligned} E[M(n)] &= E\{E[\beta(n) \mid p(n)]\} \\ &= E[\beta(n)] \end{aligned} \tag{3.13}$$

i.e., $E[M(n)]$ is the average input to the automaton. Using above definitions, we have the following:

Definition 3.1: A learning automaton is *expedient* if $\lim_{n \to \infty} E[M(n)] < M_o$.

Since $\sum_{i=1}^{r} p_i(n) = 1$, we can write $\inf M(n) = \inf_{p(n)} \{\sum_{i=1}^{r} c_i p_i(n)\} = \min_i \{c_i\} \equiv c_l$.

Definition 3.2: A learning automaton is said to be *optimal* if:

$$\lim_{n \to \infty} E[M(n)] = c_l \tag{3.14}$$

Optimality implies that the action α_l is associated with the minimum penalty probability c_l is chosen asymptotically with probability one. In spite of efforts of many researchers, the general algorithm which ensures optimality has not been found [Baba83, Kushner72, Narendra89]. It may not be possible to achieve optimality in every given situation. In this case, a suboptimal behavior is defined.

Definition 3.3: A learning automaton is *e-optimal* if:

$$\lim_{n \to \infty} E[M(n)] = c_l + \varepsilon \tag{3.15}$$

where e is arbitrarily small positive number.

Some automata satisfy the conditions stated in the definitions for specified initial conditions and for certain sets of penalty probabilities. However, we may be interested in automata that exhibit a desired behavior in arbitrary environments and for arbitrary initial conditions. These requirements are partially met by an absolutely expedient automaton, which is defined as:

Definition 3.4: A learning automaton is *absolutely expedient* if:

$$E[M(n+1) \mid p(n)] < M(n) \tag{3.16}$$

for all n, all $p_i(n)$ (0, 1) and for all possible sets $\{c_1, c_2, ..., c_r\}$. Taking the expectations of both sides, we can further see that:

$$E[M(n+1)] < E[M(n)] \tag{3.17}$$

In the application of learning automata techniques to intelligent control, we are mainly interested in the case where there is one "optimal" action α_j with $c_j = 0$[6]. In such an environment, a learning automaton is expected to reach a "pure optimal" strategy [Najim94]:

$$p(n) \rightarrow e_j^r \tag{3.18}$$

where e_j^r is the unit vector with j^{th} component equal to 1. In other words, the automaton will be optimal since $E[M(n)] = \sum_{i=1}^{r} c_i p_i(n) \rightarrow c_l$.

3.4.2 Variable Structure Automata

A more flexible learning automaton model can be created by considering more general stochastic systems in which the action probabilities (or the state transitions) are updated at every stage using a reinforcement scheme. In this section, we will introduce the general description of a reinforcement scheme, linear and nonlinear. For simplicity, we assume that each state

[6] We will discuss the feasibility of the case later.

corresponds to one action, *i.e.*, the automaton is a state-output automaton. The simulation examples in later chapters also use the same *identity* state-output mapping.

In general terms, a reinforcement scheme can be represented as follows:

$$p(n+1) = T_1[p(n), \alpha(n), \beta(n)] \tag{3.19a}$$

or

$$f^{\beta}_{ij}(n+1) = T_2[f^{\beta}_{ij}(n), \varphi(n), \varphi(n+1)\beta(n)] \tag{3.19b}$$

where T_1 and T_2 are mappings. Again, α is the action, β is the input from environment, and φ is the state of the automaton. From now on, we will use the first mathematical description (3.19a) for reinforcement schemes. If $p(n+1)$ is a linear function of $p(n)$, the reinforcement scheme is said to be linear; otherwise it is termed nonlinear.

The simplest case

Consider a variable-structure automaton with r actions in a stationary environment with $\underline{\beta} = \{0, 1\}$. The general scheme for updating action probabilities is as follows:

If $\alpha(n) = \alpha_i$

$$p_j(n+1) = p_j(n) - g_i(p(n)) \quad \text{for all } j \neq i$$

when $\beta = 0$

$$p_i(n+1) = p_i(n) + \sum_{\substack{k=1 \\ k \neq i}}^{r} g_k(p(n)) \tag{3.20}$$

$$p_j(n+1) = p_j(n) + h_i(p(n)) \quad \text{for all } j \neq i$$

when $\beta = 1$

$$p_i(n+1) = p_i(n) - \sum_{\substack{k=1 \\ k \neq i}}^{r} h_k(p(n))$$

where g_k and h_k (k=1,2,...,r) are continuous, nonnegative functions with the following assumptions:

$$0 < g_k(p(n)) < p_k(n)$$

$$0 < \sum_{\substack{k=1 \\ k \neq i}}^{r} [p_k(n) + h_k(p(n))] < 1 \tag{3.21}$$

for all i=1,2,...,r and all probabilities p_k in the open interval (0,1). The two constraints on update functions guarantees that the sum of all action probabilities is 1 at every time step.

3.5 Reinforcement Schemes

The reinforcement scheme is the basis of the learning process for learning automata. These schemes are categorized based on their linearity. A variety of linear, nonlinear and hybrid schemes exists for variable structure learning automata [Narendra89]. The linearity characteristic of a reinforcement scheme is defined by the linearity of the update functions g_k and h_k (see the

examples in the next section). It is also possible to divide all possible algorithms for stochastic learning automata into the following two classes [Najim94]:

- nonprojectional algorithms, where individual probabilities are updated based on their previous values:

$$p_i(n+1) = p_i(n) + \delta \; k_i\big(p_i(n), \alpha(n), \beta(n)\big) \tag{3.22}$$

  where δ is a small real number and k_i are the mapping functions for the update term.
- projectional algorithms, where the probabilities are updated by a function which maps the probability vector to a specific value for each element:

$$p_i(n+1) = k_i\big(p(n), \alpha(n), \beta(n)\big) \tag{3.23}$$

  where the functions k_i are the mapping functions.

The main difference between these two subclasses is that nonprojectional algorithms can only be used with binary environment response (*i.e.*, in a P-model environment). Projectional algorithms may be used in all environment types and are more complex. Furthermore, implementation of projectional algorithms are computationally more tasking.

Early studies of reinforcement schemes were centered around linear schemes for reasons of analytical simplicity. The need for more complex and efficient reinforcement schemes eventually lead researchers to nonlinear (and hybrid) schemes. We will first introduce several well-known linear schemes, and then close this section with absolutely expedient nonlinear schemes.

3.5.1 Linear Reinforcement Schemes

For an r-action learning automaton, the general definition of linear reinforcement schemes can be obtained by the substitution of the functions in Equation 3.20 as follows:

$$\begin{aligned} g_k(p(n)) &= a \;\; p_k(n) \\ h_k(p(n)) &= \frac{b}{r-1} - b \;\; p_k(n) \\ 0 &< a, b < 1 \end{aligned} \tag{3.24}$$

Therefore, the scheme corresponding to general linear schemes is given as:

If $\alpha(n) = \alpha_i$,

when $\beta = 0$

$$\begin{cases} p_j(n+1) = (1-a) \;\; p_j(n) & \text{for all } j \neq i \\ p_i(n+1) = p_i(n) + a \;\; [1 - p_i(n)] \end{cases}$$

when $\beta = 1$

$$\begin{cases} p_j(n+1) = \dfrac{b}{r-1} + (1-b) \;\; p_j(n) & \text{for all } j \neq i \\ p_i(n+1) = (1-b) \;\; p_i(n) \end{cases} \tag{3.25}$$

As seen from the definition, the parameter a is associated with reward response, and the parameter b with penalty response. If the learning parameters a and b are equal, the scheme is

called the linear reward-penalty scheme L_{R-P} [Bush58]. In this case, the update rate of the probability vector is the same at every time step, regardless of the environment response. This scheme is the earliest scheme considered in mathematical psychology.

For the linear reward-penalty scheme L_{R-P}, $E[p(n)]$, the expected value of the probability vector at time step n, can be easily evaluated, and, by analyzing eigenvalues of the resulting difference equation, it can be shown that asymptotic solution of the set of difference equations enables us to conclude [Narendra89]:

$$\lim_{n \to \infty} E[M(n)] = \frac{r}{\sum_{k=1}^{r} 1/c_k} < \frac{\sum_{k=1}^{r} c_k}{r} = M_o \tag{3.26}$$

Therefore, from Definition 3.1, the multi-action automaton using the L_{R-P} scheme is expedient for all initial action probabilities and in all stationary random environments.

Expediency is a relatively weak condition on the learning behavior of a variable-structure automaton. An expedient automaton will do better than a pure chance automaton, but it is not guaranteed to reach the optimal solution. In order to obtain a better learning mechanism, the parameters of the linear reinforcement scheme are changed as follows: if the learning parameter b is set to 0, then the scheme is named the linear reward-inaction scheme L_{R-I}. This means that the action probabilities are updated in the case of a reward response from the environment, but no penalties are assessed.

For this scheme, it is possible to show that $p_l(n)$, the probability of the action α_l with minimum penalty probability c_l, monotonically approaches 1. By making the parameter a arbitrarily small, we have $\Pr\left\{\lim_{n \to \infty} p_l(n) = 1\right\}$ as close to unity as desired. This makes the learning automata ε -optimal [Narendra89].

3.5.2 Nonlinear Learning Algorithms: Absolutely Expedient Schemes

Although early studies of automata learning were done mostly on linear schemes, a few attempts were made to study nonlinear schemes [Vor65, Chand68, Shapiro69, Vis72]. Evolution of early reinforcement schemes occurred mostly in a heuristic manner. For two-action automata, design and evaluation of schemes was almost straight forward, since actually only one action was being updated. Generalization of such schemes to multi-action case was not straightforward [Narendra89]. This problem led to a synthesis approach toward reinforcement schemes. Researchers started asking the question: “What are the conditions on the updating functions that guarantee a desired behavior?” The problem viewed in the light of this approach resulted in a new concept of absolute expediency. While searching for a reinforcement scheme that guarantees optimality and/or absolute expediency, researchers eventually started considering schemes that can be characterized as nonlinear. The update functions g_k and h_k are nonlinear functions of

penalty probabilities. For example, the following update functions are defined by previous researchers for nonlinear reinforcement schemes[7]:

- $g_j(p(n)) = h_j(p(n)) = \frac{a}{r-1} p_i(n)(1 - p_i(n))$

  This scheme is absolutely expedient for restricted initial conditions and for restricted environments [Shapiro69].

- $\begin{cases} g_j(p(n)) = p_j(n) - \varphi(p(n)) \\ h_j(p(n)) = \dfrac{p_i(n) - \varphi(p_i(n))}{r-1} \end{cases}$

  where $\varphi(x) = ax^m (m = 2,3,\ldots)$. Again, for expediency several conditions on the penalty probabilities must be satisfied (*e.g.*, $c_l < \frac{1}{m}$ and $c_j > \frac{1}{m}$ for $j \neq l$). [Chand68]

- $\begin{cases} g_j(p(n)) = a\varphi(p_1(n),1-p_1(n))p_j^{\theta+1}(1-p_j(n))^{\theta} \\ h_j(p(n)) = b\varphi(p_1(n),1-p_1(n))p_j^{\theta+1}(1-p_j(n))^{\theta} \end{cases}$

  where $\varphi(p_1,1-p_1) = \varphi(1-p_1,p_1)$ is a nonlinear function which can be suitably selected, and $\theta \geq 1$. Parameters *a* and *b* must be chosen properly to satisfy the conditions in Equation 3.21. Sufficient conditions for ensuring optimality is given in [Vor65], but they are valid only for two-action automata.

The general solution for absolutely expedient schemes is found in early 1970's by Lakshmivarahan and Thathachar [Lakshmivarahan73]. Absolutely expedient learning schemes are presently the only class of schemes for which necessary *and* sufficient conditions of design are available. This class of schemes can be considered as the generalization of the L_{R-I} scheme. Consider the general reinforcement scheme in Equation 3.20. A learning automaton using this scheme is absolutely expedient if and only if the functions $g_k(p)$ and $h_k(p)$ satisfy the following conditions:

$$\begin{aligned} \frac{g_1(\underline{p})}{p_1} &= \frac{g_2(\underline{p})}{p_2} = \quad \frac{g_r(\underline{p})}{p_r} = \lambda(\underline{p}) \\ \frac{h_1(\underline{p})}{p_1} &= \frac{h_2(\underline{p})}{p_2} = \quad = \frac{h_r(\underline{p})}{p_r} = \mu(\underline{p}) \end{aligned} \tag{3.27}$$

where $\lambda(\underline{p})$ and $\mu(\underline{p})$ are arbitrary functions satisfying[8]:

[7] Again, $\alpha(n) = \alpha_i$ is assumed. We give only the updating functions for action probabilities other than the current action; the function for current action can be obtained by using the fact that the sum of probabilities is equal to 1.

[8] The reason for the conditions on the update functions will be explained in detail in Chapter 6.

$$0 < \lambda(\underline{p}) < 1$$
$$0 < \mu(\underline{p}) < \min_{j=1,\ldots,r} \left[\frac{p_j}{1-p_j} \right] \tag{3.28}$$

Detailed proof of this theorem is given in both [Baba84] and [Narendra89]. A similar proof (for a new nonlinear absolutely expedient scheme) is given in Chapter 6.

3.6 Extensions of the Basic Automata-Environment Model

3.6.1 S-Model Environments

Up to this point we based our discussion of learning automata only on the P-model environment, where the input to the automaton is either 0 or 1. We now consider the possibility of input values over the unit interval [*a, b*]. Narendra and Thathachar state that such a model is very relevant in control systems with a continuous valued performance index. In previous sections, the algorithms (for P-model) were stated in terms of the functions g_k and h_k which are the reward and penalty functions respectively. Since in S- (and Q-) models the outputs lie in the interval [0,1], and therefore are neither totally favorable or totally unfavorable, the main problem is to determine how the probabilities of all actions are to be updated. Narendra and Thathachar [Narendra89] show that all the principal results derived for the P-model carry over the more realistic S- and Q-models also. Here, we will not discuss the Q-model; all definitions for the Q-model are similar to the P-model.

In the case where the environment outputs β_k are not in the interval [0,1], but in [*a, b*] for some *a* ,*b* IR, it is always possible to map the output into the unit interval using:

$$\bar{\beta}_k = \frac{\beta_k - a}{b - a} \tag{3.29}$$

where $a = min_i\{\beta_i\}$ and $b = \max_i\{\beta_i\}$[9]. Therefore, only the normalized S-model will be considered.

The average penalty $M(n)$ is still defined as earlier, but instead of c_i {0,1}, s_i [0,1] are used in the definition:

$$\begin{aligned} M(n) &= E[\beta(n) \mid p(n)] \\ &= \sum_{k=1}^{r} E[\beta(n) \mid p(n), \alpha(n) = \alpha_k] \Pr[\alpha(n) = \alpha_k] \\ &= \sum_{k=1}^{r} s_k p_k \end{aligned} \tag{3.30}$$

where:

$$s_k = E[\beta(n) \mid \alpha(n) = \alpha_k] \tag{3.31}$$

[9] Note that this normalization procedure requires the knowledge of *a* and *b*.

In the S-model, s_k plays the same role as the penalty probabilities in the *P*-model, and are called *penalty strengths*. Again, for a pure-chance automaton with all the action probabilities equal, the average penalty M_o is given by:

$$M_o = \frac{1}{r}\sum_{i=1}^{r} s_i \tag{3.32}$$

The definitions of expediency, optimality, ε -optimality, and absolute expediency are the same, except the fact that c_i's are replaced by s_i's. Now, consider the general reinforcement scheme (Equation 3.20). The direct generalization of this scheme to S-model gives:

$$\begin{aligned} p_i(n+1) &= p_i(n) - (1-\beta(n))g_i(p(n)) + \beta(n)h_i(p(n)) \qquad if\ \alpha(n) \neq \alpha_i \\ p_i(n+1) &= p_i(n) + (1-\beta(n))\sum_{j\neq i} g_j(p(n)) - \beta(n)p_i\sum_{j\neq i} h_j(p(n)) \quad if\ \alpha(n) = \alpha_i \end{aligned} \tag{3.33}$$

The previous algorithm (Equation 3.20) can easily be obtained from the above definition by substituting $\beta(n) = 0$ and 1. Again, the functions g_i and h_i can be associated with reward and penalty respectively. The value $1-\beta(n)$ is an indication of how far $\beta(n)$ is away from the maximum possible value of unity, and is maximum when $\beta(n)$ is 0. The functions g_i and h_i have to satisfy the conditions for Equation 3.21 for $p_i(n)$ to remain in the interval [0,1] at each step. The non-negativity condition on g_i and h_i maintains the reward-penalty character of these functions, though it is not necessary.

For example the L_{R-P} reinforcement scheme (Equation 3.25) can be rewritten for S-model (SL_{R-P}) as:

$$\begin{aligned} p_i(n+1) &= p_i(n) - [1-\beta(n)]\ a\ \ p_i(n) + \beta(n)\ [\frac{a}{r-1} - a\ \ p(n)] \quad if\ \alpha(n) \neq \alpha_i \\ p_i(n+1) &= p_i(n) + [1-\beta(n)]\ a\ \ [1-p_i(n)] - \beta(n)\ a\ \ p_i(n) \qquad if\ \alpha(n) = \alpha_i \end{aligned} \tag{3.35}$$

Similarly, the asymptotic value of the average penalty is:

$$\lim_{n\to\infty} E[M(n)] = \frac{r}{\sum_{k=1}^{r} 1/s_i} \tag{3.36}$$

Thus, the SL_{R-P} scheme has analogous properties to the L_{R-P} scheme. It is expedient in all stationary random environments and the limiting expectations of the action probabilities are inversely proportional to the corresponding s_i. The conditions of Equations 3.27 and 3.28 for absolute expediency also applies to updating algorithms for S-model environment.

3.6.2 Nonstationary Environments

Thus far we have considered only the automaton in a stationary environment. However, the need for learning and adaptation in systems is mainly due to the fact that the environment changes

with time. The performance of a learning automaton should be judged in such a context. If a learning automaton with a fixed strategy is used in a nonstationary environment, it may become less expedient, and even non-expedient. The learning scheme must have sufficient flexibility to track the better actions. The aim in these cases is not to evolve to a single action which is optimal, but to choose the actions to minimize the expected penalty. As in adaptive control theory, the practical justification for the study of stationary environments is based on the assumption that if the convergence of a scheme is sufficiently fast, then acceptable performance can be achieved in slowly changing environments.

An environment is nonstationary if the penalty probabilities c_i (or s_i) vary with time. The success of the learning procedure then depends on the environment changes as well as the prior information we have about the environment. From an analytic standpoint, the simplest case is the one in which penalty probability vector $\underline{c}(n)$ can have a finite number of values. We can visualize this situation as an automaton operating in a finite set of stationary environments, and assign a different automaton A_i to each environment E_i. If each automaton uses an absolutely expedient scheme, then ε -optimality can be achieved asymptotically for the overall system. However, this method requires that the system be aware of the "environmental changes" in order to assign the "correct" automaton, and the number of environments must not be large. Furthermore, since the automaton A_i is updated only when the environment is E_i, updating strategies may happen infrequently so that the corresponding action probabilities do not converge [Narendra89]. In our application, a similar approach is employed as explained in Chapter 4.

There are two classes of environments that have been analytically investigated: Markovian switching environments, and state-dependent nonstationary environments. If we assume that the environments E_i are states of a Markov chain described by a state transition matrix, then it is possible to view the variable-structure automaton in the Markovian switching environment as a homogeneous Markov chain. Therefore, analytical investigations are possible. Secondly, there are four different situations wherein the states of the nonstationary environment vary with the step n, and analytical tools developed earlier are adequate to obtain a fairly complete descriptions of the learning process. These are:

- Fixed-structure automata in Markovian environments
- Variable-structure automata in state-dependent nonstationary environments
- Hierarchy of automata
- Nonstationary environments with fixed optimal actions

The fundamental concepts such as expediency, optimality, and absolute expediency have to be re-examined for nonstationary environments, since, for example, the optimal action can change with time. Here, we again consider the P-model for purposes of stating the definitions. The average penalty $M(n)$ becomes:

$$M(n) = \sum_{i=1}^{r} c_i(n)\, p_i(n) \tag{3.37}$$

The pure-chance automaton has an average penalty M_o as a function of time:

$$M_o(n) = \frac{1}{r}\sum_{i=1}^{r} c_i(n) \tag{3.38}$$

Using these descriptions, we now have the following:

Definition 3.5: A learning automaton in nonstationary environment is *expedient* if there exists a n_o such that for all $n > n_o$, we have:

$$E[M(n)] - M_o(n) < 0 \tag{3.39}$$

The previous concept of optimality can be applied to some specific nonstationary environments. For example, if there exists a fixed l such that $c_l(n) < c_i(n)$ for all $i = 1,2,...,r$, $i \neq l$, and for all n (or at least for all $n > n_1$ for some n_1), then α_l is the optimal action; the original definitions of optimality and ε-optimality still hold. In general, we have:

Definition 3.6: In a nonstationary environment where there is no fixed optimal action, an automaton can be defined as *optimal* if it minimizes:

$$\lim_{T\to\infty} \frac{1}{T}\sum_{n=1}^{T} E[\beta(n)] \tag{3.40}$$

The definition of absolute expediency can be retained as before, with time-varying penalty probabilities and the new description in Equation 3.37.

3.6.3 Multi-Teacher Environments

All the learning schemes and behavior norms discussed up to this point have dealt with a single automaton in a single environment. However, it is possible to have an environment in which the actions of an automaton evoke a vector of responses due to multiple performance criteria. We will describe such environments as multi-teacher environments. Then, the automata have to "find" an action that satisfies all the teachers.

In a multi-teacher environment, the automaton is connected to N teachers (or N single-teacher environments). Then, a single-teacher environment is described by the action set $\underline{\alpha}$, the output set $\underline{\beta}^i = \{0,1\}$ (for a P-model), and a set of penalty probabilities $\{c_1^i, c_2^i, \ldots, c_r^i\}$ where $c_j^i = \Pr\{\beta^i(n) = 1 \mid \alpha(n) = \alpha_j\}$. The action set of the automaton is of course the same for all teachers/environments. Baba discussed the problem of a variable-structure automaton operating in a multi-teacher environment [Baba85]. Conditions for absolute expediency are given in his works (see also Chapter 6 for new nonlinear reinforcement schemes).

Some difficulty may arise while formulating a mathematical model of the learning automaton in a multi-teacher environment. Since we have multiple responses, the task of "interpreting" the output vector is important. Are the outputs from different teachers to be summed after normalization? Can we introduce weight factors associated with specific teachers?

If the environment is of a *P*-model, how should we combine the outputs? In the simplest case that all teachers agree on the ordering of the actions, *i.e.*, one action is optimal for all environments, the updating schemes are easily defined. However, this will not be the case in our application.

The elements of the environment output vector must be combined in some fashion to form the input to the automaton. One possibility is to use to use an AND-gate as described in [Thathachar77]. Another method is taking the average, or weighted average of the all responses [Narendra89, Baba85]. This is valid for Q- or S-model environments. Our application uses logic gates (OR) and additional *if-then* conditions as described in Chapters 4 and 5.

3.6.4 Interconnected Automata

Although we based our discussion to a single automaton, it is possible that there are more than one automata in an environment. If the interaction between different automata is provided by the environment, the case of multi-automata is not different than a single automaton case. The environment reacts to the actions of multiple automata, and the environment output is a result of the combined effect of actions chosen by all automata. If there is direct interaction between the automata, such as the hierarchical (or sequential) automata models, the actions of some automata directly depend on the actions of others.

It is generally recognized that the potential of learning automata can be increased if specific rules for interconnections can be established. There are two main interconnected automata models: synchronous and sequential. An important consequence of the synchronous model is that the resulting configurations can be viewed as games of automata with particular payoff structures. In the sequential model, only one automaton acts at each time step. The action chosen determines which automaton acts at the next time step. Such a sequential model can be viewed as a network of automata in which control passes from one automaton to another. At each step, one decision maker controls the state transitions of the decision process, and the goal of the whole group is to optimize some overall performance criterion.

In Chapters 4 and 5, the interconnection between automata is considered. Since each vehicle's planning layer will include two automata — one for lateral, the other for longitudinal actions — the interdependence of these two sets of actions automatically results in an interconnected automata network. Furthermore, multiple autonomous vehicles with multiple automata as path controllers constitute a more complex automata system as described in Chapter 7.