Multi-Automata Learning

Verbeeck Katja[1], Nowe Ann[2], Vrancx Peter[2] and Peeters Maarten[2]
[1]MiCC-IKAT Maastricht University, The Netherlands
[2] COMO Vrije Universiteit Brussel, Belgium

1. Introduction

It is well known that Reinforcement Learning (RL) techniques are able to solve Markovian stationary decision problems (MDP) with delayed rewards. However, not much is known yet about how these techniques can be justified in non-stationary environments. In a non-stationary environment, system characteristics such as probabilities of state transitions and reward signals may vary with time. This is the case in systems where multiple agents are active, the so called Multi-Agent Systems (MAS). The difficulty in a MAS is that an agent is not only subject to external environmental changes, (like for instance load changes in a telecommunication network setting) but, also to the decisions taken by other agents, with whom the agent might have to cooperate, communicate or compete. So a key question in multi-agent Reinforcement Learning (MARL) is how multiple reinforcement learning agents can learn optimal behavior under constraints such as high communication costs. In order to solve this problem, it is necessary to understand what optimal behavior is and how can it be learned.

In a MAS rewards are sensed for combinations of actions taken by different agents, and therefore agents are actually learning in a product or joint action space. Moreover, due to the existence of different reward functions, it usually is impossible to find policies which maximize the expected reward for all agents simultaneously. The latter is possible in the so-called team games or multi-agent MDP's (MMDP's). In this case, the MAS is purely cooperative and all agents share the same reward function. In MMDP's the agents should learn how to find and agree on the same optimal policy. In general, an equilibrium point is sought; i.e. a situation in which no agent on its own can change its policy to improve its reward when all other agents keep their policy fixed.

In addition, agents in a MAS face the problem of incomplete information with respect to the action choice. One can assume that the agents get information about their own choice of action as well as that of the others. This is the case in what is called joint action learning, (Littman, 2001),(Claus & Boutilier, 1998), (Hu & Wellman, 2003). Joint action learners are able to maintain models of the strategy of others and explicitly take into account the effects of joint actions. In contrast, independent agents only know their own action. The latter is often a more realistic assumption since distributed multi-agent applications are typically subject to limitations such as partial or non observability, communication costs, asynchronism and stochasticity.

Our work in MARL is mainly motivated by the early results achieved by simple learning automata (LA) that can be interconnected in games, networks and hierarchies. A learning

Source: Reinforcement Learning: Theory and Applications, Book edited by Cornelius Weber, Mark Elshaw and Norbert Michael Mayer
ISBN 978-3-902613-14-1, pp.424, January 2008, I-Tech Education and Publishing, Vienna, Austria

automaton describes the internal state of an agent as a probability distribution according to which actions should be chosen. These probabilities are adjusted with some reinforcement scheme according to the success or failure of the actions taken. Important to note is that LA are updated strictly on the basis of the response of the environment, and not on the basis of any knowledge regarding other automata, i.e. nor their strategies, nor their feedback. As such LA agents are very simple. Moreover, LA can be treated analytically, from a single automaton model acting in a simple stationary random environment to a distributed automata model interacting in a complex environment.

The past few years, a substantial amount of research has focused on comprehending (Tuyls & Nowé 2005) and solving single-stage multi-agent problems, modeled as normal form games from game theory e.g. joint-action learners (Claus & Boutilier, 1998); ESRL (Verbeeck et al, 2007 (b)) or Commitment Sequences (Kapetanakis et al, 2003). Recently, more researchers focus on solving the more challenging multi-stage game or sequential games where the agents have to take a sequence of actions. These can be modeled as Markov Games (Shapley, 1953). Many real-world problems are naturally translated into multi-stage problems. The expressiveness of multi-stage games allows us to create more realistic simulation models with less abstraction. This brings us closer to the application level. Moreover, we argue that multi-stage games help to improve the scalability of an agent system.

Here we present a summary of current LA-based approaches to MARL. We especially focus on multi-stage multi-agent decision problems of the following type : ergodic Markov Games, partial observable ergodic Markov Games and episodic problems that induce tree-based Markov games. We describe both their analytical as well as their experimental results, and we discuss their contributions to the field. As in single agent learning, we consider the different updating mechanisms relevant to sequential decision making; i.e. using global reward signals for updating called Monte Carlo updating, versus using intermediate rewards to update strategies, the so-called Bootstrapping methods.

First we start with a short description of the underlying mathematical material for analyzing multi-agent, multi-stage learning schemes.

2. Markov decision processes and Markov games

In this section we briefly explain the formal frameworks used in single and multi-agent learning.

2.1 The Markov property

A Stochastic process $\{X(t) \mid t \in T\}$ is a system that passes from one state to another governed by time. Both the state space S as the index set (time) T can be either discrete or continuous. A Markov process is a stochastic process that satisfies the *Markov property*. This property states that the future behavior of the process given its path only depends on its current state. A Markov process whose state space is discrete is also called a Markov chain, whereas a discrete-time chain is called stationary or homogenous when the probability of going from one state to another in a single step is independent of time. So, if the current state of the

Markov chain at time t is known, transitions to a new state at time $t + 1$ are independent of any of the previous states.

One of the most important questions in the theory of Markov chains is how the chain will be distributed among the states after a long time. In case the Markov chain is ergodic, a unique stationary probability distribution exists. However the process can also be absorbed into a closed set of states. An absorbing state is a state from which there is a zero probability of exiting. An absorbing Markov system is a Markov system that contains at least one absorbing state, and possesses the property that it is possible to get from each non-absorbing state to some absorbing state in one or more time-steps. More information on the properties of Markov processes can be found in (Puterman, 1994).

2.1 Definition of an MDP

A Markov Decision Process (MDP) is a discrete-time Markov process characterized by a set of states; each having several actions from which a decision maker must choose. The decision maker earns a reward for each state visited. The problem of controlling an MDP for which transition probabilities and rewards are unknown can be stated as follows. Let $S = \{s_1, \ldots, s_N\}$ be the state space of a finite Markov chain $\{X_t\}_{t \geq 0}$ and $A_i = \{a_{i1}, \ldots, a_{il}\}$ the action set 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 $T_{i\text{->}j}(a_i)$ and reward $R_{i\text{->}j}(a_i)$. The overall goal is to learn a policy α, or a set of actions, $\alpha = (a_1, \ldots, a_N)$ with $a_j \in A_j$ so that the expected average reward for policy α : $J(\alpha)$ is maximized:

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

The policies we consider, are limited to stationary, nonrandomized policies. Under the assumption that the Markov chain corresponding to each policy α is ergodic, it can be shown that the best strategy in any state is a pure strategy, independent of the time at which the state is occupied (Wheeler & Narendra, 19986). Assume the limiting distribution of the Markov chain to be $\pi(a) = (\pi_1(a), \ldots, \pi_N(a))$ with forall i, $\pi_i(a) > 0$ as $n \to \infty$. Thus, there are no transient states and the limiting distribution $\pi(a)$ can be used to rewrite Equation 1 as:

$$J(\alpha) \equiv \sum_{i=1}^{N} \pi_i(\alpha) \sum_{j=1}^{N} T^{ij}(\alpha) R^{ij}(\alpha) \tag{2}$$

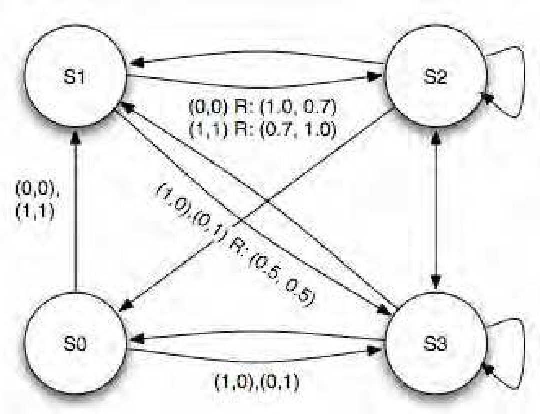
2.2 Definition of a Markov game

An extension of single agent Markov decision problems to the multi-agent case is straightforward and can be defined by Markov Games (Shapeley, 1953). In a Markov Game, actions are the joint result of multiple agents choosing an action separately. Note that $A_{i,k} = \{a_{i,k1}, \ldots, a_{i,kl}\}$ is now the action set available in state s_i for agent k, with $k: 1 \ldots n$, n being the total number of agents present in the system. Transition probabilities $T_{i\text{->}j}(a_i)$ and rewards $R_{i\text{->}j}(a_i)$ now depend on a starting state s_i, ending state s_j and a joint action a_i from state s_i, i.e. $a_i = (a_{i1}, \ldots, a_{in})$ with $a_{ik} \in A_{i,k}$. The reward function $R_{i\text{->}j}(a_i)$ is now individual to each agent k, indicated as R_k^{ij}. 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 non-trivial.

Again we will only treat non-randomized policies and we will assume that the Markov Game is ergodic in the sense that there are no transient states present and a limiting distribution on the joint policies exists. We can now use Equation 2 to define the expected reward for agent k , for a given joint policy α.

$$J_k(\alpha) \equiv \sum_{i=1}^{N} \pi_i(\alpha) \sum_{j=1}^{N} T^{ij}(\alpha) R_k^{ij}(\alpha) \tag{3}$$

Due to the existence of different reward functions, it is in general impossible to find an optimal policy for all agents simultaneously. Instead, equilibrium points are sought. In an equilibrium, no agent can improve its reward by changing its policy if all other agents keep their policy fixed. In the case of single state multi-agent problems, the equilibrium strategies coincides with the Nash equilibria of the corresponding normal form game. In the case of multi stage problems, limiting games can be used as analysis tool. The limiting game of a corresponding multi-agent multi-state problem can be defined as follows: each joint agent policy is viewed as a single play between players using the agent's policies as their individual actions. The payoff given to each player is the expected reward for the corresponding agent under the resulting joint policy. Analyzing the multi state problem now boils down to explaining the behaviour of the multi-agent learning technique in terms of Nash equilibriums in this limiting game.



		Agent 2			
		(0,0)	(0,1)	(1,0)	(1,1)
Agent 1	(0,0)	**0.29, 0.20**	0.17, 0.17	0.14, 0.12	0.08, 0.08
	(0,1)	0.17, 0.17	**0.20, 0.29**	0.08, 0.08	0.12, 0.14
	(1,0)	0.14, 0.12	0.08, 0.08	**0.29, 0.20**	0.17, 0.17
	(1,1)	0.08, 0.08	0.12, 0.14	0.17, 0.17	**0.20, 0.29**

Fig. 1. A Markov Game Problem and its corresponding limiting game

Figure 1 shows an example Markov game with 4 states and 2 agents. States s_0 and s_1 are the only action states, with 2 possible actions (*0* and *1*) for each each agent. Joint actions and nonzero rewards (R) associated with the state transitions are indicated in the figure. All transitions are deterministic, except in the non-action states s_2 and s_3 where the process goes to any other state with equal probability (1/4). The corresponding limiting game for this problem is shown in the accompanying table. Equilibria in this game are indicated in bold.
In a special case of the general Markov game framework, the so-called team games or multi-agent MDP's (MMDP's) (Boutilier, 1999) optimal policies are proven to exist. In this case, the Markov game is purely cooperative and all agents share the same reward function. This specialization allows us to define the optimal policy 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 cooperative multi-agent case.

3. Learning automata as simple policy iterators

The study of learning automata started in the 1960's by Tsetlin and his co-workers (Tsetlin, 1973). The early models were examples of fixed-structure stochastic automata. In its current form, LA are closely related to a Reinforcement Learner of the policy iteration type. Studying learning automata theory is very relevant for multi-agent reinforcement learning, since learning is treated analytically not only in the single automaton setting, but also in the case of distributed interconnected automata and hierarchies of automata interacting in complex environments (Narendra & Thathachar, 1989) . In this section, we only discuss the single automaton case in stationary environments. More complicated LA models and their behaviour will be discussed in the following sections.
A variable structure learning automaton formalizes a general stochastic system in terms of actions, action probabilities and environment responses. The action probabilities, which make the automaton mathematically very tractable, are updated on the basis of the environment input. Formally, the automata can be defined as follows: a quadruple $\{ A, r, p, U \}$ for which : $A = \{a_1, \ldots, a_l \}$ is the action or output set of the automaton, $r(t)$ is an element of the set $\{0,1\}$ and denotes the environment response at instant t, $p(t) = (p_1(t), \ldots , p_l(t))$ is the action probability vector of the automaton, with $p_i(t) = Prob(a(t) = a_i)$ and which satisfies the condition that $\Sigma^l_{i=1}\, p_i(t)=1$ for all t and U is called the learning algorithm and denotes the schema with which the action probabilities are updated on the basis of the environment input. The output a of the automaton is actually the input to the environment. The input r of the automaton is the output of the environment. In general the environment refers to all external conditions and influences affecting life and development of an organism. In the P-model learning automaton, the output $r(t)$ of the environment is considered to be binary. The output set of the environment is the set $\{0,1\}$ so that output signal $r = 0$ is identified with a failure or an unfavourable response, while $r = 1$ denotes a success or a favourable response. Static environments are characterized by penalty probabilities c_i . They represent the probability that the application of action a_i will be successful or not, i.e. $Prob(r = 0 \mid a = a_i) = c_i$. Knowing c_i , the reward probability d_i for action a_i is given by: $d_i = 1 - c_i$. Other environment models exist, depending on the nature of the environment response. In the Q-model the environment response is an element of a discrete, finite set of possible responses which has more than 2 elements, while in the S-model the environment response r is a real number in the interval $[0,1]$.
Important examples of linear update schemes are linear reward-penalty, linear

reward-inaction and linear reward-ε-penalty. The philosophy of those schemes is essentially to increase the probability of an action when it results in a success and to decrease it when the response is a failure. The general algorithm is given by:

$$
\begin{aligned}
p_m(t+1) &= p_m(t) + \alpha_r(1-\beta(t))(1-p_m(t)) - \alpha_p\beta(t)p_m(t) \\
&\quad \text{if } a_m \text{ is the action taken at time } t \\
p_j(t+1) &= p_j(t) - \alpha_r(1-\beta(t))p_j(t) + \alpha_p\beta(t)\left[(r-1)^{-1} - p_j(t)\right] \\
&\quad \text{if } a_j \neq a_m
\end{aligned}
\tag{4}
$$

with l the number of actions of the action set A. The constants α_r and α_p are the reward and penalty parameters respectively. When $\alpha_r = \alpha_p$ the algorithm is referred to as linear reward-penalty L_{R-P}, when $\alpha_p = 0$ it is referred to as linear reward-inaction
L_{R-I} and when α_p is small compared to α_r it is called linear reward-ε-penalty $L_{R-\varepsilon P}$. In the above, it is assumed that c_i and d_i are constants. This implies that the environment is stationary and that the optimal action a_m can be identified. The goal of the learning automaton is to find this optimal action, without knowing the environments' reward or penalty probabilities. The penalty probability c_m of the optimal action has the property that $c_m = min_i \{ c_i \}$. Optimality of the learning automaton can then be defined using the quantity $M(t) = E[r(t) = 0 \mid p(t)]$ which is the average penalty for a given action probability vector. Consider for instance a pure-chance automaton, i.e. the action probability vector $p(n)$ is given by: $p_i(n) = 1 / l$ for all $i: 1, \dots, l$. Then $M(t)$ is a constant (denoted by M_0) and given by:

$$M_0 = 1 \Big/ l \sum_{i=1}^{l} c_i$$

Definition 1

A learning automaton is called optimal if $\lim_{t\to\infty} E[M(t)] = c_m$

While optimality is desirable in stationary environments, practically it may not be achieved in a given situation. In this case, ε-optimality may be reached. So, put differently, the objective of the learning scheme is to maximize the expected value of reinforcement received from the environment, i.e. $E[r(t) \mid p(t) = p]$ by searching the space of all possible action probability vectors. Stated as above, a learning automata algorithm can be viewed as a policy iteration approach.

In arbitrary environments and for arbitrary initial conditions, optimality or ε-optimality may be hard to reach. Some form of desired behaviour in these cases can be specified by expediency and absolute expediency.

Definition 2

A learning automaton is called expedient if it performs better than a pure-chance automaton, i.e. $\lim_{t\to\infty} M(t) < M_0$

Definition 3

A learning automaton is said to be absolutely expedient if $E\left[M(t+1) \middle| p(t)\right] < M(t)$

Absolute expediency imposes an inequality on the conditional expectation of $M(t)$ at each instant. In (Narendra & Thathachar, 1989) it is shown that in stationary environments absolute expediency implies ε-optimality.

The reinforcement learning algorithms given above, i.e. the L_{R-P} , L_{R-I} and $L_{R-\varepsilon P}$ schemes show the following asymptotic behaviour: the L_{R-I} scheme is proved to be absolutely

expedient and thus ε-optimal in stationary environments. The L_{R-P} scheme is found to be expedient, while the $L_{R-\varepsilon P}$ scheme is also ε-optimal, (Narendra & Thathachar, 1989).
Another classification used for reinforcement schemes is made on the basis of the properties of the induced Markov process $\{p(t)\}_{\{t>0\}}$. If the penalty probabilities c_i of the environment are constant, the probability $p(t+1)$ is completely determined by $p(t)$ and hence $\{p(t)\}_{\{t>0\}}$ is a discrete-time homogeneous Markov process. The L_{R-P} and $L_{R-\varepsilon P}$ schemes result in Markov processes that are ergodic, the action probability vector $p(t)$ converges in distribution to a random variable p^*, which is independent of the initial conditions. In case of the $L_{R-\varepsilon P}$ scheme, the mean value of p^* can be made as close as desired to the optimal unit vector by choosing α_r and α_p sufficiently small. The Markov process generated by the L_{R-I} scheme is non-ergodic and converges to one of the absorbing states with probability *1*.
Choosing parameter α_r sufficiently small can make the probability of convergence to the optimal action as close to *1* as desired. More on the convergence of learning schemes can be found (Narendra & Thathachar, 1989).

4. Interconnected learning automata for ergodic Markov games

It is well known that Reinforcement Learning techniques (Sutton & Barto, 1998) are able to solve single-agent Markovian decision problems with delayed rewards. In the first subsection we focus on how a set of interconnected LA is able to control an MDP (Wheeler & Narendra, 1986) . In the next subsection, we show how to extend this result to the multi-agent case in a very natural way.

4.1 Control of MDP's

The problem of controlling a Markov chain can be formulated as a network of automata in which control passes from one automaton to another. In this set-up every state in the Markov chain has a LA that tries to learn the optimal action probabilities in that state with learning scheme given in Equation 4. Only one LA is active at each time step and transition to the next state triggers the LA from that state to become active and take some action. LA LAi active in state si is not informed of the one-step reward Ri->j(ai) resulting from choosing action ai ЄAi in si and leading to state sj. However when state si is visited again, LAi receives two pieces of data: the cumulative reward generated by the process up to the current time step and the current global time. From these, LAi computes the incremental reward generated since this last visit and the corresponding elapsed global time. The environment response or the input to LA^i is then taken to be:

$$\beta^i(t_i+1)=\frac{\rho^i(t_i+1)}{\eta^i(t_i+1)} \tag{5}$$

where $\rho_i(t_i + 1)$ is the cumulative total reward generated for action a_i in state s_i and $\eta_i(t_i + 1)$ the cumulative total time elapsed. The authors in (Wheeler & Narendra, 1986) denote updating scheme as given in Equation 4 with environment response as in Equation 5 as learning scheme T1. The following results were proved:
Lemma1 *(Wheeler & Narendra, 1986)*
The Markov chain control problem can be asymptotically approximated by an identical payoff game of N automata.

Theorem 1 *(Wheeler & Narendra, 1986)*
Let for each action state s_i of an N state Markov chain, an automaton LA^i using the Monte Carlo updates as described above 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).
The principal result derived is that, without prior knowledge of transition probabilities or rewards, the network of independent decentralized LA controllers is able to converge to the set of actions that maximizes the long-term expected reward (Narendra & Thathachar, 1989). Moreover instead of one agent visiting all states and keeping a model for all the states in the system as in traditional RL algorithms such as Q-learning; in this model there are some non-mobile LA agents who do not move around the state space but stay in their own state waiting to get activated and learn to take actions only in their own state. The intelligence of one mobile agent is now distributed over the states of the Markov chain, more precisely over the non-mobile LA agents in those states.

4.2 Control of Markov games

In a Markov Game the action chosen at any state is the joint result of individual action components performed by the agents present in the system. The LA network of the previous section can be extended to the framework of Markov Games just by putting a simple learning automaton for every agent in each state (Vrancx et al., 2007) Instead of putting a single learning automaton in each action of the system, we propose to put an automaton $LA^{i,k}$ in each state s_i with $i: 1 \ldots N$ and for each agent k, $k: 1 \ldots n$. 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 take some joint action.
As before, LA $LA^{i,k}$ active for agent k in state s_i is not informed on the one-step reward $R_{i\text{->}j,k}(a_i)$ resulting from choosing joint action $a_i = (a_{i1}, \ldots, a_{in})$ with $a_{ik} \in A_{i,k}$ in s_i and leading to state s_j. When state s_i is visited again, all automata $LA^{i,k}$ with $k: 1 \ldots n$ receive two pieces of data: the cumulative reward generated for agent k by the process up to the current time step and the current global time. From these, all $LA^{i,k}$ compute the incremental reward generated since this last visit and the corresponding elapsed global time. The environment response or the input to $LA^{i,k}$ is exactly the same as in Equation 6. The following result was proven in (Vrancx et al., 2007) :
Theorem 2 *(Vrancx et al, 07)*
The Learning Automata model proposed for ergodic Markov games with full state observability is able to find an equilibrium point in pure strategies for the underlying limited game.
The behaviour of the LA learning model on the sample problem described in section 2.2 is demonstrated in Figure 2. We show the average reward over time for both agents. Since we are interested in the long term convergence we show a typical run, rather than an average over multiple runs. To demonstrate convergence to the different equilibria, we use a single very long run in which the automata are allowed to converge and are then restarted. After every restart the automata are initialized with random action probabilities in order to allow them to converge to different equilibria.

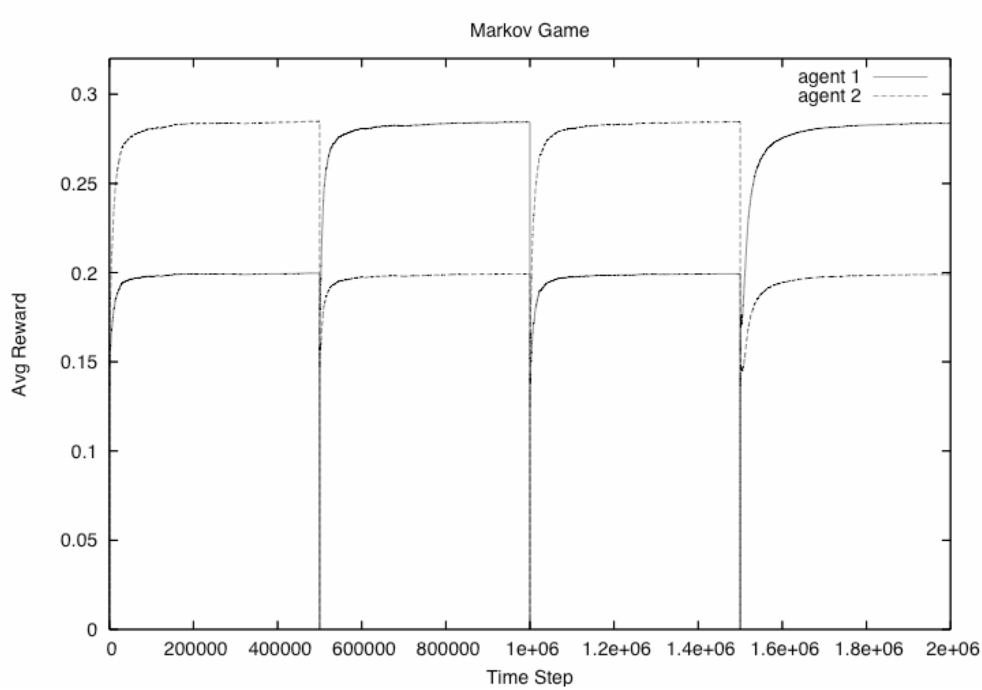


Fig. 2. Experimental results on the example Markov game of Figure 1.

5. Interconnected learning automata for partially observable ergodic Markov games

The main difference with the previous setup for learning Markov games (Vrancx et al., 2007) is that here we do not assume that agents can observe the complete system state. Instead, each agent learns directly in its own observation space, by associating a learning automaton with each distinct state it can observe. Since an agent does not necessarily observe all state variables, it is possible that it associates the same LA with multiple states, as it cannot distinguish between them. For example, in the *2*-state problem of Figure X, an agent associates a LA with each location it can occupy, while the full system state consists of the joint locations of all agents. As a consequence, it is not possible for the agents to learn all policies. For instance in the *2*-state problem, the automaton associated by agent x with location *L1* is used in state $s_1 = \{L1,L1\}$ as well as state $s_2 = \{L1,L2\}$. Therefore it is not possible for agent x to learn a different action in state s_1 and s_2. This corresponds to the agent associating actions with locations, without modelling the other agents.

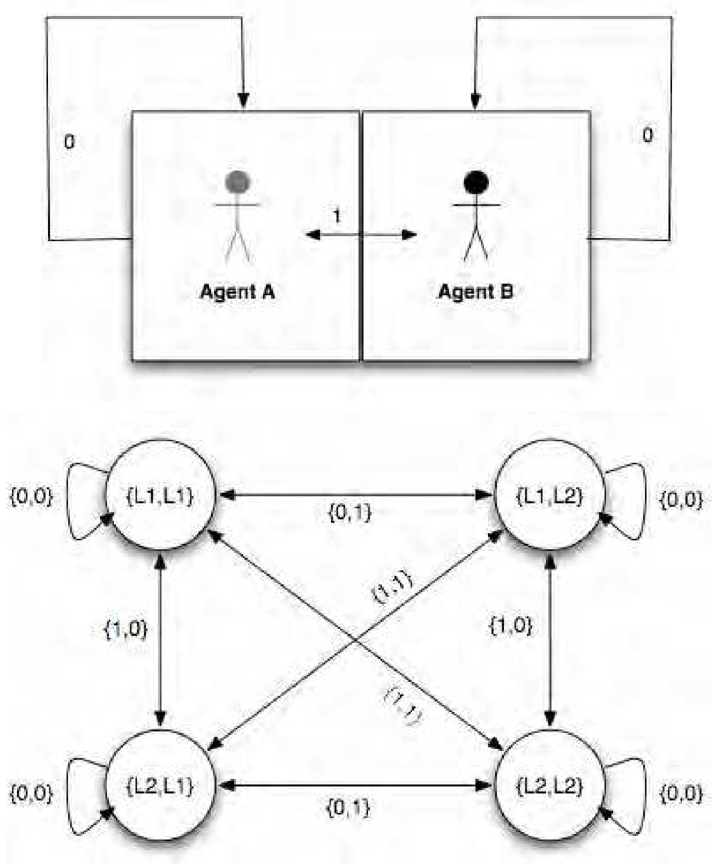


Fig. 3. A 2-location grid-world problem and its corresponding Markov game.

So the definition of the update mechanism here is exactly the same as in the previous model, the difference is that here agents only update their observable states which we will call locations to differentiate with the notion of a Markov game state. This will give the following: a LA $LA^{i,k}$ active for agent k in location L_i is not informed on the one-step reward $R_{i\text{->}j,k}(a_i)$ resulting from choosing joint action $a_i = (a_{i1}, \ldots, a_{in})$ with $a_{ik} \in A_{i,k}$ in s_i and leading to state L_j. Instead, when location L_i is visited again, automaton $LA^{i,k}$ receives two pieces of data: the cumulative reward generated for agent k by the process up to the current time step and the current global time. From these, automaton $LA^{i,k}$ compute the incremental reward generated since this last visit and the corresponding elapsed global time.
The environment response or the input to $LA^{i,k}$ is then taken to be: $\beta_{i,k}(t_i + 1) = \rho_{i,k}(t_i + 1) / \eta_{i,k}(t_i + 1)$ where $\rho_{i,k}(t_i + 1)$ is the cumulative total reward generated for action $a_{i,k}$ in location L_i and $\eta_{i,k}(t_i + 1)$ the cumulative total time elapsed. We still assume that the Markov chain of system states generated under each joint agent policy a is ergodic. In the following we will show that even when the agents have only knowledge of their own location, in some situations it is still possible to find an equilibrium point of the underlying limiting game.

		Agent 2			
		(0,0)	(0,1)	(1,0)	(1,1)
Agent 1	(0,0)	0.38	0.28	0.48	0.38
	(0,1)	0.48	0.14	**0.82**	0.48
	(1,0)	0.28	**0.42**	0.14	0.28
	(1,1)	0.38	0.28	0.28	0.38

Table 1. Limiting game for the 2-location grid world experiment of Figure 3.

Theorem 3 *(Verbeeck et al, 2007(a))*
The network of LA that was proposed here for myopic agents in Markov Games, converges to a pure equilibrium point of the limiting game provided that the Markov chain of system states generated under each joint agent policy is ergodic and agents' transition probabilities do not depend on other agents' activities.

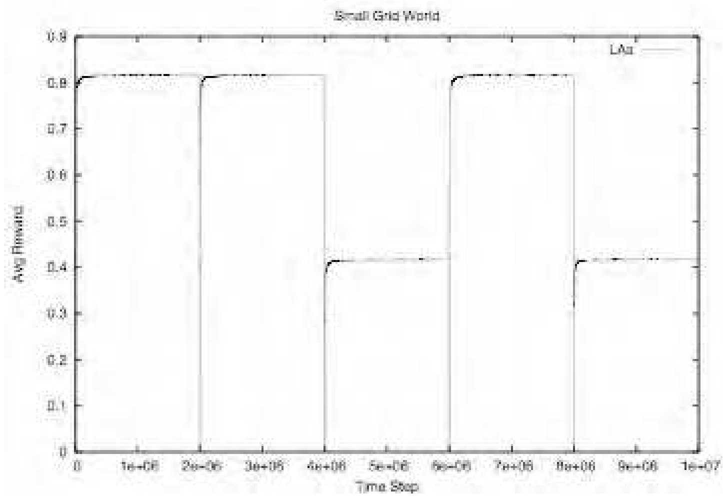


Fig. 4. Average Reward for the 2-location grid-world problem of Figure 3 .

Figure 4 shows experimental results of the LA network approach on the grid world problem of Figure 3. In these experiments both agents were given an identical reward based on their joint location after acting. The agents receive rewards *1.0* and *0.5* for joint locations *{L1,L2}* and *{L2,L1}* respectively and reward *0.01* when both agents are in the same location. The resulting limiting game matrix for this reward function is shown in Table 1. In Figure 4 we show the average reward over time for both agents, during a single long run of the algorithm, in which agents are allowed to converge and are then randomly re-initialised. We can observe that the agents move to either the optimal or the suboptimal equilibrium of the underlying limiting game, depending on their initialisation.

6. Hierarchical learning automata for episodic problems

Until now we only considered ergodic Markov games, which excludes problems that have a periodic nature or absorbing states such as finite episodic tasks. To this end we study agents constructed with hierarchical learning automata. An agent here is more complex than a simple learning automaton in the sense that the agents' internal state is now a hierarchy of learning automata. The idea is that in each step of the episode the agent chooses to activate an automaton of the next level of its hierarchy. The numbers of steps of the episodic task defines the size of the internal LA hierarchy. In (Narendra & Parthasarathy, 1991), hierarchical learning automata were introduced for multi-objective optimization. A simple problem of consistently labelling images was given. At a first stage, the object had to be recognized and in a second stage the background of the image was determined.
When several hierarchical agents play an episodic multi-agent task a corresponding Markov game can be determined as follows: at each time step, there is a new state available for each joint action possible. The resulting state space is then a tree, so no loops or joining branches are allowed. Similar to the previous section,, it is the case that learning automata can belong to different states, and we could call this setting an POMarkov game (in analogy with POMDP's). Note that in this section we only consider cooperative tasks.

6.1 Hierarchical LA agents

Learning automata can be combined into more complex structures such as hierarchies. Figure 5 shows an interaction between such hierarchies.

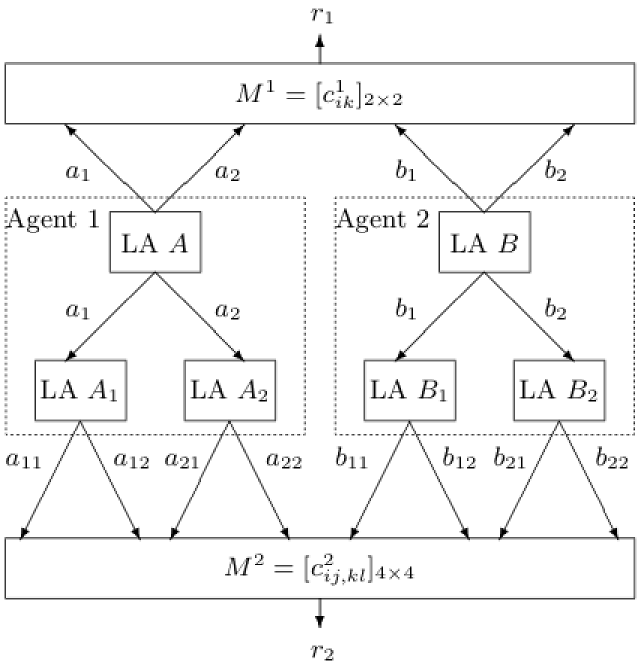


Fig. 5. An interaction between two agents constructed with a hierarchy of learning automata.

A hierarchical LA works as follows. The first automaton that is active is the root at the top of the hierarchy: LA *A* This automaton selects one of its *l* actions. If, for example, the automaton selects action *2*, the learning automaton that will become active is learning automaton LA *A2.* Then this active learning automaton is eligible for selecting an action. Based on this action, another learning automaton at the next level will become active. This process repeats itself until one of the learning automata at the bottom of the hierarchy is reached.
The interaction of the two hierarchical agents of Figure 5 goes as follows. At the top level (or in the first stage) Agent *1* and Agent *2* meet each other in a game with stochastic rewards.
They both take an action using their top level learning automata, respectively *A* and *B*. Performing actions a_i by *A* and b_k by *B* is equivalent to choosing automata A_i and B_k to take actions at the next level. The response of environment $E_1 : r_t \in \{0,1\}$, is a success or failure, where the probability of success is given by c^1_{ik} . At the second level the learning automata A_i and B_k choose their actions a_{ij} and b_{kl} respectively and these will elicit a response from the environment E_2 of which the probability of getting a positive reward is given by $c^2_{ij,kl}$. At the end of the episode all the automata that were involved in one of the games, update their action selection probabilities based on the actions performed and the responses of the environments.

6.2 Monte Carlo updating

In the Monte Carlo method, the updating of the probabilities is based on the averaged sample returns. This averaged return is usually generated at the end of an episode. Each time such a clear end state is reached, an averaged return is generated by calculating a weighted sum of all the returns obtained. This sum is then given to all learning automata that were active during the last episode in order to update their action probabilities. Thus when we reach an end stage at time step *t* we generate the following sum: $R = \theta_1 r_1 + \theta_2 r_2 + . . . + \theta_t r_t$ where r_i is the reward generated at time step *i*. Note that the weights θ_i should sum up to 1 and $0 \leq \theta_i \leq 1$ for all θ_i.
Theorem 4 *(Narendra & Parthasarathy, 1991)*
If all the automata of the hierarchical learning automata update their action probabilities at each stage using the L_{R-I} *update scheme and if the composite reward is constructed as a Monte Carlo reward and at each level the step sizes of the automata are chosen sufficiently small then the overall system is absolutely expedient.*
Stated differently, this means that the overall performance of the system will improve at each time step and convergence is assured toward a local optimum. The optima of the dynamical system under consideration, are the pure equilibrium points of the corresponding limiting single stage game.
Using a careful exploration strategy called exploring selfish reinforcement learning which is used in combination with hierarchical LA, it was shown in (Verbeeck et al., 2007) that the optimal equilibrium path can be learned.

7. Monte Carlo updating versus bootstrapping

Standard single agent reinforcement learning techniques, such as Q-learning (Watkins & Dayan, 1992), which are by nature designed to solve sequential decision problems, use the mechanism of bootstrapping to handle non-episodic tasks. Bootstrapping means that values or estimates are learned on the basis of other estimates (Sutton & Barto, 1998). The use of

next state estimates allows reinforcement learning to be applied to non-episodic tasks. Another advantage of bootstrapping over Monte Carlo methods include the fact that the former can be naturally implemented in an on-line, fully incremental fashion. As such, these methods can learn from each transition, which can sometimes speed-up learning time.
For instance the Q-learning algorithm, which is a value iteration method (see (Sutton & Barto, 1998); (Tsitsiklis, 1994)) bootstraps its estimate for the state-action value $Q_{t+1}(s,a)$ at time $t+1$ upon the estimate for $Q_t(s',a')$ with s' the state where the learner arrives after taking action a in state s :

$$Q_{t+1}(s,a) \leftarrow (1-\alpha)Q_t(s,a) + \alpha\left(r_t + \gamma \max_{a'} Q_t(s',a')\right) \tag{6}$$

With a the usual step size parameter, $\gamma \in [0,1]$ a discount factor and r_t the immediate reinforcement received at time step t. Non-bootstrapping evaluation methods such as Monte Carlo methods update their estimates based on actual returns. For instance the every-visit Monte Carlo method updates a state-action value $Q(s,a)$ at time $t+n$ (with n the time for one episode to finish) based on the actual return R_t and the previous value:

$$Q_{t+n}(s,a) \leftarrow (1-\alpha)Q_t(s,a) + \alpha(R_t) \tag{7}$$

With $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{n-1} r_n$ and t is the time at which (s,a) occurred.
Methods that learn their estimates, to some extend, on the basis of other estimates, i.e. they bootstrap are called Temporal Difference Learning Methods. The Q-learning algorithm seen in Equation 7 can be classified as a TD(0) algorithm. The back-up for each state is based on the immediate reward, and the estimation of the remaining rewards which is given by the value of the next state. One says that Q-learning is therefore a one-step TD method. However, one could also consider backups based on a weighted combination as follows:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+2} + \cdots + \gamma^{k-1} r_{t+k} + \gamma^k V_{t+k+1}(s_{t+k+1}) \tag{8}$$

In the limit, all real rewards up-until-termination are used, meaning there is no bootstrapping, this is the Monte Carlo method. So, there is a spectrum ranging from using simple one-step returns to using full-backup returns. Some of them were implemented in the set-up of section 5. The Monte Carlo technique, which was described there and which is commonly used in a LA setting, is compared with the following variations: intermediate rewards, one-step updating, n-step updating.

7.1 Intermediate rewards

In (Peeters et al., 2006) an update mechanism based on Intermediate Rewards was introduced. With this technique the learning automata at level l only get informed about the immediate reward and the rewards on the remainder of the path. The LA does not get informed about the rewards that are given to automata on the levels above because the learning automaton at this level has no direct influence over rewards depending on higher level actions and they would clutter up its combined reward. In (Van de Wege, 2006) a theoretical proof that hierarchical learning automata using only the rewards of the remainder of the path will converge to an equilibrium path in an identical pay-off multi-stage game (under the same conditions we described above for the traditional Monte Carlo technique) is given. The complete algorithm can be found in Figure 6. Because the learning automata get updated at the end of an episode, the intermediate rewards technique is still an off-line algorithm.

All the learning automata: initialise action probabilities: $\forall a \in A : p_0(a) = \frac{1}{|A|}$
for each trial **do**
  Activate the top LA of the hierarchies
  for each level l in hierarchy h **do**
    The active learning automata take action $a_t^l(h)$
      $\Rightarrow$ joint-action $\mathbf{a} = [a_t^l(1), \ldots, a_t^l(h), \ldots]$
    Store immediate reward r_t (team reward based on $\mathbf{a}$)
  end for
  for each level l in hierarchy h **do**
    Compute combined reward: $R^l(h) = \theta_t r_t + \theta_{t+1} r_{t+1} + \ldots + \theta_T r_T$
    Update the automaton that was active at level l in hierarchy h with reward $R^l(h)$
  end for
end for

Fig. 6. The pseudo code of the Intermediate Rewards algorithm.

7.2 One-step returns

In the One-Step Estimates technique, introduced in (Peeters et al., 2007), the updating of the learning automata will no longer take place at an explicit end-stage. The automata get informed immediately about the local or immediate reward they receive for their actions. In addition each automaton has estimates about the long term reward for each of its actions. These estimates are updated by combing the immediate rewards with an estimate of possible rewards that this action might give on the remainder of the path, similar to TD(0) methods. The behavior of the algorithm is controlled by three parameters: α, γ and ρ. Here, α is the step size parameter from the LR-I update scheme (Equation 8), γ is the discount factor as used in traditional bootstrapping (Equation 9), and ρ controls the influence of the difference between the combined reward and the old-estimate on the new-estimate (Note: in standard Q-learning notation this parameter is denoted by α).

7.3 n-step returns

The 1-step algorithm described above can easily be extended to the general n-step case. This creates a whole range of updating algorithms for multi-stage games, similar to the range of algorithms that exist for the single agent case. Figures 7 to 9 show the general *n*-step updating algorithm for pursuit learning automata. The parameters α, γ and ρ are equivalent to those of the *1*-step algorithm, described above.

All the learning automata: Initialise action probabilities: $\forall a \in A : p_0(a) = \frac{1}{|A|}$
Initialise the estimates of all the actions: $\forall a \in A : myEstimates(a) = 0$ (estimates for all the actions, meaning: what is the long term reward associated with this action)
Initialise estimates for the learning automata at level n $\forall a \in A : nStepEstimates_0(a) = 0$ (for each action this learning automaton has, keep an estimate of the automata at level $n+1$ of the branch connected to the action)
for each trial **do**
  Activate the top LA of the hierachies
  for each level l in hierarchy h **do**
    Take action $a_t^l(h)$
      $\Rightarrow$ joint-action $\mathbf{a} = [a_t^l(1), \ldots, a_t^l(h), \ldots]$
    Observe immediate reward r_{t+1} (reward based on $\mathbf{a}$) and store it for later use
    Propagate r_{t+1} up to the parent $\Rightarrow$ *see Figure 8: Compute the n-step reward.*
  end for
end for

Fig. 7 Pseudo code of the *n*-step algorithm. This part of the n-step algorithm shows how to handle immediate rewards.

if r_x is the n^{th} reward I receive **then**
  Compute the n-step truncated return: $R_t^{(n)} = r_t + \gamma r_{t+1} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n \; nStepEstimates(a_t^l(h))$
  Update $myEstimates(a_t^l(h)) = myEstimates(a_t^l(h)) + \rho[R_t^{(n)} - myEstimates(a_t^l(h))]$
  Update action probability $\mathbf{p}$ using $myEstimates(a_t^l(h))$ as the reward for the L_{R-I} scheme
  Propagate $myEstimates(a_t^l(h))$ up to parent $\Rightarrow$ *see Figure 9: Propagating estimates.*
else
  if this wasn't the n^{th} reward, this reward also needs to go to the parent: Propagate r_x up to the parent $\Rightarrow$ *Apply this part of the algorithm recursively.*
end if

Fig. 8 Pseudo code of the n-step algorithm. This part of the n-step algorithm computes the complete n-step reward and shows how to update the estimates.

if this estimate comes from the $n+1^{th}$ level **then**
  $nStepEstimates(a_t^l(h)) \leftarrow nStepEstimates(a_t^l(h)) + \rho(\kappa - nStepEstimates(a_t^l(h)))$
else
  Keep propagating est_x up in the hierarchy $\Rightarrow$ *Apply this part of the algorithm recursively.*
end if

Fig. 9 Pseudo code of the n-step algorithm. This part of the n-step algorithm handles the updating of the estimates of the parents in the hierarchy.

The interaction between the hierarchies remains the same as for the Monte Carlo case (and the 1-step case). The learning automata at the top of the hierarchies start by selecting an action. Based on this joint-action the environment generates a reward and this reward is handed to the automata. Since this is the immediate reward, the automata cannot yet

generate the n-step truncated return (if $n > 1$) instead they propagate this reward to their parents (Figure 7 line 10). The automata that receive this reward check whether this is the n^{th} reward they have received (Figure 8 line 1). If so, they compute the n-step truncated return (Figure 8 line 2), update the estimates of the long term reward of their own actions (Figure 8 line 3), update their probabilities (Figure 8 line 4) and keep their n^{th}-level-grandparents up to date by providing them with the updated estimates (Figure 8 line 5). If the parents didn't receive the n^{th} reward yet (thus they can't compute the n-step reward yet), they just propagate the reward to their parents (Figure 8 lines 6 and 7).
In addition to propagating the immediate rewards, the automata also propagate their updated estimates. The parents receiving an estimate from their children check whether it is the estimate they need to compute the n-step truncated return (i.e. the estimate coming from level $(n+1)^{th}$ and they adjust the estimates of their n^{th}-level-grandchildren if necessary. This process continues for each level that gets activated in the hierarchies.

7.4 Empirical results

For the Monte Carlo updating and the Intermediate Rewards method, there are theoretical proofs guaranteeing that the learning automata converge to an equilibrium path in any multi-stage game. This can be proved under the assumptions that the learning automata use the L_{R-I} update scheme and the step sizes are chosen small enough. We have however no guarantee that the automata will converge to the optimal equilibrium path. Therefore it is useful the compare the practical performance of the different update techniques. A thorough comparison can be found in Peeters et al. 2007 (b).
Here we show experiments of a series of *1000* Random Games. For each value of the learning rate we considered in our experiments, we averaged the obtained reward over *1000* randomly generated games. Thus after each of the 1000 runs, we reset the values of the reward matrices to a random number in *[0,1]*. In the experiment we used *2* hierarchies of *8* levels, with *2* actions per automaton. This gives a total of $(2^8)^2 = 65.563$ solution paths. Figure 10 shows the results for the Monte Carlo algorithm and the *4*-step reward.
The average reward when using the Monte Carlo algorithm is systematically lower compared to the average reward of any of the n-step algorithms (the plot shown is for the *4*-step algorithm, but this is observed for the whole tested range of *1*-step to *8*-step, although the performance differs). All of our results demonstrate that the performance increases when the hierarchical learning automata use an *n*-step updating algorithm.

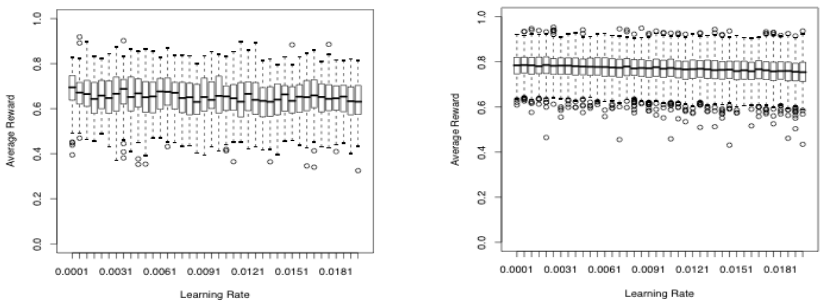


Fig. 10 The average reward using Monte Carlo (left) and the *4*-step rewards for various learning rates. The rewards are averaged over 1000 runs.

In general, using the Intermediate Reward technique, the average reward increases, compared to the Monte Carlo approach, however the variance of the rewards to which the hierarchies converge, remains the same (these results are not shown here). The results of the n-step algorithm show that the average convergence can remain at the same high level (compared to Monte Carlo updating) while the variance of the solution-paths is much less. This means that if the hierarchies converge to a sub-optimal solution, they are more likely to converge to a sub-optimal solution with an average reward that is almost as good as the optimal.

8. Conclusion

In this chapter we have demonstrated that Learning Automata are interesting building blocks for multi-agent Reinforcement learning algorithms. LA can be viewed as policy iterators, that update their action probabilities based on private information only. Even in multi-automaton settings, each LA is updated using only the environment response, and not on the basis of any knowledge regarding the other automata, i.e. nor their strategies, nor their feedback.

As such LA based agent algorithms are relatively simple and the resulting multi-automaton systems can still be treated analytically. Convergence proofs already exist for a variety of settings ranging from a single automaton model acting in a simple stationary random environment to a distributed automata model interacting in a complex environment.

The above properties make LA attractive design tools for multi-agent learning applications, where communication is often expensive and payoffs are inherently stochastic. They allow to design multi-agent learning algorithms with different learning objectives. Furthermore, LA have also proved to be able to work in asynchronous settings, where the actions of the LA are not taken simultaneously and where reward comes with delay.

We have demonstrated this design approach in 2 distinct multi-agent learning settings. In ergodic markov games each agent defers its action selection to a local automaton, associated with the current system state. Convergence to an equilibrium between agent policies can be established by approximating the problem by a limiting normal form game. In episodic multi-stage learning problems agents were designed as tree-structured hierarchies of automata, mimicking the structure of the environment. Convergence of this algorithm can again be established based on existing automata properties. By using Intermediate Rewards instead of Monte Carlo rewards, the hierarchical learning automata are shown (both empirically and theoretically) to have a faster and more accurate convergence by even using less information. However, the Intermediate Rewards update mechanism is still an off-line algorithm in which the updating happens at explicit end-states. The general n-step algorithm solves this problem by handing immediate rewards to the automata which use bootstrapping to compensate for the absence of reward of the remainder of the path. Empirical experiments show that the n-step rewards (with an appropriate value for n) outperform both the Monte Carlo technique as well as the Intermediate Rewards.

9. References

Claus, C; Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems, *Proceedings of the 15th National Conference on Artificial*

Intelligence, pp. 746 – 752, ISBN: 978-1-57735-258-7, Madison, Wisconsin, July 1998, AAAI Press

Boutilier, C; Sequential Optimality and Coordination in Multiagent Systems. *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pp. 478-485, ISBN 1- 55860-613-0, Stockholm, Sweden, August 1999, Morgan Kauffman

Hu, J.; Wellman, M. (2003). Q-learning for General-Sum stochastic Games, *Journal of Machine Learning Research*, 4, (November 2003) pp. 1039-1069, ISSN 1533-7928

Kapetanakis, S.; Kudenko, D., & Strens, M. (2003) Learning to coordinate using commitment sequences in cooperative multi-agent systems, In*: Adaptive Agents and Multi-Agent Systems II,* D. Kudenko et al. (Ed.), pp. 106-118 , Springer LNAI 3394 , ISBN 978-3-540-25260-3, Berlin/Heidelberg

Littman, M.; (2001). Friend-or-foe Q-learning in general-sum games, *Proceedings of the 18th International Conference on Machine Learning,* pp. 322-328, ISBN 1-55860-556-8, July 2001, Williamstown, MA, USA, Morgan Kaufmann Publishers, San Francisco.

Narendra, K; Parthasarathy, K. (1991). Learning Automata Approach to Hierarchical Multiobjective Analysis. *IEEE Transactions on Systems, Man and Cybernetics*, 21, 1, pp 263-272, ISSN 0018-9472.

Narendra, K.; Thathachar, M. (1989). *Learning Automata: An Introduction,* Prentice-Hall International Inc, ISBN 0-13-485558-2 , Upper Saddle River, NJ, USA.

Peeters, M.; Verbeeck, K.; Nowé A. (2006). Bootstrapping versus Monte Carlo in a Learning Automata Hierarchy. In *Proceedings of the Sixth Adaptive Learning Agents and Multi-Agent Systems Symposium*, pp. 61-71, Brussels, April 2006.

Peeters, M.; Verbeeck, K.; Nowé A. (2007) (a). The effect of bootstrapping in multi-automata reinforcement learning. *IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, ISBN 1-4244-0698-6, April 2007, Honolulu, Hawaii, USA.

Peeters, M,; Verbeeck, K.; Nowé A. (2007) (b). Solving Multi-Stage Games with Hierarchical Learning Automata that Bootstrap. To appear LNAI.

Puterman, M; (1994) *Markov Decision Processes: Discrete Stochastic Dynamic Programming* John Wiley & Sons, ISBN 0471727822, Inc. New York, NY, USA.

Ramakrishnan, K.R. (1982. Hierarchical systems and cooperative games of learning automata. Ph.D. thesis, Indian Institute of Science, Bangalore, India.

Shapley, L; (1953) Stochastic games. *Proceeding of the National Academy of Sciences USA,* 39, October 1953, pp. 1095-1100, ISSN 1091-6490

Sutton, R; Barto, A. (1998) *Reinforcement Learning: An Introduction*. MIT Press, ISBN 0-262-19398-1 , Cambridge, MA

Thathachar, M.A.L.; Ramakrishnan, K.R. (1981). A Hierarchical System of Learning Automata. *IEEE Transactions on Systems, Man and Cybernetics*, 11, 3, pp 236-241, ISSN 0018-9472.

Thathachar , M.A.L.; Sastry, P.S. (1985). A new approach to the design of reinforcement schemes for learning automata. *IEEE Transactions on Systems, Man and Cybernetics,* 15, 168–175, ISSN 0018-9472.Tsetlin, M; (1973) *Automaton Theory and Modeling of biological Systems*. Academic Press New York.

Tsitsiklis, J. (1994). Asynchronous Stochastic Approximation and Q-learning. *Machine Learning*, 16, pp 185-202, ISSN 0885-6125.

Tuyls, K.; Nowé, A. (2005). Evolutionary Game Theory and Multi-Agent Reinforcement Learning. *The Knowledge Engineering Review*, 20, 01, March 2005, pp 63–90, ISSN 0269-8889.

Van de Wege, L. (2006). Learning Automata as a Framework for Multi-Agent Reinforcement Learning. Masters thesis.

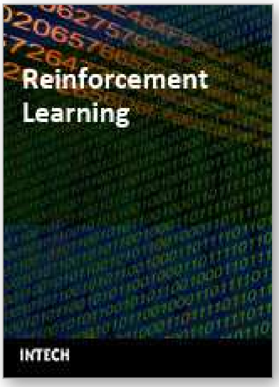
Verbeeck K.; Vrancx P., Nowé A., (2007) (a). Networks of Learning Automata and Limiting Games, *Proceedings of the 7th ALAMAS Symposium*, pp. 171-182 , ISSN 0922-8721, April 2007, Maastricht, The Netherlands, number 07-04, Maastricht University.

Verbeeck, K; Nowé, A., Parent, J., & Tuyls, K. (2007) (b). Exploring Selfish Reinforcement Learning in Repeated Games with Stochastic Rewards. *The Journal of Autonomous Agents and Multi-Agent Systems*, 14, 3, June 2007, pp. 239–269, ISSN 1387-2532 .

Vrancx, P; Verbeeck, K. & Nowé, A. (2007). Decentralized Learning in Markov Games, Tech Report Vrije Universiteit Brussel Mei 2007, Submitted.

Watkins C; Dayan P. (1992). Q-learning. *Machine Learning*, 8, 3, pp 279-292, ISSN 0885-6125.

Wheeler, R; Narendra, K. (1986). Decentralized Learning in Finite Markov Chains. *IEEE Transactions on Automatic Control*, 31, 6, 1986, pp. 519 – 526, ISSN 0018-9286



Reinforcement Learning
Edited by Cornelius Weber, Mark Elshaw and Norbert Michael Mayer

ISBN 978-3-902613-14-1
Hard cover, 424 pages
Publisher I-Tech Education and Publishing
Published online 01, January, 2008
Published in print edition January, 2008

Brains rule the world, and brain-like computation is increasingly used in computers and electronic devices. Brain-like computation is about processing and interpreting data or directly putting forward and performing actions. Learning is a very important aspect. This book is on reinforcement learning which involves performing actions to achieve a goal. The first 11 chapters of this book describe and extend the scope of reinforcement learning. The remaining 11 chapters show that there is already wide usage in numerous fields. Reinforcement learning can tackle control tasks that are too complex for traditional, hand-designed, non-learning controllers. As learning computers can deal with technical complexities, the tasks of human operators remain to specify goals on increasingly higher levels. This book shows that reinforcement learning is a very dynamic area in terms of theory and applications and it shall stimulate and encourage new research in this field.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821