Relative Reward Strength Algorithms for Learning Automata [1]

Rahul Simha & James F. Kurose

Department of Computer and Information Science
University of Massachusetts
Amherst, Mass. 01003

Abstract

We examine a new class of action probability update algorithms for learning automata that use the *relative* reward strengths of responses from the environment. Specifically, we study update algorithms for S-Model automata in which "recent" environmental responses for each of the actions are retained and used. We prove a convergence result and study the behavior of these automata through simulation. A major result of the paper is that the performance of these algorithms is superior, in several respects, to that of the well-known SL_{R-I} update algorithm. Additional results are presented on the variability of performance, the cost of learning and, in the case of static environments, modifications that result in improved convergence.

[1] This work is supported in part by the Office of Naval Research grant N00014-87-K-0304 and NSF equipment grant CER DCR 8500332

1 Introduction

Learning automata have been the subject of intense research during the past two decades [3, 14, 17, 27]. Their inherent simplicity and mathematical properties have resulted in wide applicability to various problems [7, 14, 18, 20, 24]. The environments in which learning automata are designed to operate are typically stochastic in nature, i.e., the environmental response or "reward" to a given automaton action may be considered as a random sample drawn from an associated (typically stationary) probability distribution. Broadly speaking, the goal of a learning automaton is to iteratively adjust ("optimize"), through the means of an update algorithm, its action probabilities (defined precisely below) and thus identify that action with the largest expected reward.

Traditionally, learning automata have used only the single, most recent response from the environment in updating the action probabilities [14, 20, 29]. However, it has been noted by several researchers [5, 8, 26] that the history of interaction with the environment (i.e. previous responses from the environment) may be used in designing more effective action probability update algorithms. These automata have typically used the *entire* past history of the automaton to maintain consistent *estimates* of the mean reward associated with each action; these estimates become increasingly accurate and hence the corresponding update algorithms converge faster than several previous mechanisms. However, as pointed out in [26], these automata (at least implicitly) assume a *static* environment and hence are unsuitable for tracking a non-stationary setting. This may be compensated for by the use of partial histories [26] but in this case the speed and provability of convergence may be not be guaranteed.

In this paper, we examine several update algorithms for automata that use information from previous environmental responses without losing any tracking ability. We present an S-Model automaton in which the automaton maintains and utilizes the *most recently obtained* reward value for *each* action until the next time that action is selected. A second novel feature of the automata we study is the manner in which these recently obtained reward values are used. Specifically, we study algorithms based on a gradient projection method which update action probabilities on the basis of *relative* reward strengths. We prove the convergence of these algorithms and study their performance through simulation. A major result of the paper is that the performance of these algorithms is superior, in many respects, to that of the well-known SL_{R-I} update algorithm [14, 20, 29]. In addition to providing empirical results on the speed of convergence, we investigate several other important issues such as the variance in performance and learning "cost" of update algorithms. Furthermore, in the case that a static environment may be assumed, we develop simple extensions of these relative reward algorithms which use the entire history of past responses and we demonstrate that the performance of one of the algorithms presented in this paper is superior to a recently proposed algorithm [26].

In the next section we define our model of learning automata and environments. Section 3 contains a discussion of relative reward strengths and a presentation of the algorithms studied in this paper. In addition, we derive a convergence result for one of the algorithms. Following that, in section 4 we describe our experimental results. Section 5 contains our conclusions and final remarks.

2 S-Model Learning Automata

In this section we define our model of the S-Model automaton considered in this paper. To place our work in the appropriate context, we present the standard definition (found in [3] and later in [14, 17, 30]) of a learning automaton and then discuss how the automata studied in this paper may be defined in these terms.

A learning automaton is a sextuple $M =< \alpha, \phi, \mathbf{P}, G, \sigma, A >$ where α is the set of K possible actions, ϕ is the set of internal states and $\mathbf{P}$ is the probability distribution over the set of states. G is the output function, $G : \phi \rightarrow \alpha$, and in this paper, as in several others [3, 14, 17, 19], G is a deterministic one-to-one function; thus, $\mathbf{P}$ is also the probability distribution over the set of actions. We denote $P_i^{(n)}$ as the probability of selecting the i^{th} action at step n. Next, σ is the set from which the input to the automaton is taken, i.e. the set of possible responses from the environment. In this paper, we are concerned with S-Model automata and thus [16, 29], $\sigma = [0, 1]$. Finally, A is the updating scheme that modifies the probability vector $(P_1^{(n)}, ..., P_K^{(n)})$ at each iteration n by using the response from the environment; this is typically executed after each response from the environment.

In this paper, we augment the state space with a *most-recent reward vector*, $\mathbf{s}$, and thus we may write the above definition as the septuple $< \alpha, \phi, \mathbf{P}, G, \sigma, A, \mathbf{s} >$; as with the probability vector, we use the notation $s_i^{(n)}$ to denote the most recent response for the i^{th} action at step n. Note that $s_i^{(n)}, 1 \leq i \leq K$, may also be considered as the input information to the update algorithm (defined below) at step n; in this case the input information is the most recent response obtained for action i. This is explained through an example in the following section. We note that, later, in section 4, $s_i^{(n)}$ is computed from several responses before being used in an update mechanism.

As with other S-Model automata [20, 29], for each action i, the environment responds with a reward which is taken from a continuous distribution F_i with mean $E[s_i]$ and thus, each response may be considered to be a *random sample* from the distribution F_i. In this case, the "best" action is that for which $E[s_i]$ is largest and, broadly speaking, the goal of the automaton is to incrementally adjust its action probabilities so that this action is most frequently taken. We note that in other, equivalent, formulations of the problem [3, 14, 17], the environment responds with a penalty, in which case the goal of the automaton is to modify the action probabilities so that the action that incurs the least penalty is most frequently selected. Finally, a word about the initial conditions:

the initial probability distribution is taken to be uniform and it is assumed that the initial reward vector contains a sample for each action;

3 Relative Reward Strength Algorithms

3.1 Using Old Information

At each step, the automaton can select only one of its actions (chosen randomly according to the probability distribution $(P_1^{(n)}, ..., P_K^{(n)})$), to which it will receive exactly one reward value from the environment. Thus, an update mechanism that utilizes reward values for several actions necessarily uses previously-obtained reward values. As previously discussed, the notation $s_i^{(n)}$ indicates the most recent response value on action i as of step n. For example, suppose that at the 10^{th} step, the automaton uses the 4^{th} action and obtains the reward 0.6. Next, suppose the 7^{th} action is used at the 11^{th} step and a reward of 0.3 is received. Then $s_4^{(10)} = s_4^{(11)} = 0.6$ and $s_7^{(11)} = 0.3$. Thus if the action chosen at the n^{th} step was the i^{th} action and if the corresponding response is denoted by r, then, by our notation, $s_i^{(n)} = r$.

As noted in [26], the standard update algorithms use only the value r in the update algorithm [3, 20, 29]. A high value of r causes $P_i^{(n+1)}$ to be higher than $P_i^{(n)}$ while decreasing uniformly the probabilities of the other actions. We observe that no information about the *relative* success of the actions taken in the past is taken into account. We are thus motivated to consider the case in which the relative reward strengths of all actions are used in updating the probabilities, i.e., the entire vector $\mathbf{s}$ (containing the most recent reward values received as a result of each of the K actions) is used in the update mechanism. Intuitively, we increase action probabilities *in proportion* to the relative size of the rewards last incurred.

In this paper, we only discuss previous work that is directly related to the ideas presented here; for a general survey, we refer the reader to other sources [3, 14, 17, 19]. The notion of employing the history of environmental responses has been used in [5, 25, 26]. As previously mentioned, these automata have typically used the *entire* past history of the automaton to maintain *estimates* of the mean (and variance [5]) of the reward distribution associated with each action. These estimates converge almost surely to the mean reward values and hence these algorithms exihibit faster convergence to the optimum vector of action probabilities than earlier mechanisms such as the SL_{R-I} algorithm of [29]. However, as noted in [26], these automata (at least implicitly) assume a *static* environment and cannot track a non-stationary environment. The use of a partial past history was addressed in [26], but only briefly and no convergence proofs were obtained for this case.

Relative reward strengths were also studied in [5, 25, 26]. In [25, 26] the difference between the estimated mean rewards for each action was considered in updating the action probabilities. In [5], the relative magnitudes of the estimated mean reward were considered, but as in [25, 26], these estimates were obtained over the entire history of the automaton and convergence followed primarily from the consistency of the reward estimates via the Strong Law of Large Numbers. The update algorithms studied in this paper also use the magnitude of the relative reward strengths, but in a different manner than in [5, 25, 26] (the algorithms in this paper are based on a gradient projection method) and use only a partial history (most recent rewards) as well. Since only the most recent rewards are used as estimates of the relative reward strengths, the algorithms are also well-suited for tracking a non-stationary environment. In section 4, we also consider the case of a static environment and examine extensions of the relative reward algorithms in which the entire history of interaction with the environment can be used.

3.2 Update Algorithms

At each step, n, the automaton randomly selects an action (according to the current action probabilities, $P_1^{(n)}, ..., P_K^{(n)}$), obtains the environment's random response to that action and executes the update algorithm. The update algorithm modifies the action probabilities and thus, we may write the action probability vector at step $(n+1)$ in terms of the value at step n as: $\forall i : P_i^{(n+1)} = P_i^{(n)} + a_n \Delta P_i^{(n)}$, where a_n is the stepsize or learning parameter. Therefore, an update algorithm may be specified by an expression for each $\Delta P_i^{(n)}$. We note that in choosing expressions for $\Delta P_i^{(n)}$, care must taken to ensure that the feasibility constraints, $\sum_{i=1}^{K} P_i^{(n)} = 1$ and $0 < P_i^{(n)} < 1$, are not violated. In the algorithms presented here, we have $\sum_{i=1}^{K} \Delta P_i^{(n)} = 0$ in order to satisfy the first constraint, $\sum_{i=1}^{K} P_i^{(n)} = 1$. For the latter constraints, $0 < P_i^{(n)} < 1$, we define the feasibility sets $A_1^{(n)}, A_2^{(n)}$ and $A_2'^{(n)}$ below; in optimization terminology, these sets correspond to the identification of the active and inactive inequality constraints [23]. We now present our algorithms by expressing $\Delta P_i^{(n)}$ as a function of the reward vector $s_i^{(n)}, ..., s_K^{(n)}$.

The first relative reward update algorithm we study (G_1) is derived directly from the (deterministic) gradient method of [9]. In this mechanism the probability of the action with the highest reward is increased at the expense of reducing the probabilities of the other actions. Thus if action m currently has the highest reward, i.e. $s_m^{(n)} > s_i^{(n)}, \forall i \neq m$, then we may write the change in probability for the other actions as:

Algorithm G_1:

$$\Delta P_i^{(n)} := \left(s_i^{(n)} - s_m^{(n)} \right), \qquad \forall i : i \in A_1^{(n)}, i \neq m \tag{1}$$

The corresponding change for probability P_m is computed as:

$$\Delta P_m^{(n)} := -\left(\sum_{i \in A_1^{(n)}, i \neq m} \Delta P_i^{(n)} \right)$$

whereas $\forall i \notin A_1^{(n)}$, $\Delta P_i^{(n)} = 0$. We construct the set $A_1^{(n)}$ at each iteration in order to ensure that the feasibility constraints, $0 < P_i^{(n)} < 1$, are maintained. The feasibility set $A_1^{(n)}$ for G_1 is computed as $A_1^{(n)} = \{i | P_i^{(n)} + a_n \left(s_i^{(n)} - s_m^{(n)} \right) > q_{min}\}$. The quantities q_{min}, a small positive number close to zero, and a_n, the stepsize, are discussed in below in the section on convergence.

Informally, the action probability is increased for the currently observed highest-reward action at the expense of the probabilities of the other actions. We note that, this observation is noisy and thus, the action probabilities for sub-optimal actions may frequently be increased while simultaneously lowering the probability of the action with the highest mean reward. We are, therefore, also motivated to consider alternative update mechanisms in which the probabilities are updated more cautiously. In the next algorithm that we present, there are two groups of actions, one in which the probabilities are increased and the other in which the probabilities are decreased; the two groups are implicitly created by the computation of the *average* reward at each iteration.

We now present the second relative reward update mechanism, an extension of a normative model of resource allocation initially proposed in a microeconomic context [10] and later adopted as the basis for resource allocation algorithms [11, 12]. At each iteration n, the change in probability for action i, $\Delta P_i^{(n)}$, is proportional to the difference between the reward strength, $s_i^{(n)}$, and the *average* reward strength over all actions. As before, let the action with the currently highest reward be denoted by m, i.e. $s_m^{(n)} > s_i^{(n)}$. The change in probability for each action i, $\Delta P_i^{(n)}$, is computed according to the following steps:

Algorithm G_2:

Step 1:

$$\begin{aligned} \Delta P_i^{(n)} &:= \left(s_i^{(n)} - \frac{1}{\left| A_2^{(n)} \right|} \sum_{j \in A_2^{(n)}} s_j^{(n)} \right), \qquad \forall i \in A_2^{(n)} \qquad (2) \\ &:= 0, \qquad i \notin A_2^{(n)} \end{aligned}$$

Step 2:

$$\Delta P_i^{(n)} := \left(s_i^{(n)} - s_m^{(n)} \right), \qquad \forall i \in A_2'^{(n)}$$

$$\Delta P_m^{(n)} := \Delta P_m^{(n)} + \sum_{i \in A_2'^{(n)}} \left(s_m^{(n)} - s_i^{(n)} \right) \quad (3)$$

where a_n is the stepsize parameter at step n and is discussed in detail below. Similar to algorithm G_1, the sets $A_2^{(n)}$ and $A_2'^{(n)}$ of actions are required to ensure that the feasibility constraints of the action probabilities are maintained and therefore, $\forall i \notin A_2^{(n)} \cup A_2'^{(n)}$, $\Delta P_i^{(n)} = 0$. We note that the set algorithm for $A_2^{(n)}$ originally given in [10] is incorrect in certain situations and has since been modified to maintain feasibility [12]. It is computed as follows:

/* Algorithm for computing the set $A_2^{(n)}$ */

(i) For all i, sort $s_i^{(n)}$.

(ii) Set $D = \{m|$ action m has the largest reward: $s_m^{(n)} = \max_i s_i^{(n)}\}$.

(iii) Do step (iv) for each $j, j \notin D$ in descending order of $s_j^{(n)}$.

(iv) If action j would receive a probability $P_j^{(n)} > q_{min}$, as a result of the reallocation defined by equation 2 above with $A_2^{(n)} = D \cup \{j\}$, then set $D = D \cup j$. The use of q_{min} is discussed below in the section on convergence.

(v) Set $A_2^{(n)} = D$

whereas $A_2'^{(n)} = \{i | i \notin A_2^{(n)}, i \neq m, P_i^{(n)} + a_n \left(s_i^{(n)} - s_m^{(n)} \right) > q_{min}\}$.

We offer the following intuitive explanation of the algorithm. In step 1, actions in $A_2^{(n)}$ that have incurred a higher than average reward have their action probabilities increased, whereas actions in $A_2^{(n)}$ that received rewards below the average reward over all actions have their probabilities decreased; as mentioned earlier this mechanism is intended to favour those actions receiving relatively higher rewards. It is possible to show that the algorithm G_2 without step 2 (described in the context of resource allocation in [12]) converges to the optimum vector of action probabilities. However, we have observed that it is possible to refine G_2 in order to improve performance while maintaining provability of convergence. This refinement (in step 2) consists of a reallocation of probability among those actions not in set $A_2^{(n)}$ together with the action with the currently highest reward. We note that step 2 applies algorithm G_1 over the feasibility set $A_2'^{(n)}$.

The above update mechanisms are based on gradient projection methods [2, 12, 23] (see [12] for details). In the case of linear constraints, these algorithms have been used to achieve optimal resource allocation in distributed computer systems [9, 11, 12, 28]. They have been demonstrated to possess desirable properties such as *feasibility* (always maintaining a feasible allocation) and *monotonicity* (an increase in the value of the objective function with each iteration, in the case of maximization) in addition to being amenable to decentralization and easy implementation. It is important to note that the gradient-based approaches above [2, 9, 11, 12] have been used in deterministic contexts. In this paper, the gradient-based update algorithms use noisy or random estimates (rewards) from the environment in the update process.

We note that similar gradient-based approaches have also been used in the analysis and construction of algorithms in connectionist systems [30, 31]. The emphasis in [30, 31] has been on demonstrating that several established update algorithms [1, 14, 16, 17] cause the action probability vector to move, on an average, in the direction of the gradient of the performance metric. In contrast, we note that our work is based on a *constrained* optimization formulation. Therefore, the update algorithms we describe here cause the action probability vector to move, on an average, in the direction of the *projection* of the gradient onto the constraint space, $\sum_{i=1}^{K} P_i = 1$.

We emphasize that the rewards are random samples from a distribution. Thus the algorithms only serve as a guide to improving the probability distribution of the actions. This gives rise to the question of whether the algorithms are in some sense 'optimal'. In the following section we show that under suitable, commonly assumed, restrictions on the stepsizes a_n, algorithms G_2 and G_1 converge to the desired optimum vector of action probabilities.

3.3 Convergence

In the above model, the environment responds to each action i with a random sample from distribution F_i with mean $E[s_i]$. We assume that these mean rewards are real numbers in $[0, 1]$. We define the expected reward at step n:

$$E[R_n] = \sum_{i=1}^{K} E[s_i] P_i^{(n)} \tag{4}$$

The objective of the automaton then is to maximize the expected reward by converging to an optimal distribution of action probabilities. We now show that with an appropriate choice of stepsize constants a_n (commonly made by several researchers [5, 13, 14, 22]) and some restrictions on the action probabilities, the update algorithms G_1 and G_2 converge almost surely (with probability one) to the optimum vector of action probabilities.

We restrict the action probabilities such that $q_{min} \leq P_i^{(n)} \leq q_{max}$ where q_{min} and q_{max} are real numbers such that $0 < q_{min} < q_{max} < 1$ and $q_{max} = 1 - (K-1)q_{min}$. This restriction is motivated by the following consideration. Consider the case when the i^{th} action is the optimal action and at the n^{th} iteration, $s_i^{(n)} = 0$ and the action probabilities are now adjusted so that $P_i^{(n+1)} = 0$. In this case $P_i^{(n)}$ remains *absorbed* at zero. It has been noted that previously studied update mechanisms also suffer from this problem (i.e., they converge but not necessarily to correct values) [17, 19]. To avoid this case and hence retain the ability to track a non-stationary environment, we restrict the action probabilities to strictly positive values; we note, however, that q_{min} may be arbitrarily close to zero. Stated more precisely, we have the condition $\forall i : P_i^{(n)} \geq q_{min} > 0$. Since $\sum_{i=1}^{K} P_i^{(n)} = 1$, we immediately obtain the above upper bound on $P_i^{(n)}$.

Clearly, if the i^{th} action is the optimal action, then we would like to have $P_i^{(n)}$ converge to q_{max}, which is the optimum feasible solution to (4); more formally, we would like to show that given arbitrary $\eta, \xi > 0$ there exists N such that $\forall n > N$,

$$Pr[\sup_{m>n} |P_i^{(m)} - q_{max}| > \eta] < \xi$$

We make the assumption that the environment is eventually *static*, i.e. that, although each distribution F_i may change dynamically, the distributions change only a finite number of times. Next, suppose that, after all these changes are over, the i^{th} action is the optimal action.

THEOREM 1: Let a_n be a sequence of real numbers such that $a_n > 0$, $\sum_{n=1}^{\infty} a_n = \infty$ and $\sum_{n=1}^{\infty} a_n^2 < \infty$. Furthermore if the i^{th} action is the optimal action then assume $E[s_i] > E[s_j], j \neq i$. Then $P_i^{(n)}$ converges almost surely to q_{max} under algorithms G_2 and G_1.

PROOF: (see Appendix)

3.4 Using Entire Past Histories

In this section we consider the case of a stationary environment. It has been observed [5, 25, 26] that since each response in the *entire* history of interaction with the environment contains some information pertaining to the statistics (mean rewards) of the stationary environment, past responses may be combined to provide less noisy input to the update algorithms and thus result in better performance.

We present a version of G_2 that uses the entire history of interaction with the environment. Let $r^{(k)}$ be the reward received by the automaton at the k^{th} step and let $J(i,k) = 1$ if the i^{th} action was deployed at the k^{th} step; $J(i,k) = 0$ otherwise. Then define $s_i^{(n)} = \frac{\sum_{k=1}^{n} J(i,k) r^{(k)}}{\sum_{k=1}^{n} J(i,k)}$. Earlier, $s_i^{(n)}$ represented a single response, now $s_i^{(n)}$ represents the estimated average reward obtained for action i since the first step.

We note that, since $\forall i : P_i^{(n)} > 0$, we may assume that $s_i^{(n)} \to E[s_i]$, i.e., that $s_i^{(n)}$ is a strongly consistent estimator for $E[s_i]$. In this case, the value of $s_i^{(n)}$ becomes increasingly "accurate" as n increases. The modified relative reward algorithm that we study is motivated by the following considerations. Note that G_2 is an iterative mechanism and therefore, even in the deterministic case of zero noise ($\forall i, n : s_i^{(n)} = E[s_i]$) at least some iteration is required before convergence. In a stationary, yet stochastic, environment, we observe that as $s_i^{(n)} \to E[s_i]$ and the process approaches being deterministic, more than one iteration may be used between each interaction with the environment. For any particular, large n, we would expect that $s_i^{(n)} \approx E[s_i]$ and therefore, several iterations could be performed using the values of $s_i^{(n)}$ (resembling the iteration in the deterministic case) and faster convergence might be obtained.

We use the notation $g_2(a_n, \mathbf{P}^{(n)}, \mathbf{s}^{(n)})$ to denote the computation of equations (2) and (3) where $\mathbf{P}^{(n)} = (P_1^{(n)}, ..., P_K^{(n)})$ and $\mathbf{s}^{(n)} = (s_1^{(n)}, ..., s_K^{(n)})$. Then, by this notation, the computation at each step of algorithm G_2 (equations (2) and (3)) may be described as: $\mathbf{P}^{(n+1)} := g_2(a_n, \mathbf{P}^{(n)}, \mathbf{s}^{(n)})$. We now introduce the notion of W such succesive iterations of $g_2(.)$ between interactions with the environment. Define each step of algorithm $G_2(W)$ as ($\mathbf{T}$ is a temporary variable below):

Algorithm $G_2(W)$:

$\mathbf{T} := \mathbf{P}^{(n)}$;
for $i := 1$ **to** W **do**
  $\mathbf{T} := g_2(a_n, \mathbf{T}, \mathbf{s}^{(n)})$;
$\mathbf{P}^{(n+1)} := \mathbf{T}$;

Note that we get algorithm G_2 of section 3 by taking $W = 1$ above and letting $s_i^{(n)}$ represent the most recent response. As discussed above, in this section, we have redefined $s_i^{(n)}$ as the mean reward estimate based on the entire history of responses from the environment. We study the performance of algorithm $G_2(W)$ following experimental results on G_1 and G_2 in the next section.

4 Experimental Results

In this section we present some of our simulation results and demonstrate that the relative reward algorithms produce probability vectors that result in higher performance with less variation than other algorithms, and thus, perform better than the S-Model update algorithm in [29] in several respects.

The experimental framework for these studies was as follows. The automaton had 10 actions, $(\alpha_1, ..., \alpha_{10})$, with mean rewards $(0.8, 0.74, ..., 0.8-0.06i, ..., 0.26)$ and an initial vector of equal action probabilities: $\forall i : P_i^{(0)} = 0.1$. The noise in the environment was simulated by taking truncated samples from Gaussian distributions with the above mean values. We present two sets of results: one for the case of high noise (i.e. a high common variance in the Gaussian distributions) and one for the case of zero noise. Since the goal of the automaton is to converge to action probabilities which maximize the expected reward, we are interested in observing the progress the automaton makes with each additional iteration. Consequently, our results are generally in the form of graphs plotting some performance metric as a function of the number of iterations. In our graphs, each point in each curve was obtained as an average of 300 experiments.

We note that there are several ways of selecting stepsizes to satisfy the conditions of Theorem 1, each with varying effects on the performance of the algorithm. We study two such choices for algorithms G_1 and G_2: one in which $a_n = 1/n$ and the other where $a_n = a, \forall n \leq N_a$ and $a_n = 1/n, \forall n > N_a$. Stated differently, in the latter case, a constant stepsize is maintained for a "small" horizon (N_a) before using a decreasing stepsize, whereas in the former choice, the stepsizes are monotonically decreasing. We find that an optimal choice of a constant stepsize for a finite horizon yields better performance than $a_n = 1/n$. The convergence profiles of four algorithms are presented:

- The S-Model update algorithm of [29] with stepsize parameter a (labeled as SL_{R-I}). We note that the stepsize parameter is often referred to as the *learning parameter* in the automata literature [17].
- Algorithm G_1 with $a_n = a$ and $N_a > 1000$.
- Algorithm G_2 with $a_n = 1/n$ (labeled as G_2/n).
- Algorithm G_2 with $a_n = a$ and $N_a > 1000$ (labeled as G_2). The rapidity of performance improvement depends on the choice of a. In figure 1 we plot the expected reward at the 1000^{th} iteration for various stepsizes. From this type of graph, the best stepsizes are selected for the other experiments. For example, in figures 2, 3 and 4 the stepsizes were taken to be 0.04 for algorithm SL_{R-I} and 0.02 for algorithm G_2.

In figure 2 we plot the expected reward as a function of the number of iterations in the case that the variance of the Gaussian noise process was 0.25. We note that the mean rewards produced by algorithms G_1 and G_2 are consistently higher than those produced by the other algorithms. It is possible, however, to achieve high expected rewards without converging to the optimal distribution of action probabilities. This might occur, for example, when the relative difference among the expected reward values is small. Thus, it is of interest to observe the trajectory of the probability corresponding to the highest reward action. In figure 3 we plot the probability of the highest reward action (the first action) with increasing iteration number. The trajectories of this probability under algorithms G_1 and G_2 are clearly higher than that for the other algorithms. Figure 4 shows the estimated standard deviation of the performance obtained at each iteration. Note that the standard deviation curve for the algorithms in this paper lie below the curve for algorithm SL_{R-I}. This indicates that the relative reward update mechanisms not only improve performance sooner but also produce action probability vectors with less variation than does mechanism SL_{R-I}. The curve for algorithm G_2/n is approximately the same as that of G_2.

Figure 1: High noise: Expected reward at 1000th iteration vs. stepsize

Figure 2: High noise: Expected reward vs. iterations

Figure 3: High noise: Probability of highest reward action vs. iterations

Figure 4: High noise: Standard deviation vs. iterations

Figure 5: Zero noise: Expected reward vs. iterations

Figure 6: Zero noise: Standard deviation vs. iterations

Next, we study the performance of the algorithms in a situation where the environment responds without noise, i.e., the response for each action is the corresponding mean reward for that action. This approximately indicates the behavior of the automaton in circumstances where the noise content in the environment is very low. As can be seen from figure 5, algorithms G_1 and G_2 result in higher expected rewards. Figure 6 plots the estimated standard deviation. Consider the execution of any one of the algorithms G_1, G_2 or G_2/n in the case of zero environmental noise. The algorithm samples each action once initially and since there is no noise in the rewards, the reward vector remains unchanged at each subsequent iteration. Hence, the algorithm follows a unique trajectory and therefore, demonstrates no variation in the results produced. We observe that, in contrast, algorithm SL_{R-I} displays a significant amount of variation.

Several additional comments may be made. First, consider the case when $N_a = \infty$. In this case, we found that algorithm G_2 displays slight oscillation very close to the optimum, whereas algorithm G_2/n converges to the optimum. Thus, G_2/n will eventually achieve slightly higher expected rewards than G_2 (with $N_a = \infty$). Our experimental results (not shown here) indicate that this occurs only after an extremely large number of iterations - on the order of $10,000$ to $20,000$ iterations. However, convergence is obtained only in the limit, and thus, for a finite (and more realistic) horizon of, for example, 1000 iterations, a constant stepsize produces better results; for the above experiments we took $N_a = 1000$. Second, we required that for each action, $P_i^{(n)} > q_{min}$ for the proof of convergence. For our experimental results we took $q_{min} = 10^{-7}$. Third, we have observed that the decreasing stepsize algorithm for G_1, corresponding to G_2/n, results in very similar performance to that of G_2/n and thus we have omitted any results pertaining to G_1/n. In addition, we do not plot the confidence intervals for each curve; by using a Normal approximation, we estimate the largest 95% confidence interval to be 0.011, or an error of less than 5% of each $y - axis$ unit. Finally, we note that we have required that the rewards lie in the range $[0, 1]$. In general, for the algorithms in this paper this restriction is not necessary. However, for the update mechanism SL_{R-I} [29] the rewards must be scaled to the interval $[0, 1]$.

4.1 Cost incurred during learning

The evaluation of the automaton's performance above is only concerned with convergence to an optimal set of action probabilities. Upon convergence, the actual rewards obtained by the automaton will, in an average sense, be maximized since the expected reward is maximized. However, the rewards obtained *during* the learning process are not taken into account in this performance metric. We note that during the learning process, the automaton may frequently sample suboptimal actions (actions whose corresponding expected rewards are not maximal) and thus receive suboptimal rewards. Loosely speaking, an "efficient" algorithm will converge to an optimal set of action probabilities while keeping the cost of learning (sampling suboptimal actions) to a minimum. We are thus motivated to define a performance metric that measures, at each iteration, the cumulative

Figure 7: Fractional reward ratio vs. iterations

reward obtained since the start of operation of the algorithm. We call this the fractional reward ratio (FRR) and define it as follows. Let the response at iteration i be denoted by r_i. Then if the optimal action is the j^{th} action, FRR_n, the fractional reward ratio at the n^{th} iteration, is defined by

$$FRR_n = \frac{\sum_{i=1}^{n} r_i}{nE[s_j]}$$

Note that if the optimal action, j, were being used since the start of operation then the expected sum total reward at the n^{th} iteration will be $nE[s_j]$. The metric simply computes the ratio between the actual sum total reward obtained and $nE[s_j]$. Thus, if suboptimal actions were taken frequently, then the FRR would be low; if they were taken only occasionally, then the FRR would be high.

Figure 7 shows the FRR (averaged over 300 experiments) plotted against iteration number for each of the three algorithms in the case of high noise. We note that algorithms G_1 and G_2 result in a lower learning cost (higher rewards during the learning phase) as compared to the other algorithms.

4.2 Using Entire Past Histories

We now present results pertaining to the update mechanism discussed in section 3.4, algorithm $G_2(W)$. We now show our results with algorithm $G_2(W)$ for different values of W and compare our

Figure 8: History-based algorithms

algorithm with another recently proposed update mechanism that uses entire histories of interaction [26]. Figure 8 plots convergence profiles for $W \in \{1, 2, 5, 10\}$ together with the profile for the history-based algorithm in [26]. Note that algorithm $G_2(W)$ with $W = 1$ (labelled as $G_2 : W = 1$) performs better than the algorithm in [26] (labelled as SA). Furthermore, we see that higher values of W result in better convergence profiles.

5 Conclusions

In this paper we have examined a relative reward approach towards designing update algorithms for S-Model learning automata. A strong convergence result was given and supplemented by experimental results. The performance of the relative reward update mechanisms was seen to be superior to the SL_{R-I} learning algorithm of [20, 29] in several ways: algorithms G_1 and G_2 produced consistently higher performance levels than other algorithms while achieving these levels with lower variance. In addition, the cost of learning was shown to be less than that of other algorithms. In the case that a stationary environment could be assumed, we presented a simple extension of algorithm G_2 and demonstrated its superior performance over the algorithm of [26]. Furthermore, a correspondence was made between a learning process that uses increasingly accurate estimates and a deterministic process, resulting in algorithms with even faster convergence.

Learning automata are useful in situations where the characteristics of the underlying system are changing constantly. In that case it is desirable to use algorithms that *track* the changes and maintain a high level of performance. For future work, it would be important to develop a framework to compare the tracking ability of various algorithms.

Acknowledgements:

The authors would like to thank Prof. H.K.Hsieh of the Department of Mathematics and Statistics, University of Massachusetts, for providing some valuable references.

References

[1] A.G.Barto and P.Anandan, "Pattern Recognizing Stochastic Learning Automata," *IEEE Trans. on Man, Systems & Cybernetics*, Vol. 15, pp. 360-374, 1985.

[2] D.Bertsekas, E.Gafni and R.Gallager, "Second Derivative Algorithms for Minimum Delay Distributed Routing in Networks," *IEEE Trans. on Communications*, Vol COM-32, No. 12, pp. 911-919, August 1984.

[3] B.Chandrasekharan and D.W.C.Shen, "On Expediency and Convergence in Variable Structure Automata," *IEEE Trans. on Systems, Man & Cybernetics*, pp. 52-60, Mar 1968.

[4] K.L.Chung, *A Course in Probability Theory*, New York: Academic Press, 1974.

[5] L.P.Devroye, "A Class of Optimal Performance Directed Probabilistic Automata," *IEEE Trans. on Systems, Man & Cybernetics*, Vol SMC-6, pp. 777-784, Nov. 1976.

[6] A.Dvoretzky, "On Stochastic Approximation," *Proc. 1959 Berkeley Symp. Math. Stat. and Prob.*, Vol 1, 1959.

[7] Y.M.El Fattah, P.Boyer, A.Dupuis and L.Romoeuf, "Use of a Learning Automaton for Control of Service Activity," Centre National d'Etudes des Telecommunications, Lannion, France.

[8] K.S.Fu and Z.J.Nikolic, "On Some Reinforcement Techniques and their Relation to the Stochastic Approximation," *IEEE Trans. on Automatic Control*, Vol AC-11, No. 4, pp. 756-758, Oct 1966.

[9] R.G.Gallager, "A Minimum Delay Routing Algorithm Using Distributed Computation,", *IEEE Trans. on Communications*, Vol COM-25, pp. 73-85, 1977.

[10] G.Heal, "Planning Without Prices," *Rev. of Econ Studies*, Vol 36, pp. 346-362, 1969.

[11] Y.C.Ho, L.Servi and R.Suri, "A Class of Center-Free Resource Allocation Algorithms," *Large Scale Systems*, Vol 1, pp. 51-62, 1980.

[12] J.Kurose and R.Simha, "A Microeconomic Approach to Optimal Resource Allocation in Distributed Computer Systems," *IEEE Trans. on Computers*, to appear.

[13] H.J.Kushner and D.S.Clark, *Stochastic Approximation Methods for Constrained and Unconstrained Systems*, New York: Springer Verlag, 1978.

[14] S.Lakshmivarahan, *Learning Algorithms: Theory and Applications*, New York: Springer Verlag, 1981.

[15] L.Ljung, "Analysis of Recursive Stochastic Algorithms," *IEEE Trans. on Automatic Control*, Vol AC-20, 1977, pp. 551-575.

[16] L.G.Mason, "An Optimal Learning Algorithm for S-Model Environments", *IEEE Trans. on Automatic Control*, pp. 493-496, Oct 1973.

[17] K.S.Narendra and M.A.L.Thathachar, "Learning Automata - A Survey", *IEEE Trans. on Man, Systems & Cybernetics*, pp. 323-334, July 1974.

[18] K.S.Narendra, E.A.Wright and L.G.Mason, "Applications of Learning Automata to Telephone Traffic Routing Problems," *IEEE Trans. on Systems, Man & Cybernetics*, Vol SMC-7, pp. 785-792, 1977.

[19] K.S.Narendra and S.Lakshmivarahan, "Learning Automata: A Critique," *J. of Cybernetics & Information Science*, Vol 1, 1977.

[20] O.V.Nedzelnitsky,Jr. and K.S.Narendra, "Nonstationary Models of Learning Automata Routing in Data Communication Networks," *IEEE Trans. on Systems, Man & Cybernetics*, Vol SMC-17, No. 6, pp. 1004-1015, Nov. 1987.

[21] C.R.Rao, *Linear Statistical Inference and its Applications* (First Edition), New York: Wiley, 1973.

[22] H.Robbins and S.Munro, "A Stochastic Approximation Method," *Ann. Math. Stat.*, Vol 22, pp. 400-407, 1951.

[23] D.Russell, *Optimization Theory*, New York: W.A.Benjamin Inc., 1970.

[24] P.R.Srikantakumar and K.S.Narendra, "A Learning Model for Routing in Telephone Networks," *SIAM J. Control & Optimization*, Vol 20, No. 1, pp. 34-57, Jan 1982.

[25] M.A.L.Thathachar and P.S.Sastry, "A Class of Rapidly Converging Algorithms for Learning Automata," *IEEE Int. Conf. on Cybernetics & Society*, Bombay, India, 1984.

[26] M.A.L.Thatchachar and P.S.Sastry, "A New Approach to the Design of Reinforcement Schemes for Learning Automata," *IEEE Trans. on Systems, Man & Cybernetics*, Vol SMC-15, No. 1, pp. 168-175, Jan. 1985.

[27] M.L.Tsetlin, "On the Behavior of Finite Automata in Random Media," *Automat. i. Telemekh*, Vol. 22, pp. 1345-1354, Oct. 1961.

[28] J.N.Tsitsiklis and D.P.Bertsekas, "Distributed Asynchronous Optimal Routing in Data Networks," *Proc. of the 23rd Conf. on Decision & Control*, 1984.

[29] R.Viswanathan and K.S.Narendra, "Stochastic Automata Models with Applications to Learning Systems," *IEEE Trans. on Systems, Man & Cybernetics*, pp. 107-111, Jan 1973.

[30] R.J.Williams, "Reinforcement Learning in Connectionist Networks: A Mathematical Analysis," Report 8605, *Institute for Cognitive Science*, University of California, San Diego, 1986.

[31] R.J.Williams, "Reinforcement-Learning Connectionist Systems," *Technical Report NU-CCS-87-3*, College of Computer Science, Northeastern University, Boston, 1987.

[32] J.Wolfowitz, "On the Stochastic Approximation Method of Robbins and Munro," *Ann. Math. Stat.*, Vol 25 pp. 457-461, 1952.