Behrooz MASOUMI[1], Mohammad Reza MEYBODI[2], Farnaz ABTAHI[2]

Islamic Azad University, Qazvin Branch (1), Amirkabir University of Technology (2)

Learning Automata based Algorithms for Finding Optimal Policies in Fully Cooperative Markov Games

Abstract. *Markov games, as the generalization of Markov decision processes to the multi agent case, have long been used for modeling multi-agent systems. In this paper, several learning automata based multi-agent system algorithms for finding optimal policies in fully-cooperative Markov Games are proposed. In the proposed algorithms, Markov problem is described as a directed graph in which the nodes are the states of the problem, and the directed edges represent the actions that result in transition from one state to another. Each state of the environment is equipped with a variable structure learning automata whose actions are moving to different adjacent states of that state. 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 the learning automaton in that state. The actions taken by learning automata along the path traveled by the agent is then rewarded or penalized based on the value of the traveled path according to a learning algorithm. In the second group of the proposed algorithms, the concept of entropy has been imported into learning automata based multi-agent systems to drive the magnitude of the reinforcement signal given to the LA and improve the performance of the algorithms. The results of experiments have shown that the proposed algorithms perform better than the existing learning automata based algorithms in terms of speed and the accuracy of reaching the optimal policy.*

Streszczenie. *Zaprezentowano szereg automatów uczących bazujących na algorytmach systemów typu multi-agent w celu poszukiwania optymalnej polityki w kooperatywnej grze Markova. Proces Markova jest opisany w postaci grafów których węzły opisują stan problemu, a krawędzie reprezentują akcje. (**Automat uczący bazujący na algorytmie znajdowania optymalnej strategii w kooperacyjne grze Markova**)*

Keywords: Multi-Agent Systems, Fully Cooperative Markov Games, Learning Automata, Optimal Policy.
Słowa kluczowe: system typ[u multi-agent, proces Markova, automaty uczące.

1. Introduction

A Multi-Agent System (MAS) is composed of multiple autonomous and intelligent agents, with distributed information and computational ability that interact with each other in an environment to accomplish some tasks [1]. A cooperative multi-agent system consists of a number of agents attempting to maximize the joint utility through their interactions [2]. There are several Markov models reported in the literatures for multi-agent systems. One of these models is Markov Game (MG) [3, 4]. The Markov game view of MAS is considered as a sequence of games that have to be played by multiple players, while each game belongs to a different state of the system [5]. Markov games are extensions of Markov Decision Process (MDP) to multiple agents. In a MG, actions are the result of joint action selection of all agents, while rewards and the state transitions depend on these joint actions [4, 6]. In a fully cooperative MG called a multi-agent MDP (or MMDP), all agents share the same reward function, and they should learn to agree on the same optimal policy. In multi-agent systems, the need for learning and adaption is essentially caused by the fact that the environment of agent is dynamic and just empirically observable while the environment (the reward functions and the transition states) is unknown. Hence, the reinforcement learning methods are applied in MAS 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. Assuming that agents get information about their own choice of action as well as that of the others, this is called joint action learning [6-8]. Joint action learners are able to maintain models of the strategy of others, and the explicitly takes into account the effects of joint actions. In contrast, independent agents only know their own action which is often a more realistic assumption since distributed multi-agent applications are typically subject to limitations such as partial observability, communication costs, and stochasticity.

There are several methods for finding an optimal policy in MMDPs (fully cooperative MGs). In [9], an algorithm is proposed for learning cooperative MMDPs, but it is only suitable for deterministic environments. In [8], another view on Markov Games is taken, i.e. the game can be seen as a sequence of normal form games. In [8] an algorithm called as Nash-Q is proposed which under restrictive conditions converges to Nash equilibrium policy. In [10] MMDPs are approximated as a sequence of intermediate games. The authors present optimal adaptive learning and prove convergence to a Nash equilibrium of the game. In the approach reported in [11], MMDP is decomposed to local MDPs, these MDPs are solved independently, and the overall solution is then estimated using these local solutions. In [12], an online approach is proposed for solving MMDPs which primarily selects a greedy approximate policy and then uses online search algorithms to refine it and reaches the optimal solution.

Recently, learning automata(LA) as a reinforcement learning model have been used to design multi-agent systems [13, 14]. Due to certain specifications such as structure simplicity, little need for information and feedback from the environment, learning automata are very useful in multi-agent systems. In [15], it is shown that a team of learning automata involved in a general N-person Markov game converge to Nash equilibrium if each of team members makes use of a linear learning algorithm called L_{R-I} algorithm. In [5], a model based on interconnected learning automata is reported to solve MMDPs. In [16] the authors show that a network of independent LA is able to reach equilibrium strategies in Markov games with some ergodic assumptions. In [17], the authors extend this algorithm with the intermediate rewards to accelerate learning convergence.

In the first part of this paper, we propose two new learning automata based multi agent system (MAS) algorithms. 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 MMDPs. In the proposed algorithms, the environment of Markov problem is modeled as a directed graph. The nodes of this graph represent the states and directed edges between nodes represent the actions that result in transition from one state to another. Each node of the graph is equipped with a learning automaton. The actions of each learning automaton are the outgoing edges of corresponding node. The agents move on this graph and in each state, they get help from corresponding learning automaton to choose a desirable action and move to the next state. Based on the path taken by agents and its goodness in terms of speed and cost, the probability vectors of learning automata will be updated. 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. The novelty of the algorithms are: 1) introducing a dynamic threshold to select the reward/penalty input to the learning automata, 2) maintaining a history of states visited (denoted as a path) to avoid cycles during each episode.

In the second part of this paper two more algorithms obtained by importing the concept of entropy [18, 19] into the algorithms given in the first part are proposed. Entropy is a significant concept in the thermodynamics, representing the degree of disorder in a thermodynamic system that is played an important role in various fields of computer science, such as coding theory, learning, compression, and others[18, 20] . The concept of entropy is used in order to drive the magnitude of the reinforcement signal given to the learning automata to enhance the performance of the multi-agent system leading to an optimal policy in fully-cooperative Markov Games. To evaluate the proposed methods, they have been applied to an example of MMDP called Grid Game. The results of computer simulations have shown that these algorithms outperform the previous learning automata based approaches from both cost and speed perspective. The rest of this paper is organized as follows: Section 2 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 3. In section 4, the proposed algorithm and its variations are presented. In Section 5, simulation results and discussion are reported. Section 6 concludes the paper.

2. Learning Automata

Learning Automata are adaptive decision-making devices operating on unknown random environments[21]. 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). In the following, the variable structure learning automata is described.

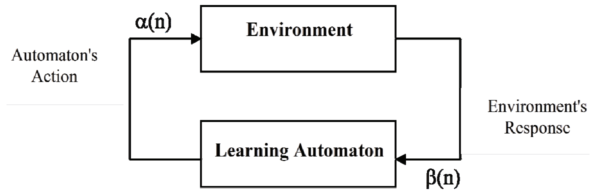


Fig 1. Learning automaton and its interaction with the environment

Variable structure learning automata can be shown by a quadruple { α , β, p, T } where α={α$_1$, α$_2$, ..., α$_r$ } which is the set of actions of the automaton, β={β$_1$, β$_2$,…, β$_m$} is its set of inputs, p={p$_1$, ..., p$_r$} is probability vector for selection of each action, and p(n +1) = T[α (n), β(n), p(n)] is the learning algorithm. We assume that action α$_i$ is selected at time-step *n*. In case of desirable response from the environment:

$$(1)\quad \begin{aligned} p_i(n+1) &= p_i(n) + a[1 - p_i(n)] \\ p_j(n+1) &= (1-a)p_j(n) \quad \forall j \;\; j \neq i \end{aligned}$$

In case of undesirable response from the environment:

$$(2)\quad \begin{aligned} p_i(n+1) &= (1-b)p_i(n) \\ p_j(n+1) &= (b/r-1) + (1-b)p_j(n) \quad \forall j \;\; j \neq i \end{aligned}$$

In equation (1) and (2), *a* and *b* are reward and penalty parameters respectively. When *a* and *b* are equal, the algorithm is called linear reward-penalty (L$_{R\text{-}P}$), when *b* is much smaller than a, the algorithm is linear reward-ε-penalty (L$_{R\text{-}\varepsilon P}$) and when *b* is zero, the algorithm is called linear reward-inaction (L$_{R\text{-}I}$). If β={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:

$$(3)\quad \begin{aligned} & p_i(n+1) = p_i(n) + \\ & \qquad a(1-\beta(n))[1-p_i(n)] - b\beta(n)\,p_i(n) \\ & \qquad \textit{if } a_i \textit{ is the action taken at time } n \\ & p_j(n+1) = p_j(n) - \\ & \qquad a(1-\beta(n))p_j(n) + b\beta(n)[\frac{1}{r-1} - p_j(n)] \\ & \quad \forall j, a_j \neq a_i \end{aligned}$$

where *r* is the number of possible actions, *a* and *b* are reward and penalty parameters, respectively.

Learning Automaton has fixed number of actions; but in many applications, a Learning Automaton with variable number of actions is needed. In each instant n, this type of Learning Automaton selects its action only from a nonempty subset *V(n)* of its action set which is called the set of active actions. Choosing the set *V(n)* is done randomly by an external element. This is how a Learning Automaton with variable number of actions works: to select an action in instant *n*, the Learning Automaton first calculates the sum of probabilities of its active actions, *K(n)*. Then, vector *p(n)* is calculated as mentioned in equation (4). Afterward, the Learning Automaton selects an action αi from its active actions set based on *p(n)* and executes it. After receiving the environment's response, the Learning Automaton will update p(n) . If the response is a reward, Learning Automaton will use equation (5) and if it is a punishment, it will use equation (6) to perform the update process.

$$(4)\quad \begin{aligned} & p_i(n) = prob[\alpha(n) = \\ & \alpha_i \mid V(n) \text{ is the set of active actions}, \\ & \quad \alpha_i \in V(n)] = \frac{p_i(n)}{K(n)} \end{aligned}$$

In case of desirable response from the environment:

$$(5)\quad \begin{aligned} p_i(n+1) &= p_i(n) + a.(1-p_i(n)) \quad \alpha(n) = \alpha_i \\ p_i(n+1) &= p_j(n) + a.p_i(n) \;\; \alpha(n) = \alpha_i, \; \forall j \;\; j \neq i \end{aligned}$$

In case of undesirable response from the environment:

(6)
$$p_i(n+1) = (1-b).p_i(n) \qquad \alpha(n) = \alpha$$
$$p_i(n+1) = \frac{b}{r-1} + (1-b)p_j(n) \quad \alpha(n) = \alpha_i,\ \forall j \ \ j \neq i$$

Then, *p(n)* will be updated using *p(n +1)*:

(7)
$$p_j(n+1) = \ p_j(n+1).K(n) \quad \text{for all j}, \alpha_j \in V(n)$$
$$p_j(n+1) = p_j(n) \qquad \text{for all j}, \alpha_j \notin V(n)$$

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 system [22]. A play a(t) = (a_1(t) . . . 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 β(t) = (β_1(t) . . . β_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 LR-εP 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 one pure equilibrium, which equilibrium is found depends on the initial conditions.

3. Markov Games

3.1. Markov Decision Process

The problem of controlling a finite Markov Chain, for which transition probabilities and rewards are unknown, called a Markov Decision Process [24] and can be stated as follows. Let s= {s_1, s_2,… s_N } be the state space of finite Markov chain {x_n}n≥0, and $A^i = \{a^i_1, a^i_2, \dots, a^i_{r_i}\}$ 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 has an associated transition probability $T^{ij}(a^i)$ and reward $R^{ij}(a^i)$. The goal is to choice the set of action, or policy, *α={a^1, a^2,…, a^n}* with $a^i \in A^i$ that maximizes the expected average reward *J(α)*.

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

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, nonrandomized policies. Hence, under the assumption that the Markov chain corresponding to each policy is ergodic, there exists the best strategy in each state is a pure strategy and is independent of the time at which the state is occupied [15].

3.2 Markov Game

Markov games are a generalization of MDPs to multiple agents and can be used as a framework for investigating multi-agent learning [25]. In a Markov game, actions are the result of joint action selection of all agents, while rewards and the state transitions depend on these joint actions [4, 6]. The action set available for agent k (1≤ k≤n) in state s_i is $A^i_k = \{a^i_{k_1}, a^i_{k_1}, \dots, a^i_{k_{ir}}\}$, *n* being the total number of agents present in the system. Transition probabilities $T^{ij}(a^i)$ and rewards $R^{ij}_k(a)$depend on a starting state si, ending state sj and joint action from state s_i, i.e. a^i= $(a^i_1, \dots, a^i_n)$ with $a^i_k \in A^i_k$.

The reward function $R^{ij}_k(a)$ for each agent *k* is individual. Different agents can receive different rewards for the same state transition. 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, or zero-sum games, are two-player games where one player's reward is always the negative of the others'. General-sum games are ones where 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, or 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 exist. Optimal policy is defined as the joint agent policy, which maximizes the payoff of all agents. An MMDP can therefore also be seen as an extension of the single agent MDP to the multi-agent case [26].

3.3 Control of Markov Game Using Learning Automata

The problem of controlling Single-Agent MDPs (and Markov Chains) can be modeled as a network of interconnected learning automata in which the control is transferred from one learning automaton to another [27]. 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:

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

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. The authors in [27] denote updating scheme as given in Equation (3) with environment response as in (9) as learning scheme T1. The following results were proved.

Lemma 1. *The Markov chain control problem can be asymptotically approximated by an identical payoff game of N automata.*

Theorem 1. *Let for each action state si of an N state Markov chain, an automaton LA_i using learning scheme T1 and having r_i actions be associated with. Assume that the Markov Chain, corresponding to each policy α is* ergodic. *Then the decentralized adaptation of the LA is globally ε-*optimal *with respect to the long-term expected reward per time step, i.e. J*(*α*).

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.[16] At each time step only are the automata of one state active; a joint action triggers the LA from that state to become active and takes some joint action. As MDP, the activated LA, LA_k^i, for agent k in state i is not informed of the one-step reward, rij(ai) resulted from choosing a joint action ai = $(a_1^i, \ldots, a_n^i)$ with action a_k^iin state i and leading to state j. When state i is visited again, all learning automata LA_k^i 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_k^i is exactly the same as in Equation 3.

4. The Proposed Algorithms

In this section, several algorithms for learning automata based multi agent system (MAS) are proposed. The multi-agent system is designed to find optimal policies in fully cooperative Markov Games. In the proposed algorithms, the multi-agent system as a fully cooperative MG problem is mapped on a directed graph. This mapping is done in such a way that the graph nodes indicate the states and directed edges represent actions of the fully cooperative MG resulting in transition from one state to another. For each state an S-model variable structure learning automaton is placed which tries to learn the optimal action probabilities in those states. The set of actions of this learning automaton is the set of permissible movements to other states. At first, all of the learning automata in all states choose their actions with equal probabilities. The agents start from “starting state” and move toward “goal state”. In each state, the agents move to the next state by the help of corresponding learning automaton and its action probability vector. Each agent continues moving on the graph until it reaches the goal state. After goal state is reached, the value of the traversed path π_j taken by agent *j*, *(Lπ_j)*, is computed based on the length of the traversed path πj divided by the reward of reaching the goal state, (RG), and then compared with a quantity called dynamic threshold, *Tj.* Depending on the result of the comparison all the learning automata along the traversed path update their action probabilities. Updating is done in direction from starting state to goal state or vice versa. If the value of the traversed path is less than or equal to the dynamic threshold then all learning automata along that path receive reward and if length of the traversed path is greater than the dynamic threshold then activated automata 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. Assume that the path π_j for agent *j* is traversed *m* times. Thus the dynamic threshold for agent *j* defined as the average value of traversed paths from start state to goal state can be updated according to equation (10). The proposed algorithm which we call it Algorithm 1 is given in more detail in Figure 2.

(10) $T_j=T_j+(L\pi_j-T_j)/m$

Algorithm 1 suffers from the possibility of existence of cycles in the paths taken by agents. This possibility increases with increase in learning rate or reward. Algorithm 2 is obtained by addressing this problem. In Algorithm 2, it is assumed that in each state *i* of the environment of the game and for each agent *k*, *k: 1..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. When the agent *j* arrives at state *s*, the learning automaton corresponding to this agent in the state is activated and agent *j* gets help from this learning automaton to find the next state to move on. By selecting action a_m in state *s* and transition to new state *m*, the action selection process is performed as follows: actions that result in transition to states which have been previously visited in agent *j's* path are deactivated and then another action from active actions set based on probability vector will be chosen. In case that the set of active actions is empty and therefore no action can be chosen, all actions will be reactivated to become available for selection by the agents. In Algorithm 2, the model of learning automata with the variable number of actions is used and the update scheme is based on Equations 4-7. Algorithm 2 in more details is shown in Figure 3.

Improving the proposed algorithms using the Concept of Entropy

In this section we first briefly talk about the concept of entropy and then introduce two new algorithms called Algorithm 3 and Algorithm 4 which are obtained by importing the concept of entropy into the Algorithms 1 and 2. Entropy which represents the degree of disorder in a thermodynamic system plays an important role in various fields of computer science [18,19]. Shannon has introduced this concept into the information theory, by the name of "information entropy". Entropy, in basic, 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 [28,29]:

(11) $H(X) = -\sum_{x\in\chi} P(x)\log(P(x))$

where *X* represents a random variable with set of values χ and probability mass function *P(x)*. Entropy is always a positive value and can change bases freely as $H_b(X) = \log_b(a).H_a(X)$. Entropy measures the uncertainty inherent in the distribution of a random variable.

Algorithm 3 and Algorithm 4 which will be described in this section are improvements over Algorithm 1 and Algorithm 2 and obtained by importing the concept of entropy into them. The concept of entropy is used to help the learning automata along the paths traversed by the agent for improving the process of learning and also faster convergence. For this purpose, when LAs in state *s* selects action *m*, and goes to new state *s'* (except goal state), the LA_s is rewarded using the entropy of the actions probability vector for the learning automata of the new state. Entropy of the action probability vector measures the degree of uncertainty that the next state of learning automaton encounters. High value of entropy indicates that the learning automaton has no information about where the goal state is and chooses its action randomly (Exploration). On the contrary, the low value of entropy indicates that the learning automaton has useful information about the goal state and, therefore, chooses its actions with higher probability (Exploitation). If $P(s) = \{p_1^s, p_2^s, \ldots, p_r^s\}$ is the action probability vector of a learning automaton with *r* actions in state *s*, then the entropy the probability vector for state *s* is computed as:

(12) $H(s) = -\sum_{j=1}^{r} p_j^s \log(p_j^s)$

where P_j^s denotes the action probability vector of the learning automaton in state *s*.

Entropy is maximized when all the actions have equal probabilities of selection and is minimized (zero value) 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 *i* is in state *s* and its learning automaton (LA_s) leads the agent to move to state *s'*. In this case, reinforcement signal β_i^s is determined using:

(13) $\beta_i^s = H(s')/Max(H(s'))$

where *Max* (*H*(s')) is the maximum entropy in state *s' and is* defined as:

(14) $Max(H(s')) = -\sum_{j=1}^{r}(1/r)\log(1/r) = \log_2 r$

ALGORITHM 1

```
Create one learning automaton for each state s and define the set of actions of the learning automaton in state s to be the set of
permissible movements from state s to other states. Initially, for a learning automaton the probability of selection of an action is set to
1/number of permissible actions;
Place all agents in starting state;
for every agent j do
    set threshold $T_j$ to zero
    set $k_j$ to zero // the number of the path traversed by agent j from starting state to goal state
end for
for every agent j do in parallel
    Repeat
        Activate learning automaton $LA_s$ corresponding to state s in which the agent j is residing and determine the
                                                        next state m to which the agent moves.
            // This is done based on the probability vector of the learning automaton, by selecting an
               action $a_m$ corresponding to directed edge (s, m) of the graph which means transition from
               state s to next state, m//
        Move Agent j to state m;
        if there is more than one agent in state m then Move agent j to state s //collision detect
        else
            $s \leftarrow m$; Add s to $\pi_j$
            if state s is the goal state then
                Compute the value of path $\pi_j$ taken by agent j, $L_j(\pi_j)$, to be $t_j(\pi_j)/R_G$
                        //$R_G$ is the reward of reaching the goal state, $t_i(\pi_j)$ is the length of the traversed path $\pi_j$ //
                if $L_j(\pi_j) < T_j$ then
                    Give reward to all the actions taken by learning automata along the traversed path $\pi_j$ according to the learning
                                                        algorithm with $\beta(k) = L_j(\pi_j)$
                else
                    Give penalty to all the actions taken by learning automata along the traversed path $\pi_j$ according to the learning
                                                        algorithm with $\beta(k) = 1 - L_j(\pi_j)$ ;
                end if
                $T_j = T_j + (L_j(\pi_j) - T_j)/k_j$;        // $T_j$ is the average value of traversed path//
                $k_j = k_j + 1$
            end if
        end if
    until ( the probability of path $\pi_j$ taken by agent j exceeds a pre-specified threshold or a fixed number of iterations are
                                                        passed)
end for
```

Fig 2. Algorithm 1

5. Simulation Results and Discussion

In order to evaluate the performance of the proposed methods, several experiments have been conducted. The environment of experiment is a Grid-world game that includes a 3×3 grid with a goal cell. Two agents start from the two bottom corners (location 1 and 3) and try, with the least possible number of moves, to find the goal square, which is the top center. After observing the current state, agents choose their actions simultaneously. They then observe the new location, both agents' rewards, and the action taken by the other agent. In the new location, agents repeat the process above. When at least one agent moves into its goal position, the game restarts. In the new episode, each agent is randomly assigned a new position (except its goal cell). This environment has also been used previously in [5,8]. To convert the problem to an MMDP, we consider each cell as a state and the allowable transitions to adjacent cells as actions. In this MMDP the objective is that both agents reach the goal state using distinct paths. If both agents try to move to the same cell, both of their moves will fail and both receive 0.01 units of punishment and stay in their previous position. If agents move to two different non-goal cells, both receive zero rewards and if one reaches the goal position, it receives 1 units of reward. The reward function for each agent *i* is based on Equation *15*, *i=1, 2*.

(15) $$r_i(t) = \begin{cases} 1 & \textit{if agent i reachedthegoalstate} \\ 0.01 & \textit{if agentsreacheda differentnongoalstate} \\ 0 & \textit{if bothagentsreachedthesamenongoalstate} \end{cases}$$

ALGORITHM 2

```
Create one learning automaton for each agent in each state and define the set of actions of the learning automaton in
state s to be the set of permissible movements from state s to other states. Initially, for a learning automaton the
probability of selection of an action is set to 1/number of permissible actions;
Activate the actions of all learning automata for each agent;
Place all agents in starting state;
for every agent j do
    set threshold $T_j$ to zero
    set $k_j$ to zero // the number of the path traversed by agent j from starting state to goal state
end for
Repeat
  s=StartState
  While s is not the goal state or a certain number of moves are made by the agent Do
      Joint-action=$\varnothing$
      for every agent j do
          Activate learning automaton $LA_j^s$
          if the active action set of $LA_j^s$ is not empty then
              Action=Select an active action $a_j^s$
          else                      // the active action set is empty//
              Reactivate all inactive actions for agent j in state s
          end if
          Joint-action = Joint-action $\cup$ Action
      end for
      NewState= GetNextState(s, Joint-action)
      for every agent j do
          Move Agent j to NewState;
          Deactivate action $a_j^s$ in Action set for LA in the current state
          if NewState is the goal state then
              Compute the value of path $\pi_j$ taken by agent j, $L_j(\pi_j)$, to be $t_j(\pi_j)/R_G$
                  //$R_G$ is the reward of reaching the goal state, $t_j(\pi_j)$ is the length of the traversed path $\pi_j$
              if $L_j(\pi_j) < T_j$ then
                  Give reward to all the actions taken by learning automata along the traversed path $\pi_j$ according to the
                      learning algorithm with $\beta(j) = L_j(\pi_j)$
              else
                  Give penalty to all the actions taken by learning automata along the traversed path $\pi_j$ according to the
                      learning algorithm with $\beta(j) = 1 - L_j(\pi_j)$
              end if
              $T_j = T_j + (L_j(\pi_j) - T_j)/k_j$      // $T_j$ is the average value of traversed path //
          end if
          $k_j=k_j+1$;
      end for
      s =NewState
  end while
```

Fig3. Algorithm 2

A path in this game represents a sequence of actions from the start to the goal position. In game terminology, such path is called *a policy* or *strategy*. The shortest path not interfering the path taken by the other agent is called the *optimal path* or *Nash path*. Figure 5 illustrates the optimal solution for Grid-world Game. In the rest of this section the results of experiments designed to show the effectiveness of the proposed algorithms will be presented.

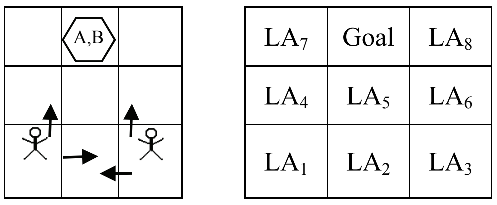


Fig 4. Grid-world game and learning automata model

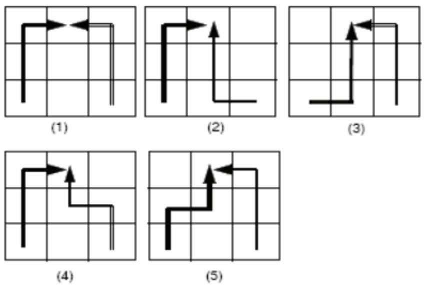


Fig 5. Optimal solutions for Grid-world Game

Experiment 1: This experiment is conducted to study the impact of learning parameter *a* of L_{R-I} learning algorithm on the convergence of learning automata in Algorithm 1. For this purpose, the probability of choosing action "up" by the agent in starting cell when the learning rate takes different values: 0.01, 0.05, 0.1 and 0.4 has been observed. Figure 6 shows the results of this experiment. From the result it is evident that the choice of value for parameter *a* has a large effect on the performance of the proposed algorithm. In this experiment the best result is obtained when *a* is set to 0.01. For values of *a* greater than *0.05* convergence rate deteriorates.

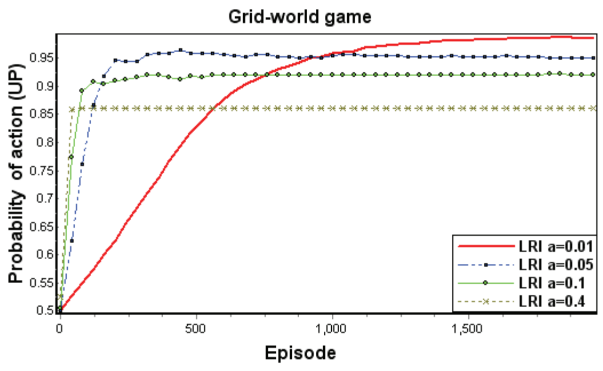


Fig 6. Probability of action "up" in starting cell of agent 1 in Algorithm 1 for different learning rates.

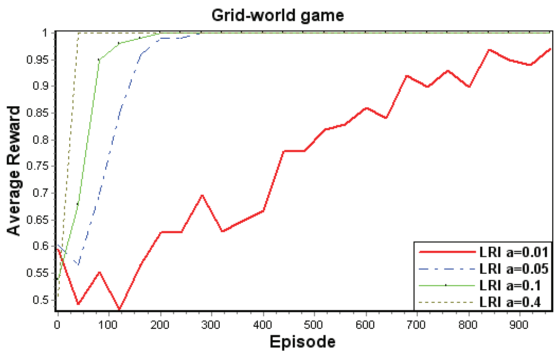


Fig 7. The impact of learning rate on the reward received by agent 1

***Experiment* 2:** This experiment is conducted to study the impact of learning parameter *a* of L_{R-I} learning algorithm on the amount of reward received by agent 1 during an episode in Algorithm 1. 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. Each value reported in this experiment is obtained by averaging over 100 runs. We assume that at the beginning of an episode, each agent starts from starting cell. Figure 7 illustrates the results of this experiment. The result illustrates clearly that choice of value for parameter *a* has a large effect on the performance of the proposed algorithm. The results indicate that higher value for parameter *a* leads to speeding up the convergence to the goal state. For values of *a* greater than *0.4* convergence rate deteriorates.

***Experiment* 3:** This experiment is conducted to study the impact of learning parameter *a* of L_{R-I} learning algorithm on the number of agent collisions during an episode in Algorithm 1. For this purpose, we plot the number of collisions between agents per episode for different values of learning parameter *a*: 0.01, 0.05, 0.1 and 0.4. We assume that at the beginning of an episode, each agent starts from starting cell. Each value reported in this experiment is obtained by averaging over 100 runs. Figure 8 summarizes the results of this experiment and shows that higher value for parameter *a* up to certain value (value *0.4 for this experiment)* results in lower number of collisions.

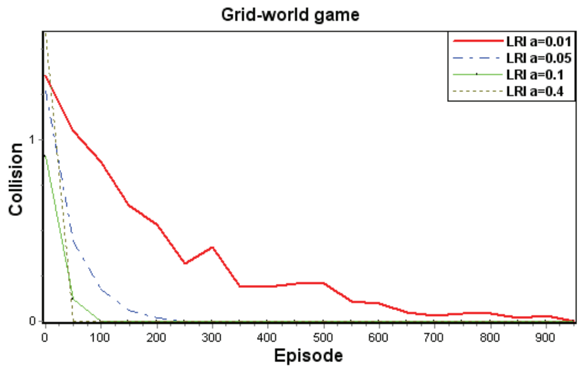


Fig. 8. The impact of learning rate on the number of agent collisions during an episode

Experiment 4: In this experiment, we study the impact of learning parameter *a* of L_{R-I} learning algorithm on the probability of optimal path in Algorithm 1. The probability of optimal path is defined as the product of probabilities of the selection of the edges along the optimal path. The plots of average probability of optimal path over the converged runs out of 100 runs are given in Figure 9. For example, It can be seen that in terms of accuracy the best result is obtained when *a= 0.01.* Increasing the value of *a* increases the speed of convergence in the expense of accuracy.

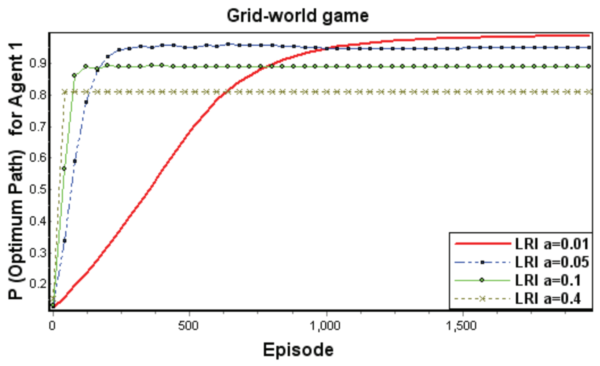
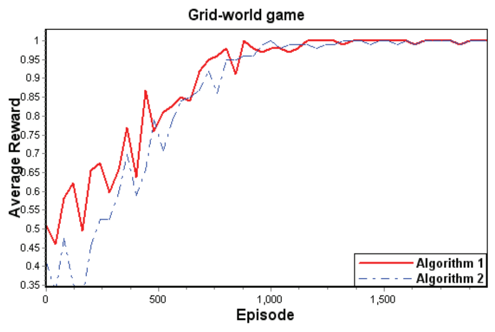


Fig. 9. Probability of optimal path for agent 1 for Algorithm 1 for different learning rate

Experiment 5: In this experiment we compare Algorithm 1 with Algorithm 2 in terms of the reward received by agent 1 and the number of agent collisions made during an episode. Learning parameter *a* for both algorithms is set to 0.01. Each value reported is the average over 100 runs. It is assumed that at the beginning of an episode, each agent starts from starting cell. Figure 10 shows the result of this experiment. As it is seen Algorithm 1 outperforms Algorithm 2 in terms of both the number of agent collisions and the average reward received during an episode.



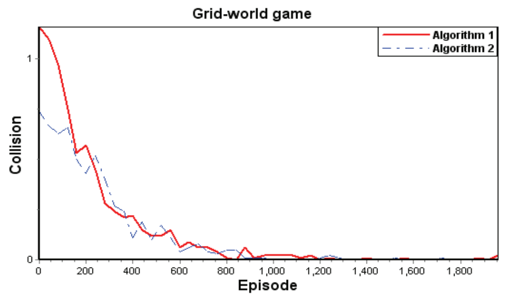
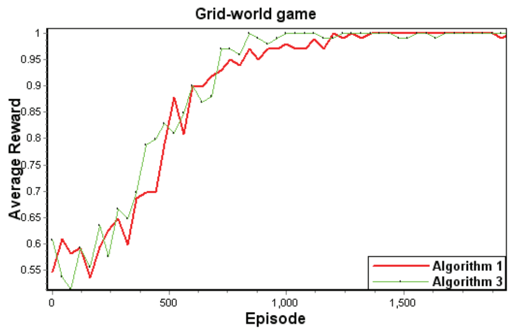
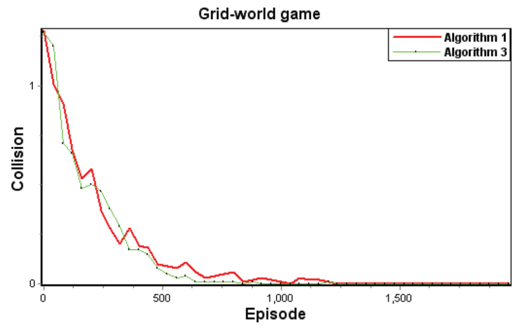
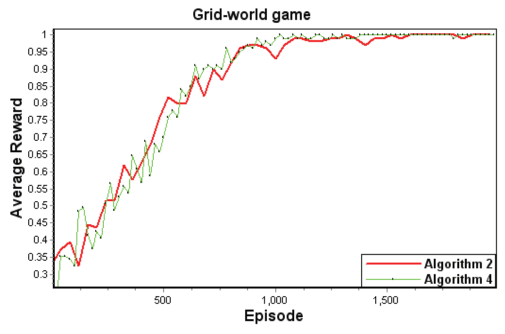


Fig. 10. Comparison of Algorithms 1-2 in terms of average reward received and the number of agent collisions







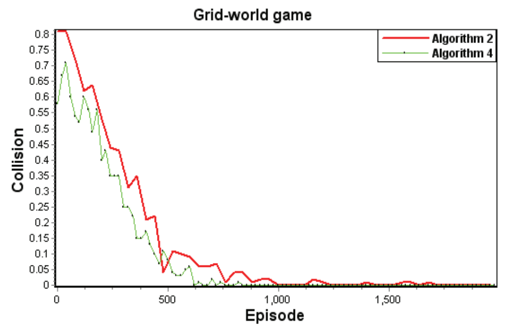
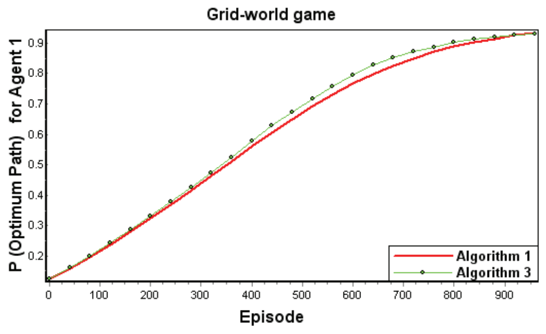


Fig. 11. Comparison of the Algorithms 2 - 4

***Experiment 6**:* This experiment is conducted to study the improvement obtained by importing entropy into Algorithms 1 and 2. To do this, we compare Algorithm 1 with Algorithm 3 and Algorithm 2 with Algorithm 4 in terms of the reward received by agent 1 per episode and the number of collusions when applied to Grid-world game. The learning parameter *a*: is set to 0.01. From the results given in figure 11 we can say that using importing in both algorithms leads to higher accuracy and also higher rate of convergence.

***Experiment 7**:* In this experiment we compare Algorithm 1 with Algorithm 3 and Algorithm 2 with Algorithm 4 with respect to optimal path probability. The learning parameter *a* is set to 0.01. Figure 11 shows the result of this experiment. As it is seen Algorithm 3 and Algorithm 4 in which we use entropy outperform their counterparts Algorithm 1 and Algorithm 2. Figure 12 show that the optimal path probability for both algorithms 3 and 4 approaching more quickly to its final value.



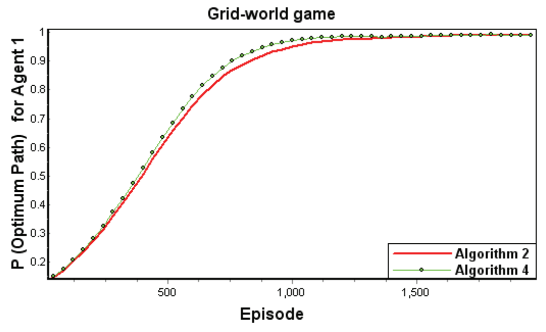


Fig.12. Comparison of the Algorithms 1 with Algorithms 3 and Algorithms 2 with Algorithms 4 with respect to optimal path probability

***Experiment 8**:* In this experiment we compare the proposed algorithms with one of the existing learning automata based algorithm called interconnected learning automata (ILA) on the same problem [5]. Similar to the proposed algorithms, in ILA algorithm each cell is equipped with a learning automaton whose actions correspond to possible movements to adjacent cells. Learning parameter *a* for the proposed algorithms and ILA algorithm is set to *0.01*. Table 1 gives the number of iterations required by each algorithm in order the probability of optimal path exceeds 0.97. As it is shown all the proposed algorithms require fewer numbers of iterations in comparison to ILA algorithm. Among the proposed algorithms, Algorithm 3 performs the best. Figure 13 shows the results of comparing Algorithm 3, Algorithm 4 and algorithm ILA in terms of the probability of optimal path and the amount of reward received per episode by agent 1. It is evident that both Algorithm 3 and Algorithm 4 outperform ILA algorithm.

Table 1: Maximum number of iterations for the first agent to reach the optimal path

Used Algorithm	*Maximum Iterations a=0.01*
Algorithm 1	1600
Algorithm 2	1700
Algorithm 3	1000
Algorithm 4	1200
ILA algorithm [5]	2800

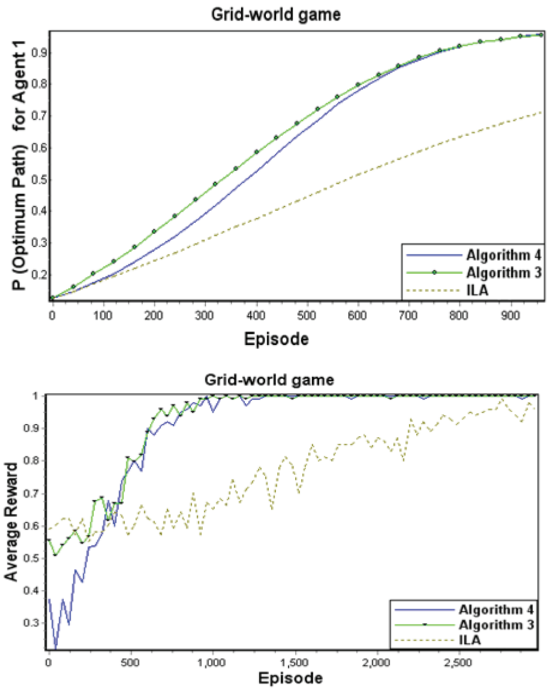


Fig. 13. Comparison of the Algorithms 3 and 4 with ILA

Experiment 9: In this experiment, we investigate the performance of the proposed algorithms and ILA algorithm when applied to a stochastic version of Grid-world game. In Grid-world game state transitions are deterministic, which means the current state and agent's joint action will uniquely determine the next state. In the stochastic version of Grid-world game which we have considered for this experiment all transitions are deterministic except those from the starting states (locations 1 and 3) to other states. An agent in its starting state moves up with probability 0.5 and remains in starting state with probability 0.5. This example is borrowed from [5]. Figure 14 illustrates the optimal solutions for stochastic version of Grid-world Game. In this experiment, the optimal path probability and the reward received by agent 1 per episode have been observed. Experimentations have shown that all five algorithms perform best when learning parameter *a* is set to 0.01 and for this reason in this experimentation *a* is chosen to be 0.01 for all algorithms. Figure 15 shows the result of this experiment. As it is seen Algorithm 3 outperforms all the other algorithms. Notice that algorithm ILA performs the worst.

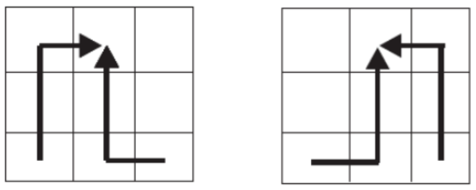


Fig. 14. Optimal solutions (Nash Path) for stochastic version of Grid-world Game

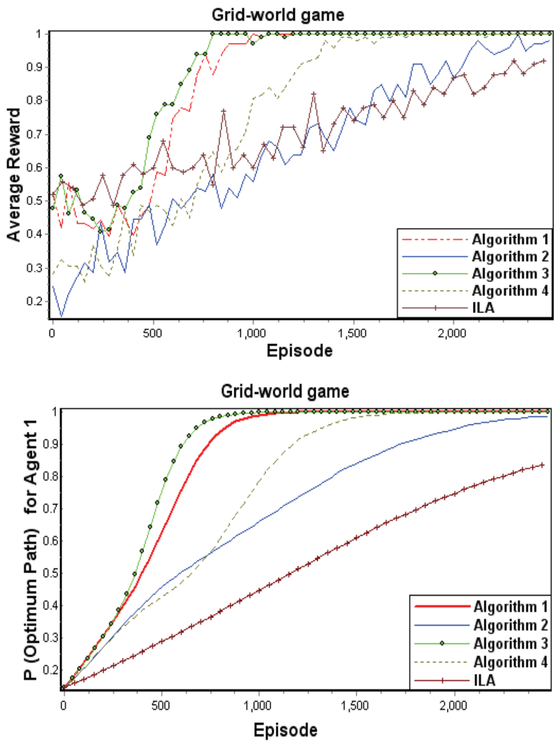


Fig. 15. Comparison of the proposed Algorithms and ILA

6. Conclusion

In this paper several learning automata based multi agent system (MAS) algorithms for finding optimal policy in fully cooperative Markov Games were proposed. In the proposed algorithms, the environment of Markov problem is modeled as a directed graph. The nodes of this graph represent the states and directed edges between nodes represent the actions that result in transition from one state to another. Each node of the graph is equipped with a learning automaton whose actions are the outgoing edges of the corresponding node. The agents move on this graph and in each state, they get help from corresponding learning automaton to move to the next state. The proposed multi-agent systems were evaluated by applying them to an example of a MMDP called Grid Game. Simulation results showed that the choice of the learning rate has a great impact on the performance of all the proposed algorithms. The concept of entropy was also used to enhance the performance of the proposed multi-agent systems. The results of experimentations showed that the proposed multi-agent systems outperform the previous learning automata based multi agent system in terms of cost and speed of convergence. In the proposed algorithms there are limitations, such as requiring that the problem involves reaching a goal state.

REFERENCES

[1] Vlassis N., A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence, (Morgan and Claypool Publishers, Amesterdam, (2007).

[2] L. Panait, S. Luke, Cooperative Multi-Agent Learning: The State of the Art, Autonomous Agents and Multi-Agent Systems, 11 (2005) 387-434.

[3] L. Shapley, Stochastic Games, in: The National Academy of Sciences. USA, (1953), 1095-1100.

[4] M.J. Osborne, A. Rubinstein, A Course in Game Theory, (Cambridge, MA: MIT Press, 1994).

[5] A. Nowe, K. Verbeeck, Colonies of Learning Automata, IEEE Transactions on Systems, Man and Cybernetics, 32 (2002) 772-780.

[6] C. Claus, C. Boutilier, The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems, in: The 15th National Conference on Artificial Intelligence. Wisconsin, USA, 1998), 746-752.

[7] G. Chalkiadakis, C. Boutilier, Coordination in Multiagent Reinforcement Learning: A Bayesian Approach, in: The 2nd International Joint Conference on Autonomous Agents and Multiagent Systems. Melbourne, Australia, 2003), 709-716.

[8] J. Hu, M.P. Wellman, Nash Q-Learning for General-Sum Stochastic Games, Journal of Machine Learning Research, 4 (2003) 1039-1069.

[9] M. Lauer, M. Riedmiller, An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-Agent Systems, in: The 17th International Conference on Machine Learning. (Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, 2000), 535 - 542.

[10] X. Wang, T. Sandholm, Reinforcement Learning to Play an Optimal Nash Equilibrium in Team Markov Games, in: in Advances in Neural Information Processing Systems. (MIT Press, 2002), 1571-1578.

[11] P. Laroche, Y. Boniface, R. Schott, A New Decomposition Technique for Solving Markov Decision Processes, in: The 2001 ACM Symposium on Applied Computing. Las Vegas, Nevada, United States, 2001), 12 -16.

[12] L. Peret, F. Garcia, On-line search for solving large Markov Decision Processes, in: Sixth European Workshop on Reinforcement Learning (EWRL-6). Nancy, FRANCE, 2003).

[13] M.R. Khojasteh, M.R. Meybodi, Evaluating Learning Automata as a Model for Cooperation in Complex Multi-Agent Domains, in: RoboCup 2006: Robot Soccer World Cup. (Springer, Berlin, 2007), 410-417.

[14] A. Now´e, K. Verbeeck, M. Peeters, Learning Automata as a Basis for Multi-agent Reinforcement Learning, in: Learning and Adaption in Multi-Agent Systems (Springer, Berlin, 2006), 71-85.

[15] P. Sastry, V. Phansalkar, M.A.L. Thathachar, Decentralized Learning of Nash Equilibria in Multi-person Stochastic Games with Incomplete Information, IEEE Transactions on Systems, Man, and Cybernetics, 24 (1994) 769-777.

[16] P. Vrancx, K. Verbeeck, A. Nowé, Decentralized Learning in Markov Games, IEEE Transactions on Systems, Man and Cybernetics (Part B: Cybernetics), 38 (2008) 976-981.

[17] B. Masoumi, M.R. Meybodi, Utilization of Networks of Learning Automata for Solving Decision Problems in Decentralized Multiagent Systems, in: 17th Iranian Conference on Electrical Engineering (ICEE2009). Tehran, Iran, 2009).

[18] X. Zhuang, Z. Chen, Strategy Entropy as a Measure of Strategy Convergence in Reinforcement Learning, in: First International Conference on Intelligent Networks and Intelligent Systems. Wuhan, China, 2008), 81-84.

[19] E.H. Lieb, J. Yngvason, The Physics and Mathematics of the Second Law of Thermodynamics, Physics Report, 310 (1999) 1-96.

[20] L. Guana, I.U. Awanb, I. Phillipsa, A. Griggc, W. Dargied, Performance analysis of a threshold-based discrete-time queue using maximum entropy, Simulation Modelling Practice and Theory, 17 (2009) 558-568.

[21] K.S. Narendra, M.A.L. Thathachar, Learning Automata: an Introduction, (Prentice-Hall Inc, 1989).

[22] M.A.L. Thathachar, P.S. Sastry, Networks of Learning Automata: Techniques for Online Stochastic Optimization, (Kluwer Academic Publishers, 2004).

[23] S. Lakshmivarahan, K. Narendra, Learning Algorithms for Two-person Zero-sum Stochastic Games with Incomplete Information, Mathematics of Operations Research, 6 (1981) 379 - 386.

[24] R.A. Howard, Dynamic Programming and Markov Processes, (Cambridge, MA: MIT Press, 1960).

[25] M. Littman, Markov Games as a Framework for Multi-agent Reinforcement Learning, in: 11th Int. Conf. on Machine Learning. San Francisco, Morgan Kaufmann, 1994), 157-163.

[26] C. Boutilier, Planning, Learning and Coordination in Multiagent Decision Processes, in: The 6th Conference on Theoretical Aspects of Rationality and Knowledge. Renesse, Holland, 1996), 195-210.

[27] R.M. Wheeler, K.S. Narendra, Decentralized Learning in Finite Markov Chains, IEEE Transactions on Automatic Control, 31 (1986) 519-526.

[28] M. Costa, A. Goldberger, C. Peng, Multi-scale Entropy Analysis of Complex Physiologic Time Series, Physical Review Letters, 89 (2002) 1-4.

[29] Z. Dianhu, F. Shaohui, D. Xiaojun, Entropy-A Measure of Uncertainty of Random Variable, Systems Engineering and Electronics, 11 (1997) 1-3.

Authors: *Dr Behrooz Masoumi, Department of Computer and Electrical Engineeing, Islamic azad university, Qazvin Branch, E-mail:* Masoumi@Qiau.ac.ir; *Prof. dr Mohammad Reza Meybodi, Department of Computer engineering and Information Technology, Amirkabir University of Technology, Iran, Tehran,E-mail:* *mmeybodi@aut.ac.ir* ; *Farnaz Abtahi, Department of Computer engineering and Information Technology, Amirkabir University of Technology, Iran, Tehran, E-mail:* *Abtahi@aut.ac.ir.*

The correspondence address is:
e-mail: Masoumi@Qiau.ac.ir;