Reinforcement Learning for Stochastic Cooperative Multi-Agent-Systems

Martin Lauer
Department of Computer Science
University of Osnabrück
48069 Osnabrück, Germany
martin.lauer@uos.de

Martin Riedmiller
Department of Computer Science
University of Osnabrück
48069 Osnabrück, Germany
martin.riedmiller@uos.de

Abstract

We present a distributed variant of Q-learning that allows to learn the optimal cost-to-go function in stochastic cooperative multi-agent domains without communication between the agents. We motivate this approach from a theoretical standpoint by showing its relation to standard Q-learning. Also, a practical variant of the algorithm is proposed - called the 'reduced-lists algorithm' - that deals with the problem of the combinatorial explosion of the number of joint actions. The principle behaviour of the algorithm is shown on a benchmark problem proposed by Boutilier and Claus.

1. Introduction

Multi-agent reinforcement learning is faced with the problem of several independent agents acting in a shared environment with the objective to learn a strategy to act optimally with respect to a given reward function [1, 2, 6]. Here we focus on cooperative learning [3] where all agents use the same reward function. Besides general problems of RL (large state spaces, ...), cooperative reinforcement learning poses two further challenges: a) how to coordinate action selection and b) how to accumulate *distributed* past experiences in order to learn joint optimal behavior. We discuss the case of model-free learning where the agents have to learn only from the information they get from state-to-state transitions in experimental runs. Claus and Boutilier [3] distinguish between two cases: a) every agent can observe its own as well as the other agents' actions, called 'Joint action learners' (JAL) and b) 'Independent learners' (IL) that are not able to observe the actions taken by their teammates. In the JAL case standard learning algorithms like Q-learning [4] are applicable and converge to the optimal cost-to-go function. The IL case is much more difficult to solve. In [5], we presented a cooperative learning algorithm for the IL case that converges for *deterministic* environments. This paper now focuses on IL learners for the more general case of *stochastic* environments.

A key problem in stochastic domains is to distinguish between the random influence caused by the environment and the structural influence of other agents. [7] uses an approach that factorizes the environment using independencies between agents and that clamps the strategy of several agents for a certain time interval. Although factorization is desirable to deal with large tasks it is not always possible in general environments.

We present an algorithm for general stochastic environments for the model-free IL case. No explicit communication between the agents is assumed. In its 'full-lists'- variant, convergence to the optimal joint policy can be shown. In a more efficient 'reduced-lists'-variant, theoretical convergence currently is only known for deterministic environments. As a proof of concept, the algorithm is applied to Claus and Boutiliers 'climbing' game [3].

2. A distributed learning algorithm

2.1. Principle settings

The framework we consider here is a standard Markov Decision Process (MDP). An MDP is a 4-tuple, (S, A, T, r), where S denotes a finite set of states, A denotes the action space, T is a probabilistic transition function $T : S \times A \times S \to [0, 1]$, that denotes the probability for a transition from state s to state s' when a certain action a is applied. Finally, $r : S \times A \to \mathbb{R}$ is a reward function, that denotes

the immediate reward for applying a certain action a in a certain state s.

In the multi-agent case considered here, actions are *vectors* of individual decisions. Each agent i contributes its own action component to the joint action, that is eventually applied to the environment and determines the transition. All agents share the same reward function r. The goal is to find an optimal policy π^* that maximizes the sum of discounted rewards [8].

2.2. A learning algorithm for independent learners (IL)

Independent learners suffer from the lack of information about the actions taken by their teammates. They only know about their own contribution a_i to the joint action. A value estimation based on that information alone would therefore mix the rewards of several different joint action vectors and thus become meaningless. The problem is therefore, that the independent learners must somehow be able to distinguish between the different joint actions. The key idea of the following algorithm is, that the agent can distinguish between different joint actions without knowing, what exactly the joint actions are. This will be realized by the idea of 'implicit coordination'. We will develop our algorithm in four basic steps:

1. Idea of implicit coordination A naive approach to realize implicit coordination would be to determine the sequence of actions to be played in an initialization phase ahead of learning. However, such a scheme is rather inflexible, e.g. in course of learning some actions turn out to be more attractive than others. A more flexible scheme can be built using the following idea: Each agent i maintains a list for every state s, $L_i(s)$, where each entry corresponds to exactly one joint action. A list entry contains a tuple of the form $L_i(s)[l] = \langle Q, a_i, n\rangle$ where a_i is component i of the (implicitly) referenced joint action vector, Q the respective Q-value and n counts the number of occurrences of the pair $(s, \underline{a})$, where $\underline{a} = (a_1, a_2, \ldots a_m)$ is the joint action of the m agents. If we can assure, that at every single point in time, all agents select the same index l to access their individual lists and select the action $L_i(s)[l].a_i$, then the reward can be uniquely assigned. The reward is caused by the joint action implicitly referenced by the list index. Accordingly, the Q-value of that entry, $L_i(s)[l].Q$, can be correctly updated.

2. The Select_Index procedure The Select_Index procedure is realized in every agent and has to assure, that at every point in time all agents get the same index. As a necessary condition this therefore assumes, that all agents are aware of the same time, e.g. by reliably counting the number of decision cycles. One possibility to implement such an index selection, is to take a random number generator and to initialize the seed with the same number for all agents ahead of learning. Additionally, the select index procedure has to fulfill some fairness properties (see section 3).

3. Efficient exploration To enforce efficient exploration, the lists are sorted according to decreasing Q-values. This means, that the most promising actions are at the beginning of the list. The Select_Index procedure can then be constructed to prefer entries at the beginning of the list with a higher frequency. As long as the chance for any action to be selected is larger than zero, this does not hinder theoretical convergence.

4. Reduced lists When the lists contain an entry for every possible joint action, then the approach is called the 'full lists'-variant of the algorithm. However, dealing with full lists will be rather impractical. A more feasible way is to only consider lists with a reduced number of joint actions ('reduced lists'-variant). The size of a list is denoted by l_{max}. Certainly, ignoring some joint actions yields the danger of missing the optimal solution. Therefore, after some training cycles (the 'training interval'), the list is partially reset. Only the l_{keep} best entries are preserved; the rest of the list is reinitialized and new candidates get a chance. This means, every list entry from $l_{keep} + 1$ to l_{max} is assigned a new individual action; the Q and n entries are reset to zero. To enforce a stabilization of the learning process, the length of the training intervals is increased in course of learning.

2.3. Description of the algorithm

Algorithm 1 gives a description of the reduced-lists approach in terms of list-based data-structures. Every agent i keeps for every state s a list, where every entry in the list, $L^i(s)[l]$ stores three values: the individual action $L^i(s)[l].a \in Actions(i)$, the number $L^i(s)[l].n \in \mathbb{N}$ of times that the index l has been selected up to now, and the current Q-value $L^i(s)[l].Q \in \mathbb{R}$. Each list contains at most l_{max} entries.

Procedure **Main()** controls the learning process. After initializing all lists, learning is started. The procedure **SelectState()** selects the visited state. E. g. **SelectState()** might be implemented, such that learning on trajectories is realized, but the formulation also allows to present states in an arbitrary fashion. The procedure **LearnTransition(state s)** is realized in every agent. It selects an individual action (corresponding to a selected index l_i) which is applied to the en-

vironment. The observed successor state is then used to update the Q-value of state s at index l_i. Learning of transitions is repeated until a certain number (num_traincycles) of cycles is done. Then, each agent resets parts of his list (**Reset()**): First, the lists are sorted by decreasing Q-values, then all but l_{keep} entries are deleted. The freed list entries are filled again by new candidate actions. Finally, num_traincycles is optionally modified, for example to implement a strategy of training intervals with increasing length. The whole process is repeated, until the optimal policy is found or approximated to a sufficient level.

3. Relation to standard Q-learning

The proof of the theoretical soundness of the 'full lists'-approach for ILs given in algorithm 2 is based on exploiting its relationship to standard Q-learning. Standard Q-learning [4] is based on a cost-to-go function of a Markov Decision Process (MDP) with finite state and action sets S and A, stochastic transition kernel T, reward function r and discount factor γ. $Q(s,a)$ describes the discounted expected reward for taking action a in state s. After a transition from state s to state s' with action a and observed reward r the expected reward is updated by:

$$Q(s,a) \leftarrow (1-\alpha_{n(s,a)})\cdot Q(s,a)+\alpha_{n(s,a)}\cdot(r+\gamma \max_{a'} Q(s',a')) \quad (1)$$

where $n(s,a)$ counts the number of occurrences of action a in state s. [8] state that the Q-table updated in the above manner converges for every MDP to the true expected rewards whenever conditions (i) and (ii) hold:

(i) the sequence $(\alpha_i)_{i=1}^{\infty}$ fulfills: $\alpha_i \in [0,1]$, $\sum_{i=1}^{\infty} \alpha_i = \infty$ and $\sum_{i=1}^{\infty} \alpha_i^2 < \infty$.

(ii) all pairs of states and actions (s,a) occur infinitely often .

In the distributed cooperative learning model for MDPs with a fixed number of agents n the transition and reward functions depend not only on a single action but on an n-dimensional action vector $\underline{a} \in A^n$. Each agent selects one component of the action-vector. Thus in the JAL case, where every agent has the information about the actions taken by its teammates, distributed MDPs are only a small variant of standard MDPs with a special action set. In the IL case however, things get more complicated because the knowledge of other agents actions is not available and thus the entries in the Q-tables are no longer unique. This is discussed in the following.

Initialize()
FOR all agents $i = 1 \dots m$,
all states $s = 1 \dots N$,
all indeces $l = 1 \dots l_{max}$ {
  $L^i(s)[l].Q := 0.0$;
  $L^i(s)[l].n := 0$;
  $L^i(s)[l].a :=$*RandomAction(Actions$_i$(s))*;
}

Reset()
FOR all agents $i = 1 \dots m$, all states $s = 1 \dots N$ {
  SortListByDecreasingQ($L^i(s)$)
  FOR all indices $l = \mathbf{l_{keep}} + \mathbf{1} \dots l_{max}$ {
    $L^i(s)[l].Q := 0.0$;
    $L^i(s)[l].n := 0$;
    $L^i(s)[l].a :=$ *RandomAction*($Actions_i(s)$);
  }
}

LearnTransition(state s)
FOR all agents $i = 1 \dots m$ {
  SELECTINDEX: $l_i =$ *SelectIndex*$_i(t)$
  OUTPUT $a_i := L^i(s_t)[l_i].a$
}
APPLY $\underline{a} = (a_1, \dots, a_m)$
OBSERVE s', r
FOR all agents $i = 1 \dots m$ {
  $\alpha_i =$ *GetLearnrate*($L^i(s)[l_i].n$)
  $L^i(s)[l_i].Q \leftarrow (1-\alpha_i)\, L^i(s)[l_i].Q$
    $+\alpha_i(r + \gamma \max_j L^i(s')[j].Q)$
  $L^i(s)[l_i].n \leftarrow L^i(s)[l_i].n + 1$
}

Main()
Initialize()
REPEAT {
  FOR $k = 1 \dots$ *num_traincycles*
    LearnTransition(*SelectState()*)
  Reset()
  num_traincycles = ModifyTraininginterval()
} UNTIL TASKISLEARNED

Algorithm 1: An algorithm for distributed Q-learning in stochastic domains as described in section 2.3.

3.1. List representation and index addressing

So far $Q(s,\underline{a})$ has been used to describe a mathematical function, not a data structure. Beneath the most convenient way to represent the Q-function in table form it is as well possible to use a list representation with $L(s)$ indicating a list of triples

$L(s)[i] = \langle Q, \underline{a}, n \rangle$ with $L(s)[i].Q := Q(s, L(s)[i].\underline{a})$ and $L(s)[i].n := n(s, L(s)[i].\underline{a})$. As long as the lists contain exactly one entry for each joint action $\underline{a}$ the list and the table representation of Q-values are isomorphic. Furthermore, the question whether the list is sorted does not change the convergence properties at all.

For a joint action learner with list representation the update rule for Q-learning originally given in (1) can be rewritten as follows. After a transition from state s to s' with joint action $L(s)[l].\underline{a}$ and reward r we set:

$$\begin{aligned} L(s)[l].Q \leftarrow & (1 - \alpha_{L(s)[l].n}) \cdot L(s)[l].Q \\ & + \alpha_{\alpha_{L(s)[l].n}} \cdot (r + \gamma \max_{l'} L(s')[l'].Q) \\ L(s)[l].n \leftarrow & L(s)[l].n + 1 \end{aligned} \quad (2)$$

In the case of 'full lists' every joint action is uniquely mapped to a list index. Thus we can address an action using the list index instead of the action itself. This allows us to abstract from the action set. The action selection step is replaced by an index selection and the action is set to $L(s)[l].\underline{a}$ with l the selected index.

Combining index addressing and resorting needs careful design of the action selection scheme. The condition of every state-list index pair (s, l) occurring infinitely often is in general not sufficient to guarantee condition (ii) due to the ongoing resorting. A random based strategy which is at least sufficient for condition (ii) is to assign non-vanishing probabilities to each list index and to select the index randomly according to the specified probabilities. The condition of non-vanishing probabilities and the finiteness of index space ensures that every state-action pair $(s, \underline{a})$ almost surely occurs infinitely often in the sequence of state-action pairs as long as every state s occurs infinitely often.

3.2. Distributed computation in the IL case

The above list structure is the basic data structure for our learning algorithm for independent learners (ILs). Each agent manages its own Q-list. The initial Q-lists are set identically with respect to the entry-lines as well as the list ordering. The agents initially agree on a fair index selection scheme and afterwards use the same resorting algorithm of their Q-lists. Then they act according to algorithm 2.

We can proof by induction that everytime the Q-lists of all agents are identical. Let us consider the lists of agent i and j: the initial Q-tables $L^i(s)$ and $L^j(s)$ are identical by construction. In every step: due to the common index selection scheme both agents choose the same index, i.e. $l_i = l_j$. The induction hypothesis tells us that $L^i(s)[l_i] = L^j(s)[l_j]$ holds and therefore both agents refer to the same joint action. After performing the update (2) both agents resort their lists in the same way. Thus after one iteration the lists $L^i(s)$ and $L^j(s)$ are identical again.

Notice that each agent actually does not need to know the complete joint action but only its own contribution since the identification of the relevant list line is done by the index l, not by the joint action. Thus, it is possible to replace the action vectors in the agent-individual Q-lists by the respective component.

Initialize()
FOR all agents $i = 1 \dots m$,
all states $s = 1 \dots N$,
all indices $l = 1 \dots l_{max}$ {
  $L^i(s)[l].Q := 0.0$;
  $L^i(s)[l].n := 0$;
  $L^i(s)[l].a :=$*ChooseCoordinatedAction()*;
}

LearnTransition(state s)
FOR all agents $i = 1 \dots m$ {
  SELECTINDEX: $l_i = SelectIndex_i(t)$
  OUTPUT $a_i := L^i(s_t)[l_i].a$
}
APPLY $\underline{a} = (a_1, \dots, a_m)$
OBSERVE s', r
FOR all agents $i = 1 \dots m$ {
  $\alpha_i =$*GetLearnrate*$(L^i(s)[l_i].n)$
  $L^i(s)[l_i].Q \leftarrow (1 - \alpha_i)\, L^i(s)[l_i].Q$
    $+\alpha_i(r + \gamma \max_j L^i(s')[j].Q)$
  $L^i(s)[l_i].n \leftarrow L^i(s)[l_i].n + 1$
  SortListByDecreasingQ($L^i(s)$)
}

Main()
Initialize()
REPEAT {
  LearnTransition(*SelectState()*)
} UNTIL TASKISLEARNED

Algorithm 2: Distributed Q-learning in the 'full-lists' variant which is guaranteed to converge whenever the procedures *ChooseCoordinatedActions()*, *SelectIndex$_i$(t)*, *GetLearnrate(i)* and *SelectState()* fulfill the conditions described in section 3.

4. Empirical Results

The example we consider is the 'climbing game' taken from [3] (see figure 1). Claus and Boutilier showed, that their learning scheme for ILs will find an equilibrium, which not necessarily is the optimal solution. In fact, for the climbing game they reported that

	a_0	a_1	a_2
b_0	11	-30	0
b_1	-30	7	0
b_2	0	6	5

Figure 1. Reward matrix of the 'climbing game' [3].

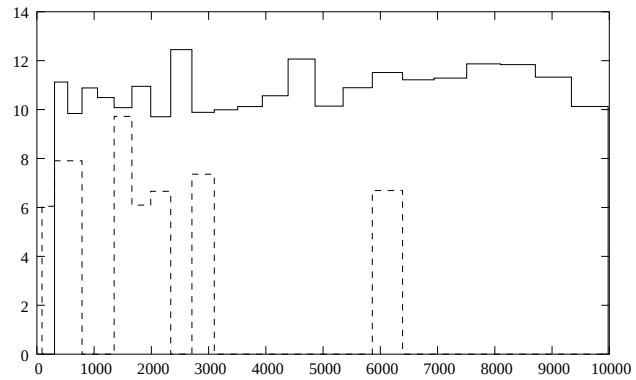

Figure 2. Reduced-lists algorithm for stochastic climbing game with noise variance 1. Q-values of optimal (solid) and second best action (dashed) at the end of training cycle as learned by the agents.

the agents learned to play (a_1, b_1), which yields a reward of 7, whereas the maximum reward is 11 for joint action (a_0, b_0).

Since the full-lists-variant of our algorithm can be theoretically shown to find the optimal solution it is no surprise that the two agents correctly learned to play the optimal joint action (a_0, b_0). Due to the deterministic nature of the original climbing game, once the optimal action has been played, it stays at the top of the individual lists and will never be replaced any more. Therefore, the optimal solution is found rather quickly. For the *reduced-lists* variant the same result holds: Once the optimal action is considered in the list (which is guraranteed to happen if the selection procedure is fair), then it gets the highest reward so far and will never be replaced any more. Thus both variants of our IL algorithm are guaranteed to find the *optimal* solution of the original (deterministic) climbing game, which is a clear improvement over previous results [3].

Next, we extended the climbing game by adding Gaussian noise to the rewards, to test our algorithm in stochastic envrionments. We investigate the behaviour of the reduced-lists variant with list length 4 ($l_{max} = 4$) and $l_{keep} = 2$ (note that the full-lists have a length of 9).

First, a noise variance of 1 is examined, which adds a small random perturbation to the environment. The diagram in 2 shows the learned Q-values for the best action ((a_0, b_0), solid line) and the second best action ((a_1, b_1), dashed line) for the first 10,000 cycles. The Q-values plotted are those from the end of each training interval. If the values are zero, this means that the action is not part of the current list. The optimal action is considered as a list entry at cycle 200 for the first time. After that time, it is never removed again from the list. Additionally, its Q-value is always better than that of any other action. Hence, in that environment, the reduced-lists algorithm finds the optimal action quickly and shows stable behavior. The second best action (a_1, b_1) (dashed line) is only partly a member of the lists. In the other cases, the second permanent list place is occupied by other actions. This is plausible, since the rewards of actions (a_1, b_2) and (a_2, b_2) are very close to the reward of (a_1, b_1).

When the noise variance is further increased to 4, the principle behavior of the learning algorithm is preserverd (see figs. 3 and 4). The learned Q-values oscillate more (which can be expected, since the observed rewards have a higher variance now). Again, once the optimal action got considered in the list, it achieves to remain in the list for the rest of training. The diagram in 3 shows the learning behaviour in some more detail: It plots the expected reward of the current greedy action. During learning intervals, a suboptimal action can look better for a short amount of time (typically one to a few cycles). This might happen, if it gets an occasionally high reward. However, when learning proceeds, the values are averaged again and therefore at the end of each learning interval, the greedy action was observed to be the actual optimal action. Even for noise levels of 5 and higher (current work, not shown here), finally the optimal action can be reliably learned. However, it occurs more often, that the optimal action is removed from the lists in course of learning.

5. Conclusion

We propose a reinforcement learning scheme for distributed learning in a cooperative stochastic environment for the independent learner (IL) case. In contrast to joint action learners (JALs), where the complete action vector of all agents is assumed to be known (making explicit communication necessary), we only assume, that the agents are doing synchronized decisions. The key idea of our approach is to represent the expected costs-to-go in list based data structures instead of ta-

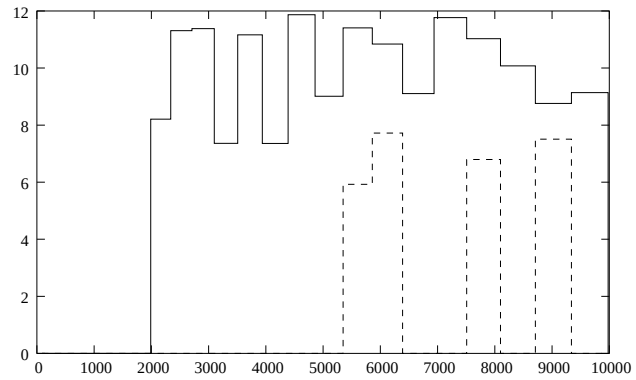

Figure 3. Results for stochastic climbing game with noise variance 4. Q-values of optimal and second best action at the end of training cycle as learned by the agents.

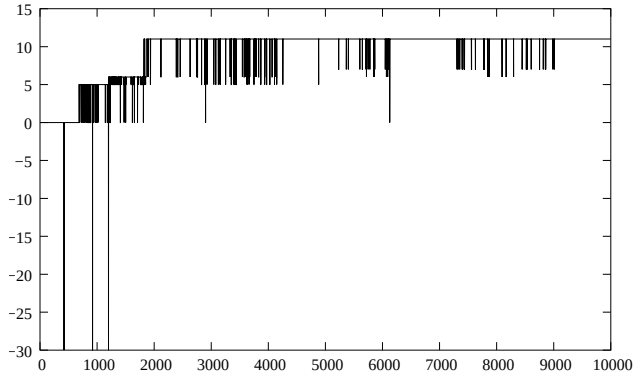

Figure 4. Results for stochastic climbing game with noise variance 4. Expected reward of the current greedy action.

bles. This allows to reference an action vector using a list index instead of the joint action itself. With the assumption of synchronized decisions, a select_index procedure can be built that guarantees that at a certain point in time, every agent independently selects the same index. This allows them to do implicitly coordinated decisions which need no further communication during training and application. By exploiting the relationship to standard Q-learning, convergence of the proposed IL algorithm can be prooved.

For better efficiency we proposed the sorting of the list entries. Prefering small list indices induces a preference of promising action vectors and guides the exploration in order to reduce the computational effort. Furthermore, a *reduced-lists* variant was proposed, which concentrates on a small working subset of action vectors. From time to time the worst actions are replaced by randomly chosen new actions. Due to their large Q-values, the best actions have a high probability of remaining in the working set while the poorer ones are likely to be replaced.

The proposed approach overcomes the elementary problem in distributed learning of distinguishing between random noise and structural influence of other agents. Therefore this approach is able to solve Claus and Boutilier's climbing game problem described in section 4. Future work is aimed at further investigating both empirically and theoretically the convergence behaviour of the reduced-lists variant with respect to the noise level of the envionment. A key point for analysis will be the use of training intervals with increasing length.

References

[1] J. Hu and M. P. Wellman. Multiagent reinforcement learning: theoretical framework and an algorithm. In *Proc. 15th ICML*, 1998.

[2] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proc. 11th Int'l Conf. on Machine Learning*, pp. 157–163, 1994.

[3] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *15th National Conf. on Artificial Intelligence*, 1998.

[4] C. Watkins. *Learning from delayed rewards.* PhD thesis, Department of Computer Science, Cambridge University. Cambridge, UK, 1989.

[5] M. Lauer and M. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proc. 17th ICML*, 2000.

[6] J. Schneider, W. Wong A. Moore and M. Riedmiller, Distributed value functions In *Proc. 16th Int'l Conf. on Machine Learning*, pp. 371–378, 1999.

[7] C. Guestrin, M. Lagoudakis, and R. Parr. Coordinated reinforcement learning. In *Proc. 19th Int'l Conf. on Machine Learning*, pp. 227–234, 2002.

[8] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.