Asian Journal of Control, Vol. 14, No. 4, pp. 1–16, July 2012
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/asjc.315

LEARNING AUTOMATA BASED MULTI-AGENT SYSTEM ALGORITHMS FOR FINDING OPTIMAL POLICIES IN MARKOV GAMES

Behrooz Masoumi and Mohammad Reza Meybodi

ABSTRACT

Markov games, as the generalization of Markov decision processes to the multi-agent case, have long been used for modeling multi-agent systems (MAS). The Markov game view of MAS is considered as a sequence of games having to be played by multiple players while each game belongs to a different state of the environment. In this paper, several learning automata based multi-agent system algorithms for finding optimal policies in Markov games are proposed. In all of the proposed algorithms, each agent residing in every state of the environment is equipped with a learning automaton. Every joint-action of the set of learning automata in each state corresponds to moving to one of the adjacent states. Each agent moves from one state to another and tries to reach the goal state. The actions taken by learning automata along the path traversed by the agent are then rewarded or penalized based on the comparison of the average reward received by agent per move along the path with a dynamic threshold. In the second group of the proposed algorithms, the concept of entropy has been imported into learning automata based multi-agent systems to improve the performance of the algorithms. To evaluate the performance of the proposed algorithms, computer experiments have been conducted. The results of experiments have shown that the proposed algorithms perform better than the existing algorithms in terms of speed and accuracy of reaching the optimal policy.

Key Words: Markov games, multi-agent systems, learning automata, optimal policy.

I. INTRODUCTION

A multi-agent system (MAS) is comprised of a collection of autonomous and intelligent agents that interact with each other in an environment to optimize a performance measure [1]. Multi-agent systems are applied in a wide variety of domains including robotic teams, distributed control, resource management, collaborative decision support systems, data mining, and are useful in the modeling, analysis and design of systems where control is distributed among several autonomous decision makers [2–5]. There are several models proposed in the literature for multi-agent systems MASs based on Markov models. One of these models is the Markov Game (MG) which is the generalization of the Markov decision process (MDP) to the Multiple agent [6]. The Markov game view of MAS is considered as a sequence of games having to be played by multiple players while each game belongs to a different state of the environment [7]. In an MG, actions are the result of joint action selection of all agents, while rewards and the state transitions depend on these joint actions [8]. As a special case, when only is one state assumed, the Markov game is

Manuscript received February 6, 2010; revised July 14, 2010; accepted September 3, 2010.

Behrooz Masoumi is with the Department of Computer Engineering, Islamic Azad University, Science and Research Branch, Tehran, Iran (e-mail: masoumi@Qiau.ac.ir).

Mohammad Reza Meybodi is with the Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran (e-mail: mmeybodi@aut.ac.ir).

known as a repeated normal form game in game theory [6]. In addition, when only is one agent assumed, the Markov game is known as MDP. In multi-agent system research, two main perspectives are found in the literature; the cooperative and non-cooperative perspective. In cooperative MASs, the agents pursue a common goal and the agents can be built expect benevolent intentions from other agents [9]. In contrast, a non-cooperative MAS setting has non-aligned goals, and individual agents try to obtain only to maximize their own profits. In multi-agent systems, the need for learning and adaption is essentially caused by the fact that the environment of an agent is dynamic and just empirically observable while the environment (the reward functions and the transition states) is unknown. Hence, the reinforcement learning methods may be applied in MASs to find an optimal policy in MGs. In addition, agents in a multi-agent system face the problem of incomplete information with respect to the action choice. If agents get information about their own choice of action as well as that of the others, then we have joint action learning [8]. Joint action learners are able to maintain models of the strategy of others, and explicitly takes into account the effects of joint actions.

Learning automata (LA) are adaptive decision making devices suited for operation in unknown environments [10]. Currently, learning automata are among the valuable tools to design reinforcement learning algorithms and multi-agent systems [11–13]. Due to certain specifications such as structure simplicity, little need for information, and feedback from the environment, LAs are very useful in multi-agent systems. Many algorithms based on learning automata have been developed for learning optimal strategies in Markov games. In the past few years, a significant part of the research has focused on comprehending and solving single-stage multi-agent problems modeled as a normal form game and multi–stage game modeled as Markov games [14]. There are several methods based on learning automata for finding an optimal policy in MGs. In [15] it is shown that a team of learning automata involved in a general N-person Markov game converges to Nash equilibrium if each of the team members makes use of a linear learning algorithm called L_{R-I} algorithms. In [16], LAs are used for playing stochastic games at multiple levels. In [7], a model based on interconnected learning automata is suggested to solve MMDPS. In [17], the authors showed a network of independent LA which is able to reach equilibrium strategies in Markov games with some ergodic assumptions. In [18], the authors proposed two new algorithms based on learning automata that can be effectively used to solve multi-agent MDPs and to find the optimal policy. In [19], a new model based on learning automata for finding optimal policies in general-sum stochastic game is proposed. In [20], the authors extended the model presented in [17] with the intermediate rewards to accelerate learning convergence.

In the first part of this paper, several learning automata based multi-agent system algorithms for finding optimal policies in Markov games are proposed. These algorithms consist of multiple agents that use learning automata in order to optimize their own behavior that can be effectively used to find the optimal policy in Markov games. In all the proposed algorithms, each agent residing in every state of the environment is equipped with a learning automaton. These learning automata control the agents' behavior for the state transitions in such a way that the best solution to maximize the expected reward is found. Each action of the learning automaton corresponds to moving to one of the adjacent states. Each agent moves from one state to another and tries to reach the goal state. In each state, the agent chooses its next transition with help of its learning automaton in that state. The actions taken by learning automata along the path traversed by the agent are then rewarded or penalized using the cost of the traversed path according to a learning algorithm. This process is performed in parallel by all agents and it iterates several times until the path taken by each agent converges to the optimal path.

In the second part of the paper, the concept of entropy [21] has been used to improve the algorithms proposed in the first part of the paper. The concept of entropy is imported into these algorithms with the aim of improving the learning process. To evaluate the performance of the proposed algorithms computer experiments have been conducted. The proposed algorithms have been applied to two grid games as examples of Markov games. The results of experiments have shown that the proposed algorithms perform better than the existing algorithms in terms of cost and the speed of reaching the optimal policy.

The rest of this paper is organized as follows: Section II is a brief review on different types of learning automata. Definitions for Markov decision process and Markov games as well as the concept of solution in them are discussed in Section III. Definition of the entropy concept is discussed in Section IV. In Section V, the proposed algorithm and its variations are presented. In Section VI, simulation results and discussion are reported. Section VII concludes the paper.

II. LEARNING AUTOMATA

Learning automata are adaptive decision-making devices operating on unknown random environments [22]. The learning automaton has a finite set of actions and each action has a certain probability (unknown for the automaton) of getting rewarded by the environment of the automaton. The aim is to learn to choose the optimal action (*i.e.* the action with the highest probability of being rewarded) through repeated interaction on the system. If the learning algorithm is chosen properly, then the iterative process of interacting on the environment can be made to select the optimal action. Figure 1 illustrates how a stochastic automaton works in feedback connection with a random environment. Learning automata can be classified into two main families: fixed structure learning automata and variable structure learning automata (VSLA) [22]. In the following, the variable structure learning automata is described.

Variable structure learning automata can be shown by a quadruple $\{\alpha, \beta, p, T\}$ where $\alpha=\{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ which is the set of actions of the automaton, $\beta=\{\beta_1, \beta_2, \ldots, \beta_m\}$ is its set of inputs, $p=\{p_1, \ldots, p_r\}$ is probability vector for selection of each action, and $p(n+1)=T[\alpha(n), \beta(n), p(n)]$ is the learning algorithm. If $\beta=\{0, 1\}$, then the environment is called *P-Model*. If β belongs to a finite set with more than two values, between 0 and 1, the environment is called *Q-Model* and if β is a continuous random variable in the range [0, 1] the environment is called *S-Model*. Let a VSLA operate in an *S-Model* environment. A general linear schema for updating action probabilities when action i is performed is given by:

$$\begin{aligned} p_i(n+1) &= p_i(n)+a\,(1-\beta(n))(1-p_i(n)) \\ &\quad -b\beta(n)\,p_i(n) \\ p_j(n+1) &= p_j(n)-a\,(1-\beta(n))\,p_j(n) \\ &\quad +b\beta(n)\left[\frac{1}{r-1}-p_j(n)\right] \quad \forall j \;\; j\neq i \end{aligned} \tag{1}$$

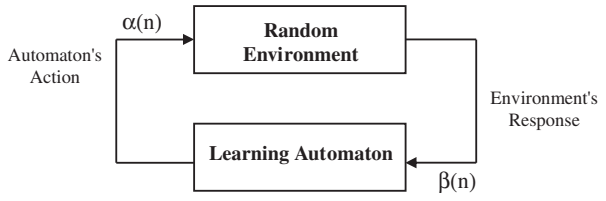

Fig. 1. The interaction between learning automata and environment.

where a and b are reward and penalty parameters. When $a=b$, the automaton is called $S-L_{R-P}$. If $b=0$ and if $0<b<<a<1$, the automaton is called $S-L_{R-I}$ and $S-L_{R-\varepsilon P}$, respectively. The overall operation of learning automaton is summarized in Fig. 2.

Learning automata games. Automata games were introduced to see if automata could be interconnected in useful ways so as to exhibit group behavior that is attractive for either modeling or controlling complex systems [10]. A play $a(t)=(a_1(t)\ldots a_n(t))$ of n automata is a set of strategies chosen by the automata at stage t, such that $a_j(t)$ is an element of the action set of the jth automaton. Correspondingly the outcome is now also a vector $\beta(t)=(\beta_1(t)\ldots\beta_n(t))$. At every time-step, all automata update their probability distributions based on the responses of the environment. Each automaton participating in the game operates without information concerning the number of other participants, their strategies, actions or payoffs. In zero-sum games, the L_{R-I} scheme converges to the equilibrium point if it exists in pure strategies, while the $L_{R-\varepsilon P}$ scheme can arbitrarily close approach a mixed equilibrium [23]. In general non zero-sum games it is shown that when the automata use a L_{R-I} scheme and the game is such that a unique pure equilibrium point exists, convergence is guaranteed [15]. In cases where the game matrix has more than a pure equilibrium, which equilibrium is found depends on the initial conditions. Summarized we have the following:

Theorem 1. When the automata game is repeatedly played with each player making use of the L_{R-I} learning algorithm with a sufficiently small step size, then local convergence is established towards pure Nash equilibrium.

Theorem 2. Players in an n-person non-zero sum game who use independently a reward-inaction update scheme with an arbitrarily small step size will always converge to a pure equilibrium point. If the game has Nash equilibrium, the equilibrium point will be one of the Nash equilibriums.

VSLA Algorithm

Initialize $\boldsymbol{p}$ to [1/r, 1/r,…,1/r] where r is the number of actions
while not done
Select an action i based on the probability vector p
Evaluate the action and return a reinforcement signal β
Update the probability vector using the learning algorithm
end while

Fig. 2. Pseudo code of variable-structure learning automaton.

Theorem 1 guarantees convergence to a pure equilibrium point. If the game has a pure Nash equilibrium, the LA will converge to one of the Nash equilibrium. Nevertheless, Nash equilibrium is often proposed as an interesting solution of a game, there might be other solution concepts of interest depending on the type of the game.

III. MARKOV DECISION PROCESS AND MARKOV GAMES

3.1 Markov decision process

The problem of controlling a finite Markov Chain, for which transition probabilities and rewards are unknown, is called a Markov decision process and can be stated as follows [24]. Let $s=\{s_1, s_2, \ldots, s_N\}$ be the state space of finite Markov chain $\{x_n\}_{n\geq 0}$, and $A^i=\{a_1^i, a_2^i, \ldots, a_{r_i}^i\}$ be the finite set of actions available in state s_i. Each starting state s_i, action choice $a^i \in A^i$, and ending state s_j have an associated transition probability $T^{ij}(a^i)$ and reward $R^{ij}(a^i)$. The goal is to choose the set of actions or policy, $\alpha=\{a^1, a^2, \ldots, a^n\}$, with $a^j \in A^j$ that maximizes the expected average reward $J(\alpha)$ as follows:

$$J(\alpha)=\lim_{l\to\infty}\frac{1}{l}E\left[\sum_{t=0}^{l-1} R^{x(t)x(t+1)}(\alpha)\right] \tag{2}$$

where $R^{x(t)x(t+1)}(\alpha)$ is the reward generated by a transition from $x(t)$ to $x(t+1)$ using policy α. The set of policies is limited in this formulation to stationary and nonrandomized policies. Hence, under the assumption that the Markov chain corresponding to each policy is ergodic, the best strategy in each state is a pure strategy and is independent of the time at which the state is occupied [24].

3.2 Markov games

Markov games are a generalization of MDPs to multiple agents and can be used as a framework for investigating multi-agent learning [13]. In a Markov game, actions are the joint action which is the result of joint action selection of all agents, while rewards and the state transitions depend on these joint actions [9]. The action set available for agent $k(1\leq k\leq n)$ in state s_i is $A_k^i=\{a_{k_1}^i, a_{k_1}^i, \ldots, a_{k_{i_r}}^i\}$, n being the total number of agents present in the system. Transition probabilities $T^{ij}(a^i)$ and rewards $R_k^{ij}(a)$, depend on a starting state s_i, ending state s_j and joint action from state s_i, *i.e.* $a^i=(a_1^i, \ldots, a_n^i)$ with $a_k^i \in A_k^i$. The reward function $R_k^{ij}(a)$ for each agent k is individual, meaning that different agents can receive different rewards for the same state transition. The objective for each agent in the game is to find a policy which maps each state to a strategy in order to maximize its reward. Since each agent k has its own individual reward function, defining a solution concept becomes difficult.

Markov games are categorized based on the agents' rewards into cooperative and non-cooperative games. Non-cooperative games may be classified as competitive games and general-sum games. Strictly competitive games, zero-sum games, are two-player games where one player's reward is always the negative of the others'. General-sum games are the ones whose the reward sum is not restricted to zero or any constant and allow the agents' rewards to be arbitrarily related. However, in full cooperative games, team games, rewards are always positively related. In a fully cooperative MG (or team MG) called a multi-agent MDP (or MMDP), all agents share the same reward function and optimal policies are obtained [9]. Nevertheless, in general MG there is no constraint on the sum of the agents' rewards and the agents should learn to find and agree on the same optimal policy. Since each agent has its own individual reward function, finding an optimal policy becomes difficult. Instead, an equilibrium point is sought. In this situation no agent can improve its reward by changing its policy if all other agents keep their policy fixed. The baseline solution concept for general-sum games is the Nash equilibrium. Nash equilibrium is a strategy profile from which none of the players has any incentive to deviate. The strategies that constitute Nash equilibrium can be stationary strategies. In [25] it is shown that every discounted Markov game possesses at least one Nash equilibrium in stationary strategies.

3.3 Control of markov game using learning automata

The problem of controlling single-agent MDPs can be modeled as a network of interconnected learning automata in which the control is transferred from one learning automaton to another [24]. Each state in MDP has a learning automaton that tries to learn the optimal probability distribution of actions during the process. Agents move on this network and in each state, they get help from the learning automaton assigned to that state to move to the next state. This is done by using the probability vector of the corresponding learning automaton. In this model, only one learning automaton is active at each time and the transition from one state to another will activate the learning automaton of that

state. The activated learning automaton LA^i in state i will not be informed about the immediate reward $r^{ij}(a^i)$ yielded from its action a^i in transition from state i to state j. Instead, when state i is visited again, LA^i receives two parts of data: the cumulative reward from the beginning of the process up to the current time step and current global time. Using these data, LA^i calculates the reward received since the last visit of state i and the corresponding elapsed global time. Then, the input to LA^i is calculated using the following equation:

$$\beta^i(t_i+1)=\frac{\rho^i(t_i+1)}{\eta^i(t_i+1)} \tag{3}$$

where $\rho^i(t_i+1)$ is the cumulative total reward generated for action a_i in state i and $\eta^i(t_i+1)$ is the cumulative total elapsed time. This process continues until all probability vectors converge or a pre-specified condition is met. When the number of agents increase and the model extends to multi-agent case, more than one learning automaton should be active simultaneously because the states depend on the problem and the environment and the agents could be in different states.

In a Markov game, actions are the joint action which is the result of joint action selection of all agents. The network of learning automata applied to MDP is extended to Markov game by putting a learning automaton for each agent in each state instead of putting a single learning automaton in each state of the system [17]. At each time step only the automata of one state are active; a joint action triggers the LA from that state to become active and takes some joint action. As MDP, the activated LA LA^i_k, for agent k in state i is not informed of the one-step reward, $r^{ij}(a^i)$ that resulted from choosing a joint action $a^i=(a^i_1,\ldots,a^i_n)$ with action a^i_k in state i and leading to state j. When state i is visited again, all learning automata LA^i_k receive two pieces of data: the cumulative reward generated by the process up to the current time step and the current global time. The environment responses or the input to LA^i_k is exactly the same as in Equation (3).

3.4 Grid game environment as a markov game

One of the non-competitive Markov games used to test multi-agent learning is the Grid Game. This game has different varieties all of which are two player general sum games. Two types of Grid Game are illustrated in Fig. 3. The first one, Grid Game 1 (GG1), is a version of the Chicken game having several states and one goal [26]. The second one, Grid Game 2 (GG2), a Markov game that is reminiscent of Bach or Stravinsky [8]. In GG2, there is one goal and two

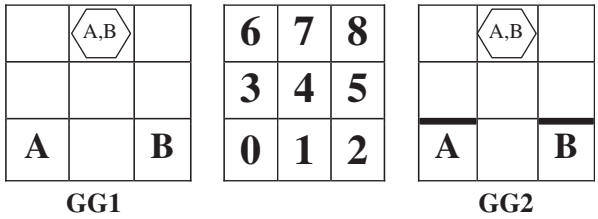

Fig. 3. Two types of Grid Game and illustration of game coordinates.

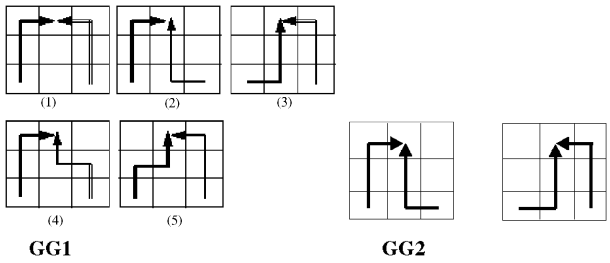

Fig. 4. Optimal solutions to two Grid Games.

barriers: if an agent attempts to move through one of the barriers, then with probability 1/2 this move fails. Players' actions are defined as four actions in four different directions namely up, down, left and right. State space set is defined as $S=\{s|s=(l_1,l_2)\}$, in which each state $S=\{s=(l_1,l_2)\}$ indicates the coordinates of agents 1 and 2. In GG1, the state transitions are deterministic, *i.e.* the next state is uniquely determined by the current state and the joint action of the agents. In the second game, GG2, most state transitions are deterministic except in the following case: if an agent chooses 'up' from position (0, 2) or (2, 0), it moves up with probability 0.5 and remains in its previous position with probability 0.5. In both games, two agents start from two bottom corner of the page and try to reach goal with the least possible number of moves. Agents cannot take the same coordinates at the same time. That is, if both agents try to move to the same square, both of their moves will fail. If agents move to two different non-goal positions, both receive zero rewards and if one reaches the goal position, it receives 100 units of reward. However, if they collide with each other both receive one unit of punishment and stay in their previous position. In these games, agents are assumed not to know the goal position and the other agent's reward functions. Agents choose their actions simultaneously and can only know about the previous moves of the other agents and their current state (the joint position of both agents). They also observe the immediate rewards after both agents choose their actions. Figure 4 illustrates the optimal solution to games GG1 and GG2.

IV. THE ENTROPY CONCEPT

Entropy is a significant concept in thermodynamics, representing the degree of disorder in a thermodynamic system that plays an important role in various fields of computer science, such as coding theory, learning, compression, and others [27, 28]. Shannon has introduced this concept into information theory, by the name of information entropy. Entropy, in its basic form, indicates a measure of uncertainty rather than a measure of information. More specifically, the information entropy is a case of the entropy of random variables defined as follows [29]:

$$H(X) = -\sum_{X \in \chi} P(X) \log(P(X)) \tag{4}$$

where X represents a random variable with set of values χ and probability mass function $P(\mathrm{X})$. Entropy is always a positive value and can change bases freely as $H_b(X) = \log_b(a).H_a(X)$. For the random variable x and with $\log(x)$ tending to zero as x tends to zero. Entropy measures the uncertainty inherent in the distribution of a random variable. The entropy of random variables has problem-dependent meanings in different applications. In this paper, entropy is introduced as a measure of learning process in discrete state space in which each state s_i is assigned a set of decision probability values. For each state s_i there is a corresponding random variable D_i representing the probability distribution of decision probability values in the state. In the learning process, the entropy of D_i represents the uncertainty of the decision under state s_i. The larger D_i's entropy causes the more uncertain the decision maker. In any particular state s_i a local entropy which represents the uncertainty of the decision in that state is defined as follows [21]:

$$H_{\mathrm{local}}(s_i) = -\sum_{j=1}^{N} p_j(s_i) \log(p_j(s_i)) \tag{5}$$

where N is the number of the elements in the action set and $P_j(s_i)$ is the probability to select the j-th action under state s_i. The local entropy can only represent the uncertainty of the decision for a single state. According to all states' local entropy and to evaluate overall the uncertainty of the whole system, the global entropy concept is introduced as the average of all the entropies [21]:

$$\begin{aligned} H_{\mathrm{global}}(s_i) &= -\frac{1}{M} \sum_{i=1}^{M} H_{\mathrm{local}}(s_i) \\ &= -\frac{1}{M} \sum_{i=1}^{M} \sum_{j=1}^{N} p_j(s_i) \log(p_j(s_i)) \end{aligned} \tag{6}$$

where M is the number of the elements in the state set. In reinforcement learning, the process of optimization increases the probability values of selecting optimal actions, *i.e.* the initial random strategy will converge to a specific optimal strategy. Hence, the uncertainty of both the strategy and the global entropy will decrease. Therefore, the entropy can represent the degree of convergence in the learning process [21].

V. THE PROPOSED ALGORITHMS

In this section, several learning automata based multi-agent system algorithms for finding optimal policies in Markov games are proposed. The multi-agent system considered in this paper consists of a set of n agents, an environment that consists of a finite set of states $S = \{s_1, s_2, \ldots, s_m\}$, and a finite set of actions A_i for every agent i. The action selected by agent i is denoted by $a_i \in A_i$ so that the joint action $a \in A = A_1 \times \cdots \times A_n$ is the vector of all individual actions. There is a state transition function $T : S \times A \times S \to [0, 1]$ which gives the transition probability $p(s'|a, s)$ when the joint action a is performed in state s, the system moves to state s' while the reward function $R_i : S \times A \to R$ provides agent i with a reward $r_i \in R_i(s, a)$ based on the joint action a taken in the state s.

In the proposed multi-agent system algorithms, in each state i of the environment of the game and for each agent k, $k : 1 \ldots n$, a learning automaton LA_k^i is placed. LA_k^i in state i tries to learn the optimal action probability of agent k. The number of adjacent states (neighbors) determines the number of actions of each learning automaton in each state and every joint action corresponds to a transition to an adjacent state. The state with the highest payoff value is the goal state. At first, all learning automata in all states choose their actions with equal probabilities. The agents start from 'start state' and move toward 'goal state'. The selected actions of learning automata in each state determine the next state of the agents. This joint action activates the learning automata in the next state. The process of choosing joint action, moving to the next state and activating automata in this state is repeated until the goal state is reached. If the joint action taken by learning automata in state s causes agents to move to the goal state, the cumulative reward of reaching the goal state from the start state for agent k, (R_k^G), is computed. $A\pi_k$, the average reward received by agent k per move along the path taken by agent k is then computed by dividing R_k^G by the length of traversed

Algorithm 1

Initialize: *In each state, a Learning Automaton of type P for each agent is placed. The set of actions of this LA is the set of permissible movements to other states.*

for *all states s, agents k* ***do***
  P(s, k) =1/number of permissible actions for agent k
end for
for *every agent k* ***do***
  Set dynamic threshold T_k *to zero*
end for
m = 1 // m counts the number of paths traversed by agents from starting state to goal state//
Repeat
  OldState = start state //
  While *OldState is not the goal state or a certain number of moves are made by the agent* ***Do***
    Joint Action =∅;
    for *every* ***agent*** *k* ***do***
      Action = Select Action (P(Old State, k))
      LastAction (Old State, k)=Action
      JointAction = Joint Action ∪ Action
    end for
    NewState= GetNextState (Old State, JointAction)
    if *NewSate = Goal State* ***then***
      for *every agent k* ***do***
        Compute *the average reward received by agent k during traversal of path* π_k ($A\pi_k = R_k^G / L\pi_k$)
          // R_k^G *is the cumulative reward of reaching the goal state for agent k,* $L\pi_k$ *is the length of the traversed path* π_k *taken by agent* along path π_k until it reaches the goal state //
        if $A\pi_k >= T_k$ ***then*** *give reward to all the actions taken by learning automata along path* π_k
        else *Penalize all the actions taken by learning automata along path* π_k
        end if
        $T_k(m) \leftarrow T_k(m-1) + (A\pi_k(m) - T_k(m-1))/m$
      end for
      m = m + 1
    end if
    OldState = NewState
  end while
Until *the path probability is greater than a certain threshold or a pre-specified numbers of paths (episodes)are Traversed*

Fig. 5. The proposed Algorithm 1.

path π_k taken by agent k, ($L\pi_k$). Then $A\pi_k$ is compared with a quantity called the dynamic threshold of agent k, T_k. Depending on the result of the comparison, all the learning automata along the path update their action probabilities. Updating is carried out in direction from starting state to goal state or vice versa. If the $A\pi_k$ is greater than or equal to the T_k, then the actions chosen by all learning automata along that path receive reward otherwise they receive penalty. This process stops if the path probability is greater than a certain threshold or a pre-specified numbers of paths (episodes) traversed. The path probability is defined as the product of the probability of choosing actions of learning automata in the states of the traversed path. Dynamic threshold T_k is updated every time that agent k reaches the goal state. In the mth entrance to the goal state (stage m) $T_k(m)$ is updated according to Equation (7) where $A\pi_k(m)$ is the average reward of agent k in its mth entrance to the goal state. Dynamic threshold for every agent is in fact the average of the average reward per move for agent k in traversed paths from start state to goal state by that agent.

$$T_k(m) = T_k(m-1) + (A\pi_k(m) - T_k(m-1))/m \quad (7)$$

The proposed algorithm, called Algorithm 1, is given in more detail in Fig. 5. Algorithm 1 can be improved in several ways some of which are described below.

Continuous valued feedback for learning Automata. In Algorithm 1, all learning automata are operating in *P-model* environment in which the output of the environment is defined to be binary. An improvement of Algorithm 1 which we call Algorithm 2, can be obtained by allowing the output of the environment to take a continuous range of values measuring the closeness of the traversed path to the optimal path. In Algorithm 2, if the average reward for agent k($A\pi_k$) is greater than or equal to T_k then the selected actions

of all learning automata along path π_k will be updated according to (8).

$$\begin{aligned} p_i^k(n+1) &= p_i^k(n)+a\left[1-\left(\frac{L\pi_k}{R_k^G}\right)\right](1-p_i^k(n)) \\ p_j^k(n+1) &= p_j^k(n)-a\left[1-\left(\frac{L\pi_k}{R_k^G}\right)\right]p_j^k(n) \quad \forall j \quad j\neq i \end{aligned} \tag{8}$$

Variable learning rate. In Algorithm 1, all the learning automata use the same fixed learning rate. This gives an equal degree of importance to all states along the path being traversed, which may not be appropriate. States which are closer to the starting state should be given higher degree of participation in the optimal path (higher probability of being part of the optimal path) than those which are far from the starting node. Therefore, the amount of the reward given to an action taken at a particular time must increase as we approach the goal state. To achieve this, we halve the learning rate while traversing back toward the starting state. Algorithm 1 in which this modification is made will be called Algorithm 3.

Modified Dynamic Threshold. In Algorithm 1, the dynamic threshold for each agent is the average reward received by the agent along the path leading to the goal state (7). Algorithm 4 is obtained from Algorithm 1 by changing the definition of dynamic threshold, $T_k(m)$, which is the threshold for agent k in its mth entrance to the goal state. $T_k(m)$, is updated according to (9) where $L\pi_k$ is the length of the traversed path taken by the agent during the mth episode at the end which the mth entrance to the goal state occurs.

$$T_k(m) = T_k(m-1)+(A\pi_k(m)-T_k(m-1))/L\pi_k \tag{9}$$

Using Entropy to Improve Learning. Another improvement can be obtained by importing the concept of entropy into each of the above mentioned algorithms that is Algorithms 1 through 4. We call these new algorithms, Algorithms 5 through 8. For instance, Algorithm 5 is obtained from Algorithm 1 by importing the concept of entropy into Algorithm 1. In Algorithm 5 if the set of selected actions (joint action) of learning automata in state s_i leads to state s_j (except the goal state), then the selected action of the learning automaton for agent k in state s_i, $(LA_k^{s_i})$ is updated based on the entropy of the probability vector of the learning automaton of agent k in state $s_j(LA_k^{s_j})$. Otherwise, it is updated as specified in Algorithm 1. Entropy of the action probability vector measures the degree of uncertainty that the learning automaton encounters. A high value of entropy indicates that the learning automaton does not have useful information about where the goal state is and chooses its action randomly. On the contrary, a low value for entropy indicates that the learning automaton has useful information about the goal state. If $P_k^s = \{p_k^s(1), p_k^s(2), \ldots, p_k^s(r)\}$ is the action probability vector of a learning automaton in state s for agent k with r actions, then the entropy of this probability vector, H_k^s, is computed according to (10) as given below. P_k^s denotes the action probability vector for LA_k^s.

$$H_k^s = \sum_{j=1}^{r} p_k^s(j)\log(p_k^s(j)) \tag{10}$$

Entropy has its maximum value when all the actions have equal probabilities of selection and has value zero (its minimum) when the action probability vector is a unit vector. In order to be able to use entropy as a reinforcement signal for *S-Model* variable structure learning automata, the entropy needs to be rescaled in the range of [0,1]. Suppose that agent k is in the state s and its learning automaton, that is LA_k^s, leads the agent to the state s'. In this case, reinforcement signal (β_k^s) as given in (11) is then used in (1) to update the probability vector of A_k^s.

$$\beta_k^s = \frac{H_k^{s'}}{\mathrm{Max}(H_k^{s'})} \tag{11}$$

where $\mathrm{Max}(H_k^{s'})$ is maximum entropy in state s' for agent k defined as:

$$\mathrm{Max}(H_k^{s'}) = \sum_{j=1}^{r(k)} \frac{1}{r(k)}\log\left(\frac{1}{r(k)}\right) = \log_2^{r(k)} \tag{12}$$

Table I gives a summary of the proposed algorithms.

In what follows we use Fig. 6 to describe the behavior of Algorithm 1 in more detail. It is assumed that two agents are in each state and state s_8 is the goal state. In the mth entrance, learning automata of agent 1 $LA_1^{s_1}$ has two actions $\{0, 1\}$ and $LA_2^{s_1}$ has two actions $\{0, 1\}$. Let $LA_1^{s_1}$ chooses its action and learning automaton of agent 2, $LA_2^{s_1}$ chooses its action in state s_1. This joint action (0, 1) activates the state s_2 and

Table I. A summary of the proposed algorithms.

The property used in Algorithm	Without entropy	With entropy
Binary feedback for LA	Algorithm 1	Algorithm 5
Continuous valued feedback	Algorithm 2	Algorithm 6
Variable learning rate	Algorithm 3	Algorithm 7
Modified dynamic threshold	Algorithm 4	Algorithm 8

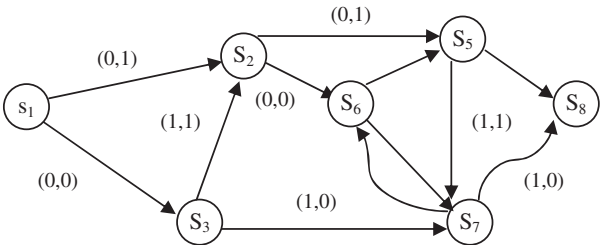

Fig. 6. An example of state diagram.

the learning automata corresponding to agents in this state. The joint action (0,0) in state s_2 activates state s_6 and so on. The process of selection of joint action and activating learning automata in states is repeated until state 8 is reached or for some reasons certain number of moves are made by the agents. Assume that in Fig. 6, the path with tick lines is traversed for agent 1, that is, $\pi_1 = (s_1, s_3, s_5, s_7, s_8)$ After the destination (goal) state is reached, $A\pi_1$, the average reward received by agent 1 per move along the path taken by agent computed and then compared with the dynamic threshold of agent 1, T_1. Depending on the result of the comparison, all the learning automata (except in state 8) along the path update their action probabilities. If the $A\pi_1$ is greater than or equal to the T_1, then the actions chosen by all learning automata along that path receive reward otherwise they receive a penalty.

Assume that up to the stage m (the mth entrance to the goal state), path π_k taken by agent k is selected. Let $A\pi_k$, be the average reward received by agent k per move along the path taken by agent k and T_k be the dynamic threshold of agent k, which is average of the average reward per move for agent k in traversed paths from start state to goal state by that agent. Let $q_k(m)$ be the probability of traveling along path π_k by agent k in stage m and If $P_k^s = \{p_k^s(0), p_k^s(1), \ldots, p_k^s(r-1)\}$ is the action probability vector of a learning automaton in state s for agent k with r actions then $q_1(m)$ which is the probability of traveling along path $\pi_1 = (s_1, s_2, s_5, s_7, s_8)$ in stage m is computed as:

$$q_1(m) = p_1^{s1}(0).p_1^{s2}(0).p_1^{s5}(1).p_1^{s7}(1) \tag{13}$$

In Section VI through simulation we show that the algorithm converges to the optimal path with a probability very close to unity.

VI. SIMULATION RESULTS AND DISCUSSIONS

To evaluate the performance of the proposed methods, several experiments have been conducted whose results are reported below. The Markov games considered in the experiments are borrowed from [8, 26] For all experiments, each reported value is obtained by averaging over 100 runs. It is assumed that at the beginning of an episode, the agents start from the position (0 2). The *POP* curve gives the plot of the values of probability of the optimal path (*POP*) during the process of learning the optimal path. Probability of the optimal path is computed at the end of every episode and is the product of probabilities of selection of actions of learning automata along the optimal path at the time when the episode ends. A point with coordinate (x, y) on the POP curve provides us with the average number of required number of iterations for obtaining the optimal path with a specific probability. Each algorithm terminates when probability of the optimal path reaches 0.93 or a predetermined number of episodes are run. This predetermined number of episodes for GG1 is 2000 and for GG2 is 4000.

Experiment 1. This experiment is conducted to study the impact of learning parameter a of L_{R-I} learning algorithm on the convergence behavior of Algorithm 1 when applied to both GG1 and GG2. For this experiment, learning rate takes values 0.01, 0.05, 0.1 and 0.4. POP curves for agent 1 in GG1 and GG2 are given in Fig. 7. Each point in this plot is an average taken over the converged runs. A run converges when the value of *POP* reaches a predefined threshold within in a predetermined number of episodes. For example, it can be seen that in terms of accuracy the best result is obtained when $a = 0.01$. The figures also show that when we increase the learning parameter we gain higher speed but instead we lose accuracy.

Experiment 2. This experiment is performed in order to study the impact of learning parameter a on the amount of reward received by agent 1 during an episode when Algorithm 1 is used. For this purpose, we plot the reward received by agent 1 per episode for different values of learning parameter a: 0.01, 0.05, 0.1, and 0.4. Figure 8 illustrates the result of this experiment for both GG1 and GG2. From the result, it is evident that the value of parameter $a$has a large impact on the performance of Algorithm 1. In this experiment, the best result is obtained for both GG1 and GG2 when a is set to 0.1.

Experiment 3. This experiment is designed to compare Algorithms 1through 4 with respect to speed and accuracy of convergence. To do this, we evaluate these algorithms in terms of: (i) the probability of optimal path (*POP*) when the algorithm terminates; (ii) the number of movements made by agent 1 to reach the optimal path (MOV); and (iii) the total number of

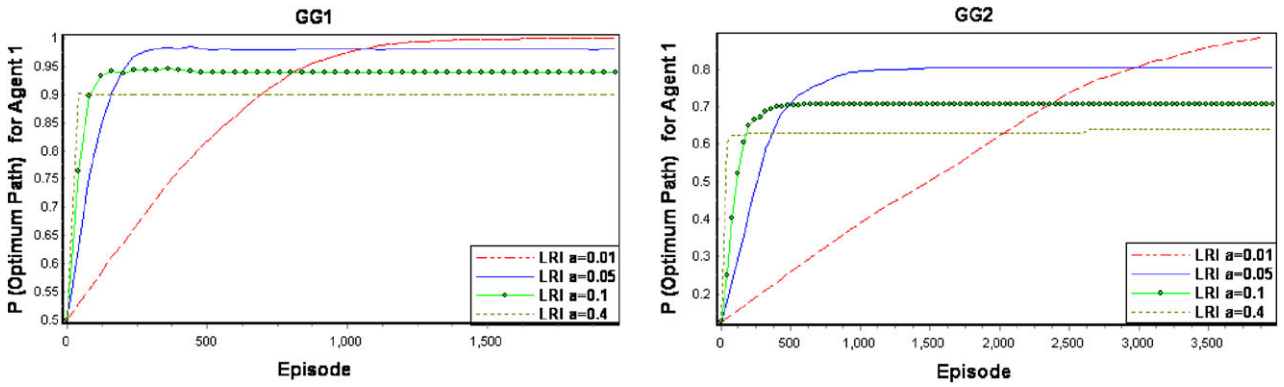

Fig. 7. The impact of learning rate on POP curve for agent 1 in GG1 and GG2.

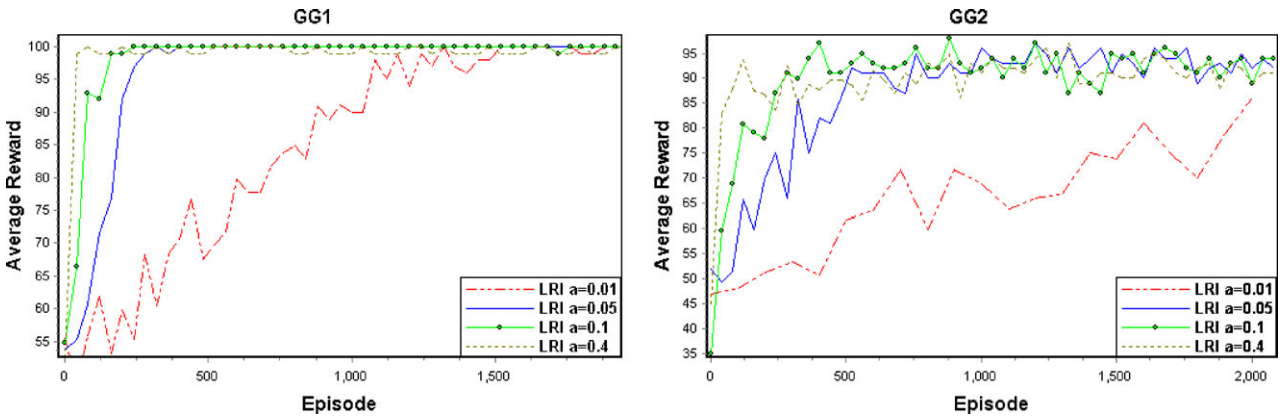

Fig. 8. The impact of learning rate on the reward received by agent 1 in GG1 and GG2.

Table II. The simulation results of proposed algorithms for GG1 in 2000 steps.

	Algorithm 1			Algorithm 2			Algorithm 3			Algorithm 4		
a	POP	MOV	COL	POP	MOV	COL	POP	MOV	COL	POP	MOV	COL
0.01	***0.93***	***7700***	***512***	***0.94***	***7761***	***540***	**0.75**	**7580**	**890**	**0.98**	**7740**	**560**
0.05	*0.84*	*7960*	*112*	*0.86*	*7970*	*120*	*0.78*	*7934*	*197*	*0.94*	*7956*	*120*
0.1	*0.82*	*7988*	*60*	*0.82*	*7988*	*61*	*0.75*	*7970*	*100*	*0.81*	*7977*	*68*
0.2	*0.68*	*8000*	*31*	*0.68*	*7998*	*29*	*0.7*	*7985*	*52*	*0.77*	*7993*	*39*
0.4	***0.5***	***8100***	***18***	***0.48***	***8100***	***20***	***0.56***	***8000***	***30***	***0.67***	***8037***	***25***

collisions occurred during the learning process (COL). Learning parameter a for all algorithms is set to 0.01. Tables II and III summarize the results of this experiment. Every value in both tables is the average over 100 runs. From the results, the following points can be made.

1. Algorithm 4 has the highest accuracy for both games. The algorithms in increasing order of their accuracy are 4, 2, 1, and 3 for GG1 and 4, 1, 3, and 2 for GG2.
2. Maximum POP value is 0.93 which is obtained for Algorithm 4 when $a = 0.01$. For this case the number of movements is 7,700 while the number of collisions is 512.
3. To reach the same accuracy for GG2 the algorithms need to make more moves. This can be contributed to the fact that in GG2, due to the

Table III. The simulation results of proposed algorithms for GG2 in 4000 steps.

	Algorithm 1			Algorithm 2			Algorithm 3			Algorithm 4		
a	POP	MOV	COL	POP	MOV	COL	POP	MOV	COL	POP	MOV	COL
0.01	***0.82***	***15519***	***627***	***0.85***	***15 420***	***700***	***0.85***	***14 961***	***995***	***0.85***	***15 519***	***630***
0.05	*0.81*	*15 430*	*123*	*0.79*	*15 438*	*125*	*0.95*	*15 688*	*210*	*0.82*	*15 620*	*125*
0.1	*0.78*	*15 440*	*57*	*0.76*	*15 497*	*60*	*0.93*	*15 715*	*100*	*0.79*	*15 660*	*58*
0.2	*0.56*	*15 570*	*22*	*0.52*	*16 190*	*22*	*0.9*	*15 734*	*55*	*0.63*	*15 676*	*24*
0.4	***0.5***	***15 800***	***13***	***0.45***	***17 295***	***10***	***0.7***	***15 739***	***30***	***0.52***	***15 849***	***14***

Table IV. The simulation results for experiment 4 for GG1.

	Algorithm 1			Algorithm 5			Algorithm 2			Algorithm 6		
a	POP	MOV	COL	POP	MOV	COL	POP	MOV	COL	POP	MOV	COL
0.01	***0.93***	*7700*	*512*	***0.96***	*7776*	*483*	***0.94***	*7761*	*540*	**0.957**	7785	490
0.05	*0.84*	*7960*	*112*	*0.86*	*7992*	*98*	*0.86*	*7970*	*120*	0.87	7974	100
0.1	*0.82*	*7988*	*60*	*0.84*	*7986*	*48*	*0.82*	*7988*	*61*	0.845	7977	51
0.2	*0.68*	*8000*	*31*	*0.72*	*7689*	*25*	*0.68*	*7998*	*29*	0.73	7418	29
0.4	*0.5*	*8100*	*18*	*0.55*	*7334*	*14*	*0.48*	*8100*	*20*	0.56	7089	16
	Algorithm 3			**Algorithm 7**			**Algorithm 4**			**Algorithm 8**		
a	POP	MOV	COL	POP	MOV	COL	POP	MOV	COL	POP	MOV	COL
0.01	**0.75**	7580	890	***0.92***	*7654*	*700*	0.98	7740	560	**0.999**	7779	520
0.05	0.78	7934	197	*0.80*	*7941*	*149*	0.94	7956	120	0.96	7952	115
0.1	0.75	7970	100	*0.77*	*7984*	*74*	0.81	7977	68	0.83	7977	65
0.2	0.7	7985	52	*0.73*	*7992*	*37*	0.77	7993	39	0.79	7690	37
0.4	**0.56**	**8000**	**30**	***0.61***	***7693***	***25***	**0.67**	**8037**	**25**	**0.7**	**7142**	**24**

existence of barriers in the environment, agents may need to make extra moves in order to bypass the barriers.

4. For all algorithms, when learning parameter a increases, the number of collisions (*COL*) decreases and the number of movement (*MOV*) increases.
5. For GG2 all algorithms have almost the same rate of convergence whereas for GG1 Algorithms 1, 2, and 4 have almost the same rate of convergence and Algorithm 3 perform worse.
6. The highest *POP* value for Algorithms 1, 2 and 4 decreases as the learning parameter $a$increases. For both GG1 and GG2, all algorithms except Algorithm 3 attain their highest *POP* values when a is set to 0.01. The highest *POP* value for Algorithm 3 is obtained when a is set to 0.05.
7. Total number of collisions for Algorithm 3 is higher than those for Algorithms 1, 2, and 4.
8. Algorithm 2 outperforms Algorithm 1 in terms of convergence to the optimal path (POP) but it has higher number of collisions and movements.
9. Algorithm 3 when applied to GG**2** improves the probability of the optimal path but increases the number of collisions.
10. Algorithm 4 has a higher rate of convergence as compared to Algorithms 1 and 2.

Experiment 4. In this experiment we compare algorithms 1 through 4 with algorithms 5 through 8 with respect to three parameters: (i) the probability of optimal path (*POP*) when the algorithm terminates; (ii) the number of movements made by agent 1 to reach the optimal path (MOV); and (iii) the total number of agent collisions during the learning process (COL). Learning parameter a for all algorithms is set to 0.01. This experiment is conducted for two games GG1 and GG2. Table IV presents the results for GG1 when the learning process continues for 2,000 episodes and Table V presents the results for GG2 when the learning process continues for 4,000 episodes. Every value in these tables is averaged over 100 runs. From the

Table V. The simulation results for experiment 4 for GG2.

	Algorithm1			Algorithm 5			Algorithm 2			Algorithm 6		
a	POP	MOV	COL	POP	MOV	COL	POP	MOV	COL	POP	MOV	COL
0.01	*0.82*	**15 519**	*627*	0.95	16 490	780	*0.85*	*15 420*	*700*	**0.98**	16 340	840
0.05	*0.81*	*15 430*	*123*	0.88	14 370	158	*0.79*	*15 438*	*125*	0.85	14 490	156
0.1	*0.78*	*15 440*	*57*	0.73	12 880	71	*0.76*	*15 497*	*60*	0.77	12 300	77
0.2	*0.56*	*15 570*	*22*	0.53	10 165	29	*0.52*	*16 190*	*22*	0.51	10 640	33
0.4	*0.5*	*15 800*	*13*	0.45	9870	17	*0.45*	*17 295*	*10*	0.25	10 120	14
	Algorithm 3			**Algorithm 7**			**Algorithm 4**			**Algorithm 8**		
a	POP	MOV	COL	POP	MOV	COL	POP	MOV	COL	POP	MOV	COL
0.01	*0.85*	*14961*	*995*	**0.99**	16300	930	*0.85*	*15519*	*630*	**0.98**	16400	830
0.05	*0.95*	*15688*	*210*	0.96	15780	160	*0.82*	*15620*	*125*	0.84	14060	175
0.1	*0.93*	*15715*	*100*	0.86	15150	77	*0.79*	*15660*	*58*	0.74	11400	70
0.2	*0.9*	*15734*	*55*	0.76	12164	38	*0.63*	*15676*	*24*	0.53	10177	45
0.4	***0.7***	***15739***	***30***	**0.32**	**9540**	**23**	***0.52***	***15849***	***14***	**0.38**	**7020**	**28**

Table VI. The percentage of improvement in performance when entropy is used.

The algorithms	GG1 Percentage of improvement in performance (%)	GG2 Percentage of improvement in performance (%)
Algorithm 5 to Algorithm 1	3	15.8
Algorithm 6 to Algorithm 2	1	15.2
Algorithm 7 to Algorithm 3	22.5	16.4
Algorithm 8 to Algorithm 4	1	15.2

result of this experimentation, the following points can be made.

1. Algorithms 5 through 8 which use the entropy concept outperform their counterparts in terms of highest value obtained for *POP* (the value of POP when the algorithm terminates). This is because utilization of entropy improves the process of selection of policy at each transition.
2. For Algorithm 5 through 8 we obtain a fewer number of collisions and higher number of moves as they are compared with Algorithms 1 through 4 (Tables V and VI).
3. Simulation results show that in Algorithms 5–8, the *POP* and *MOV* values decrease as the learning parameter *a* increases. Note that when learning rate is greater than 0.1, Algorithms 5 through 8 no longer perform better.
4. The value of COL for Algorithm 7 is much greater than the value of COL for Algorithms 5, 6, and 8.

Experiment 5. In this experiment, we compare mbox-Algorithms 1 through 4 with Algorithms 5 through 8 in order to study the impact of entropy on the amount of reward received by agent 1 during an episode for both games GG1and GG2. Again, like experiment 3, learning parameter *a* is chosen in such a way that the best result can be obtained. Figure 9 shows the result of this experiment. As seen, again the entropy based version of an algorithm performs better than its counterpart which does not use the concept of entropy. In Table VI we have also shown how much greater performance will be gained if the concept of entropy is used.

Experiment 6. In this experiment, the best three of the proposed algorithms are compared with two existing learning automata based algorithms, ILA [7] and Vrancx *et al.* [17]. Each algorithm terminates when probability of the optimal path reaches 0.93. The learning rate parameter for all algorithms is set to 0.01. Figs 10 and 11 show the POP curves of all algorithms for GG1 and GG2, respectively. The results show that for GG1 Algorithm 8 and for GG2 Algorithm 7 outperform the other algorithms in terms of the speed of reaching the optimal policy. Table VII gives the values of *POP*, *MOV*, *COL* for GG1 and GG2. The maximum number of episodes for GG1 and GG2 are set to 2,000 and 4,000 steps, respectively. As it shown the highest value of POP for GG1 is obtained by Algorithm 8 and for GG2 is obtained by Algorithm 7. Note that ILA and Vrancx algorithms have lower numbers of moves and higher number of collusion comparing with Algorithms 6, 7, and 8.

Experiment 7. In this experiment, we compare the proposed algorithms with the existing learning automata

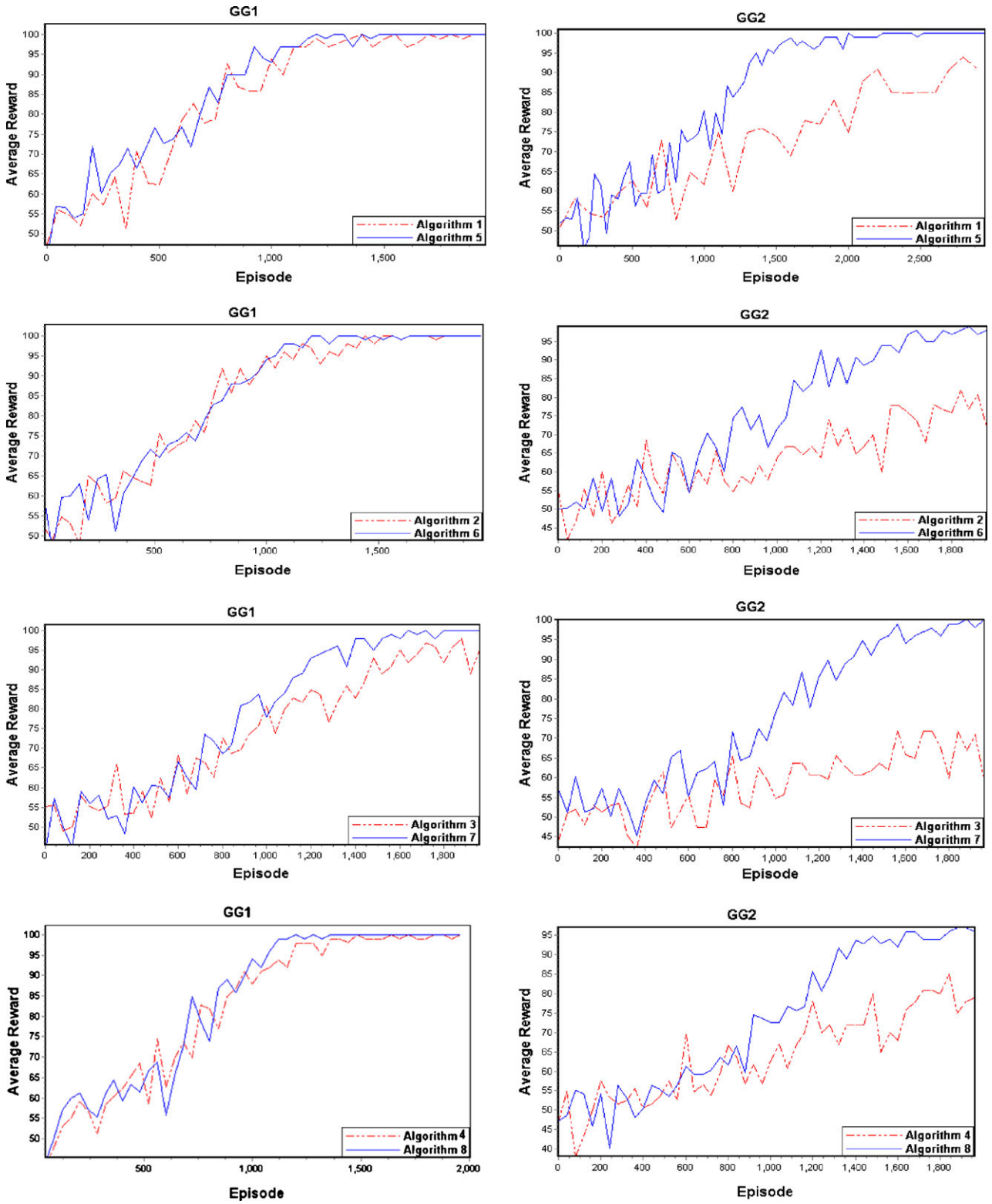

Fig. 9. Comparison of Algorithms 1–4 with 5–8 in terms of average reward received by agent 1.

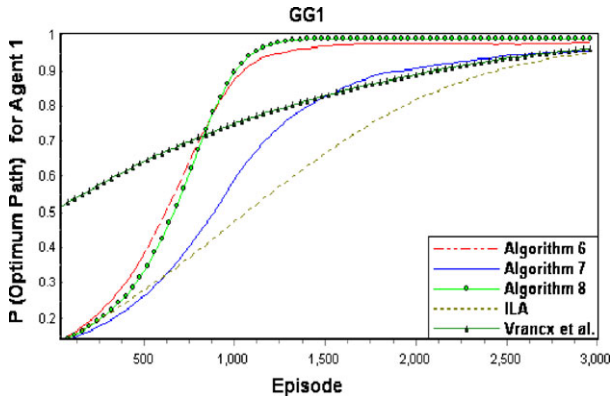

Fig. 10. POP curves for different methods for GG1.

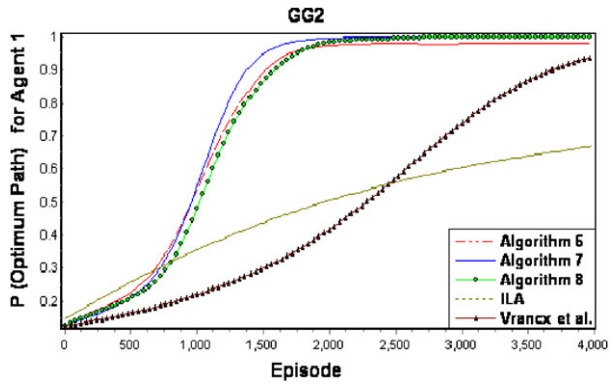

Fig. 11. POP curves for different methods for GG2.

Table VII. Comparison of the proposed algorithms with Vrancx and ILA algorithms.

	Parameter					
	GG1			GG2		
Algorithm	POP	MOV	COL	POP	MOV	COL
Algorithm 6	0.957	7785	490	0.98	8500	840
Algorithm 7	*0.92*	*7654*	*700*	0.99	8200	900
Algorithm 8	0.999	7779	520	0.98	8400	830
Vrancx *et al.*	0.88	4978	1300	0.70	6750	1400
ILA	0.8	3950	750	0.60	5060	870

based algorithms in terms of execution time. Each algorithm terminates when probability of the optimal path reaches 0.93. Learning rate parameter for all algorithms is set to 0.01. Each reported value is obtained by averaging over 100 runs. We have measured the execution time using a laptop computer with processor Intel Core 2 Duo with a clock frequency of 1.66 Hz which supports Microsoft Windows XP Media Center Edition 2005 on 1 GB RAM. Table VIII shows the results of this experiment. As is shown, Algorithms 5 through 8 which use entropy, require more execution time than their counterparts. All the proposed algorithms run faster than ILA and Vrancx algorithms.

Table VIII. Comparison of the proposed algorithms with Vrancx and ILA algorithms in term of execution time.

	Parameter	
Algorithm	GG1 Execution time (ms)	GG2 Execution time (ms)
Algorithm 1	5.47	8.7
Algorithm 2	*5.47*	8.4
Algorithm 3	6.41	12
Algorithm 4	5.46	7.8
Algorithm 5	9.5	13
Algorithm 6	9.5	14
Algorithm 7	10.7	16.3
Algorithm 8	9.6	15
Vrancx *et al.*	14	16.2
ILA	11.5	15.3

VII. CONCLUSION

In this paper, several multi-agent system algorithms based on learning automata for finding optimal policy in Markov Games were proposed. The proposed methods use a network of interconnected learning automata for optimizing their behavior in order to effectively find the optimal policy. Then, the concept of entropy has been used in these algorithms to further improve their performance. Several experiments were conducted to investigate the learning performance of the proposed algorithms. The experimental results showed that the proposed algorithms have better learning performance in terms of the cost, speed and accuracy of reaching the optimal policy than the existing learning automata based algorithms.

REFERENCES

1. Vlassis, N., *A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence*, Morgan and Claypool Publishers, Amesterdam, The Netherlands (2007).
2. Goel A. K., V. Kumar, and S. Srinivasan, "Application of multi-agent system & agent coordination," *2nd Natl. Conf. Math. Tech.-Emerg. Paradigms Electron. IT Ind.*, New Delhi, India, pp. 328–337 (2008).

3. Boccadoro, M., F. Martinelli, and P. Valigi, "A multi-agent control scheme for a supply chain model," *Asian J. Control*, Vol. 10, No. 2, pp. 260–266 (2008).
4. Piovesan, J. L., C. T. Abdallah, and H. G. Tanner, "A hybrid framework for resource allocation among multiple agents moving on discrete environments," *Asian J. Control*, Vol. 10, No. 2, pp. 171–186 (2008).
5. González, E. J., A. Hamilton, L. Moreno, R. L. Marichal, G. N. Marichal, and J. Toledo, "A MAS implimentation for system identification and process control," *Asian J. Control*, Vol. 8, No. 4, pp. 417–423 (2006).
6. Shoham, Y. *Multiagent Systems*: *Algorithmic Game Theoretic and Logical Foundations*, Cambridge University Press, Cambridge, UK (2009).
7. Nowe, A. and K. Verbeeck, "Colonies of learning automata," *IEEE Trans. Syst.*, *Man Cybern.*, Vol. 32, pp. 772–780 (2002).
8. Hu, J. and M. P. Wellman, "Nash Q-learning for general-sum stochastic games," *J. Mach. Learn. Res.*, Vol. 4, pp. 1039–1069 (2003).
9. Claus, C. and C. Boutilier, "The dynamics of reinforcement learning in cooperative multiagent systems," *15th Natl. Conf. Artif. Intell.*, Madison, WI, pp. 746–752 (1998).
10. Thathachar, M. A. L. and P. S. Sastry, *Networks of Learning Automata*: *Techniques for Online Stochastic Optimization*, Kluwer Academic Publishers, Dordrecht, The Netherlands (2004).
11. Khojasteh, M. R. and M. R. Meybodi, "Evaluating learning automata as a model for cooperation in complex multi-agent domains," *RoboCup 2006*: *Robot Soccer World Cup*, Bremen, Germany, Vol. 4434, pp. 410–417, (2007).
12. Nowé, A., K. Verbeeck, and M. Peeters, "Learning automata as a basis for multi-agent reinforcement learning," *Learn. Adapt. Multi-Agent Syst.*, Vol. 3898, pp. 71–85 (2006).
13. Verbeeck, K., A. Nowe, P. Vrancx, and M. Peeters, "Multi-automata learning," In *Reinforcement Learning*, Weber, C. M. Elshaw, N. M. Mayer (eds). I-Tech Education and Publishing, Vienna, pp. 167–185 (2008).
14. Verbeeck, K., A. Nowé, M. Peeters, and K. Tuyls, "Multi-agent reinforcement learning in stochastic single and multi-stage games," *Adapt. Agents Multi-Agent Syst. Part III.*, Vol. 3394, pp. 275–294 (2005).
15. Sastry, P., V. Phansalkar, and M. A. L. Thathachar, "Decentralized learning of nash equilibria in multi-person stochastic games with incomplete information," *IEEE Trans. Syst. Man Cybern.*, Vol. 24, pp. 769–777 (1994).
16. Billard, E. A. and S. Lakshmivarahan, "Learning in multilevel games with incomplete information-part I," *IEEE Trans. Syst. Man Cybern. Part B*, Vol. 29, pp. 329–339 (1999).
17. Vrancx, P., K. Verbeeck, and A. Nowé, "Decentralized learning in Markov games," *IEEE Trans. Syst. Man Cybern., Part B*, Vol. 38, pp. 976–981 (2008).
18. Abtahi, F. and M. R. Meybodi, "Solving multi-agent Markov decision processes using learning automata," *6th Int. Symp. Intell. Syst.*, Subotica, Serbia, pp. 1–6 (2008).
19. Masoumi, B., M. R. Meybodi, and B. Jafarpour, "Solving general sum stochastic games using learning automata," *2nd Joint Cong. Fuzzy Intell. Syst.*, Tehran, Iran (2008).
20. Masoumi, B. and M. R. Meybodi, "Utilization of networks of learning automata for solving decision problems in decentralized multiagent systems," *17th Iranian Conf. Electr. Eng.*, Tehran, Iran (2009).
21. Zhuang, X. and Z. Chen, "Strategy entropy as a measure of strategy convergence in reinforcement learning," *1st Int. Conf. Intell. Netw. Intell. Syst.*, Wuhan, China, pp. 81–84 (2008).
22. Narendra, K. S. and M. A. L. Thathachar, *Learning Automata*: *an Introduction*, Prentice-Hall, Upper Saddle River, NJ (1989).
23. Lakshmivarahan, S. and K. Narendra, "Learning algorithms for two-person zero-sum stochastic games with incomplete information," *Math. Oper. Res.*, Vol. 6, pp. 379–386 (1981).
24. Wheeler, R. M. and K. S. Narendra, "Decentralized learning in finite Markov chains," *IEEE Trans. Autom. Control*, Vol. 31, pp. 519–526 (1986).
25. Fink, A. M. "Equilibrium in a stochastic N-person game," *J. Sci. Hiroshima Univ. Ser. A-I*, Vol. 28, pp. 89–93 (1964).
26. Greenwald, A. and K. Hall, "Correlated Q-learning," *20th Int. Conf. Mach. Learn.*, Washington, DC, pp. 242–249 (2003).
27. Golan, A., "Information and entropy econometrics–volume overview and synthesis," *J. Econom.*, Vol. 138, pp. 379–387 (2007).
28. Xie, L., "Topological entropy and data rate for practical stability: a scalar case," *Asian J. Control*, Vol. 11, No. 4, pp. 376–385 (2009).

29. Costa, M., A. Goldberger, and C. Peng, "Multiscale entropy analysis of complex physiologic time series," *Phys. Rev. Lett.*, Vol. 89, pp. 1–4 (2002).

Behrooz Masoumi received his BS and MS degrees in Computer Engineering in 1995 and 1998, respectively. He is currently pursuing his PhD degree in Computer Engineering at the Science and Research University, Tehran, Iran. He joined the faculty of Computer and IT Engineering Department at Qazvin Azad University, Qazvin, Iran, in 1998. His research interests include learning systems, multi-agent systems, multi-agent learning, and soft computing.

M. R. Meybodi received his BS and MS degrees in Economics from the Shahid Beheshti University in Iran, in 1973 and 1977, respectively. He also received his MS and PhD degrees in Computer Science from the Oklahoma University, U.S.A., in 1980 and 1983, respectively. Currently he is a Full Professor in the Computer Engineering Department, Amirkabir University of Technology, Tehran, Iran. Prior to his current position, he worked from 1983 to 1985 as an Assistant Professor at Western Michigan University, and from 1985 to 1991 as an Associate Professor at Ohio University, U.S.A. His research interests include channel management in cellular networks, learning systems, parallel algorithms, soft computing, and software development.