Multiagent Q-Learning by Context-Specific Coordination Graphs

Jelle R. Kok   Nikos Vlassis
Intelligent Autonomous Systems Group, Informatics Institute
Faculty of Science, University of Amsterdam, The Netherlands[*]
{jellekok,vlassis}@science.uva.nl

Abstract.
One of the main problems in cooperative multiagent learning is that the joint action space is exponential in the number of agents. In this paper, we investigate a sparse representation of the joint action space in which value rules specify the coordination dependencies between the different agents for a particular state. Each value rule has an associated payoff which is part of the global Q-function. We will discuss a Q-learning method that updates these context-specific rules based on the optimal joint action found with the coordination graph algorithm. We apply our method to the pursuit domain and compare it with other multiagent reinforcement learning methods.

1 Introduction

A multiagent system (MAS) consists of a group of agents that can potentially interact with each other [9]. In fully cooperative multiagent systems all agents share the same common goal. One of the key problems in such systems is the problem of *coordination*: how to ensure that the individual decisions of the agents result in jointly optimal decisions for the group.

A recent approach to solve this problem is to use context-specific coordination graphs [3] which specify the coordination requirements of the system. Each node in the graph represents an agent and edges between the nodes indicate that the corresponding agents have to coordinate their actions. Each agent has a set of value rules, which define an agent's payoff for a specific context. This context is defined as a propositional rule over state variables and actions of that agent and its neighbors. These sets can be regarded as a sparse representation of the complete state-action space. Using a variable elimination algorithm, the optimal joint action can be derived corresponding to the maximal payoff for the current context.

In [4] we applied coordination graphs to coordinate a group of agents in a dynamic environment using predefined value rules for which the payoffs were set based on a priori information. In this paper, we investigate a method to learn the payoffs using Q-learning. We first specify the coordination requirements of the system by creating value rules that specify in which context certain agents have to coordinate their actions. The global Q-function is then decomposed as the linear sum of the payoffs contained in the agents' value rules which are applicable in the current context. During learning, these payoffs are updated using a Q-learning update which takes the optimal joint action into account.

[*]Appeared in the proceedings of IAS-8: The 8th Conference on Intelligent Autonomous Systems.

2 Coordination Graphs

A coordination graph (CG) represents the coordination requirements of a system [2]. A node in the graph represents an agent $A_i \in A$, while an edge defines a (possible directed) dependency between two agents. Only interconnected agents have to coordinate their actions. The left graph in Fig. 1 shows a possible CG for a 4-agent problem in which agent 3, denoted by A_3, has to coordinate with A_2, A_4 has to coordinate with A_3, and A_1 has to coordinate with both A_2 and A_3. If we assume that the global payoff function can be decomposed as a sum of local payoff functions, the global coordination problem is replaced by a number of easier local coordination problems involving fewer agents. The optimal joint action is found using a variable elimination algorithm in combination with a message passing scheme [2].

The algorithm assumes that each agent knows its neighbors in the graph. Each agent is 'eliminated' from the graph by solving a local optimization problem that involves only this agent and its neighbors: the agent collects from its neighbors all payoff functions in which it is involved, then optimizes its decision conditionally on all possible neighbors' decisions, and finally communicates the resulting 'conditional' payoff function back to its neighbors. A next agent is selected and the process is repeated. This continues until only one agent remains. This last agent fixes its strategy and communicates its decision to its neighbors in order for them to fix their strategy. This procedure repeats itself until all agents have fixed their strategy. The outcome of this algorithm is independent of the elimination order and the initial distribution of the rules, and always results in an optimal joint action [2].

The local payoff functions can be matrix-based [2] or rule-based [3]. In the latter case the payoff functions are defined using 'value rules', which specify how an agent's payoff depends on the current context. The context is defined as a propositional rule over the state variables and the actions of the agent's neighbors. These value rules are a sparse representation of the complete payoff matrices since not all action combinations have to be defined.

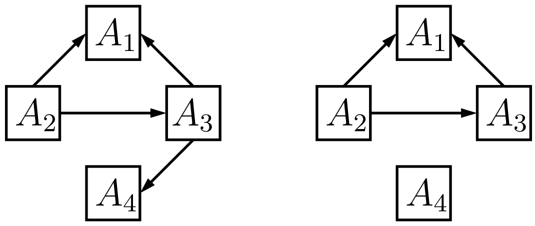
As an example, consider the case where two persons have to coordinate their actions to pass through a narrow door. This situation is described with the value rule

$$\begin{aligned}\langle p1 \quad ; \quad & \text{in-front-of-same-door}(A_1, A_2) \quad \wedge \\ & a_1 = \text{passThroughDoor} \quad \wedge \quad a_2 = \text{passThroughDoor} : -50\rangle.\end{aligned}$$

This rule indicates that when the two agents are located in front of the same door and both select the same action (passing through the door), the global payoff value is reduced by 50. When the state is not consistent with the above rule (and the agents are not located in front of the same door), the rule does not apply and the agents do not have to coordinate their actions. In order to condition on the current state, each agent only needs to observe that part of the state included in its value rules. Based on this information, it discards all irrelevant rules and as a consequence the CG is dynamically updated and simplified.

For a more extensive example, see Fig. 1. Beneath the left graph all value rules, defined over binary action and context variables[1], are depicted together with the agent the rule applies to. The coordination dependencies between the agents are represented by directed edges, where each (child) agent has an incoming edge from the (parent) agent that affects its decision. After the agents observe the current state, $x = true$, they condition on the context.

[1] a_1 corresponds to a_1 = *true* and $\overline{a_1}$ to a_1 = *false*.



A_1	$\langle a_1 \wedge \overline{a_3} \wedge x \quad : 4\rangle$	$\langle a_1 \wedge \overline{a_3} \quad : 4\rangle$
	$\langle a_1 \wedge \overline{a_2} \wedge x \quad : 5\rangle$	$\langle a_1 \wedge \overline{a_2} \quad : 5\rangle$
A_2	$\langle \overline{a_2} \wedge x \quad : 2\rangle$	$\langle \overline{a_2} \quad : 2\rangle$
A_3	$\langle a_3 \wedge a_2 \wedge x \quad : 5\rangle$	$\langle a_3 \wedge a_2 \quad : 5\rangle$
A_4	$\langle a_3 \wedge a_4 \wedge \overline{x} \quad : 10\rangle$	

Figure 1: Initial coordination graph (left) and graph after conditioning on the context $x = true$ (right).

The rule of A_4 does not apply and is removed. As a consequence, the optimal joint action is independent of A_4 and its edge is deleted from the graph as shown in the graph on the right.

After the agents have conditioned on the state variables, the agents are one by one eliminated from the graph. Let us assume that we first eliminate A_3 in the above example. Agent 3 first collects all rules from its children in which it is involved and then maximizes over the rules $\langle a_1 \wedge \overline{a_3} : 4\rangle\langle a_3 \wedge a_2 : 5\rangle$. For all possible actions of A_1 and A_2, A_3 determines its best response and then distributes this conditional strategy, in this case equal to $\langle a_2 : 5\rangle\langle a_1 \wedge \overline{a_2} : 4\rangle$, to its parent A_2. After this step, A_3 has no children in the coordination graph anymore and is eliminated. The procedure then continues and after A_2 has distributed its conditional strategy $\langle a_1 : 11\rangle\langle \overline{a_1} : 5\rangle$ to A_1, it is also eliminated. Finally, A_1 is the last agent left and fixes its action to a_1. Now a second pass in the reverse order is performed, where each agent distributes its strategy to its parents, who then determine their final strategy. This results in the optimal joint action, $\{a_1, \overline{a_2}, \overline{a_3}\}$ and a global payoff of 11.

3 Markov Decision Processes and Q-learning

In this section, we review the Markov Decision Process (MDP) framework. An observable MDP is a tuple $\langle S, A, R, P\rangle$ where S is a finite set of world states; $\mathcal{A}$ is a set of actions; $R : S \times \mathcal{A} \rightarrow \mathbb{R}$ is a reward function that returns the reward $R(s, a)$ obtained after taking action a in state s and $P : S \times \mathcal{A} \times S \rightarrow [0, 1]$ is the Markovian transition function that describes the probability $P(s'|s, a)$ of ending up in state s' when performing action a in state s. The Markov property implies that the state of the world at time t provides a complete description of the history before time t. An agent's policy is denoted by $\pi : S \rightarrow \mathcal{A}$. The objective is to find an optimal policy π^* that maximizes the utility $U^*(s) = \max_\pi E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi, s_0 = s\right]$ for each state s. The expectation operator $E[\cdot]$ averages over reward and stochastic transitions and $\gamma \in [0, 1)$ is the discount factor. We can also represent this using Q-values which explicitly store the expected discounted future reward for each state s and possible action a: $Q^*(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q^*(s', a')$. At any state s, the optimal policy is obtained by choosing an action $\arg\max_a Q^*(s, a)$.

Reinforcement learning (RL) [7] can be applied to estimate $Q^*(s, a)$. Q-learning is a widely used learning method when the transition model is unavailable. This method starts with an initial estimate $Q(s, a)$ for each state-action pair. During exploration it updates the

Q-values based on the received reward and the perceived state transitions using

$$Q(s,a) := (1-\alpha)Q(s,a) + \alpha[R(s,a) + \gamma \max_{a'} Q(s',a')] \tag{1}$$

where $\alpha \in (0,1)$ is the learning rate that specifies the incremental update. It can be shown that $Q(s,a)$ converges to the optimal $Q^*(s,a)$ when all state-action pairs are visited infinitely often by means of an appropriate exploration strategy [7].

Extending the above methods to multiagent systems involves several issues, e.g., whether the agents share the same reward, whether they are allowed to communicate, whether they model each other, etc. In our case, we assume that all agents are allowed to communicate with each other and all receive an individual reward $R_i(s,a)$ based on the selected joint action a in state s. In coordinated states this reward can thus depend on the actions of the other agents.

The most direct approach to extend the MDP framework and Q-learning to a multiagent environment is to regard the complete system as one large single agent in which the joint action is regarded as a single action. In order to apply this approach a central controller should communicate each agent its individual action or all agents should model the complete MDP and select the individual action that corresponds to their own identity[2]. In the latter case, no communication is needed between the agents. Although this approach leads to the optimal solution, it is infeasible for problems with many agents since the joint action space, which is exponential in the number of agents, becomes intractable.

A second approach is to let each agent learn its strategy independently from the other agents and regard the other agents as part of the environment [1]. However, the convergence proof for Q-learning does not hold in this case, since the transition model becomes dependent on the policy of the other learning agents, making it non-stationary [10]. Despite the lack of guarenteed convergence, the method is applied successfully in multiple cases [8, 6].

4 Q-learning in Context-Specific Coordination Graphs

We now focus on our context-specific approach. The main idea is to use a sparse representation of the (joint) Q-values. The value rules described in section 2 offer a convenient representation. First of all, each value rule specifies which agents have to coordinate their actions in a specific context. Secondly, the payoff defined in each value rule can be regarded as a local Q-value. The global Q-function then corresponds to the sum of all local Q-values that are consistent with the current context

$$Q(s,a) = \sum_{j=1}^{n} Q_j(s,a) \tag{2}$$

where n is the number of agents, s is the current state and a is the joint action. An instantiated Q-function Q_i^s in state s corresponds to all applicable value rules in state s for player i. The involved agents in Q_i^s are denoted by Agents$[Q_i^s] = \{A_j \in A \mid A_j \in \mathit{Scope}[Q_i^s]\}$.

For example, a value rule p_1 for an agent i in state s_1 in which it does not have to coordinate with any other agent is defined as $\langle p_1^i; s_1 \wedge a_i : Q_i^{s_1} \rangle$. All action combinations with the other agents are mapped to the rule p_1^i since we assume that the action of agent i is independent of the actions of the other agents in s_1. In this case, Agents$[Q_i^{s_1}]$ is thus equal to $\{i\}$. If

[2] The problem of exploration can be solved by using the same random number generator (and the same seed) for all agents [9].

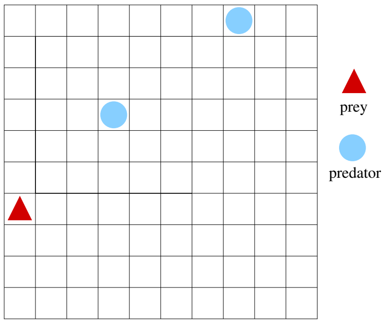


Figure 2: An example situation with two predators and one prey.

in state s_2 we do not make this assumption, we have to add value rules for every possible action combination of agent i with another agent (e.g. agent k), one of which looks as follows: $\langle p_2^i; s_2 \wedge a_i \wedge \overline{a_k} : Q_i^{s_2} \rangle$. In this case Agents$[Q_i^{s_2}]$ is equal to $\{i, k\}$.

The update rule used in Eq. 1 cannot be used anymore, since the global Q-value is distributed over multiple agents and each agent only stores a local Q-function corresponding to its contribution to the global payoff. In order to update a local Q-function, we adapt Eq. 1 to

$$Q_j(s,a) := (1-\alpha)Q_j(s,a) + \alpha[\sum_{k \in A_j^s} R_k(s,a) + \gamma \sum_{l \in A_j^{s,s'}} \frac{|A_j^s \cap A_l^{s'}|}{|A_l^{s'}|} Q_l(s', \arg\max_{a'} Q(s',a'))] \tag{3}$$

where $A_j^s = \text{Agents}[Q_j^s]$ and $A_j^{s,s'} = \{l \in A \mid \text{Agents}[Q_j^s] \cap \text{Agents}[Q_l^{s'}] \neq \emptyset\}$. The updated Q-function holds the future discounted reward for the involved agents. This value is updated based on the direct received reward for all involved agents A_j^s and the expected discounted future reward for the next state s'. The latter is based on the optimal joint action a' that is computed using the coordination graph algorithm from section 2. To determine the future reward for the involved agents A_j^s we need the payoff of all value rules that are consistent with the state s' and the action(s) in a' and incorporate an action of an agent in A_j^s. Since an agent's action can be listed in the value rule of any other agent l, this corresponds to the Q-functions for which an agent in A_j^s is listed in $A_l^{s'}$. Note that l is always a neighbor in the coordination graph and the associated Q-value can be communicated together with the final strategy during the second pass of the variable elimination algorithm. The fraction is needed to distribute the payoffs proportionally over the agents in case $A_l^{s'}$ involves more agents than A_j^s, e.g., when an agent moves from a non-coordinated state to a state in which it has to coordinate with one other agent only half of the payoff is used in the update.

For an agent remaining in a non-coordinated state, this update rule is equal to Eq. 1 since the rules are only defined over single actions and the summations thus sum over single agents. See the next section for a detailed example.

5 Experiments

We applied our method to the well-known Predator-Prey (or Pursuit Domain) [5] in which it is the goal of the predators to capture the prey as fast as possible in a discrete grid-like world. We concentrate on a coordinated problem in which two predators in a 10×10 toroidal grid have to capture a single prey. Each agent can move to one of its adjacent cells or remain on its current position. The prey is captured when both predators are located in an adjacent cell to the prey and only one of the two agents moves to the location of the prey. An example situation is depicted in Fig. 2. When two predators move to the same cell or a predator moves to the prey position without a nearby predator, it is penalized and placed on a random position on the field. The policy of the prey is fixed. It stays on its current position with a probability of 0.2. In all other cases it moves to one of the free adjacent cells with uniform probability.

To apply our approach to this problem, we initialize the first predator with a complete set of value rules, corresponding to all possible states (predator and prey position) and possible action combinations of the two predators. All payoffs are initialized with a value of 75[3]. The initial value rule corresponding to the situation in Fig. 2 looks as follows:

$$\begin{aligned}\langle p_1^i \quad ; \quad &\text{prey}(-3,-3) \quad \wedge \quad \text{pred}(4,3) \quad \wedge \\ &a_i = \text{move\_none} \quad \wedge \quad a_j = \text{move\_north} : 75\rangle\end{aligned}$$

Next, the specific coordination requirements between the two predators are added. Since the predators only have to coordinate their actions when they are close to each other, we remove the action of the second predator from the value rules in which the Manhattan distance to the other predator is larger than two cells or one of the two predators is located more than two cells away from the prey. For these non-coordinated states, new value rules are added to the set of value rules for the second predator containing both this state and all possible actions. Such a value rule looks as $\langle p_1^i; \text{prey}(-3,-3) \wedge \text{pred}(4,3) \wedge a_i = \text{move\_none} : 75\rangle$.

This results in the generation of 73,470 value rules for the first predator (42,270 for the 8,454 non-coordinated states and 31,200 for the 1,248 coordinated states). The second predator holds a set of 42,270 rules for the non-coordinated states. Note that in the coordinated states the action for the second predator will be determined based on the rules from the first predator. During learning we use Eq. (3) to update the payoffs of the rules. In all cases each predator receives a reward of 37.5 when helping in capturing the prey and receives a negative reward of -50.0 when colliding with another predator. This relates to a global reward of respectively 75.0 and -100.0 in the case of two agents. When moving to the prey without support that agent receives a reward of -5.0. In all other cases the reward is -0.5. We use an ϵ-greedy exploration step of 0.2, a learning rate α of 0.3, and a discount factor γ of 0.9.

We compare our method to two other Q-learning methods. First, we compare it against the independent learners. In this case, each Q-value is derived from a state that consists of both the position of the prey and the other predator and one of the five possible actions. This corresponds to $48,510\ (= 99 \cdot 98 \cdot 5)$ different state-action pairs for each agent.

We also model both agents as a complete MDP with the joint action represented as a single action. In this case, the number of state action-pairs equals $242,550\ (= 99 \cdot 98 \cdot 5^2)$.

Figure 3 shows the capture times for the learned policy during the first 500,000 episodes for the different methods. The results are generated by running the current learned policy after each interval of 500 episodes five times on a fixed set of 100 starting configurations. During these 500 test episodes no exploration actions were performed. This complete procedure was repeated for 10 different runs. The 100 starting configurations were selected randomly beforehand and were used during all 10 runs.

Both the independent learners and our proposed method learn quickly in the beginning with respect to the MDP learners since learning is based on fewer state-action pairs. However, the independent learners do not converge to a single policy but keep oscillating. This

[3]The value of 75 corresponds to the maximal reward at the end of an episode. This ensures that the predators explore all possible action combinations sufficiently.

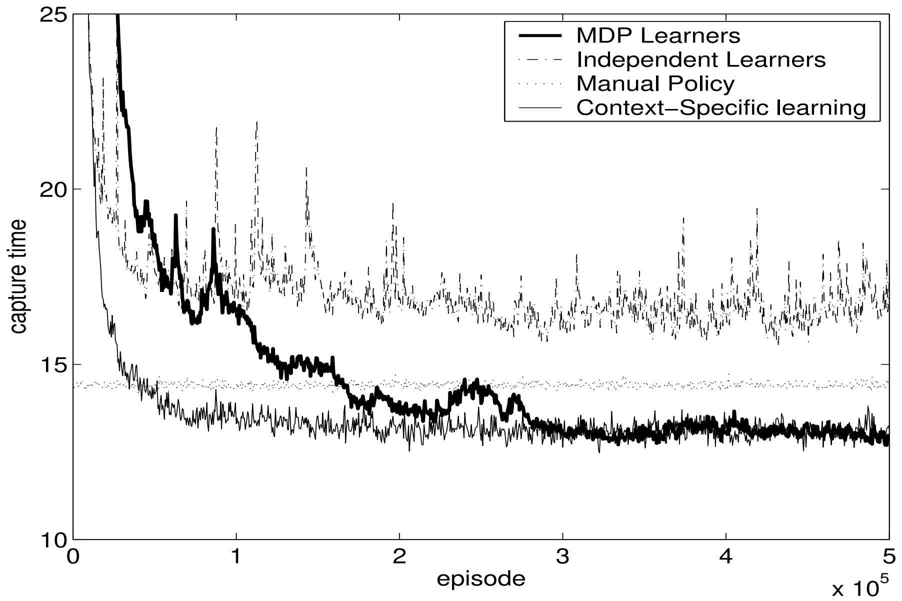


Figure 3: Capture times for the learned policy for the three different methods for the first 500,000 episodes. Results are averaged over 10 runs.

Method	**Avg. capt. time**	**Method**	**Avg. capt. time**
Independent learners	16.86	Context-Specific Learning	12.99
Manual policy	14.43	MDP Learners	12.87

Table 1: Results of different methods after learning. All results are averaged over 5,000 episodes.

is caused by the fact that they do not take the action of the other agent into account. When both predators are located next to the prey and one predator moves to the prey position, this predator is not able to distinguish between the situation where the other predator remains on its current position or performs one of its other actions. In the first case a positive reward is returned, while in the second case a large negative reward is received. However, in both situations the same Q-value is updated.

These coordination dependencies are explicitly taken into account in the two other approaches. In the MDP learners approach, these dependencies are taken into account for every state which results in a slowly decreasing learning curve; it takes longer before all state-action pairs are explored. The context-specific approach has a quicker decreasing learning curve because only joint actions are considered for these states in which the agents have to coordinate their actions. Nevertheless, both methods result in an almost identical policy.

Table 3 shows the average capture times for the three different approaches after learning for the last 10 test runs from Fig. 3 and a manual implementation in which both predators first minimize the distance to the prey and then wait till both predators are located next to the prey. When both predators are arrived, social conventions are used to determine which of the two predators moves to the prey position.

The context-specific learning approach converges to a slightly higher capture time than that of the MDP Learners. A possible explanation for this small difference is the fact that not all necessary coordination requirements are added as value rules. We assume agents do not have to coordinate when they are located far away from each other, but already coordinating in these situations might have a positive influence on the final result. It is possible to add these constraints as extra value rules, but that would decrease the learning time since the state-action space would increase. The independent learners are not able to converge to a good policy. They keep oscillating between different policies for reasons mentioned earlier.

6 Conclusions and future work

In this paper, we discussed a context-specific Q-learning approach for cooperative multiagent systems. Value rules are used to specify the coordination requirements and can be regarded as a sparse representation of the complete state-action space. During learning, the agents only learn to coordinate in specified states. In all other states, single agent learning is applied. Results in the predator-prey domain show that this method improves the learning time considerably and the final results are comparable to the optimal policy.

As future work, we would like to apply our approach to larger domains with more agent dependencies. Furthermore, we would like to investigate methods to learn the coordination requirements of the systems automatically.

Acknowledgements

This research is supported by PROGRESS, the embedded systems research program of the Dutch organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the Technology Foundation STW, project AES 5414.

References

[1] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proc. 15th Nation. Conf. on Artificial Intelligence*, Madison, WI, 1998.

[2] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems 14*, pages 1523–1530. The MIT Press, 2002.

[3] C. Guestrin, S. Venkataraman, and D. Koller. Context-specific multiagent coordination and planning with factored MDPs. In *Proc. 8th Nation. Conf. on Artificial Intelligence*, pages 253–259, Edmonton, Canada, July 2002.

[4] J. R. Kok, M. T. J. Spaan, and N. Vlassis. Multi-robot decision making using coordination graphs. In A. de Almeida and U. Nunes, editors, *Proceedings of the 11th International Conference on Advanced Robotics, ICAR'03*, pages 1124–1129, Coimbra, Portugal, June 30-July 3 2003.

[5] J. R. Kok and N. Vlassis. The pursuit domain package. Technical Report IAS-UVA-03-03, Informatics Institute, University of Amsterdam, The Netherlands, Aug. 2003.

[6] S. Sen, M. Sekaran, and J. Hale. Learning to coordinate without sharing information. In *Proc. 12th Nation. Conf. on Artificial Intelligence*, Seattle, WA, 1994.

[7] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[8] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proc. 10th Int. Conf. on Machine Learning*, Amherst, MA, 1993.

[9] N. Vlassis. A concise introduction to multiagent systems and distributed AI. Informatics Institute, University of Amsterdam, Sept. 2003. http://www.science.uva.nl/˜vlassis/cimasdai.

[10] C. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.