An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-Agent Systems

Martin Lauer LAUER@IRA.UKA.DE
Martin Riedmiller RIEDML@IRA.UKA.DE
Department of Computer Science, University of Karlsruhe, Am Fasanengarten 5, D-76128 Karlsruhe

Abstract

The article focuses on distributed reinforcement learning in cooperative multi-agent-decision-processes, where an ensemble of simultaneously and independently acting agents tries to maximize a discounted sum of rewards. We assume that each agent has no information about its teammates' behaviour. Thus, in contrast to single-agent reinforcement-learning each agent has to consider its teammates' behaviour and to find a cooperative policy. We propose a model-free distributed Q-learning algorithm for cooperative multi-agent-decision-processes. It can be proved to find optimal policies in deterministic environments. No additional expense is needed in comparison to the non-distributed case. Further there is no need for additional communication between the agents.

1. Introduction

Reinforcement learning has originally been discussed for Markov Decision Processes (MDPs): a single agent has to learn a policy that maximizes the discounted sum of rewards in a stochastic environment. Since this framework does not allow multiple agents working in the same environment the MDP model has been extended to multi-agent MDPs (MAMDPs) in which several autonomous agents with different reward-functions make decisions in order to maximize their own discounted reward.

Special cases of this general framework have been examined. Firstly there are so-called zero-sum games with two agents playing against each other. This is realized using reward-functions for both agents which sum to zero. Secondly there are cooperative MAMDPs. They can be identified as environments in which all agents are given the same reward-function. The latter ones will be discussed in this paper.

What is the difference between general MAMDPs and cooperative MAMDPs? In general MAMDPs the existence of different reward-functions makes it impossible to find policies which maximize the summed discounted rewards for all agents. Instead, an equilibrium point is sought. This is a situation in which no agent can improve its reward by changing its own policy if all other agents fix their policies. In contrast, cooperative MAMDPs use the same reward function for all agents. Therefore policies can be found which represent not only equilibrium points but even optimal policies in the sense of maximal discounted reward for all agents.

Why can't a cooperative MAMDP be seen as a single-agent MDP, in which the agent is represented by the whole team? In a single-agent MDP there is only one agent that decides which policy to use. In contrast, in MAMDPs the system's behaviour is influenced by the whole team of simultaneously and independently acting agents. It can be seen as a kind of distributed learning and decision-making. Since the agents are not forced to harmonize their behaviour, it is the task of the learning-algorithm to create cooperative behaviour.

In addition to the problem of 'harmonization' between the agents we are faced with the difficulty of incomplete information with respect to the choice of action. Two cases can be distinguished: a) the case of agents that get information about their own choice of action as well as their partners' choice, and b) the case of agents which only know their own action. The former case was called "joint actions learners" in Claus and Boutilier (1998) while the latter one was named "independent learners". Since a transition in a MAMDP depends on the current state and the *ensemble* of actions, holding back information on the other agents' choice in the case of independent learners results in the succeeding state being unpredictable for an individual agent. In both cases, we assume that every agent knows the current state and the reward gained from a transition.

In this paper we present a distributed algorithm for independent learners which computes an optimal policy in a cooperative MAMDP without additional information or communication between the agents. Two main difficulties have to be solved: a) The agents must learn which policies are optimal and b) if different optimal policies exist the agents must agree on a single policy such that the combination of behaviour of every agent results in an optimal policy of the whole team.

2. Related Work

Recently several algorithms for learning in MAMDPs have been proposed. They are based on Q-learning for single-agent MDPs (Watkins, 1989; Watkins & Dyan, 1992).

Littman (1994) has discussed the case of zero-sum-games and he developped an algorithm for finding equilibrium points for these problems. This approach was extended by Hu and Wellman (1998) to general MAMDPs. Both algorithms need the whole information of joint action learners. They cannot be used for independent learners.

The case of cooperative MAMDPs was examined by Claus and Boutilier (1998). They used ordinary Q-learning for independent learners, trying to compute an equilibrium point by continuously reducing the exploration frequency. But there is no guarantee that an equilibrium point or an optimal policy is learned.

Another approach in distributed learning was made by Wolpert, Wheeler and Tumer (1999). They created individual reward-functions for every agent from the given one. But this procedure complicates the analysis and enlarges the computational expense.

The differences between independent learners and joint action learners are developped in Claus and Boutilier (1998). Some suggestions for solving the problem of harmonization are given in Boutilier (1999).

Beside the theoretical examinations, several successful applications of distributed reinforcement learning have been proposed like routing in a network with the network-nodes as agents (Boyan & Littman, 1993), the control of a group of elevators (Crites & Barto, 1998) and the control of a power network (Schneider et al., 1999). They work with ordinary Q-learning without any adaption to the existence of other agents.

Although there is empirical evidence that these approaches can be used sucessfully there is no theoretical guarantee that an optimal behaviour is obtained.

3. Problem Description

For the analysis of distributed reinforcement-learning we use a model close to Markov Decision Processes. It extends MDPs in the fact that an action is a vector composed of elementary actions, chosen by m independently acting agents. Therefore the set of all possible actions U is the set of all vectors that can be combined by the elementary actions of the m agents: $U = A^m$ where A is the finite set of all elementary actions that can be chosen by an agent.[1]

Like in single-agent MDPs there are a finite set of states S, a transition rule δ and a reward-function $r : S \times U \to \mathbb{R}$. As a simplification we use a deterministic form of MDPs, i. e. the transition rule δ is a deterministic function $\delta : S \times U \to S$.

The aim of learning is to find an optimal policy $\Pi : S \to U$ that maximizes the summed discounted rewards $\sum_{i=0}^{\infty} \beta^i r(s_i, \Pi(s_i))$ with $s_{i+1} = \delta(s_i, \Pi(s_i))$ for all $s_0 \in S$. $\beta \in [0, 1)$ is the discount factor.

Due to the broad analogy between single-agent MDPs and distributed MDPs differing only in the special choice of the action-set, it is well known that for every cooperative MAMDP $\mathcal{M} = (S, A^m, \delta, \beta, r)$ there is an optimal policy Π^*. It can be split into m component-policies $\pi_j : S \to A$ with $\Pi^*(s) = (\pi_1(s), \ldots, \pi_m(s))$.

One way to compute an optimal policy is to treat the distributed MDP as a single-agent MDP with action-set U, computing a policy using Q-learning or Value-iteration and afterwards splitting it up into component-policies.

Q-learning for deterministic MDPs is described by the iteration rule:

$$Q_0(s,u) = 0 \qquad \text{for all } s \in S,\ u \in U$$

$$Q_{t+1}(s,u) = \begin{cases} Q_t(s,u) & \text{if } s \neq s_t \text{ or } u \neq u_t \\ r(s,u) + \beta \cdot \max\limits_{u' \in U} Q_t(\delta(s,u), u') & \\ & \text{if } s = s_t \text{ and } u = u_t \end{cases} \tag{1}$$

The Q_t-values converge towards the optimal Q^*-function if every pair $(s, u) \in S \times U$ occurs infinitely often in the sequence of current states s_t and actions u_t. The Q^*-values allow to derive an optimal policy.

In the case of a reward-function with

$$r(s,u) \geq 0 \qquad \text{for all } s \in S,\ u \in U \tag{2}$$

[1] Throughout the paper we denote elementary actions with the small letter $a \in A$ and action-vectors with the letter u: $u = (a^{(1)}, \ldots, a^{(m)}) \in A^m = U$.

it can be proved by induction on t that the Q_t-values in algorithm (1) grow monotonically with increasing index t. Furthermore, assumption (2) is no real restriction to the MDPs since optimal policies stay optimal even if a constant is added to the reward-function.

However, this algorithm needs a *central* control, to be applicable. In contrast we are interested in *distributed* learning of componentwise policies which constitute an optimal behaviour. Each agent should act like an independent learner only using the information about the current state, its own choice of action and the reward gained by a transition.

4. Distributed Q-learning: Basic Idea

Independent learners have no possibility to distinguish between several action-vectors with the same own elementary action. Therefore they aren't able to compute Q-tables of the form $Q : S \times A^m \to \mathbb{R}$ depending on the current state and the ensemble of the agents' elementary actions $(s, a^{(1)}, \ldots, a^{(m)})$. Only smaller tables of the size $S \times A$, depending only on the current state and one agent's elementary action $(s, a^{(j)})$, are appropriate.[2] Thus, a projection is needed which compresses the information of the large Q-table to the smaller ones. This projection can also be interpreted as a kind of 'assumption': every agent assumes its teammates' behaviour and reduces the information of the central Q-table by selecting only these items which represent the assumed behaviour.

Several of such projections are possible. By using Q-learning in a distributed manner without any adaption to the multi-agent-model individual $q^{(j)}$-tables are computed by creating a weighted sum on the items of the larger table. They are the unique solution of the system of equations:

$$q^{(j)}(s,a) = \\ = \sum_{\substack{u=(a^{(1)},\ldots,a^{(m)}) \\ a=a^{(j)}}} \Big(\Pr(u|a^{(j)}) \cdot \big(r(s,u) + \beta \cdot \max_{a' \in A} q^{(j)}(s',a') \big) \Big) \tag{3}$$

where $\Pr(u|a^{(j)})$ denotes the probability of action-vector u appearing in state s when elementary action $a^{(j)}$ is chosen by Agent j.

[2] We denote the central, large Q-table with the capital letter Q and the small, distributed tables with the small letter $q^{(j)}$. Note that for every agent an individual table has to be computed, indicated by the additional index j.

Such a projection leads to results that depend on parameters of the learning algorithm itself (i.e. the exploration strategy). There are examples for which this algorithm doesn't create optimal or near-optimal policies (see the results of Claus, and Boutilier (1998) in example 1).

Another kind of projection is the so-called 'pessimistic assumption'. It takes into account that other agents may act disadvantagous. Therefore the individual $q^{(j)}$-tables are computed from the large one by selecting the worst item. Although the idea of the pessimistic assumption seems to create robust policies, it turned out that this kind of projection results in a too cautious behaviour which is not very useful. Therefore it is omitted here.

The dual form of the pessimistic assumption is to set the item $q^{(j)}(s,a)$ to the value of $\max_{\substack{u=(a^{(1)},\ldots,a^{(m)}) \\ a^{(j)}=a}} Q(s,u)$, i.e. to select the best item in the central Q-table.

From the fact that the greedy optimal policy can be computed from the central Q-table by selecting an action-vectors $\hat{u} = (\hat{a}^{(1)}, \ldots, \hat{a}^{(m)})$ with:

$$Q(s,\hat{u}) = \max_{u \in U} Q(s,u) \tag{4}$$

concludes that also equation (5) holds:

$$\max_{\substack{a^{(j)}=\hat{a}^{(j)} \\ u=(a^{(1)},\ldots,a^{(m)})}} Q(s,u) = \max_{a \in A} \max_{\substack{a^{(j)}=a \\ u=(a^{(1)},\ldots,a^{(m)})}} Q(s,u) \tag{5}$$

Written in terms of the $q^{(j)}$-tables, it is $q^{(j)}(s,\hat{a}^{(j)}) = \max_{a \in A} q^{(j)}(s,a)$. But equation (5) describes only a necessary, not a sufficient condition for the optimal choice of elementary actions. It can happen that each agent selects an elementary action from equation (5) but the combination of all agents' elementary actions isn't optimal. This is the difficulty of coordination between the agents. It is described in Section 6.

Since every agent picks out from the large table only these items which are maximal, this projection can also be interpreted as an 'optimistic assumption': every agent assumes that the other agents will act optimally, i.e. the chosen elementary-actions can be combined to an optimal action-vector. As described in the previous paragraph it can happen that this assumption fails. Nevertheless it is possible to compute optimal policies using this kind of distributed Q-values and additional coordination-techniques, as explained in the following paragraphs.

5. Learning Distributed Q-values

To begin with we show how the small, agent-individual $q^{(j)}$-tables can be computed using the optimistic assumtion. For that purpose an algorithm is presented which simulates common Q-learning as described by equation (1).

Given a sequence of states s_t and actions $u_t = (a_t^{(1)}, \ldots, a_t^{(m)})$, compute for every agent j and iteration-index t:

$$q_0^{(j)}(s,a) = 0$$

$$q_{t+1}^{(j)}(s,a) = \begin{cases} q_t^{(j)}(s,a) & \text{if } s \neq s_t \text{ or } a \neq a_t^{(j)} \\ \max\{q_t^{(j)}(s,a),\ r(s_t,u_t) + \\ \qquad \beta \cdot \max_{a' \in A} q_t^{(j)}(\delta(s_t,u_t),a')\} & \\ \text{otherwise} & \end{cases} \tag{6}$$

The central idea of the iteration-rule is that the returns of appropriate transitions are collected in a $q^{(j)}$-value by successively taking the maximum.

Proposition 1
Let $\mathcal{M} = (S, A^m, \delta, \beta, r)$ be a deterministic cooperative MAMDP with m agents for which assumption (2) holds.
Then for every index t, agent j, state s and elementary action a the following equation holds:

$$q_t^{(j)}(s,a) = \max_{\substack{u=(a^{(1)},\ldots,a^{(m)}) \\ a^{(j)}=a}} Q_t(s,u) \tag{7}$$

whereby Q_t-values are computed by algorithm (1) and the $q_t^{(j)}$-tables are created like (6) with the same sequences of current states s_t and actions u_t.

Proof (by induction on t):
$t = 0$: the induction holds for the initial values

$t \rightsquigarrow t+1$: Fix an agent j and let be $u_t = (a_t^{(1)}, \ldots, a_t^{(m)})$ and $u = (a^{(1)}, \ldots, a^{(m)})$.

1. For all $(s,a) \neq (s_t, a_t^{(j)})$, the induction step is trivially true, since neither in (1) nor in (6) an update is performed.
2. $(s,a) = (s_t, a_t^{(j)})$
Applying the learning rule (6) and replacing the individual $q_t^{(j)}$-values by the central Q_t-values as assumed valid by the induction hypothesis leads to:

$$\begin{aligned} & q_{t+1}^{(j)}(s_t, a_t^{(j)}) = \\ & = \max\{q_t^{(j)}(s_t, a_t^{(j)}), \\ & \qquad r(s_t,u_t) + \beta \cdot \max_{a' \in A} q_t^{(j)}(\delta(s_t,u_t),a')\} \\ & = \max\{ \max_{\substack{u=(a^{(1)},\ldots,a^{(m)}) \\ a^{(j)}=a_t^{(j)}}} Q_t(s_t,u), \\ & \qquad r(s_t,u_t) + \beta \cdot \max_{u' \in U} Q_t(\delta(s_t,u_t),u')\} \end{aligned}$$

Applying update-rule (1) to the second element of the set and rewriting the set yields:

$$\begin{aligned} & q_{t+1}^{(j)}(s_t, a_t^{(j)}) = \\ & = \max\left(\{Q_t(s_t,u) \big| a_t^{(j)} = a^{(j)}\} \cup \{Q_{t+1}(s_t,u_t)\}\right) \\ & = \max\Big(\{Q_t(s_t,u) \big| a_t^{(j)} = a^{(j)},\ u \neq u_t\} \cup \\ & \qquad \{Q_t(s_t,u_t)\} \cup \{Q_{t+1}(s_t,u_t)\}\Big) \end{aligned}$$

Observing that no update is performed by iteration-rule (1) for the first part of the set, and that the monotonicity of the Q_t-values allows to reject the second part, we get:

$$\begin{aligned} & q_{t+1}^{(j)}(s_t, a_t^{(j)}) = \\ & = \max\Big(\{Q_{t+1}(s_t,u) \big| a_t^{(j)} = a^{(j)},\ u \neq u_t\} \cup \\ & \qquad \{Q_{t+1}(s_t,u_t)\}\Big) \\ & = \max_{\substack{u=(a^{(1)},\ldots,a^{(m)}) \\ a^{(j)}=a_t^{(j)}}} Q_{t+1}(s_t,u) \end{aligned}$$

Algorithm (6) makes it possible to learn distributed Q-tables, which are independent of the training data set and for which equation (5) holds. The only condition for valid results is, that all state-action pairs occur infinitely often in the sequence of current states and actions. This necessity is already known from the single-agent-case.

Example 1
For a more intuitive understanding, we give a simple example which was taken from Claus and Boutilier (1998). It consists of one state s_0, two agents and three elementary actions ($A = \{a', a'', a'''\}$). The reward-function $r(s_0, (a^{(1)}, a^{(2)}))$ is given by the following table:

		$a^{(1)} =$		
		a'	a''	a'''
$a^{(2)} =$	a'	11	−30	0
	a''	−30	7	6
	a'''	0	0	5

For simplification we set the discount factor $\beta = 0$. Therefore the above table shows the reward-function as well as the optimal Q^*-values computed by the central learning algorithm (1). Obviously, the best choice

of action would be for both agents to take a' which yields the highest possible reward.

The distributed algorithm given in (6) computes the following distributed $q^{(j)}$-tables:

	$a =$		
	a'	a''	a'''
$q^{(1)}(s_0, a) =$	11	7	6
$q^{(2)}(s_0, a) =$	11	7	5

A greedy evaluation of both $q^{(j)}$-tables chosing the elementary actions with the highest value also leads to the optimal action (a', a') with expected reward 11.

In contrast, the approach of Claus and Boutilier is supposed to find a sub-optimal policy which selects the action (a'', a'') with reward 7.

6. Coordination between Agents

The existence of proper Q-tables, which can be computed with algorithm (6), is the first essential of the computation of optimal policies. Greedy policies can be created by chosing the best action in every state: $\pi^{(j)}(s) = \arg\max_{a \in A} q^{(j)}(s, a)$. But in contrast to single-agent MDPs there is no guarantee that these policies are optimal although the $q^{(j)}$-tables are correct in the sense of equation (7). An example shows a typical case:

Example 2
This one-state two-agent example is also taken from Claus and Boutilier (1998). Three elementary actions a', a'', a''' can be chosen by each agent. The discount factor is set to 0, the reward-function is described in the following table (k is assumed to be a parameter less than 10):

		$a^{(1)} =$		
		a'	a''	a'''
$a^{(2)} =$	a'	10	0	k
	a''	0	2	0
	a'''	k	0	10

Algorithm (6) computes the distributed $q^{(j)}$-tables:

	$a =$		
	a'	a''	a'''
$q^{(1)}(s_0, a) =$	10	2	10
$q^{(2)}(s_0, a) =$	10	2	10

Four greedy policies can be generated from the distributed $q^{(j)}$-functions, yielding the actions (a', a'), (a''', a'''), (a', a''') and (a''', a'), respectively. But only the first two are optimal with expected reward 10, while the other ones yield a lower gain of k.

This states the necessity of an additional mechanism of coordination between the agents.

During the process of learning it is necessary that every state-action-pair occurs infinitely often. Therefore the optimal actions appear as well and (due to (7)) they can be detected by every individual agent. By means of storing the first occuring optimal action in a current policy, an optimal policy is computed in parallel with learning the $q^{(j)}$-tables. The update-rule for the current agent-individual policies $\pi_t^{(j)}$ is given in (8). It is based on the computation of the $q^{(j)}$-values like in algorithm (6).

$\pi_0^{(j)}(s) \in A$ arbitrarily

$$\pi_{t+1}^{(j)}(s) = \begin{cases} \pi_t^{(j)}(s) & \text{if } s \neq s_t \text{ or} \\ & \max_{a \in A} q_t^{(j)}(s, a) = \max_{a \in A} q_{t+1}^{(j)}(s, a) \\ a_t^{(j)} & \text{otherwise} \end{cases} \tag{8}$$

with $u_t = (a_t^{(1)}, \ldots, a_t^{(m)})$.
The central idea of the learning algorithm is to update the current policy only if an improvement in the $q^{(j)}$-values happens.

Proposition 2
Let $\mathcal{M} = (S, A^m, \delta, \beta, r)$ be a deterministic cooperative MAMDP with m agents for which assumption (2) holds.
Then for every index t and state s it is:

$$Q_t\Big(s, \big(\pi_t^{(1)}(s), \ldots, \pi_t^{(m)}(s)\big)\Big) = \max_{u \in U} Q_t(s, u) \tag{9}$$

This means that the combination of the current policies $(\pi_t^{(1)}(s), \ldots, \pi_t^{(m)}(s))$ computed by algorithm (8) is greedy with respect to the non-distributed Q-table created by algorithm (1) with the same sequence of current states s_t and actions u_t.

Proof: In the following, we consider an arbitrary iteration step $t \in \{0, 1, 2, 3, \ldots\}$ and an arbitrary state $s \in S$:
1. $\max_{u \in U} Q_t(s, u) = 0$:
then every policy is greedy in s
2. $\max_{u \in U} Q_t(s, u) > 0$:
due to the initialization of Q_0-table with 0 and the monotonicity property of the Q_t-values there exists a greatest index $0 \leq t' < t$ with:

$$\max_{u\in U} Q_{t'}(s,u) < \max_{u\in U} Q_{t'+1}(s,u) \tag{10}$$

and

$$\max_{u\in U} Q_{t'+1}(s,u) = \cdots = \max_{u\in U} Q_t(s,u) \tag{11}$$

Using (10) and (11) and the equality between the Q and the q-values (proposition 1) we can conclude for the q-values, that

$$\max_{a\in A} q_{t'}^{(j)}(s,a) < \max_{a\in A} q_{t'+1}^{(j)}(s,a) = \max_{a\in A} q_t^{(j)}(s,a)$$

Considering the policy-update rule (equation 8) that means for the policy that

$$a_{t'}^{(j)} = \pi_{t'+1}^{(j)}(s) = \pi_t^{(j)}(s)$$

which means that the composite action $u_{t'}$ can be written as

$$u_{t'} = (\pi_t^{(1)}(s), \ldots, \pi_t^{(m)}(s)) \tag{12}$$

On the other hand, using equation (10) and since $Q_{t'+1}$ only changed at $(s, u_{t'})$, we can conlude that

$$Q_{t'+1}(s,u_{t'}) = \max_{u\in U} Q_{t'+1}(s,u)$$

and further, using (11) again with the monotonicity argument

$$Q_{t'+1}(s,u_{t'}) = Q_t(s,u_{t'})$$

Using expression (12) for $u_{t'}$ we finally conlcude that

$$Q_t(s,(\pi_t^{(1)}(s),\ldots,\pi_t^{(m)}(s))) = \max_{u\in U} Q_t(s,u)$$

The convergence of the current policies Π_t into an optimal policy follows from proposition 2 and the convergence of algorithm (1). Thus, optimal policies can be learned in distributed, deterministic environments using the described algorithm. It even doesn't require more computational expense than a central control would need for the same task.

Reconsidering example 2, the additional policy-update-rule selects exactly one optimal policy from the set of greedy policies, i.e. either (a', a') or (a''', a''') is chosen depending on the order of occurance in the sequence of training patterns. In contrast, the approach of Claus and Boutilier (1998) finds an optimal behaviour only if the parameter k is close to 10.

Some elements of the coordination-technique were originally suggested by Boutilier (1999). He called it "Randomization with Learning". It was used in context with joint action learners which can observe the complete action-vector and therefore have the ability to distinguish between all actions. In contrast, independent learners only know their own elementary action. Thus, Boutilier's version of "Randomization with Learning" is not applicable. The new variant instead uses the current q-tables, that are computed in parallel and thereby allow to detect optimal actions.

7. Distributed Learning in Stochastic Environment

The algorithms (6) and (8) show a principle procedurale scheme for distributed learning in cooperative MAMDPs: computing distributed Q-tables using the optimistic assumption from Section 4 as a projection of the large central Q-table onto the distributed small functions and creating an optimal policy with the help of an additional coordination-technique.

This works excellently for deterministic environments. But can these results be transfered to stochastic MDPs?

The difference between deterministic and stochastic MDPs is the change to a stochastic transition-relation. Instead of the successor-function $\delta : S \times U \to S$ a function $T : S \times U \times S \to [0,1]$ is used with $\sum_{s'\in S} T(s,u,s') = 1$ for all $s \in S,\ u \in U$. $T(s,u,s')$ indicates the probability of a successor-state s' when action u is chosen in state s. All transitions are thought to run independently.

This difference in transition rule has also effects on the computation of the optimal Q-values. In deterministic environment they are the unique solution of the Bellman-equation:

$$Q^*(s,u) = r(s,u) + \beta \cdot \max_{u'\in U} Q^*(\delta(s,u),u') \tag{13}$$

while for stochastic MDPs the Bellman-equation has the form:[3]

$$\begin{aligned} Q^*(s,u) &= r(s,u) + \beta \cdot \mathop{\mathsf{E}}_{\text{successor state } s'} \left[\max_{u'\in U} Q^*(s',u')\right] \\ &= r(s,u) + \beta \cdot \sum_{s'\in S} \left(T(s,u,s') \cdot \max_{u'\in U} Q^*(s',u')\right) \end{aligned} \tag{14}$$

It is obvious that equation (13) is a special case of (14) for a unique successor state. A suitable extension of update-rule (6) therefore would be:

[3] $\mathsf{E}[X]$ is the expectation value of X.

$$q_{t+1}^{(j)}(s,a) = \begin{cases} q_t^{(j)}(s,a) & \text{if } s \neq s_t \text{ or } a \neq a_t^{(j)} \\ \max\left\{ q_t^{(j)}(s,a),\ r(s_t,u_t) + \beta \cdot \sum_{s' \in S} \left(T(s_t,u_t,s') \cdot \max_{a' \in A} q_t^{(j)}(s',a') \right) \right\} & \\ \text{otherwise} & \end{cases} \tag{15}$$

But in the above equation (15) the value of the sum $\sum_{s' \in S} \left(T(s_t,u_t,s') \cdot \max_{a' \in A} q_t^{(j)}(s',a') \right)$ is unknown at instance t. It has to be estimated during the learning process. By collecting rewards of several transitions from the same state with the same action this value could be estimated using a Robbins-Monro-approximation. On the other hand every independently learning agent only knows a part of the current action-vector and therefore cannot distinguish different action-vectors which doesn't differ in the visible part. Such an agent would mix the estimated rewards of different actions and therefore could not find the proper $q^{(j)}$-value.

Thus, the difficulty with distributed reinforcement-learning in stochastic MDPs can be explained as follows: there are two kinds of influence which determine the successor state: a) the behaviour of other agents and b) the random influence of the stochastic environment. Both influences must be considered in different ways: the behaviour of the agents has to be maximized while the random influences have to be taken into account with their expectation values. The need of considering two different kind of noise is not only a difference to distributed Q-learning in deterministic environment but also to non-distributed learning in stochastic domain.

As we have not yet found a feasible way to distinguish between both kinds of influence in the given framework, the possibility of distributed reinforcement-learning in stochastic environment remains an open question.

8. Discussion

In this paper we investigated the possibility of developping an algorithm for distributed reinforcement learning on the basis of Q-learning.

In deterministic environments there are two difficulties: a) the question of suitable Q-tables and a projection which gives the relation between the central, large Q-table and the smaller, distributed ones and b) the coordination-problem.

For (a) we introduced a projection that can be interpreted as a kind of 'optimistic assumption': every agent works as if its teammates complete its behaviour to an optimal way of acting. It was shown that this assumption is a suitable technique for distributed learning. It reduces the whole information of the central Q-table to the necessary parts which are available for every agent. A model-free iteration-rule was given that was proved to compute the distributed 'optimistic' Q-tables without additional communication or harmonization between the agents. Especially in the case of deterministic MDPs with a unique optimal action in every state the policies which are computed greedily from the distributed Q-tables could be shown to be optimal.

In the case of several optimal actions in a single state the optimal behaviour of the whole team cannot be guaranteed; the optimistic assumption can be violated. Therefore an additional procedure for cooperation inbetween the team was developped. An explicit current policy is updated which can be proved to be greedy with respect to the current central Q-table. Therefore these policies converge towards an optimal behaviour.

The computational expense of distributed learning is as large as of central learning, as the simulation proofs have shown. The necessary storage volume is $|S| \cdot |A|^m$ for central learning and $m \cdot |S| \cdot |A| + m \cdot |S|$ items in distributed learning (size of Q-tables plus size for current policies).

In stochastic environment a suitable learning algorithm has not yet been found. The fundamental difficulty is to distinguish the unknown random influence from the effects of other agent's behaviour. As there is a difference in considering both influences – maximizing the other agent's behaviour and taking the expectation value of the random effects – the approach for deterministic environments cannot be transfered.

Nevertheless, this paper shows that distributed reinforcement learning cannot be based only on the computation of value-functions. Additional coordination-techniques are necessary. On the other hand, the evaluation of a given policy needs the computation of a kind of quality-function. Thus, we expect the combination of policy-update and value-function-update to be a promising approach for distributed reinforcement learning in cooperative context.

References

Boutilier, C. (1999). Sequential optimality and coordination in multiagent systems. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (pp. 478–485). San Francisco: Morgan Kaufmann.

Boyan, J. A., & Littman, M. L. (1993). Packet routing in dynamically changing networks: A reinforcement learning approach. *Advances in Neural Information Processing Systems, 6*, (pp. 671–678). San Francisco: Morgan Kaufmann.

Claus, C., & Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. *Proceedings of the Fifteenth National Conference on Artificial Intelligence* (pp. 235–262). Cambridge, MA: MIT Press

Crites, R. H., & Barto, A. G. (1998). Elevator group control using multiple reinforcement learning agents. *Machine Learning, 33*, 235-262.

Hu, J., & Wellman, M. P. (1998). Multiagent reinforcement learning: Theoretical framework and an algorithm. *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 242–250). San Francisco: Morgan Kaufmann.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. *Proceedings of the Eleventh International Conference on Machine Learning* (pp. 157–163). San Francisco: Morgan Kaufmann.

Schneider, J. & Wong, W. K., & Moore, A. W., & Riedmiller, M. (1999). Distributed value functions. *Proceedings of the Sixteenth International Conference on Machine Learning* (pp. 371-378). San Francisco: Morgan Kaufmann.

Watkins, C. (1989). *Learning from delayed rewards.* Ph.D. thesis, Department of Computer Science, Cambridge University, Cambridge, UK.

Watkins, C., & Dyan, P. (1992). Q-learning. *Machine Learning, 8*, 279–292.

Wolpert, D. H., Wheeler, K. R., & Tumer, K. (1999). General principles of learning-based multi-agent-systems. *Proceedings of the Third Annual Conference on Autonomous Agents* (pp. 77–83). New York: ACM