Solving Multi-Agent Markov Decision Processes Using Learning Automata

Farnaz Abtahi*, Mohammad Reza Meybodi*
* Amirkabir University of Technology, Department of Computer Engineering and IT, Tehran, Iran
{abtahi, mmeybodi}@aut.ac.ir

***Abstract*— Multi-Agent Markov Decision Processes (MMDPs) are widely used for modeling many types of multi-agent systems. In this paper, two new algorithms based on learning automata are proposed for solving MMDPs and finding optimal policies. In the proposed algorithms, Markov problem is described as a directed graph. The nodes of this graph are the states of the problem, and the directed edges represent the actions that result in transition from one state to another. Each node in the graph is equipped with a learning automaton and its actions are the outgoing edges of that node. Each agent moves from one node to another and tries to reach the goal state. In each node, the agent chooses its next transition with help of the learning automaton in that node. The actions taken by learning automata along the path traveled by the agent is then rewarded or penalized based on the cost of the traveled path according to a learning algorithm. This way the optimal policy for the agent will be gradually reached. The results of experiments have shown that our proposed algorithms perform better than the existing learning automata based algorithms in terms of cost and the speed of reaching the optimal policy.**

***Keywords*— Multi-Agent Systems, Multi-Agent Markov Decision Process, Learning Automata, Optimal Policy.**

I. INTRODUCTION

So far, many models have been proposed for multi-agent systems. One of these models is Multi-Agent Markov Decision Process (MMDP). A Single-Agent Markov Decision Process (MDP) can be defined as the quadruple $< S, A, P, R >$ in which $S = \{i\}$ and $A = \{a\}$ are finite sets of states and actions respectively. $P : S \times A \times S \to [0,1]$ defines the transition function; so, the probability of transition from state *i* to *j* by performing action *a* is $P(i, a, j)$. $R : S \to \mathbb{R}$ is the reward function; when an agent observes state *i*, the reward it will receive will be $R(i)$. The solution to an MDP is a policy which is defined as a procedure for action selection by agents in each state. These policies which we represent them by π, are actually mappings from states to actions defined as $\pi : S \to A$.

Now, consider a multi-agent environment with a set of *n* agents defined as $M = \{m\}$. Each agent has a set of states $S_m = \{i_m\}$, and a set of actions $A_m = \{a_m\}$. The easiest and most common approach to extend the concept of single-agent MDP to multi-agent case is to assume that each agent affects the state transition and reward of other agents. As a result, an MMDP can be easily defined as a "big" MDP $< S_M, A_M, P_M, R_M >$. In this quadruple, the joint action space S_M is define as outer product of state space of all agents, $S_M = S_1 \times ... \times S_n$. The joint action space is similarly defined as $A_M = A_1 \times ... \times A_n$.

The transition and reward functions are also defined on joint state space and joint action space as standard as $P_M : S_M \times A_M \times S_M \to [0,1]$ and $R_M : S_M \to \mathbb{R}$ respectively [10]. The above representation scheme which is called "flat scheme" is the most general way to represent MMDPs because having joint space and action spaces, agents can interact arbitrarily with each other.

There are several methods for finding an optimal policy in an MMDP. In Ref. [3], an algorithm based on Q-Learning is proposed for solving these problems. In the approach of Ref. [4], MMDP is decomposed to local MDPs, these MDPs are solved independently and the overall solution is then estimated using these local solutions. In Ref. [5], 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 reach the optimal solution. Another way of solving these problems which is based on reinforcement learning is Learning Automata (LA). In Ref. [6], a model based on Interconnected Learning automata is suggested to solve MMDPs.

In this paper, we propose two new algorithms based on Learning Automata that can be effectively used to solve MMDPs and to find the optimal policy. In our 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 then equipped with a Learning Automaton. The actions of each Learning Automaton are the outgoing edges of corresponding node. Agents move on this graph and in each state, they get help from corresponding Learning Automaton to choose a desirable action and move to 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 Experiments have shown that these algorithms perform better that previous approaches based on Learning Automata from 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. In section 3, we will discuss our approach for solving MMDPs using Learning Automata and we will also

1-4244-2407-8/08/$20.00 ©2008 IEEE

present the result of our experiments. Finally, we will discuss the conclusions in section 4.

II. LEARNING AUTOMATA

Learning Automaton is a machine that can perform finite number of actions. Each selected action is evaluated by a random environment. The result is presented to the Learning Automaton as a positive or negative signal and then the Learning Automaton uses this response to choose its next action. The final goal of Learning Automaton is to learn to select the best action among all its actions. The best action is an action that maximizes the probability of receiving reward from the environment. Learning Automaton and its interaction with the environment are shown in figure 1 [6, 11, 12].

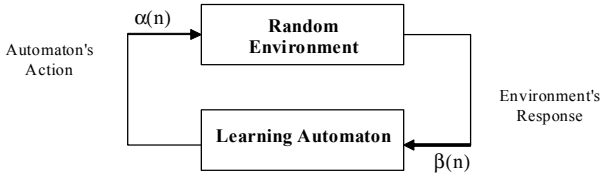

Figure 1. Learning Automaton and its interaction with the environment [6]

The environment is represented by triple $E \equiv \{\alpha, \beta, c\}$. Here, $\alpha \equiv \{\alpha_1, \alpha_2, ..., \alpha_r\}$ is the set of inputs, $\beta \equiv \{\beta_1, \beta_2, ..., \beta_m\}$ is the set of outputs and $c \equiv \{c_1, c_2, ..., c_r\}$ is the set of penalty probabilities. If β is a two member set, the environment is of type *P*. In such an environment, $\beta_1 = 1$ and $\beta_2 = 0$ are considered as punishment and reward respectively. In a *Q* environment, β can take a discrete value from finite values in [0,1]. In an environment of type *S*, $\beta(n)$ is a random variable in [0,1]. c_i is the probability that action α_i has an undesirable result. In a static environment, c_i values are fixed, but in a dynamic environment, they might change as time passes.

Fixed Structure Learning Automata is represented by quintuple $\{\alpha, \beta, F, G, \phi\}$ in which $\alpha \equiv \{\alpha_1, \alpha_2, ..., \alpha_r\}$ is the set of actions, $\beta \equiv \{\beta_1, \beta_2, ..., \beta_r\}$ is the set of inputs, $\phi(n) \equiv \{\phi_1, \phi_2, ..., \phi_k\}$ is the set of internal states at time *n*, $F : \phi \times \beta \to \phi$ is the state transition function and $G : \phi \to \alpha$ is the output function that maps the current state of the Learning Automaton to next output (action).

Variable Structure Learning Automaton can be defined as $\{\alpha, \beta, p, T\}$. In this quadruple, $\alpha \equiv \{\alpha_1, \alpha_2, ..., \alpha_r\}$ is the set of actions, $\beta \equiv \{\beta_1, \beta_2, ..., \beta_r\}$ is the set of inputs, $p = \{p_1, .., p_r\}$ is the probability vector of actions and $p(n+1) = T[\alpha(n), \beta(n), p(n)]$ is the learning algorithm. The following algorithm is a sample of linear learning algorithms. We assume that action α_i is selected at time-step *n*.

In case of desirable response from the environment:

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

In case of undesirable response from the environment:

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

In equation (1) and (2), *a* and *b* are reward and penalty parameters respectively. When *a* and *b* are equal, the algorithm is called L_{RP}, when *b* is much smaller than *a*, the algorithm is $L_{R\varepsilon P}$ and when *b* is zero, the algorithm is called L_{RI}.

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 relation (3). Afterward, the Learning Automaton selects an action α_i 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 relation (4) and if it is a punishment, it will use relation (5) to perform the update process.

$$\begin{aligned} & p_i(n) \\ &= prob[\alpha(n) = \alpha_i \mid V(n) \text{ is the set of active actions}, \alpha_i \in V(n)] \\ &= \frac{p_i(n)}{K(n)} \end{aligned} \tag{3}$$

In case of desirable response from the environment:

$$\begin{aligned} p_i(n+1) &= p_i(n) + a.(1 - p_i(n)) \quad \alpha(n) = \alpha_i \\ p_i(n+1) &= p_j(n) + a.p_i(n) \quad \alpha(n) = \alpha_i, \;\; \forall j \;\; j \neq i \end{aligned} \tag{4}$$

In case of undesirable response from the environment:

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

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

$$\begin{aligned} p_j(n+1) &= p_j(n+1).K(n) \quad \text{for all j}, \alpha_j \in V(n) \\ p_j(n+1) &= p_j(n) \quad \text{for all j}, \alpha_j \notin V(n) \end{aligned} \tag{6}$$

III. CONTROL OF MULTI –AGENT MARKOV DECISION PROCESS 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. Each state in MDP has a Learning Automaton that tries to learn the optimal probability distribution of actions during the process using rule (1) and (2). Thus, in this model, actions belong to the states rather than the agents. 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 corresponding Learning Automaton. In this model, only one Learning Automaton is active at a time and the transition from one state to another will activate the Learning Automaton of that state. This process continues until all probability vectors converge or a pre-specified condition satisfies. When the number of agents increases and the model extends to multi-agent case, more than one Learning Automaton should be active simultaneously, because since the states depend on the problem and the environment, agents could be in different states.

In this paper, two new algorithms are proposed for solving MMDPs. In these algorithms, the Markov problem is mapped on a directed graph. This mapping is done so that graph nodes indicate the states and directed edges represent actions of the MMDP that result in transition from one state to another. Next, a Learning Automaton is assigned to each node corresponding to the state that is assigned to that node. Agents start from "start node" and move toward "end node" or "goal state" in parallel. In each state, agents move to the next state with help of corresponding Learning Automaton and its probability vector. Each agent continues moving on the graph until it reaches the goal state. At that moment, the probability vector of Learning Automata included in agent's path will be updated based on cost of the path. This process will repeat several times for each agent and gradually, the path taken by each agent will converge to the optimal path or a pre-defined condition will be satisfied.

We have performed an experiment to investigate how a network of Learning Automata operates in control process of an MMDP. The environment of our experiment is shown in figure 2. This MMDP includes a 3*3 grid with a goal cell and two agents who try to reach it. Each agent starts its motion from the states specified in figure 3. This environment has also been used previously in [3] and [6].

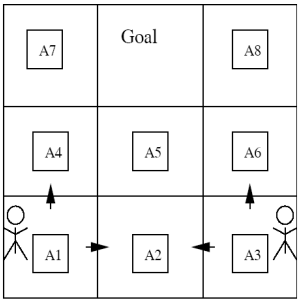

Figure 2. Sample MMDP used in our experiments [3,6]

To convert the problem to an MMDP, we consider each cell as a state and allowable transitions to adjacent cells as actions. In this MMDP, the objective is that both agents reach the goal state using distinct paths. When agents decide to move to the same cell (except the goal state), their transition will be prevented and they stay in their current cells. In addition to this objective, it is desirable that agents take shortest paths they can.

Thus, this multi-agent process is noteworthy from several viewpoints. First, this process is an MMDP that can be solved by a network of Learning Automata. Second, we can consider it as a multi-agent optimization problem for finding the shortest path in which agents cooperate to find the optimal Path. Third, according to the constraints that exist on agents' motions when they decide to move to the same cell, coordination between agents is necessary.

Our first proposed algorithm, which we call Algorithm1, is as follows:

Algorithm1:

1. *A directed graph corresponding to the problem is created; states and actions are mapped on nodes and edges respectively.*
2. *In each node of the graph, a Learning Automaton of type* L_{RP} *is placed. The set of actions of this Learning Automaton is the set of permissible movements to other states. The probability vector elements of the* Learning Automaton *are set to 1/number of permissible actions to other states.*
3. *The following steps are taken by all agents in parallel:*
   - 3.1. *Agent j is placed in start state.*
   - 3.2. *Dynamic threshold* T_k^j *is set to zero for the agent.*
   - 3.3. *At* k^j *-th step:*
     - 3.3.1. *When agent j arrives at state s, the Learning Automaton* A_s *corresponding to this state is activated and agent j gets help from this Learning Automaton to transit to the next state. This is done based of 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 s to next state, m.*
     - 3.3.2. *Agent j transits to state m.*
     - 3.3.3. $s \leftarrow m$, *we return to 3.3.1 until agent j reaches the goal state.*
     - 3.3.4. *When agent j reaches the goal state, the cost of path* π_i *taken by j is calculated using equation* $L^j_{\pi_i} = t^j_{\pi_i} / R_G$

       In this equation, R_G *is the reward of reaching the goal state, and* $t^j_{\pi_i}$ *is the time elapsed since starting from start state until reaching the goal state using path* π_i.
     - 3.3.5. *If condition* $L^j_{\pi_i} < T^j_k$ *is satisfied, the path is rewarded and if it is not, the path is punished. We consider the reward and the punishment equal to* $L^j_{\pi_i}$ *.this reward or punishment is given to all Learning Automata that contributed in the path and their probability vectors will be updated.*

3.3.6. $k^j \leftarrow k^j + 1$,

$$T_k^j \leftarrow T_{k-1}^j + \frac{1}{k^j}(L_{\pi_i}^j - T_{k-1}^j)$$

3.3.7. The termination condition of the algorithm (if the probability of agent j's path exceeds a pre-specified threshold) is verified. If this condition is not met, we return to 3.2.

If we consider the probability of a path as product of probabilities of its edges and set the termination condition to 0.95, mean number of iterations until convergence is approximately 2100. This is shown in figure 4.

As we can see, Algorithm1 suffers from possibility of existence of cycles in the paths taken by agents. This possibility increases with increase of learning rate or reward. The next algorithm is proposed to address this problem.

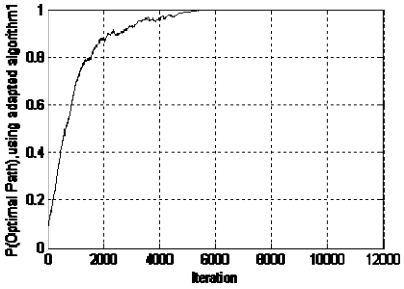

Figure 3. Probability of optimal Path for first agent using Algorithm1

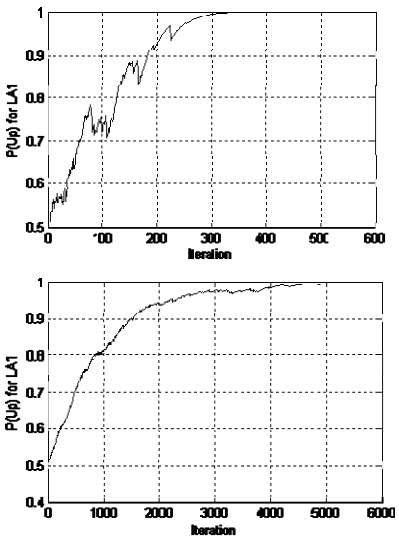

Figure 4. Probability of action "up" in starting cell of agent 1 using algorithm1 with learning rate 0.1 (1) and 0.01 (2)

Algorithm2:

1. *A directed graph corresponding to the problem is created; states and actions are mapped on nodes and edges respectively.*
2. *In each node of the graph, a Learning Automaton of type* L_{RP} *with variable number of actions is placed. The set of actions of the Learning Automaton is the set of permissible movements to other states. The probability vector elements of the Learning Automaton are set to 1/number of permissible actions to other states.*
3. *The following steps are taken by all agents in parallel:*
   - 3.1. *Agent j is placed in start state.*
   - 3.2. *Dynamic threshold* T_k^j *is set to zero for the agent.*
   - 3.3. *At* k^j *-th step:*
     - 3.3.1. *When agent j arrives at state s, the Learning Automaton* A_s *corresponding to this state is activated and agent j gets help from this Learning Automaton to transit to the next state. This is done based on the probability vector of Learning Automaton, by selecting an action* a_m *corresponding to directed edge (s,m) of the graph which means transition from s to next state, m. The action selection process is performed as follows:* A_s *deactivates actions that result in transition to states which have been previously visited in agent j's path.* A_s *then selects an action from active actions set based on its probability vector. If all actions are inactive and no action can be chosen by* A_s, $k^j \leftarrow k^j + 1$, *all actions will be activated and we go to 3.3.7; else, after selecting an action, all inactive actions will be activated to become ready to be used by other agents.*
     - 3.3.2. *Agent j transits to state m.*
     - 3.3.3. $s \leftarrow m$, *we return to 3.3.1 until agent j reaches the goal state.*
     - 3.3.4. *When agent j reaches the goal state, the cost of path* π_i *taken by j is calculated using equation* $L_{\pi_i}^j = t_{\pi_i}^j / R_G$. *In this equation,* R_G *is the reward of reaching the goal state, and* $t_{\pi_i}^j$ *is the time elapsed since starting from start state until reaching the goal state taking path* π_i.
     - 3.3.5. *If condition* $L_{\pi_i}^j < T_k^j$ *is satisfied, the path is rewarded and if it is not, the path is punished. We consider the reward and the punishment equal to* $L_{\pi_i}^j$ *.this reward or punishment is given to all Learning Automata that contributed in the path and their probability vectors will be updated.*

3.3.6. $k^j \leftarrow k^j + 1$,

$$T_k^j \leftarrow T_{k-1}^j + \frac{1}{k^j}(L_{\pi_i}^j - T_{k-1}^j)$$

3.3.7. *The termination condition of the algorithm (if the probability of agent j's path exceeds a pre-specified threshold) is verified. If this condition is not met, we return to 3.2.*

As shown in figure 5, using Algorithm2 with learning rate 0.01 and threshold 0.95, the optimal path is approximately reached in 2500 iterations. The reason why Algorithm2 has operated slower than Algorithm1 is that our test MMDP environment is small and cycles are often created. In the situation of getting stuck in a cell with no active actions, the agent has to terminate its motion and start from the beginning. In this case, none of the probability vectors are updated.

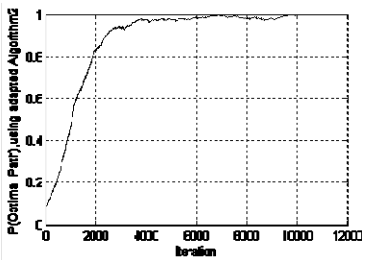

Figure 5. Probability of optimal Path for first agent using Algorithm2

To compare the functionally of our proposed algorithms with other existing algorithms based on Learning Automata, we have done another experiment. In this experiment, the approach of Ref. [6] for solving MMDPs has been applied on the same problem. Similar to our algorithms, each cell is equipped with a Learning Automaton and its actions are set to possible movements to adjacent cells. When Learning Automaton A_i in state i is activated, it will not be informed about the immediate reward $r_j^i(k)$ yielded from action k that results in transition from state i to j; instead, when state i is visited again, A_i receives two parts of data: the cumulative reward from the beginning of the process up to then, and the global current time. Using these data, A_i calculates the reward received since the last visit of state i and corresponding elapsed time. Then, the input to A_i is calculated using the following equation:

$$\beta^i(n_i+1) = \frac{\rho_k^i(n_i+1)}{\eta_k^i(n_i+1)} \qquad (7)$$

In equation (7), $\rho_k^i(n_i+1)$ is the whole reward for action k in state i, and $\eta_k^i(n_i+1)$ is the elapsed time.

The result of this experiment is shown in figure 6. According to this diagram, it takes approximately 4000 iterations for the agents to reach the optimal path while the leaning rate is 0.01 and the threshold is 0.95.

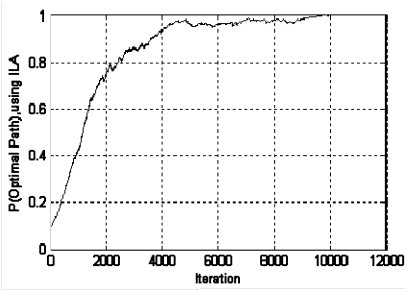

Figure 6. Probability of optimal Path for first agent using Algorithm2

Table 1 is a comparison between the three algorithm discussed in this paper. We should consider that the optimal path achieved by these algorithms, is in fact an optimal policy for the MMDP.

TABLE I.
MAXIMUM NUMBER OF ITERATIOS FOR THE FIRST AGENT TO REACH THE OPTIMAL PATH. (THRESHOLD= 0.95, LEARNING RATE= 0.01)

Used Algorithm	*Maximum Iterations*
Algorithm1	2016
Algorithm2	2863
The Algorithm of Ref [6]	3995

IV. CONCLUSION

In this paper, two new algorithms are suggested for solving MMDPs. These algorithms use a network of interconnected Learning Automata. The agents get help from these Learning Automata in their decision process in order to transit between states and to find the optimal policy. To assess our proposed approach, we have solved an MMDP which is also a distributed optimization problem some coordination and cooperation aspects among the agents. The results of our experiments have shown that these algorithms perform much better in terms of cost and speed in comparison with existing algorithms that make use of Learning Automata to find optimal policies in MMDPs.

REFERENCES

[1] G. Weiss, Multi-Agent Systems, a Modern Approach to Distributed Artificial Intelligence, *MIT Press, 2000*.

[2] A. Nowe, K. Verbeeck, and M. Peeters, "Learning Automata as a Basis for Multi-Agent Reinforcement Learning", First International Workshop on Learning and Adaptation in Multi-Agent Systems (LAMAS), pp. 71-85, 2006.

[3] J. Hu, and M. P. Wellman, "Multi-Agent Reinforcement Learning: Theoretical Framework and an Algorithm", Proceedings of the Fifteenth International Conference on Machine Learning, 1998.

[4] P. Laroche, Y. Boniface, and R. Schott, "A New Decomposition Technique for Solving Markov decision Processes", Proceedings of the 2001 ACM Symposium on Applied Computing, Las Vegas, Nevada, United States, pp. 12 – 16, 2001.

[5] L. Peret, and F. Garcia, "On-line search for solving large Markov Decision Processes", Sixth European Workshop on Reinforcement Learning (EWRL-6), 2003.

[6] A. Nowe, and k. Verbeeck, "Colonies of Learning Automata", IEEE Transactions on Systems, Man and Cybernetics, vol. 32, pp. 772-780, 2002.

[7] H. 7. Beigy, and M. R. Meybodi, "Utilizing Distributed Learning Automata to Solve Stochastic Shortest Path Problem", International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, vol. 14, pp. 591-615, 2006.

[8] L. Busoniu, B. De Schutter, and R. Babuska, "Learning and Coordination in Dynamic Multi-Agent Systems", Delft Center for Systems and Control, 2005.

[9] H. 9. Hanna, and A. Mouaddid, "Markov Decision Process for Allocation Task in Multi-Agent Systems", International Conference IPMU, France, 2002.

[10] M. Spaan, N. Vlassis, and F. Groen, "High Level Coordination of Agents based on Multi-Agent Markov Decision Processes with Roles", IROS'02 *Workshop on Cooperative Robotics*, 2002.

[11] M. R. Shirazi, and M. R. Meybodi, "Application of Learning Automata to Cooperation in Multi-Agent Systems", Proceedings of First International Conference on Information and Knowledge Technology (IKT2003), pp. 338-349, 2003.

[12] H. Hetland, T. L. Eriksen, "Interconnected Learning Automata Playing Iterated Prisoner's Dilemma", Master Thesis, Agder University College, 2004.

[13] A. Nowe, K. Verbeeck, M. Peeters, and K. Tuyls, "Multi-Agent Reinforcement Learning in stochastic Single and Multi-Stage Games", Lecture Notes in Computer Science, Springer Berlin, vol. 3394, pp. 275-294, 2005.