# Learning multi-agent state space representations

Yann-Michaël De Hauwere
Computational Modeling Lab
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussel, BELGIUM
ydehauwe@vub.ac.be

Peter Vrancx
Computational Modeling Lab
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussel, BELGIUM
pvrancx@vub.ac.be

Ann Nowé
Computational Modeling Lab
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussel, BELGIUM
anowe@vub.ac.be

## ABSTRACT

This paper describes an algorithm, called CQ-learning, which learns to adapt the state representation for multi-agent systems in order to coordinate with other agents. We propose a multi-level approach which builds a progressively more advanced representation of the learning problem. The idea is that agents start with a minimal single agent state space representation, which is expanded only when necessary. In cases where agents detect conflicts, they automatically expand their state to explicitly take into account the other agents. These conflict situations are then analyzed in an attempt to find an abstract representation which generalises over the problem states. Our system allows agents to learn effective policies, while avoiding the exponential state space growth typical in multi-agent environments. Furthermore, the method we introduce to generalise over conflict states allows knowledge to be transferred to unseen and possibly more complex situations. Our research departs from previous efforts in this area of multi-agent learning because our agents combine state space generalisation with an agent-centric point of view. The algorithms that we introduce can be used in robotic systems to automatically reduce the sensor information to what is essential to solve the problem at hand. This is a must when dealing with multiple agents, since learning in such environments is a cumbersome task due to the massive amount of information, much of which may be irrelevant. In our experiments we demonstrate a simulation of such environments using various gridworlds.

## Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

## General Terms

Algorithms

## Keywords

Reinforcement Learning, Knowledge representation

## 1. INTRODUCTION

Reinforcement learning (RL) has already been shown to be a powerful tool for solving single agent Markov Decision Processes (MDPs) [14]. It allows a single agent to learn a policy that maximises a possibly delayed reward signal in an initially unknown stochastic stationary environment. However, when multiple agents are present in the environment and influence each other, the convergence guarantees of RL no longer hold since the agents now experience a non-stationary environment [15].

A straightforward approach to deal with this issue is to provide the agents with sufficient information to make the environment they experience stationary. Generally, this means allowing them to observe the state information and selected actions of all the agents in the environment. This becomes untractable very quickly since usually both the state and the action space in which the agents now learn are exponential in the number of agents. As such, this approach is unsuitable for all but the smallest environments with only a few agents present.

Recently a lot of attention has gone to mitigating this problem by learning interdependencies between agents [11, 10] and learning when action coordination is beneficiary [12]. These approaches however are still focused on determining specific states in which coordination is necessary. Even though we start from this same assumption, we also learn to generalise over these states in order to reduce our state space even further and to be able to tranfer our knowledge to different environments. Such generalisations can be very useful in applications such as Robocup soccer where some tactics might be applicable in a defensive as well as in an offensive phase.

Our approach starts with independently trained agents and maintains statistics on observed returns. If the statistics indicate that a change in the policy is needed, the independent state space of an agent is expanded with relevant state information about other agents, in order to allow the agent to learn a better policy. Furthermore we adapt the state space representation to an agent-centric representation in order to be able to generalise from experience. Unlike the work of Ghavamzadeh et al. in [6] coordination among agents is done at the level of primitive actions and not at a subtask level since this requires prior knowledge of the problem at hand.

The remainder of this paper is organised as follows. In Section 2 we elaborate on the necessary background and related work to understand the algorithm introduced in this paper. Section 3 introduces our approach for learning in which states the influence of other agents must be taken into account. We learn to generalise over these states in Section 4. Experiments which illustrate our method in various gridworld environments are presented in Section 5 . We show that our approach can learn an accurate set of states in which the agents influence each other, as well as learn a good generalisation for these states. Finally, we conclude in Section 6.

**Cite as:** Learning multi-agent state space representations, Y-M. De Hauwere, P. Vrancx and A. Nowé, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. 715-722

## 2. BACKGROUND INFORMATION

### 2.1 MDPs and Q-learning

Reinforcement Learning (RL) is an approach to solving a Markov Decision Process (MDP), where an MDP can be described as follows. Let $S = \{s_1, \ldots, s_N\}$ be the state space of a finite Markov chain $\{x_l\}_{l \geq 0}$ and let $A = \{a^1, \ldots, a^r\}$ be the action set available to the agent. Each combination of starting state $s_i$, action choice $a^i \in A^i$ and next state $s_j$ has an associated transition probability $T(s_i, a^i, s_j)$ and immediate reward $R(s_i, a^i)$. The goal is to learn a policy $\pi$, which maps an action to each state so that the expected discounted reward $J^\pi$ is maximised:

$$J^\pi \equiv E\left[\sum_{t=0}^{\infty} \gamma^t R(s(t), \pi(s(t)))\right] \quad (1)$$

where $\gamma \in [0,1)$ is the discount factor and expectations are taken over stochastic rewards and transitions. This goal can also be expressed using Q-values which explicitly store the expected discounted reward for every state-action pair:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a') \quad (2)$$

So in order to find the optimal policy, one can learn this Q-function and subsequently use greedy action selection over these values in every state. Watkins described an algorithm to iteratively approximate $Q^*$. In the Q-learning algorithm [17], a table consisting of state-action pairs is stored. Each entry contains the value for $\hat{Q}(s,a)$ which is the learner's current hypothesis about the actual value of $Q(s,a)$. The $\hat{Q}$-values are updated according to following update rule:

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha_t [R(s,a) + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)] \quad (3)$$

where $\alpha_t$ is the learning rate at time step $t$.

Provided that all state-action pairs are visited infinitely often and a appropriate learning rate is chosen, the estimates $\hat{Q}$ will converge to the optimal values $Q^*$ [15].

### 2.2 Markov Game Definition

In a Markov Game, actions are the joint result of multiple agents choosing an action individually. $A_k = \{a_k^1, \ldots, a_k^r\}$ is now the action set available to agent $k$, with $k : 1 \ldots n$, $n$ being the total number of agents present in the system. Transition probabilities $T(s_i, a^i, s_j)$ now depend on a starting state $s_i$, ending state $s_j$ and a joint action from state $s_i$, i.e. $a^i = (a_1^i, \ldots, a_n^i)$ with $a_k^i \in A_k$. The reward function $R_k(s_i, a^i)$ is now individual to each agent $k$, meaning that agents can receive different rewards for the same state transition.

In a special case of the general Markov game framework, the so-called team games or multi-agent MDPs (MMDPs) optimal policies still exist [1, 2]. In this case, all agents share the same reward function and the Markov game is purely cooperative. This specialisation allows us to define the optimal policy as the joint agent policy, which maximises the payoff of all agents. In the non-cooperative case typically one tries to learn an equilibrium between agent policies [9, 7, 16]. These systems need each agent to calculate equilibria between possible joint actions in every state and as such assume that each agent retains estimates over all joint actions in all states.

### 2.3 Related work

In recent multi-agent research a lot of attention has been given to the problem of local coordination problems in a RL setting. Kok & Vlassis introduced an approach where agents learn when it is beneficial to coordinate with other agents in fully cooperative games [10]. They attempt to reduce the action space which agents need to consider, by letting the agents learn individually, without taking into account the other agents. Only in states where a dependency between the agents exists they learn in the joint action space. To this end they use coordination graphs (CGs) [8] to describe the dependencies between agents. Figure 1 shows a graphical representation of a simple CG for a given situation where the effect of the actions of agent 4 depend on the actions of agent 2 and the actions of agent 2 and 3 both depend on the actions of agent 1.

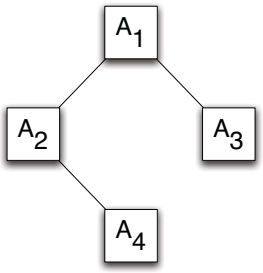



**Figure 1: Simple coordination graph. In the depicted situation the effect of the actions of agent 4 depends on the actions of agent 2 and the actions of agent 2 and 3 both depend on the actions of agent 1.**

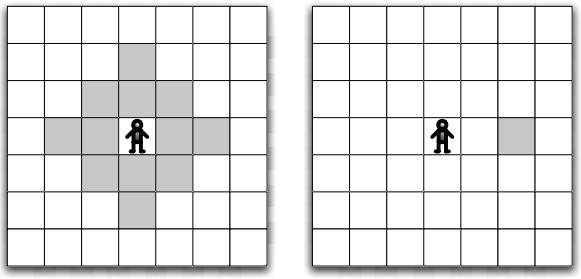
Kok & Vlassis [11] explicitly specify these graphs beforehand, while in Kok et al. [10] this approach is extended to also learn the graphs using statistical information about the obtained rewards conditioned on the states and actions of the other agents. As such, the approach always uses complete information about the joint state-action space in which the agents are learning (i.e. agents are fully observable). From this information they manage to learn a compact representation for the action space of the agents, based on CGs. This approach however is limited to fully cooperative MAS.

Spaan and Melo approached the problem of coordination from a different angle [13]. They introduced a new model for multi-agent decision making under uncertainty called *interaction-driven Markov games* (IDMG). This model contains a set of interaction states which lists all the states in which coordination should occur. In later work, Melo and Veloso [12] introduced an algorithm where agents learn in which states they need to condition their actions on other agents. As such, their approach can be seen as a way of solving an IDMG where the states in which coordination is necessary is not specified beforehand. To achieve this they augment the action space of each agent with a pseudo-coordination action. This action will perform an active perception step. This could for instance be a broadcast to the agents to divulge their location or using a camera or sensors to detect the location of the other agents. This active perception step will decide whether coordination is necessary or if it is safe to ignore the other agents. Since the penalty of miscoordination is bigger than the cost of using the active perception, the agents learn to take this action in the interaction states of the underlying IDMG. Note here that the outcome of this algorithm is strongly dependent on the value of this penalty. A too low value of this parameter could cause the agents to always coordinate, while a too high value would cause them to never choose their coordi-

nate action. In terms of independence this approach however still requires a common reward component, based on the *overall* state of the game and on the actions of *all* agents.

De Hauwere et al. [4, 3] proposed another solution to coordination problems, i.e. a two layer approach called 2observe. The first layer uses a Generalized Learning Automaton (GLA) to decide whether agents could act independently on the second layer, effectively ignoring the presence of other agents, or if a form of coordination mechanism has to be used in that layer in order to avoid that the agents would influence each other in a negative way. Figure 2(a) shows a graphical representaton of a generalisation in gridworld navigation tasks that was learned with the *2observe* algorithm. The GLA received the manhattan distance between two agents as input and based on this distance, the agents learn in which surrounding area attention had to be payed to the presence of other agents. This approach however implies that the environment has to inform the GLA about the correct action to take by means of an appropriate reward signal. This is a requirement that can not always be met.

The rationale behind the new approach presented in this paper is that we do not rely on the environment to inform the agent explicitly about its danger zone. Furthermore, we can even reduce the generalised area by providing more relevant state information, such as the preferred action of the agent. Figure 2(b) shows the reduction we learn. In the next section we explain our approach in depth.



(a) (b)

**Figure 2: (a) Generalisation learned by 2observe and (b) the generalisation that was obtained using a neural network and relevant state information of the CQ-Learning algorithm.**

## 3. LEARNING COORDINATION STATES

Our first step towards reducing our state space is to start from a single agent representation and identify those states in which agents are experiencing negative influence due to the presence of other agents. To this end we developed CQ-Learning, which stands for *Coordinating Q-learning*. This algorithm will identify states in which an agent should take other agents into account when choosing its preferred action and will condition the need for coordination on the preferred action of the other agent. This means that if both agents are adjacent to each other, but their respective preferred actions do not influence each other, the algorithm will not coordinate since there is no need for coordination.

We assume that our agents have already learnt an optimal *single agent* policy when acting alone in the environment. Using this prior knowledge, every agent has a model of its expected rewards for every state action pair. The algorithm then uses a Student's t-test to detect if there are changes in the observed rewards for the selected state-action pair. Two situations can occur:

1. The statistics detect a change in the received immediate rewards. In this situation, the algorithm will mark this state and search for the cause of this change by collecting new samples from the joint state space and as such identifies the joint state-action pairs in which collisions occur. These state-action pairs are then marked as being *dangerous* and the state space of the agent is augmented by adding this joint state. State-action pairs that did not cause collisions are marked as being *safe*, i.e. the agent's actions in this state are independent from the states of other agents. So the algorithm will first attempt to detect changes in the rewards an agent receives, solely based on its own state, before trying to identify for which states of the other agent these changes occur.

2. The statistics indicate that the rewards the agent receives are from the same distribution as if the agent was acting alone. No special action is taken in this situation and the agent continues to act as if it was alone.

Each time an agent comes in a marked state, it will observe whether it is in one of the joint states in which it must take the other agent into consideration. If so, an action is selected using this joint state and following update rule is used:

$$Q_k^j(js,a_k) \leftarrow (1-\alpha_t)Q_k^j(js,a_k) + \alpha_t[r(js,a_k) + \gamma \max_{a'_k} Q(s',a'_k)]$$

where $Q_k$ stands for the Q-table containing the independent states, and $Q_k^j$ contains the joint states ($js$). Note that this second Q-table is initially empty. The Q-values of the single states of an agent are used to bootstrap the Q-values of the states that were augmented to joint states. When an agent comes in a state which has not been marked by the agent, no updates are executed. The algorithm is more formally described in Algorithm 1.

A graphical representation of CQ-Learning is given in Figure 3. Agents begin with 9 independent states. After a while states 4 and 6 of an agent are expanded to include the states of another agent. This means that states 4 and 6 were marked at some point by the statistical test on line 9 of the algorithm and aftwerwards the test on line 11 showed that states 1, 2 and 3 of another agent were responsible for the changes in the reward for state 4 and states 1 and 2 were responsible for the changes that occured when the agent selected its preferred action in state 6. In the following section we explain how the second step of the approach works, i.e. generalise over the found coordination states.

## 4. GENERALISING OVER COORDINATION STATES

As mentioned previously, the size of the state space an agent has to deal with in a multi-agent system is exponential in the number of agents. When joint states and joint actions are always observed, learning in such systems is a cumbersome task with no guarantees of finding good solutions. In the previous section we already introduced an algorithm capable of only taking joint states into consideration when past experience has proven that this was necessary.

Our goal is to further reduce this augmented state space by learning a generalisation over the states in which coordination was necessary. In the gridworld problems we use in our experiments, it is immediately obvious for a human observer that agents need to coordinate to avoid collisions, whenever they are close and moving towards each other. However, with the CQ-learning approach

**Algorithm 1** CQ-Learning algorithm for agent k

```
Initialise Q_k and Q_k^j;
while true do
    if ∀ Agents k, state s_k of Agent k is a safe state then
        Select a_k for Agent k from Q_k
    else
        Select a_k for Agent k from Q_k^j
    end if
    ∀ Agents A_k, sample ⟨s_k, a_k, r_k⟩
    if t-test detects difference in observed rewards vs expected rewards then
        for ∀ seen states s_k of agent A_k do
            if t-test detects difference between independent state s and joint state js then
                add js to Q_k^j
                mark js as dangerous
            else
                mark js as safe
            end if
        end for
    end if
    if s_k is safe for Agent k then
        No need to update Q_k(s).
    else
        Update Q_k^j(js) ← (1 − α_t)Q_k^j(js) + α_t[r(js, a_k) + γ max_a Q(s'_k), a]
    end if
end while
```

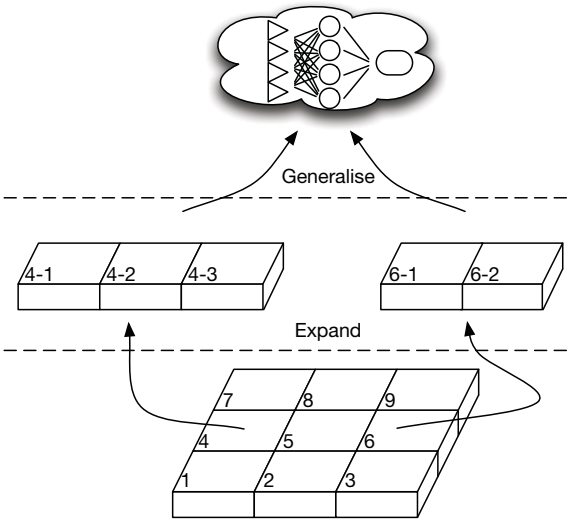



**Figure 3: Independent single states are expanded to joint-states where necessary. Here some states of agent 2 are added to the state representation of agent 1 in states 4 and 6 because these states are identified by the algorithm as states in which coordination is necessary. Over these expanded states we generalise to reduce the state space again.**

above, agents have to learn for each possible conflict location and for each other agent, which agent location and action combinations can cause collisions. This is clearly not the most efficient method, since relatively simple rules exist that can guide an agent through all possible problem locations. We will now introduce a method that attempts to learn such a generalisation over the conflict situations.

To this end we use the knowledge gathered during the execution of CQ-learning, and adapt the state space representation to an agent-centric factored representation. An agent-centric representation only has meaning relative to the agent using it and to the context in which it is being used [5]. The main advantage of such a representation is that if our state information is represented as *'There is an agent North-East of your current position'*, it doesn't matter what the current position of the agent is, nor does it matter which agent it is. Furthermore, this kind of information is a better match for a mobile robot learning problem. In such a setting robots are more likely to have access to sensor readings of their immediate surroundings, rather than full information on the absolute positions of all robots. A factored representation means that a state is represented using a set of random variables $X = X_1, \ldots, X_n$ where each state variable $X_i$ can assume values in a finite domain $Dom(X_i)$. Each possible system state corresponds to a value assignment $x_i \in Dom(X_i)$ for every state variable $X_i$.

Different machine learning techniques are able to generalise over these states, going from rule learning to kernel based approaches. Here we used a feedforward neural network, which was trained with the samples collected during the execution of CQ-learning which are refactored to an agent-centric representation, i.e.:

$$s_1, s_2, a_1, a_2 \rightarrow \Delta(x), \Delta(y), a_1, a_2$$

where $a_1$ and $a_2$ are the preferred actions of agent 1 and 2 respectively. This means that the absolute location, $s_1$ and $s_2$, of the agents for the states CQ-Learning marked as safe or dangerous, is refactored to a relative distance between the agents, $\Delta(x)$ and $\Delta(y)$, respectively in the horizontal and vertical axis. Each agent uses one neural network to generalise over its safe and dangerous states. This neural network is trained as shown in Figure 4.

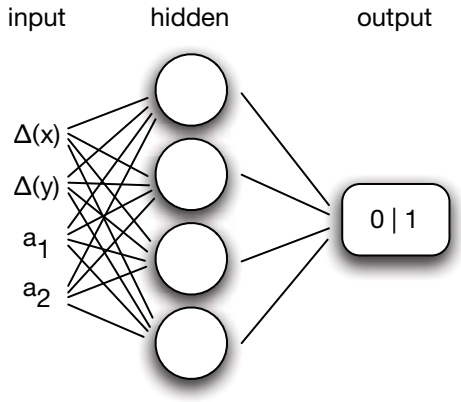



**Figure 4: Neural network used for the generalisation after CQ-Learning. The number of units in the hidden layer were varied over the experiments**

For every sample, the $\Delta(x)$ and $\Delta(y)$ are determined and stored together with a boolean value that indicates if this is a sample that resulted in a collision, or if it was a safe sample. When enough

samples are gathered, the neural network is trained with these samples as input and the boolean value as output.

In practice this neural network can then be used to allow the agent to choose whether it should observe certain locations when choosing an action or not (as shown in the example of Figure 2(b)). There the agent wants to select the action that would take it to the East of its current position. Using the neural network the agent can find out which of its surrounding locations it must take into account to avoid colliding with other agents by querying the network with some possible values for the location and action of the other agent. As another example, in a predator-prey setting, an agent can use the neural network to find out when he has to coordinate with another agent in order to capture the prey. We illustrate our algorithm in this paper in function of our application, which is a set of gridworld environments, but our approach can be applied to most multi-agent reinforcement learning problems by adopting a suitable agent-centric representation for the coordination problem that occurs in that particular setting.

# 5. EXPERIMENTAL RESULTS

The testbed for our algorithms is a set of two-agent gridworld games with varying difficulty in terms of size complexity and number of possible encounters with other agents. We compared our algorithms to independent Q-learners (Indep) that learned without any information about the presence of other agents in the environment, joint-state learners (JS), which received the joint location of the agents as state information but chose their actions independently and joint-state-action learners (JSA) which also received a joint location as input, but selected a joint action (a so-called superagent). For this latter, the reward was given when both agents reached their goal state, but once an agent was in his goal state, he remained in the goal. As such, we could apply joint-state-action learners, because this approach was developed for pure cooperative MAS whereas in most of our environments the agents have a different reward function. The environments we used are depicted in Figure 5. Environments (b) to (e) are an adaptation from the games used by Melo & Veloso [12]. In our games collisions are not limited to a small predetermined set, but can occur in every location of the gridworld. All experiments were run with two agents each having their own goal, except for *TunnelToGoal*. If both agents would adopt the shortest solution in this environment they would constantly collide at the entrance of the tunnel.

All experiments with the independent learners, joint-state learners, joint-state-action learners and the joint-state learners used in CQ-Learning, were run with a learning rate of 0.05 and exploration was regulated using a fixed ε-greedy policy with $\varepsilon = 0.1$. Transitions are deterministic and rewards are given as follows: $+20$ for reaching the goal state, $-1$ for colliding into a wall, $-10$ for colliding into another agent. For CQ-Learning the rewards are chosen from a normal distribution (for implementation reasons) with the mean as mentioned above. Agents were trained for $10,000$ iterations with this configuration. The resulting policy was then played greedily for 1000 iterations and averaged over 10 runs.

Table 1 shows the size of the state space for the different algorithms, as well as the size of the action space, the number of collisions and the average number of steps for each agent to reach the goal in the different environments. We see that all approaches manage to find a collision free path to the goal, but the number of steps greatly varies. Independent learners learn a policy where one agent will not take its shortest path, but take a detour to reach the goal. CQ-Learning does not have to do this since it can include the location of the other agent, in states where this is necessary and as such make the detour much smaller (just one state is enough) to allow the other agent to pass through. Throughout almost all experiments the joint-state-action learners perform worse than all other approaches. This is partly due to the size of the state-action space in which they have to learn which slows their learning process considerably compared to the other approaches and partly due to the fact that this approach is not suited for RL problems where agents do not share the same reward function.

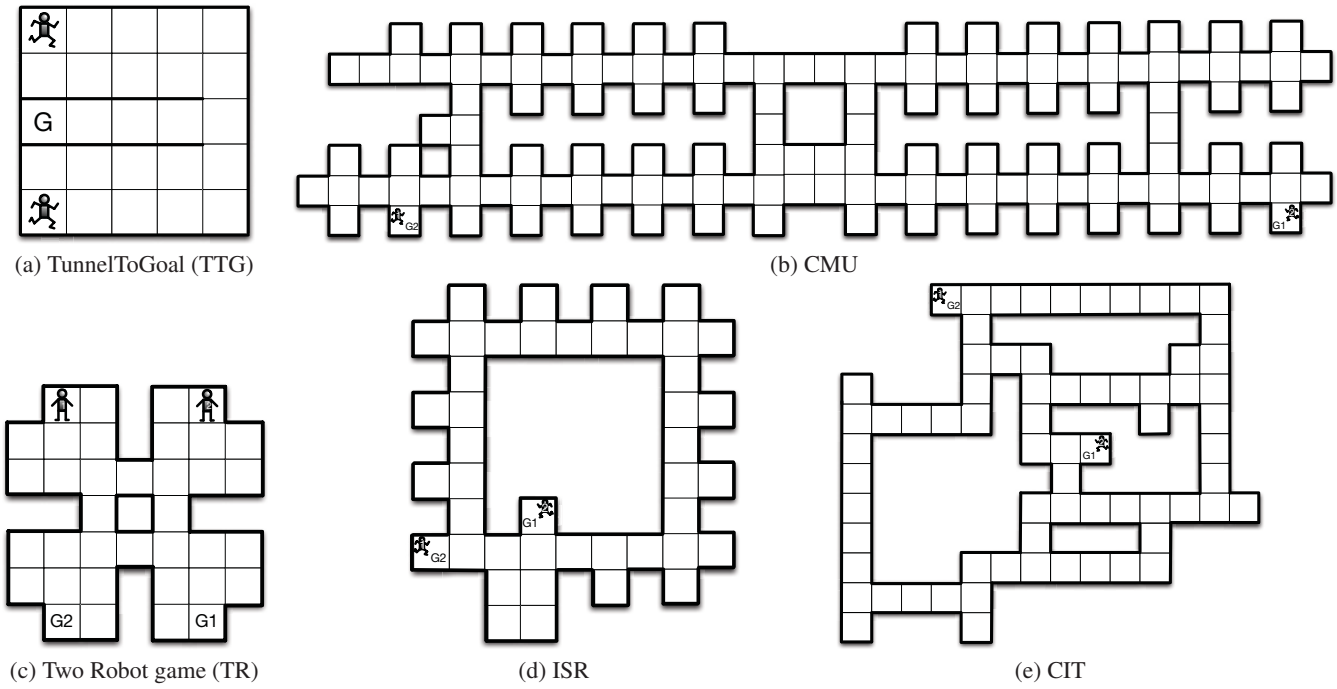
As an example let us consider the TunnelToGoal environment. Using CQ-learning both agents learned to include the other agent's location in their state representation at the entrance of the tunnel, as well as some other states in which changes in the reward were detected. As such the algorithm is on average learning in a state space consisting of approximately 27 states. Using an even higher confidence interval for the Student-t test might reduce the inclusion of unnecessary states, but we found that using a value of 99% certainty gave good overall results in all environments.

In general we can say that CQ-Learning learns a collision free-policy with a minimal extension of the state space. Moreover our algorithm conditions on its preferred action. This means that if agent 1 is situated right next to agent 2, but the preferred action of agent 1 is to move away from the other agent, it will never see this situation as being possibly dangerous. This is a significant difference to previous work [12], where the avoidance of coordination in such a situation is highly dependent on the penalty for miscoordination in that situation and how often it occurs.

| Env | Alg | #states | #actions | #coll | #steps |
|---|---|---|---|---|---|
| TTG | Indep | 25 | 4 | 0 | $11 \pm 0.0$ |
| | JS | 625 | 4 | 0 | $13.3 \pm 1.1$ |
| | JSA | 625 | 16 | 0 | $15.3 \pm 1.89$ |
| | CQ | $27 \pm 0.67$ | 4 | 0 | $\mathbf{10.6 \pm 0.1}$ |
| TR | Indep | 36 | 4 | 0 | $\mathbf{9 \pm 0.0}$ |
| | JS | 1296 | 4 | 0 | $12.1 \pm 0.32$ |
| | JSA | 1296 | 16 | 0 | $13.7 \pm 2.06$ |
| | CQ | $38.0 \pm 2.1$ | 4 | 0 | $9.05 \pm 0.03$ |
| ISR | Indep | 43 | 4 | 0 | $7.7 \pm 18.9$ |
| | JS | 1849 | 4 | 0 | $5.8 \pm 0.89$ |
| | JSA | 1849 | 16 | 0 | $7 \pm 1.5$ |
| | CQ | $48.3 \pm 1.2$ | 4 | 0 | $\mathbf{5 \pm 0.28}$ |
| CIT | Indep | 69 | 4 | 0 | $18.9 \pm 17.9$ |
| | JS | 4761 | 4 | 0 | $15.4 \pm 0.5$ |
| | JSA | 4761 | 16 | 0 | $20.4 \pm 1.43$ |
| | CQ | $77.0 \pm 6.5$ | 4 | 0 | $\mathbf{11.2 \pm 0.9}$ |
| CMU | Indep | 133 | 4 | 0 | $35.9 \pm 2.8$ |
| | JS | 17689 | 4 | 0 | $47.7 \pm 5.6$ |
| | JSA | 17689 | 16 | 0 | $67.1 \pm 4.2$ |
| | CQ | $140.5 \pm 4.3$ | 4 | 0 | $\mathbf{31.3 \pm 0.9}$ |

**Table 1: Size of the state space, number of collisions and number of steps for different approaches in the different games. (Indep = Independent Q-Learners, JS = Joint-state learners, JSA = Joint-state-action learners, CQ = CQ-Learners.)**

After our initial experiments we tested our generalisation approach in the 5 environments. For these experiments we started from the same assumption as for CQ-learning, i.e. agents know how to act independently in the environment if they are alone. For




(a) TunnelToGoal (TTG) (b) CMU

(c) Two Robot game (TR) (d) ISR (e) CIT

**Figure 5: The different games used throughout this experiment. (b), (c), (d) and (e) are variations on games from Melo et al. In all these games, collisions can occur in every state.**

our neural network this means that we started with a set consisting of only safe samples. To this set we added the samples that were gathered during the execution of CQ-Learning by converting the states that were marked safe and dangerous to an agent centric representation. We then trained the neural network with this entire set. We used 80% of the collected samples to train our neural network. The remaining 20% were used to validate the network. We experimented with different quantities of hidden units using a logistic output unit activation function and tested the quality of the neural network. Table 2 shows the accuracy of the neural networks. We can see that as soon as we use 4 hidden units or more, the performance of our network is at least 97% in all environments.

If avoiding miscoordination is critical in the particular situation at hand, the set could also be initialized with samples that classify every situation as being dangerous. As such the neural network will learn high output values for situations that did not occur during the execution of CQ-Learning.

Figure 6 shows two contour plots of the policy of one of the neural networks for actions EAST,WEST for agent 1 and agent 2 respectively. The horizontal distance between the agents, $\Delta x$, is represented on the x-axis and the vertical distance, $\Delta y$ on the y-axis. A high output value (close to 1) means there is a high need for coordination for that situation. A low output value (close to 0) means there is no danger and an agent can select its preferred action without taking other agents into consideration. For instance, $\langle \Delta x = -2, \Delta y = 0, a_1 = EAST, a_2 = WEST \rangle$. This means that the agents are two cells apart on the same height. With actions EAST, WEST the agents would certainly collide since this brings them to the same location. In the left figure we can see that this situation has a very high value and is correctly identified as being a dangerous situation. Notice also that in this figure we clearly see the effect of initialising the neural network with only safe samples. Every situation maps to a low output value, unless where the network was trained with samples which indicated that coordination was necessary. In the figure on the right this same situation also has a high output value, but this network was initialised with dangerous samples. This results in a network that will output a high value for unseen situations, until the network is trained more with safe samples for these cases. Depending on the particular problem at hand, one would prefer one approach over the other. Initialising with dangerous states is much safer to avoid collisions in unseen situations, but will result in more coordinations than if the network was trained with safe samples for unseen situations.

## 6. DISCUSSION

In this paper we presented CQ-learning, a learning algorithm capable of exploiting independent experience in a multi-agent environment while learning a multi-agent representation for states where single agent representations are insufficient to reach a good policy. This is done by means of statistical tests to determine if changes in the immediate rewards an agent receives are significant and if this is the case, augment the representation of that state to include the location of the other agent. The second contribution of this paper is the introduction of a generalisation technique over the augmented state representations by means of a neural network. With this generalisation agents can, in a robotic system, configure the sensors of the robot in order to minimise the flow of information from these sensors to the minimum of what is necessary to predict collisions with other robots and thus avoid them.

Our experiments show that our algorithm is capable of learning a collision free policy in all environments with a limited number of states in which the joint location of the agents had to be observed. This is a vast improvement over joint-state and joint-state-joint-action learners where the size of the state-action space quickly grows out of reasonable bounds. Moreover, unlike the approach de-

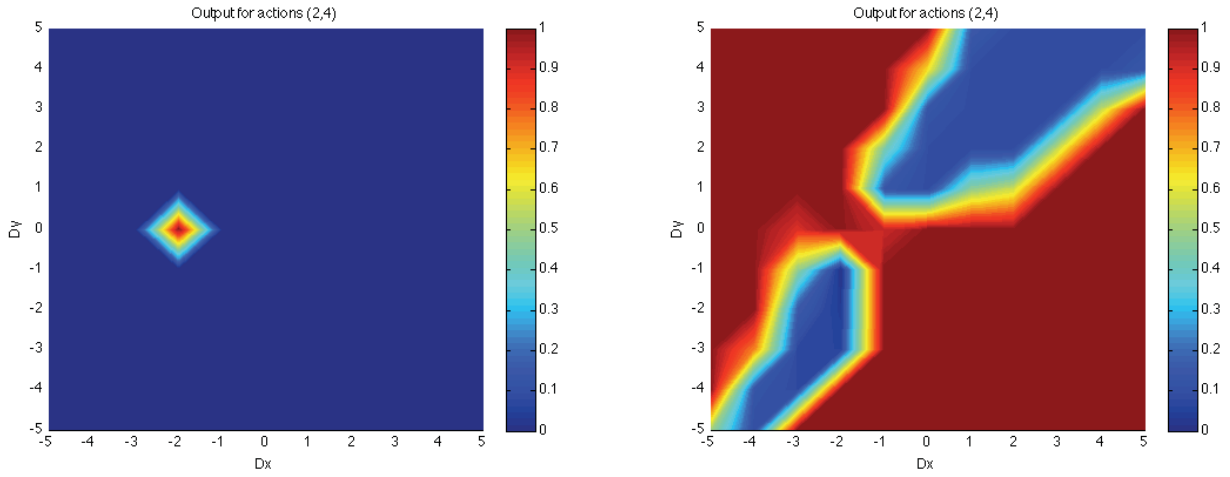



**Figure 6: Output of the neural network for actions EAST, WEST for respectively agent 1 and agent 2. A high output value (close to 1) means there is a high need for coordination for that situation, a low output value (close to 0) means there is no need for coordination. (a) the neural network was initialised with all safe samples and (b) the network was initialised with dangerous samples.**

| Game | Agent | # hidden units in the neural network | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 4 | 6 | 8 |
| TunnelToGoal | 1 | 88.21% | 93.70% | 98.53% | 98.54% | 98.71% |
| | 2 | 97.74% | 98.52% | 98.63% | 98.64% | 98.64% |
| TwoRobot | 1 | 98.35% | 98.35% | 98.35% | 98.35% | 98.35% |
| | 2 | 99.62% | 99.62% | 99.62% | 99.62% | 99.62% |
| ISR | 1 | 89.93% | 92.54% | 96.65% | 97.31% | 97.53% |
| | 2 | 90.46% | 95.24% | 97.87% | 98.43% | 98.97% |
| CIT | 1 | 81.97% | 93.00% | 97.00% | 97.82% | 98.04% |
| | 2 | 86.88% | 91.95% | 97.62% | 98.67% | 98.84% |
| CMU | 1 | 76.93% | 96.53% | 96.63% | 97.11% | 97.21% |
| | 2 | 97.90% | 98.67% | 99.23% | 99.29% | 99.53% |

**Table 2: Accuracy of the neural network in the different games with different quantities of hidden units.**

scribed by Melo & Veloso [12], the performance of our algorithm does not depend on the cost of using the pseudo-coordinate action and does not require a common reward component for all agents.

One interesting issue to explore is the scalability and the performance of our approach to situations with more than two agents and situations in which coordination might be needed among more than two agents; we might even treat the agents as heterogeneous entities. Our algorithm is currently still limited to situations where the change needed to solve the coordination problem is in the same state as where the change in immediate rewards occur. It would be interesting to be able to solve problems at time step $t$ even if the rewards only change at time step $t+n$ with $n \geq 1$. This situation does not occur in the settings we explored here, but imagine that the order in which the agents reach the goal in the TunnelToGoal environment has importance. This would be a case where the immediate reward only changes around the goal, but where the coordination must happen at the entrance of the tunnel. One possible approach to solve this would be to perform the statistical test of CQ-Learning on the Q-values instead of on the immediate rewards.

Another interesting approach would be to reduce the assumption of trained agents, by augmenting the state space while the agents are learning and also training the neural network online. This would make it possible for agents to rapidly learn which situations are dangerous and in a robotic context, to learn quickly how to configure the sensors. Thanks to the generalisation this would mean that the agent could decide at runtime which state information is relevant. It could switch from the independent level to the augmented multi-agent level and even further to the more abstract generalised level without requiring any human interaction.

## Acknowledgments

Yann-Michaël De Hauwere and Peter Vrancx are both funded by a Ph.D grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT Vlaanderen)

## 7. REFERENCES

[1] C. Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 195–210, Renesse, Holland, 1996.

[2] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 746–752. AAAI Press, 1998.

[3] Y.-M. De Hauwere, P. Vrancx, and A. Nowé. Learning what to observe in multi-agent systems. In *Proceedings of the 20th Belgian-Netherlands Conference on Artificial Intelligence*, pages 83–90, 2009.

[4] Y.-M. De Hauwere, P. Vrancx, and A. Nowé. *Multi-agent systems and large state spaces*, chapter to appear. Multi-agent system technology for Internet and Enterprise Systems. Springer, 2010.

[5] S. Finney, P. Wakker, L. Kaelbling, and T. Oates. The thing that we tried didn't work very well : Deictic representation in reinforcement learning. In *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence*, pages 154–160, San Francisco, CA, 2002. Morgan Kaufmann.

[6] M. Ghavamzadeh, S. Mahadevan, and R. Makar. Hierarchical multi-agent reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 13:197–229, 2006.

[7] A. Greenwald and K. Hall. Correlated-Q learning. In *AAAI Spring Symposium*, pages 242–249. AAAI Press, 2003.

[8] C. Guestrin, M. Lagoudakis, and R. Parr. Coordinated reinforcement learning. In *Proceedings of the 19th International Conference on Machine Learning*, pages 227–234, 2002.

[9] J. Hu and M. Wellman. Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4:1039–1069, 2003.

[10] J. Kok, P. 't Hoen, B. Bakker, and N. Vlassis. Utile coordination: Learning interdependencies among cooperative agents. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG05)*, pages 29–36, 2005.

[11] J. Kok and N. Vlassis. Sparse cooperative Q-learning. In *Proceedings of the 21st International Conference on Machine learning*. ACM New York, NY, USA, 2004.

[12] F. Melo and M. Veloso. Learning of coordination: Exploiting sparse interactions in multiagent systems. In *Proceedings of the 8th International Conference on Autonomous Agents and Multi-Agent Systems*, 2009.

[13] M. Spaan and F. Melo. Interaction-driven Markov games for decentralized multiagent planning under uncertainty. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*, pages 525–532, 2008.

[14] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[15] J. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Journal of Machine Learning*, 16(3):185–202, 1994.

[16] P. Vrancx, K. Verbeeck, and A. Nowé. Decentralized learning in markov games. *IEEE Transactions on Systems, Man and Cybernetics (Part B: Cybernetics)*, 38(4):976–981, 2008.

[17] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.