Parallel Algorithms for Modules of Learning Automata

M. A. L. Thathachar, *Fellow, IEEE,* and M. T. Arvind

***Abstract*— Parallel algorithms are presented for modules of learning automata with the objective of improving their speed of convergence without compromising accuracy. A general procedure suitable for parallelizing a large class of sequential learning algorithms on a shared memory system is proposed. Results are derived to show the quantitative improvements in speed obtainable using parallelization. The efficacy of the procedure is demonstrated by simulation studies on algorithms for common payoff games, parametrized learning automata and pattern classification problems with noisy classification of training samples.**

I. Introduction

A stochastic learning automaton (LA) [1]–[3] is a model based on the stimulus–response learning characteristics studied in psychology and is used to solve decision and control problems under uncertainty. LA have the capability to learn optimal action combinations based on stochastic feedbacks from the environment in which they operate. The conventional LA model can handle finite action sets. Generalizations of the LA model have enabled handling associative information [4], [6], [7] and continuous action sets [5]. It is also possible to construct feedforward networks using teams of LA and applications of such structures to pattern classification have been well studied [6], [7]. Other applications include control of Markov chains [8], telephone traffic routing, and queuing networks [1].

In its simplest form, an LA has a finite set of actions, that form the stimuli to a random environment. The response (payoff) of the environment to an action selected by the LA is based on a fixed (but unknown) distribution for each action. The goal of the LA is to asymptotically choose that action which results in the maximum expected payoff. The LA maintains a probability distribution over the action set. At every instant it chooses an action based on this distribution, and employs the reinforcement signal of the environment to refine the action probability distribution using a learning algorithm.

Most of the LA algorithms use a learning parameter to refine the action probability distribution in order to obtain improved performance. The learning parameter can be considered both as a step size as well as a confidence measure of the reinforcement signal. It often represents the tradeoff between speed and accuracy of the algorithm. Small values of learning parameter imply smaller weight to the stochastic payoff and thus a greater accuracy, but at the cost of speed of convergence. For large problems, even with reasonable demands on accuracy, permissible learning parameter values are quite small, thus rendering the algorithms slow and limiting their applications. There is hence a need to improve the speed characteristics of the LA without sacrificing the accuracy.

The basic LA model is essentially sequential; only one action is applied to the environment at a time and a single response is elicited. This model is possibly motivated by applications such as telephone traffic routing [1] where only one route is selected at a time for routing a call. However, there are several applications where the LA is used as more of a computational device, and several actions can be tried at a time. An example is a pattern recognition problem in which actions correspond to parameter values and the binary reinforcement signal indicates matching or otherwise of the classification resulting from the parameter selection [9]. In such a situation, several actions could be tried at the same instant for a given sample pattern and all the reinforcement signals could be used to refine the action probability distribution. Since the response of the environment to an action is stochastic, a decision based on several responses would have less expected error than a decision based on a single response. The refinements to the action probability vector would thus be more accurate and facilitate faster convergence. A first step in this direction is made in [10], where improvements in speed of convergence are demonstrated for a group of LA operating in parallel.

This paper presents a parallel version of the well-known L_{R-I} [1], [2] algorithm applicable to a module of LA operating in parallel in a stationary environment. This parallel algorithm for a module of LA forms the subject of Section II. The basic idea here is the simultaneous operation of n LA in a module to result in a nearly n-fold increase in speed of convergence. Results are derived to show the ε-optimality and improved rate of convergence of the proposed procedure. Bounds are derived for the value of the learning parameter as well as the module size for the required accuracy of convergence. Simulation studies supplement the results and demonstrate the efficacy of the proposed procedure.

The development of the parallel LA scheme is shown to yield a general procedure for parallelizing a large class of LA schemes. The schemes are derived in Section III; justifications will be provided for the improved rate of convergence using an ordinary differential equation (ODE) approach. The subsequent sections are devoted to studies of several examples of

Manuscript received April 26, 1995; revised February 14, 1996 and October 26, 1996.

The authors are with the Department of Electrical Engineering, Indian Institute of Science, Bangalore 560012, India (e-mail: malt@vidyut.ee.iisc.ernet.in).

Publisher Item Identifier S 1083-4419(98)00220-9.

parallel LA algorithms derived using the general framework. The problems considered include common payoff games and feedforward networks comprising of LA. Pattern classification examples with noisy classification of training samples are also considered. The studies clearly indicate the efficacy of parallelization in improving the speed of convergence.

II. Parallel Algorithm for a Module of LA

An algorithm is proposed in this section to improve the speed of convergence of LA by using a number of LA operating in parallel, which could be regarded as forming a module. Each LA of the module selects an action at every instant, and the environment gives a payoff to each action. The motivation for the algorithm is that n agents acting simultaneously should speedup a process roughly by a factor n; the idea is similar to that of using a population in handling a complex task as in genetic algorithms [11].

Motivation is also derived from the L_{R-I} algorithm, where the action probability of the selected action is incremented proportional to the product of reward received by it and a term linear in the current value of the action probability. Since a number of LA in the module could choose the same action, the idea in this paper is to change an action probability by a quantity proportional to the difference between the net payoff obtained by that action and the estimated fraction of the total payoff associated with that action. The algorithm presented in this paper has the characteristic that a module of size one corresponds to the well known L_{R-I} [1] learning algorithm; the scheme can thus be viewed as a generalization of L_{R-I}. The algorithm exhibits better speed of convergence characteristics, but retains all other properties of L_{R-I}. The features of the proposed scheme for a module of LA are as follows:

- the action probability distribution is common to all members of the module;
- each member of the module selects an action based on the common action probability distribution, independent of other members;
- the updating depends on actions selected by all members of the module, as well as the payoffs obtained by these actions.

The setup is shown in Fig. 1. It uses a module of n identical LA, with the common action set $\alpha \triangleq \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$. Each of them selects an action (independent of others) based on the common action probability distribution $p(k) \triangleq [p_1(k), p_2(k), \ldots, p_r(k)]$. $\alpha^i(k)$ is the action selected by the ith module member at instant k, $\beta^i(k)$ denotes the corresponding payoff; $\alpha(k) \triangleq [\alpha^1(k), \alpha^2(k), \ldots, \alpha^n(k)]$ denotes the action vector at k; and $\beta(k) \triangleq [\beta^1(k), \beta^2(k), \ldots, \beta^n(k)]$ denotes the corresponding payoff vector. It is assumed that $\beta^i(k) \in [0,1]\ \forall i, k$. The outputs of the environment are fed to a fuser[1] that merges them suitably and supplies the required

[1] Although the fuser is depicted as a single block in the diagram, it can be implemented in a distributed fashion in several ways depending on the application and the number of processors. This fact is elaborated upon in Section II-C.

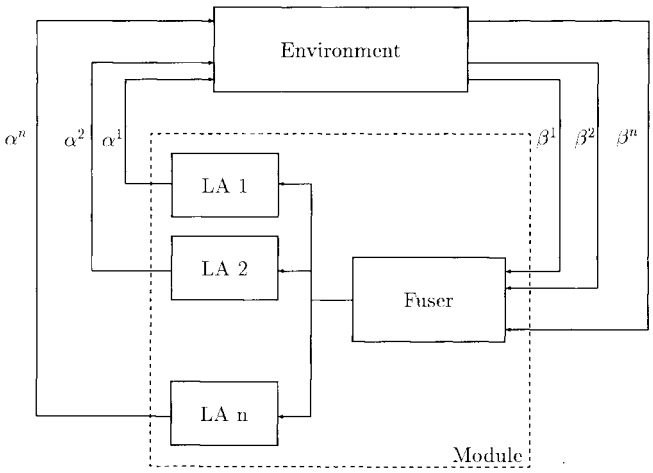

Fig. 1. Module of learning automata.

quantities to all the LA for updating the action probability distribution. The fuser at every instant computes

- total payoff to α_i at k : $q_i(k) \triangleq \sum_{j=1}^{n} \beta^j(k) I\{\alpha^j(k) = \alpha_i\}$;
- total payoff at k : $q(k) \triangleq \sum_{j=1}^{n} \beta^j(k) = \sum_{i=1}^{r} q_i(k)$.

$\lambda \in (0,1]$ is the learning parameter for the algorithm and the quantity $\tilde{\lambda} \triangleq \frac{\lambda}{n}$ is the normalized value of the learning parameter. $I\{\cdot\}$ is the indicator function taking values 1 or 0. The actual algorithm is

$$p_i(k+1) = p_i(k) + \tilde{\lambda}(q_i(k) - q(k)p_i(k)) \quad \forall i \in \{1,2,\ldots,r\}. \quad (1)$$

At any instant k, the expected fraction of choices of α_i is $p_i(k)$. The quantity $q_i(k)/q(k)$ could be considered as a figure of merit of the performance of α_i, and the update term could be regarded as moving $p_i(k)$ toward $q_i(k)/q(k)$. It should be noted that the update preserves the probabilistic nature of the common state vector. In practice, the action probability $p(k)$ is updated once and not necessarily by each LA; the updated value $p(k+1)$ is shared by all the LA in the module.

A. Analysis

In this section, it is shown that optimal action being unique, the proposed algorithm (1) is ε-optimal. Cases not satisfying this assumption are commented upon subsequently (see Remark 4). The explicit dependence on k is omitted whenever there is no scope for ambiguity. A learning algorithm is said to be ε-optimal [1] if it is possible to make the limiting expected payoff arbitrarily close to its best possible value. To demonstrate ε-optimality it is sufficient to show that for any given $\delta \in (0,1)$, that there exists a learning parameter $\lambda_0 \in (0,1)$ such that $\forall \lambda \in (0, \lambda_0)$

$$\Pr\{\text{prob. of choosing the best action} \to 1\} > 1 - \delta.$$

The following definitions are used during the analysis:

$$d_i \triangleq E[\beta^j(k) \mid \alpha^j(k) = \alpha_i]; \quad \forall j.$$

The environment is assumed to be stationary; the d_i are hence constant

$$\Delta p_i(k) \triangleq E[p_i(k+1) - p_i(k) \mid p(k)].$$

From the algorithm, (omitting k)

$$\Delta p_i = \frac{\lambda}{n} E[q_i - p_i q \mid p]. \tag{2}$$

Substituting for q_i and q

$$\Delta p_i = \frac{\lambda}{n} \sum_{j=1}^{n} E[\beta^j (I\{\alpha^j = \alpha_i\} - p_i) \mid p]. \tag{3}$$

Remark 1: $\beta^j(k)$ does not depend on $\alpha^s(k)$; $s \neq j$.

Now,

$$E[\beta^j (I\{\alpha^j = \alpha_i\} - p_i) \mid p] = p_i(1-p_i)d_i - p_i \sum_{s \neq i} p_s d_s. \tag{4}$$

Substituting this in (3),

$$\Delta p_i = \lambda \left(p_i (1 - p_i)\, d_i - p_i \sum_{s \neq i} p_s\, d_s \right). \tag{5}$$

Let $d_l = \max d_i$; $1 \leq i \leq r$. Then $\Delta p_l \geq 0$. It is easy to see that $p(k)$ is a Markov process and 0 and 1 are absorbing states for $p_l(k)$. To eliminate other possible zeros of Δp_l, the condition that d_l is unique, is enforced. Then,

$$\Delta p_l > 0; \quad \forall p \in (0,1)^r, \text{ such that } \sum_{i}^{r} p_i = 1. \tag{6}$$

Proposition 1: Let d_l be unique. Then $p_l(k)$ converges to 0 or 1 w.p. 1.

Proof: Since $\Delta p_l \geq 0$, $p_l(k)$ is a sub-Martingale. Since d_l is unique, by (5) and (6), $\Delta p_l = 0$ only when $p_l = 0$ or 1. The proposition follows from the Martingale convergence theorem [12]. □

1) ε-Optimality: In this section, sufficiency conditions are derived on $\tilde{\lambda}$ such that the algorithm is ε-optimal in all stationary random environments. The concept of subregular functions [1], [2] is used for this purpose. A continuous function $\phi : S_r \to \Re$ is said to be subregular if

$$E[\phi(y(k+1)) \mid y(k)] \geq \phi(y(k))$$

where, S_r is the simplex $S_r \triangleq \{p \in [0,1]^r : \sum_{i=1}^{r} p_i = 1\}$. Also, let

$$\Gamma_i(p) \triangleq \Pr\{p_i(\infty) = 1 \mid p(0) = p\}.$$

Proposition 2: Let $\phi_i(p)$ be subregular with, $\phi_i(e_i) = 1$ and $\phi_i(e_j) = 0$; $\forall j \neq i$ (e_j is the unit vector with 1 in the jth component). Then $\Gamma_i(p) \geq \phi_i(p)$.

Proof: See [1]. □

Remark 2: Let

$$\phi_i(p) \triangleq \frac{1 - e^{-x_i p_i}}{1 - e^{-x_i}}; \quad x_i > 0.$$

Then $\phi_i(p)$ satisfies the boundary conditions of Proposition 2.

Now, using (1) it can be shown that

$$\begin{aligned} &E[\phi_l(p(k+1)) \mid p(k)] - \phi_l(p(k)) \\ &\quad = \frac{1}{1 - e^{-x_l}} (e^{-x_l p_l(k)} - E[e^{-x_l p_l(k+1)} \mid p(k)]) \\ &\quad = \frac{e^{-x_l p_l}}{1 - e^{-x_l}} (1 - (A_l + B_l)^n) \end{aligned} \tag{7}$$

where, for any $s \in \{1, 2, \ldots, n\}$

$$A_l \triangleq p_l E[e^{-x_l \tilde{\lambda} \beta^s (1 - p_l)} \mid p, \alpha^s = \alpha_l], \tag{8}$$

$$B_l \triangleq \sum_{i \neq l} p_i E[e^{x_l \tilde{\lambda} \beta^s p_l} \mid p, \alpha^s = \alpha_i]. \tag{9}$$

Hence, $A_l + B_l \leq 1$ is sufficient to ensure subregularity of $\phi_l(p)$. The following inequalities are used while obtaining sufficiency conditions:

i) for $y \geq 0$, $e^y \leq 1 + y + \frac{y^2}{2} e^y$;

ii) for $y \geq 0$, $e^{-y} \leq 1 - y + \frac{y^2}{2}$.

Let $d_m = \max_{i \neq l} d_i$; $1 \leq i \leq r$; α_m is the second best action. Let $\theta \triangleq d_l - d_m$.

Lemma 1: Let $\theta > 0$. Then $\phi_l(p)$ is subregular whenever $x_l \tilde{\lambda} \leq ((1+\theta)^2 + 4\theta)^{\frac{1}{2}} - (1+\theta)$.

Proof: From (8) and (9) and the inequalities given above,

$$\begin{aligned} A_l + B_l &\leq p_l E\left[\left(1 - x_l \tilde{\lambda} \beta^s (1 - p_l) \right.\right. \\ &\qquad \left.\left. + \frac{(x_l \tilde{\lambda} \beta^s (1 - p_l))^2}{2} \right) \middle| p, \alpha^s = \alpha_l \right] \\ &\quad + \sum_{j \neq l} p_j E\left[\left(1 + x_l \tilde{\lambda} \beta^s p_l \right.\right. \\ &\qquad \left.\left. + \frac{(x_l \tilde{\lambda} \beta^s p_l)^2}{2} e^{x_l \tilde{\lambda} \beta^s p_l} \right) \middle| p, \alpha^s = \alpha_j \right] \\ &\leq 1 - x_l \tilde{\lambda} p_l \sum_{j=1}^{r} p_j (d_l - d_j) + \frac{(x_l \tilde{\lambda} (1 - p_l))^2 p_l}{2} \\ &\quad + \frac{(x_l \tilde{\lambda} p_l)^2 (1 - p_l)}{2} e^{x_l \tilde{\lambda}} \\ &\leq 1 - x_l \tilde{\lambda} p_l (1 - p_l) \left(\theta - \frac{(x_l \tilde{\lambda})}{2} e^{x_l \tilde{\lambda}} \right). \end{aligned} \tag{10}$$

For the quantity on the right hand side to be smaller than unity, it is necessary that $x_l \tilde{\lambda} \in (0,1)$. Using the inequality,

$$e^x \leq \left(\frac{2 + x}{2 - x} \right); \quad x \in (0,1)$$

(10) simplifies to

$$A_l + B_l \leq 1 - x_l \tilde{\lambda} p_l (1 - p_l) \left(\theta - \frac{(x_l \tilde{\lambda})}{2} \frac{2 + x_l \tilde{\lambda}}{2 - x_l \tilde{\lambda}} \right). \tag{11}$$

The result follows by enforcing $(\theta - \frac{(x_l\tilde{\lambda})}{2}\frac{2+x_l\tilde{\lambda}}{2-x_l\tilde{\lambda}}) > 0$. The main result of this section is now proved. □

Theorem 1: The proposed algorithm is ε-optimal in all stationary random environments with $\theta > 0$.

Proof: It is enough to prove that for any given $\delta \in (0,1)$, it is possible to have $\Gamma_l(p) \geq (1-\delta)$, by a suitable choice of $\tilde{\lambda}$.

Now, by definition $\phi_l(p) > (1 - e^{-x_l p_l})$ where $p_l = p_l(0)$. Choosing $x_l = \ln(\frac{1}{\delta})/p_l(0)$, $\phi_l(p) > (1-\delta)$ is ensured. By Lemma 1 subregularity of $\phi_l(p)$ is assured whenever $\tilde{\lambda} \in (0, \tilde{\lambda}_0)$, where

$$\tilde{\lambda}_0 = \left(\frac{p_l(0)}{\ln(\frac{1}{\delta})}\right)\left(((1+\theta)^2 + 4\theta)^{\frac{1}{2}} - (1+\theta)\right). \quad (12)$$

Choosing $\tilde{\lambda} \in (0, \tilde{\lambda}_0)$, and using Proposition 2, the result follows. □

Remark 3: It is seen that the accuracy of the process is controlled by the parameter $\tilde{\lambda}$, while n and λ can be varied to control the speed of convergence for any given accuracy. Since both speed and accuracy can be controlled independently, this algorithm is more flexible when compared with the sequential L_{R-I} algorithm.

Corollary 1: If the optimal action is unique, given any $\delta \in (0,1)$, $n \geq 1$ and $\theta_0 \in (0,1)$, there exists $\lambda_0 \in (0,1)$ such that $\forall \lambda \in (0, \lambda_0)$, $\Pr\{\lim_{k\to\infty} p_l(k) = 1\} \geq (1-\delta)$ in all stationary random environments with $\theta \geq \theta_0$.

Corollary 2: If the optimal action is unique, given any $\delta \in (0,1)$, $\lambda \in (0,1)$, $\theta_0 \in (0,1)$, there exists $n_0 \geq 1$ such that $\forall n > n_0$, $\Pr\{\lim_{k\to\infty} p_l(k) = 1\} \geq (1-\delta)$ in all stationary random environments with $\theta \geq \theta_0$.

These corollaries indicate the freedom in selection of parameters. While Corollary 1 gives sufficiency conditions on the step size of the algorithm for a given module size and given accuracy, Corollary 2 gives sufficiency conditions on the module size for a given step size. Since a larger step size indicates larger speed, as shown in the following subsection, it means that speed of convergence of the algorithm can be improved, without sacrificing the accuracy, by using a module of LA. Sufficiency conditions on λ_0 and n_0 are given by

$$\lambda_0 = \min\left(1, \left(\frac{np_l(0)}{\ln(\frac{1}{\delta})}\right)\left(((1+\theta)^2 + 4\theta)^{\frac{1}{2}} - (1+\theta)\right)\right)$$

$$n_0 = \lambda\left(\left(\frac{p_l(0)}{\ln(\frac{1}{\delta})}\right)\left(((1+\theta)^2 + 4\theta)^{\frac{1}{2}} - (1+\theta)\right)\right)^{-1}.$$

Remark 4: In situations where there are multiple optimal actions and there is at least one suboptimal action, similar analysis holds if p_l is interpreted as the sum of action probabilities corresponding to all the optimal actions. It can be shown that the combined probability of choosing one of the optimal actions at a time, can be made as close to unity as desired with a high probability, by choosing a small enough learning parameter. Such a result intuitively makes sense, since, if there is more than one optimal action, best performance is obtained when any of them is attempted; it is sufficient that the sum of the probabilities of attempting them goes to unity. With p_l interpreted this way, exactly the same analysis holds as earlier for demonstrating ε-optimality.

TABLE I
FIVE-ACTION PROBLEM, $p_i(0) = 0.2;\ \forall i$

n	λ	$k_{avg}(0.9)$	$k_{avg}(0.95)$	$k_{avg}(0.99)$
1	0.01	2732	3727	5811
2	0.02	1388	1831	2871
4	0.04	679	942	1412
8	0.08	341	473	745
16	0.16	189	261	365
32	0.32	87	119	176
64	0.64	43	56	82

2) Speed of Convergence and Module Size: A popular method of analyzing algorithms of type (1) for any given n is to study an associated ordinary differential equation (ODE). The ODE of the algorithm is given by

$$\frac{df_i}{dt} = n\left[f_i(1-f_i)\,d_i - f_i\sum_{s\neq i} f_s\,d_s\right]; \quad f_i(0) = p_i(0),\ \forall i \quad (13)$$

where $f(t)$ is the variable in the associated ODE corresponding to $p(k)$ and $t = k\tilde{\lambda}$. Using the weak convergence theory of [13] it can be shown that the large time behavior of the algorithm can be approximated by that of the associated ODE (13) for small enough values of $\tilde{\lambda}$. It can also be verified that the only stable equilibrium point of the associated ODE of the algorithm is the unit vector e_0 with 1 at the position of the optimal action. Under these conditions, it can be shown [2], [8] that $E[p(k) - f(\tilde{\lambda}k)] = O(\sqrt{\tilde{\lambda}})$ and $E[(p(k) - f(\tilde{\lambda}k))^2] = O(\tilde{\lambda})$.

These expressions help analyze the speed of convergence of the algorithm to its ODE and/or to the desired point e_0. Specifically, $\tilde{\lambda} = \lambda/n$ can be viewed as an accuracy parameter, relating the averaged process to the associated ODE. However, the speed of convergence of the ODE is completely characterized by λ for a given $p(0)$. If $\tilde{\lambda}$ is fixed, the algorithm for a module of size n tracks an ODE which is n times faster, but retains the same level of accuracy. This amounts to a n fold compression of the time axis. On the other hand, if λ is fixed, $\tilde{\lambda}_1 < \tilde{\lambda}_2$ whenever $n_1 > n_2$ and the algorithm and the ODE trajectory match better in the former case.

B. Simulation Studies

A five-action problem (i.e., $r = 5$) with expected payoffs 0.9, 0.81, 0.7, 0.6, and 0.5 is considered for simulation studies. $k_{\text{avg}}(a)$ is the average value of k in 100 runs for which $p_{\text{opt}}(k)$ remained beyond a. To demonstrate sufficiency conditions, the same value of $\tilde{\lambda}$ was used for all module sizes in Table I. The table indicates speedups of roughly the order of module size, thus demonstrating the efficacy of the approach. For this problem, $\theta = 0.09$. With $\delta = 0.01$, the sufficiency condition (12) yields $\tilde{\lambda} = 0.0067$, which is in good agreement with the value $\tilde{\lambda} = 0.01$ employed in simulations. This shows that the sufficiency bounds are quite reasonable.

C. Some Implementation Issues

This section addresses some implementation issues for the parallel model described earlier. Questions relating to performance of the algorithm on various parallel random access machine (PRAM) models and those relating to fuser implementation on multiprocessor machines, are briefly discussed.

1) Speedup and Efficiency on PRAM Models: The speedups of the order of the number of LA, indicated by simulation studies in Table I are idealistic, in the sense that they do not account for communication overheads and other implementation aspects involved while programming on shared memory systems. The goal of this subsection is to obtain more realistic estimates of the speedups on PRAM [14] models. PRAM models model the ideal parallel machine with a shared memory and synchronized read-memory, compute, and write-memory cycles. Speedup figures obtained from such models constitute an upper bound on those achievable in practice.

It is assumed that each LA is, in reality, a processor or a processing element (PE). Since each processor has to update the common state vector, exclusive write (EW) model is assumed. Since the state vector can be read simultaneously by several PE, both common read (CR) and exclusive read (ER) policies are acceptable. Hence the PRAM model could be either EREW or CREW. Assume each memory read and write takes t_r and t_w units of time respectively, and the computation by each processor which includes choosing an action randomly, computing or obtaining a stochastic payoff for the action and computing its contribution of state update, take t_c units of time. Further, it is assumed that the ideal speedup is n, the number of processors. This is consistent with the simulation results observed till now, which indicate average speedups of the order of the number of processors.

Assuming that all processors read the state information simultaneously, the speedup (S_{CREW}) and efficiency (η_{CREW}) on the CREW model are

$$S_{\text{CREW}} = \frac{t_r + t_w + t_c}{\frac{1}{n}(t_r + nt_w + t_c)} = n - \frac{n(n-1)t_w}{t_r + nt_w + t_c}$$
$$\eta_{\text{CREW}} = 1 - \frac{(n-1)t_w}{t_r + nt_w + t_c}$$

where the factor nt_w appears in the denominator as n write operations are required in the CREW model. Since the state is assumed to be read simultaneously, only one read overhead is incurred, as given by the t_r term in the denominator. The corresponding quantities for the EREW model are

$$S_{\text{EREW}} = \frac{t_r + t_w + t_c}{\frac{1}{n}(nt_r + nt_w + t_c)} = n - \frac{n(n-1)(t_r + t_w)}{nt_r + nt_w + t_c}$$
$$\eta_{\text{EREW}} = 1 - \frac{(n-1)(t_r + t_w)}{nt_r + nt_w + t_c}$$

where the factor $n(t_r + t_w)$ appears in the denominator as n read and write operations are required in the EREW model. It is clear from the expressions that good speedups and efficiencies can be achieved for problems that involve large overheads in payoff computation. Such applications arise in complex stochastic optimization problems, where actions of LA are parameter values, based on which a noisy version of the function to be optimized is given as a payoff. As is true with most of parallel processing applications, good speedups can be achieved in applications where read-write overheads are a small percentage of the overall computational burden.

2) Fuser Implementation: The fuser indicated in Fig. 1 is actually a summing element. It is meant to collect the updates of the n processors, sum them, scale and add the net update to the common internal state. When n is small, the processors themselves could add their updates to the common internal state directly, making the fuser implicit (this would correspond to the uniform memory access shared memory model [14, Sec. 1.2]). However, when n is large, this strategy is not efficient as it creates huge memory write overheads. To alleviate this problem, a distributed memory implementation of the fuser could be considered.

When the processors are divided into clusters, with each cluster having its own common shared memory, it makes sense to have a *local* fuser that collects the updates computed by each processor in its cluster. These partial updates are summed to form the final update by a *global* fuser and added to the common internal state vector. This leads to a distributed cum shared implementation of the fuser (a hierarchical cluster nonuniform memory access shared memory model is suitable in this case). The concept can be generalized to have a tree of fusers, whenever the number of processors is large. However, feasibility of such implementations is directed by specific applications and the computation versus communication overheads involved. Such implementations do not in any way affect the ε-optimality of the algorithm. In fact, as long as the sequential version of the algorithm is ε-optimal, the parallel version also is; a small enough learning parameter can always be chosen, depending on the number of processors, to obtain the required accuracy.

III. General Procedure

In this section, a general parallelization paradigm applicable to a large class of LA algorithms is proposed. The design of the proposed parallelization procedure closely follows the developments of the previous section. The setup includes several learning agents operating in a stochastic environment which presents them with a common situation at each instant. Each learning agent responds to the situation by selecting an action, and obtains a payoff. The learning agents belonging to a module *cooperate* in some sense, as they have a common internal state representation. A shared memory system housing the common internal state is particularly suitable for such a setup. Actions selected by each learning agent depend on the common internal state, but the selected actions, conditioned on the internal state, are independent of one another.

Let each learning agent, in the sequential case, use the following algorithm for updating its internal state $X(k)$, represented usually by the action probability vector

$$X(k+1) = X(k) + \lambda H(X(k), Y(k)); \quad X(0) = x_0. \quad (14)$$

Here $Y(k)$ denotes the observation vector comprising the chosen action and payoff.

Although the proposed parallelization procedure is applicable in general, special attention is paid here to algorithms satisfying the following assumptions [13], [15].

- The vector $Y(k)$ has a Semi-Markov representation controlled by $X(k)$. $Y(k)$ is a function of the extended state $Z(k) \triangleq (X(k), Y(k-1))$, which is a Markov chain with transition probability $\pi_X(Z, dZ)$ a function of X

$$\Pr\{Z(k) \in dZ \mid Z(k-1), Z(k-2), \ldots; X(k-1), X(k-2), \ldots\} = \pi_{X(k-1)}(Z(k-1), dZ).$$

  It is assumed that for fixed X in the effective domain of the algorithm, the Markov chain $\{Z(k)\}_{k\geq 0}$ has a unique stationary asymptotic behavior. A special case of this is when the transition probability does not depend on X.
- The function $H(X, Y)$ may admit discontinuities, but it is assumed that there exists a *mean vector field* which is regular and is defined by

$$h(x) \triangleq \lim_{k\to\infty} E_x[H(x, Y(k))]$$

  where P_x denotes the distribution of $\{Y(k)\}_{k\geq 0}$ for a fixed value $X = x$ and E_x is the corresponding expectation.

These assumptions are fairly general, and most of the LA algorithms fit into this framework. To avoid possible confusion Y will be explicitly denoted by the tuple (β, α) in the sequel. The advantage of such assumptions is that the algorithms' behavior becomes tractable.

Under these assumptions, the associated ODE of the algorithm (14) is given by

$$\frac{dx}{dt} = h(x); \quad x(0) = x_0. \tag{15}$$

In addition, the large time behavior of the algorithm can be approximated by the trajectory of the associated ODE [13]. This also follows from the following theorem of [15].

Theorem 2: Given $\varepsilon > 0, \lambda$ sufficiently small and $T < \infty$. Then there exists a constant $C(\lambda, T)$, with $\lim_{\lambda\to 0} C(\lambda, T) = 0$ and

$$\Pr\left\{\max_{k\lambda<T} ||X(k) - x(k\lambda)|| > \varepsilon\right\} \leq C(\lambda, T).$$

Thus, the algorithm *tracks* the ODE for as large a time as required, provided the learning parameter is small. Further, whenever the ODE is globally stable with a unique stable equilibrium point f_0, the following result from [15] characterizes the asymptotic behavior of the algorithm, with this additional assumption:

- $h(x)$ grows at most at a polynomial rate, and the growth of the random term $H(X, \cdot) - h(X)$ is similarly controlled.

Theorem 3: Given $\varepsilon > 0, \lambda$ sufficiently small; ODE (15) globally stable with unique stable equilibrium point f_0. Then there exists a constant $C(\lambda)$, with $\lim_{\lambda\to 0} C(\lambda) = 0$ and

$$\limsup_{k\to\infty} \Pr\{||X(k) - f_0|| > \varepsilon\} \leq C(\lambda).$$

Thus for a small enough value of the learning parameter, the algorithm tracks its associated ODE asymptotically with the required accuracy.

In deriving the parallel version of algorithm (14) in the framework just described, it is useful to rewrite (1) as

$$p_i(k+1) = p_i(k) + \tilde{\lambda} \sum_{j=1}^{n} \beta^j(k)(I\{\alpha^j(k) = \alpha_i\} - p_i(k)). \tag{16}$$

The parallel version of the L_{R-I} algorithm is thus seen to have an update vector which is just the sum of the update vectors of n LA employing the L_{R-I} algorithms individually. Using the same methodology, the parallel version of (14) is obtained as

$$X(k+1) = X(k) + \tilde{\lambda} \sum_{j=1}^{n} H(X(k), \quad (\beta^j(k), \alpha^j(k))); \quad X(0) = x_0 \tag{17}$$

where $X(k)$ replaces $p(k)$.

Following the same arguments as in the sequential case, the associated ODE of (17) is

$$\frac{dx}{dt} = nh(x); \quad x(0) = x_0. \tag{18}$$

For a given n, ODE's (15) and (18) have the same set of stationary points with identical properties. Further, (18) is nothing but ODE (15) with the time axis compressed n-fold. If $\tilde{\lambda}$ were fixed at the same value for both the ODE's and n were such that λ is within permissible limits, if any, then for a given T, ODE (18) which is faster by a factor of n in comparison with ODE (15), is tracked for a time T within desired levels of accuracy by algorithm (17). As regards asymptotic accuracies in case of a unique stable equilibrium point, the same arguments hold for both the algorithms. In particular, the proposed procedure does not alter the asymptotic properties of the sequential algorithm while parallelizing it.

Remark 5: The comments above would lead one to believe that the speed of convergence could be improved arbitrarily by increasing the value of n as desired. This is in general not possible, as in most cases, the increase in n for a fixed $\tilde{\lambda}$ is limited by a ceiling on λ. A reason for this could be the constraints on the state space; for example, most of the LA algorithms maintain a state vector whose elements are probabilities and it is necessary that $\lambda \in (0, 1]$. It is easy to see that the behavior of the scheme when λ is fixed and n, and hence $\tilde{\lambda}$, are variable, is characterized by the strong law of large numbers.

IV. Common Payoff Games

Common payoff games have traditionally served as models for decentralized decision making and multivariable optimization. A common payoff game among LA has each player represented by an LA and the actions of the LA corresponding to the player's strategies [16]. Schemes wherein each LA uses the L_{R-I} algorithm have been studied in [8], [16], [17]. It has been demonstrated in [8] that whenever the expected payoff

matrix is unimodal, this strategy is ε-optimal. The multimodal case is studied in [17]; it is shown that all the modes of the matrix are stable equilibrium points of the associated ODE, and the process weakly converges to the ODE trajectory as the learning parameter value is reduced. Estimator algorithms have also been considered for solving such games [9]. It has been demonstrated that whenever the globally optimal strategic combination is unique, the algorithm converges to that combination with a high probability. However, the memory requirement becomes a bottleneck as an estimate of the entire payoff matrix has to be maintained.

A. Parallel L_{R-I} Algorithm

The common payoff game considered in this section involves N players, each of whom comprises of a module of n identical LA. There are n plays of the game at each instant; play s involves selection of action by the sth member of each module and their receiving a common payoff. Thus there are n teams, the sth team consisting of the sth members of each module (player), $s = 1, 2, \ldots, n$. All the features of algorithm (1) carry over to the game situation also. The same notations apply here with the corresponding change in interpretation. Specifically, each member of module i has r_i actions and its action probability vector is $p_i(k) \triangleq [p_{i1}(k), p_{i2}(k), \ldots, p_{ir_i}(k)]$. $p(k)$ now denotes the action probability vectors of all the players put together and $f(t)$ denotes the corresponding quantity in the associated ODE. The update for $p_{ij}(k)$, the probability of selecting α_{ij}, the jth action of the ith player, is given by

$$p_{ij}(k+1) = p_{ij}(k) + \tilde{\lambda}(q_{ij}(k) - q(k)p_{ij}(k)) \quad \forall j = 1, 2, \ldots, r_i; \quad \forall i = 1, 2, \ldots, N \tag{19}$$

where $q_{ij}(k)$ is the total payoff to α_{ij} at k; $q(k) \triangleq \sum_{j=1}^{r_i} q_{ij}(k)$ for any i; and $q(k)$ is the total payoff received by any player at k.

A brief outline of the analysis of this algorithm for the unimodal and multimodal case is given here. Since this algorithm directly comes under the framework of the previous section, all the analysis of that section holds here about the speed of convergence.

1) Unimodal Case: Let α_0 be the unique optimum pure strategy combination and f_0 be the corresponding pure strategy action probability vector. Under this assumption, it is now shown that algorithm (19) is ε-optimal.

Theorem 4: For a common payoff game with a unique pure strategy equilibrium, algorithm (19) is ε-optimal.

2) Outline of Proof: 1) It can be shown that f_0 is the only stable equilibrium point of the associated ODE (see previous section). Hence whenever $p(0)$ has all nonzero entries, the ODE trajectory converges to f_0. 2) The algorithm is strictly distance diminishing [8] and the state space of the overall Markov process is compact. Hence $p(k)$ converges to the set of absorbing states w.p. 1. 3) Given $p(0)$ with all nonzero entries, using arguments similar to those of [2], [8] it can be shown that $E[|p(k) - f(k\tilde{\lambda})|] = O(\tilde{\lambda})$, where f is the corresponding variable of p in the associated ODE. Hence for any $\varepsilon \in (0,1), \exists \tilde{\lambda}_0 \in (0,1)$ such that $\forall \tilde{\lambda} \in (0, \tilde{\lambda}_0)$, $\Pr\{\lim_{k\to\infty} p(k) = f_0\} > 1 - \varepsilon$. 4) The algorithm tracks an ODE which converges to f_0 faster by a factor of n, with a high degree of accuracy, for sufficiently small values of $\tilde{\lambda}$. □

TABLE II
MULTIMODAL GAME, L_{R-I}, $\bar{\lambda} = 0.00625$

n	$k_{avg}(0.95)$	n	$k_{avg}(0.95)$
1	3995	16	263
2	2402	32	132
4	1001	64	65
8	520	128	33

3) Multimodal Case: Whenever the common payoff matrix is multimodal, it is observed that all the modes of the matrix are stable equilibrium points of the associated ODE [17], and the large time behavior of the algorithm can be approximated by the ODE trajectory for small learning parameter values [13], [15]. The situation is similar to the single player case in which the optimal action is not unique; the algorithm converges to one of the modes with a high probability, whenever the learning parameter is small. Comments made in Section III about the speed of convergence readily hold in both the unimodal and multimodal cases. It only remains to verify these using simulation studies.

4) Example: Simulation results are presented for a two player, two actions per player multimodal game. The game matrix is given by

$$\begin{bmatrix} 0.9 & 0.6 \\ 0.7 & 0.8 \end{bmatrix}.$$

Entries 0.9 and 0.8 are the two modes of the matrix and convergence to either mode is observed to occur. Convergence is assumed when action probabilities corresponding to either of the modes exceeds 0.95. Algorithm (19) is used for updating the action probabilities of both modules of players. For each n, 50 runs of simulation were conducted, and $\tilde{\lambda}$ was chosen such that no wrong convergence resulted in any of these runs. $\tilde{\lambda}$ was set to 0.006 25 and $\lambda = \tilde{\lambda}n$ in each case. All action probabilities start from 0.5. Speedups of the order of the module size are observed from Table II.

B. Parametrized Learning Automata

Parametrized learning automata (PLA) have been employed in [6] to achieve convergence to a global optimum in common payoff games as well as feedforward networks. The updating procedure in PLA involves a random walk term along with the gradient following term, thus facilitating wide exploration of the search space. Unlike simulated annealing, the random term does not tend to zero; adaptation to time varying environments is hence possible. With the addition of the random term, it is difficult to preserve the internal state of the conventional LA as a probability vector. This is circumvented by allowing the internal state to be a vector in $\Re^n$, and using a probability generating function to obtain the action probabilities.

Although PLA are useful in achieving global convergence, they tend to be quite slow. It is necessary to improve their speed characteristics to make them utilizable in practical problems. A parallel version of the PLA algorithm is presented

here. The algorithm does not fit into the framework of the analysis given in the previous section directly, as it involves an additional random term. The analysis of [6] shows that the large time behavior of the algorithm, with a small step size, can be approximated by a stochastic differential equation (SDE). A similar result holds here also and simulation studies clearly indicate the improved speed performance.

1) Algorithm for Modules of PLA: A common payoff game among N players is considered. Player i has r_i actions and his internal state is represented by $u_i(k) \triangleq [u_{i1}(k), u_{i2}(k), \ldots, u_{ir_i}(k)]$. $u(k)$ is the vector comprising $u_i(k) \in \Re^{r_i}$, $\forall i = 1,2,\ldots,N$ and $f(t)$ is the quantity corresponding to $u(k)$ in the associated SDE. $p_{ij}(k)$, the probability of selecting action j of player i, is obtained by means of the probability generating function g_i

$$p_{ij} = g_i(u_i, \alpha_{ij}) \triangleq \frac{e^{u_{ij}}}{\sum_{s=1}^{r_i} e^{u_{is}}}; \quad \forall j = 1,2,\ldots,r_i; \quad \forall i = 1,2,\ldots,N.$$

As in the previous sections, each player now corresponds to a module of size n. Thus there are n plays of the game at any instant and the sth play involves the sth members of each module selecting their actions and receiving a common payoff $(s = 1,2,\ldots,n)$. q_{ij} and q denote the payoffs as in (19). The update for u_{ij} is given by

$$u_{ij}(k+1) = u_{ij}(k) + \frac{\lambda}{n}(q_{ij}(k) - p_{ij}(k)q(k)) + \tilde{\lambda}\frac{dh(u_{ij}(k))}{du_{ij}(k)} + (\tilde{\lambda})^{\frac{1}{2}} S_{ij}(k) \tag{20}$$

where h is given by

$$h(x) = \begin{cases} -K(x-L)^{2m} & x \geq L \\ 0 & |x| \leq L \\ -K(x+L)^{2m} & x \leq -L \end{cases} \tag{21}$$

where $K \geq 0, L \geq 0$ are real and m is a positive integer. $\{S_{ij}(k) : 1 \leq j \leq r_i, 1 \leq i \leq N, k \geq 0\}$ is a sequence of i.i.d. random variables with zero mean and variance σ^2.

The third term on the right side is employed to bound the solutions, while the last term facilitates exploration of the search space. The algorithm could be regarded as the parallel version of the algorithm of [6] as this can be obtained by setting $n = 1$ in (20). A brief outline of the analysis is now presented.

- The large time behavior of the algorithm can be approximated by the SDE [13]

$$df = n\nabla V(f)\,dt + n^{1/2}\sigma\,dw; \; f(0) = u(0) \tag{22}$$

where

$$V(f) = E[\beta^s \mid f] + \sum_{(i,j)} h(u_{ij}) \text{ for any } s \in \{1,2,\ldots,n\}$$

and w is the standard Brownian motion of the appropriate dimension.

- The invariant probability distribution of this equation can be shown to be concentrated around the global optimum of $V(f)$ [18].
- Globally optimizing the function $V(f)$, can be shown to be equivalent to globally optimizing the given common payoff game [6].

The SDE could be regarded as a perturbation over an ODE which is essentially the same equation without the random term. Since the ODE in this case has a multiplicative factor n associated with it, the SDE could be regarded as a perturbation over an ODE that converges faster by a factor of n in comparison with the sequential case.

2) Example: This section considers simulation results for algorithm (20) on a bimodal two player, two actions per player game with the following payoff matrix

$$\begin{bmatrix} 0.9 & 0.7 \\ 0.4 & 0.8 \end{bmatrix}.$$

It is easy to see that options corresponding to 0.9 and 0.8 entries of the matrix are locally optimal; options corresponding to the former being globally optimal. Each player has a scalar internal state, since each of them has only one independent action probability. Designating the internal states as u and v, and ith player's jth action as α_{ij}, the probability generating functions are chosen as

$$\Pr\{\alpha_1^j(k) = \alpha_{11}\} = 1 - \Pr\{\alpha_1^j(k) = \alpha_{12}\} = \frac{1}{1+e^{-u(k)}}$$
$$\Pr\{\alpha_2^j(k) = \alpha_{21}\} = 1 - \Pr\{\alpha_2^j(k) = \alpha_{22}\} = \frac{1}{1+e^{-v(k)}}$$

where α_i^j denotes the action selected by the jth member of player (module) i.

To demonstrate the efficacy of PLA, starting points biased toward convergence to the local but not global optimum are considered. The algorithm makes use of a large value of standard deviation in the beginning, progressively reduces it to a small nonzero value, and maintains this value constant later on. No change in the analysis is required for handling this case. For each value of n, 20 simulation runs were conducted and $\tilde{\lambda}$ was chosen such that no wrong convergence resulted in any of these runs. The value of σ was initially set to ten and reduced as

$$\sigma(k+1) = a\sigma(k)$$

where $a \in (0,1)$. σ is maintained constant at 0.01 once $\sigma(k)$ falls below 0.01. The results are given in Table III for the minimum value of a for which correct convergence resulted in every run. The other parameters of the algorithm are $K = 1, L = 3, m = 2$. From the table it can be seen that the improvement in speed is not linear in n. This could be attributed to the effect of the random term which has an effective standard deviation of $n^{1/2}\sigma$ for a given n, and the different reduction parameters a employed in each case.

V. Pattern Classification Example

In this section, classification results on the iris data problem are presented using feedforward networks of modules of LA. The objective of the study is to demonstrate the improvement in speed of convergence because of parallelization. Pattern classification under both deterministic as well as stochastic

TABLE III
MULTIMODAL GAME, PLA

n	λ	a	$k_{avg}(0.95)$
4	0.08	0.99	1001
8	0.16	0.95	245
16	0.32	0.8	190
32	0.64	0.5	55

setups are considered. A feedforward network is employed for the purpose. The network in general, comprises three layers [19]. The first layer has several units; each unit has several LA which choose weights that multiply each attribute of the input pattern. Each unit of the first layer outputs a binary value which is computed as in case of the standard perceptron. Thus the network is capable of learning as many hyperplanes as the number of first layer units. Units in the second layer perform the AND operation on the outputs of selected units in the first layer and thus learn convex sets with piecewise linear boundaries. The third layer performs an OR of the outputs of the second layer units to give the classification. The environment compares this classification with its internal classification and provides a payoff to the network. It is assumed that a unit payoff results on correct classification; otherwise there is no payoff.

Convergence results for such a feedforward network are identical to those of a common payoff game. Hence no elaboration of these results is made here. A simplified version of the iris data [20] is considered for studies using the parallel L_{R-I} algorithm for common payoff games (19). This is a three class (iris-setosa, iris-versicolor, and iris-verginica) problem with four features. Setosa is linearly separable from the other two classes and the problem solved here is a two class version ignoring this class. There are 50 samples from each class with known classification. The network comprises of nine first layer units and three second layer units. Each first layer unit has five LA corresponding to the four features and the threshold. All first layer LA have nine common weight choices for their actions and the common weight set is $\{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$. All the first layer LA use a starting value of 1/9 for all the action probabilities. Simulations are reported both for 0% and 10% noise. In the latter case, 10% noise means that the given classification of a randomly chosen sample pattern is changed with probability 0.1. Comparisons with backpropagation with momentum (BPM) algorithm [7] have shown the superiority of the L_{R-I} scheme; BPM did not converge in noisy situations. Comparison was also carried out with the quickprop algorithm [21]. In the zero noise case, quickprop took, on the average, about 16 000 iterations. This is comparable to the best performance obtained using the proposed parallel algorithm with 4 LA. However, like backpropagation with momentum, quickprop did not converge in the noisy case. Even after 100 000 iterations, errors kept fluctuating between 10% and 75%. Superiority of LA algorithms in noisy circumstances is thus clearly established.

The objective of the following study is to improve the speed performance of the sequential algorithm by employing a module of LA. Simulation studies are presented in Fig. 2

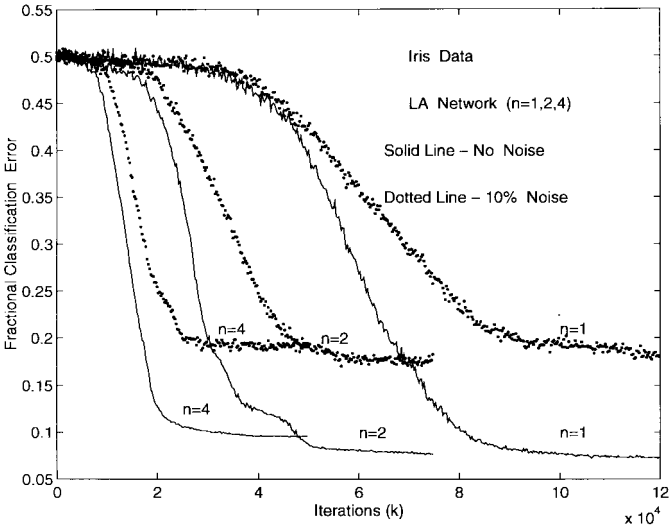

Fig. 2. Learning curves based on average fractional error in classification of Iris data.

for module sizes of 1, 2, and 4, respectively. The figure shows the fractional error in classification averaged over ten runs as a function of the number of iterations. The algorithm was run in two phases; learning phase and the error computation phase. During the learning phase no error computation is performed. A sample pattern is chosen at random, and its classification is altered with probability 0.1 in the noisy case. Otherwise the given classification itself is maintained. The network of modules classifies the pattern using its internal action probability vectors and based on the payoffs obtained by each module, the probability vectors are updated using (19). The error computation phase was performed once every 250 iterations. The probability vectors were not updated during this phase. The set of 100 sample patterns was presented sequentially to the team of modules. This was repeated over ten cycles and the fraction of patterns classified wrongly was computed. Repeated presentations are meant to average out the stochastic classification effects of the network to yield realistic error estimates. As in the learning phase, classification of the input pattern is altered with probability 0.1 in the noisy case.

The figures indicate the faster speed of convergence for the larger module sizes, with learning parameters chosen such that the limiting values of the error are approximately same for the noise free and noisy cases respectively. The chosen values for the limiting errors in the noise free and noisy cases were 0.1 and 0.2, respectively. The values of $(\tilde{\lambda}_1, \tilde{\lambda}_2)$, the learning parameters used in the first and second layer units are. respectively, (0.005, 0.002).

VI. CONCLUSION

An algorithm was proposed to operate a module of LA in parallel. The algorithm was shown to be ε-optimal and sufficiency conditions were derived on the learning parameter value as well as the module size for the desired accuracy. Improvements resulting in speed of convergence were theoretically shown as well as demonstrated using simulations. The algorithm was shown to yield a general procedure for

parallelizing a large class of LA algorithms in a shared memory setup. The procedure was used to develop parallel L_{R-I} and PLA algorithms for teams of modules of LA to solve common payoff games. Finally, pattern recognition examples under uncertainty were considered to demonstrate the improvements in speed of convergence resulting from parallel operation. Applicability of the ideas of this paper to associative as well as delayed reinforcement schemes and their operation in more general setups such as multiteacher environments will be reported elsewhere.

REFERENCES

[1] K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

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

[3] K. Najim and A. S. Poznyak, *Learning Automata: Theory and Applications.* Oxford, U.K.: Pergamon, 1994.

[4] A. G. Barto and P. Anandan, "Pattern-recognizing stochastic learning automata," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-15, no. 3, pp. 360–374, 1985.

[5] G. Santharam, P. S. Sastry, and M. A. L. Thathachar, "Continuous action set learning automata for stochastic optimization," *J. Franklin Inst.*, vol. 331B, no. 5, pp. 607–628, 1994.

[6] M. A. L. Thathachar and V. V. Phansalkar, "Learning the global maximum with parametrized learning automata," *IEEE Trans. Neural Networks*, vol. 6, no. 2, pp. 398–406, 1995.

[7] ——, "Convergence of teams and hierarchies of learning automata in connectionist systems," *IEEE Trans. Syst., Man, Cybern.*, vol. 25, pp. 1459–1469, Nov. 1995.

[8] R. M. Wheeler, Jr. and K. S. Narendra, "Decentralized learning in finite Markov chains," *IEEE Trans. Automat. Contr.*, vol. 31, no. 6, pp. 519–526, 1986.

[9] M. A. L. Thathachar and P. S. Sastry, "Learning optimal discriminant functions through a cooperative game of automata," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-17, no. 1, pp. 73–85, 1987.

[10] M. A. L. Thathachar and M. T. Arvind, "A parallel algorithm for operating a stack of learning automata," in *Proc. 4th IEEE Symp. Intelligent Systems*, Bangalore, India, Nov. 1994.

[11] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Cambridge, MA: MIT Press, 1992.

[12] J. L. Doob, *Stochastic Processes*. New York: Wiley, 1953.

[13] H. J. Kushner, *Approximation and Weak Convergence Methods for Random Processes*. Cambridge, MA: MIT Press, 1984.

[14] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. New York: McGraw-Hill, 1993.

[15] A. Benveniste, M. Metivier, and P. Priouret, *Adaptive Algorithms and Stochastic Approximations*. New York: Springer-Verlag, 1987.

[16] M. A. L. Thathachar and K. R. Ramakrishnan, "A cooperative game of a pair of learning automata," *Automatica*, vol. 20, pp. 797–801, 1984.

[17] P. S. Sastry, V. V. Phansalkar, and M. A. L. Thathachar, "Decentralized learning of Nash equilibria in multiperson stochastic games with incomplete information," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, pp. 769–777, May 1994.

[18] F. Aluffi-Pentini, V. Parisi, and F. Zirilli, "Global optimization and stochastic differential equations," *J. Optim. Theory Appl.*, vol. 47, no. 1, pp. 1–26, 1985.

[19] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, pp. 4–22, Apr. 1987.

[20] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.

[21] S. E. Fahlman, "An empirical study of learning speed in backpropagation networks," CMU-CS-88-162, Carnegie Mellon Univ., Pittsburgh, PA, June 1988.

M. A. L. Thathachar (SM'79–F'91) received the B.S. degree in electrical engineering from the University of Mysore, India, in 1959, the M.S. degree in power engineering, and the Ph.D. degree in control systems from the Indian Institute of Science, Bangalore, in 1961 and 1968, respectively.

He was a Faculty Member at the Indian Institute of Technology, Madras, from 1961 to 1964. Since 1964, he has been with the Indian Institute of Science where he is currently Professor in the Department of Electrical Engineering and Chairman of the Division of Electrical Sciences. He has held visiting faculty positions at Yale University, New Haven, CT, Concordia University, Montreal, P.Q., Canada, and Michigan State University, East Lansing. His current research interests are learning automata, neural networks, and fuzzy systems.

Dr. Thathachar is a Fellow of the Indian National Science Academy, the Indian Academy of Sciences, and the Indian National Academy of Engineering.

M. T. Arvind received the B.E. degree in electronics and communication engineering from the University of Mysore, India, in 1989 and the M.E. degree in electrical engineering from the Indian Institute of Science, Bangalore, in 1992. He is currently pursuing the Ph.D. in electrical engineering at the Indian Institute of Science.

His current interests include learning systems, neural networks, and distributed computing.