ABSTRACT
Recent research has demonstrated that considering local interactions among agents in specific parts of the state space, is a successful way of simplifying the multi-agent learning process. By taking into account other agents only when a conflict is possible, an agent can significantly reduce the state-action space in which it learns. Current approaches, however, consider only the immediate rewards for detecting conflicts. This restriction is not suitable for realistic systems, where rewards can be delayed and often conflicts between agents become apparent only several time-steps after an action has been taken.

In this paper, we contribute a reinforcement learning algorithm that learns where a strategic interaction among agents is needed, several time-steps before the conflict is reflected by the (immediate) reward signal. To do this, we make use of statistical information about the future returns and the state information of the agents. This allows the agent to determine when it should expand its state representation with information on the other agents and when it can safely rely on its own state information. We apply our method to a set of representative grid world problems and show that with our approach, agents successfully manage to expand their state information to solve delayed coordination problems.

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, coordination problems, multi-agent learning

1. INTRODUCTION
Reinforcement Learning (RL) is an unsupervised learning technique which allows agents to learn policies in initially unknown, possibly stochastic, environments, steered by a scalar reward signal they receive from the environment. This signal can however be delayed, such that agents only see the effect of a certain action, several timesteps after the action was performed. Using an appropriate backup diagram which backpropagates these rewards still ensures convergence to the optimal policy [12]. When multiple agents are present in the environment, these guarantees no longer hold, since the agents now experience a non-stationary environment due to the influence of other agents [13].

Most multi-agent learning approaches alleviate the problem by providing the agents with sufficient information about each other. Generally this information means the state information and selected actions of all the other agents. As such, the state-action space becomes exponential in the number of agents.

Recent research has illustrated that it is possible to identify in which situations this extra state information is necessary to obtain good policies [10, 3] or in which states agents have to explicitly coordinate their actions [9, 8]. These techniques rely on sparse interactions with other agents and only use the state information of the other agents if this is needed. In all these techniques however, it is assumed that the need for coordination is reflected in the immediate reward signal. However, in RL-systems a delayed reward signal is common. Likewise, in a multi-agent environment the effect of the joint action of the agent is often only visible several time steps in the future.

In this paper we describe an algorithm which will determine the influence of other agents on the total reward until termination of the learning episode. By means of statistical test on this information it is possible to determine when the agent should take other agents into consideration even though this is not yet reflected by the immediate reward signal. By augmenting the state information of the agents in these situations to include the (local) state of the other agents, agents can coordinate without always having to learn in the entire joint-state joint-action space. An example for such situations are for instance mobile robots which can not cross each other in small alleys. Coordination should occur at the entrance of such an alley, but robots will only observe
the problem when they bump into each other when they are already in the alley. In our experiments we evaluate a simplified version of this problem using gridworld environments.

The remainder of this paper is organised as follows: in Section 2 we introduce the necessary background on reinforcement learning and describe related work around sparse interactions. Section 3 describes our approach of solving coordination problems which are not reflected in the immediate reward. We illustrate our algorithm in Section 4 in various gridworlds. Such environments are a representative simplified version of mobile robots and are thus a good testbed for learning future coordination problems. Finally, we conclude in Section 5.

2. BACKGROUND INFORMATION

2.1 Reinforcement Learning

Reinforcement Learning (RL) is an approach to solving a Markov Decision Process (MDP), where an MDP can be described as follows. Let Markov Decision Process (MDP), where an MDP can be expressed using state transition and reward. We illustrate our algorithm in Section 4 in various gridworlds. Such environments are a representative simplified version of mobile robots and are thus a good testbed for learning future coordination problems. Finally, we conclude in Section 5.

2.2 Markov Game Definition

In a Markov Game, actions are the joint result of multiple agents choosing an action individually. \( A_k = \{a^k_1, \ldots, a^k_n\} \) is now the action set available to agent \( k \), with \( k = 1 \ldots n \). 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 Learning with sparse interactions

Recent research around multi-agent reinforcement learning is trying to make a bridge between a complete independent view of the state of the system and a fully cooperative system where agents share all information. Terms such as local or sparse interactions where introduced to describe this new venue in MARL.

Kok & Vlassis use a sparse representation of the joint action space of the agents. They describe a set of states in which the agents explicitly have to coordinate their actions[9]. These dependencies between the actions of the different agents are represented by coordination graphs (CGs) [6]. The authors later expanded this approach to also learn the CGs using statistical information about the obtained rewards conditioned on the states and actions of the other agents [8]. This approach always uses complete information about the joint state space in which the agents are learning (i.e. agents are fully observable), but only learn using the joint action space in the coordination states. By observing the actions taken by other agents in a given state, they could identify in which states a dependency existed between the actions selected by the agents. For states in which dependencies were detected and for which a CG existed, the agents execute a variable elimination algorithm to select a joint action. This approach however is limited to fully cooperative MAS.

Spaan and Melo approached the problem of coordination from a different angle [11]. 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 list all the states in which coordination should occur, or, in other words, in which states the local state of other agents should be observed. In later work, Melo and Veloso [10] introduced an algorithm where agents learn in which states they need to condition their actions on other agents. This approach is
called Learning of Coordination and will be referred to in this paper as LoC. 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. This approach solves the coordination problem by deferring it to the active perception mechanism.

De Hauwere et al. introduced CQ-learning for dealing with sparse interactions [3]. This algorithm maintains statistics on the obtained immediate rewards and compares these against a baseline, which it received from training the agents independent of each other or by tracking the evolution of the rewards over time [4]. As such, states in which coordination should occur, could be identified and the state information of these states was augmented to include the state information of the other agents. These are states in which there is a statistical significant difference exists between the rewards of acting alone in the environment and acting with multiple agents or when the rewards radically change over time. This technique can also be seen as a way of solving an IDMG, since it also learns the states in which coordination is necessary. However, it does not rely on external mechanisms, such as active perception, to do so.

All of these approaches however assume that states in which coordination is required can be identified using the immediate rewards that are received in those states. In the following section we will describe that this assumption might not always be met and thus there is need for more general algorithms capable of dealing with this issue.

3. DELAYED COORDINATION PROBLEMS

One of the main features of reinforcement learning is the capability of dealing with delays in the reward signal. This capability has not yet been ported to the framework of sparse interactions and thus, to the best of our knowledge, all work in this area depends on a penalty for miscoordination being available immediately. If we think about mobile robots navigating in an environment, it is possible that there are some bottleneck areas, such as small alleys where robots will only see the fact that they had to coordinate when it is already too late, i.e. both robots are already in the alley. A similar situation in which coordination must occur is when the order in which agents enter the goal is important for the reward they can earn. In the experiments section we will illustrate our approach in such problem environments. We will begin by explaining how our approach works.

3.1 FCQ-learning

The technique we describe here uses the same basic principles as CQ-learning, but has been adapted to be able to deal with future coordination problems. This is why we call our approach FCQ-learning, which stands for Future Coordinating Q-learning. As for CQ-learning, the idea is that agents learn in which of their local states they will augment their state information to incorporate the information of other agents and use a more global system state. This idea is represented in Figure 1. The local states for one agent are represented at the bottom. In its local states labeled 4 and 6 it augmented its information to include global state information illustrated at the top. For now, we use the same initial assumption of CQ-learning, that the agents have already learnt an optimal single agent policy when acting alone in the environment and that their Q-values have converged to the correct values.

![Figure 1: The state information of states 4 and 6 is augmented to incorporate additional information in order to solve coordination problems.](image)

Given this information the most important challenge is detecting in which states, the state information must be augmented. FCQ-learning makes use of a Kolmogorov-Smirnov test for goodness of fit to trigger an initial sampling phase. This statistical test can determine the significance of the difference between a given population of samples and a specified distribution. Since the agents have converged to the correct Q-values, the algorithm will compare the evolution of the Q-values when multiple agents are present to the values it learned when acting alone in the environment. In Figure 2 we show the states in which this statistical test has observed a difference for one agent. The darker the shade of the cell, the earlier the change was detected. The goal was to reach the cell marked by G starting from the cell marked by an x. We first allowed the agent to learn the correct Q-values, after which we changed the value of the reward received for reaching the goal. The KS-test detected this change first in the Q-values of the cell adjacent to the goal state. Since the Q-values were still being updated, the KS-test continued detecting changes further down, back to the initial state of the agent.

If a change is detected in the Q-values of a state of an agent, it will start observing the local state information of the other agents and start sampling the rewards it collects, starting from that local state until termination of the episode. Using these samples, the agent can perform a Friedmann statistical test which can identify the significance of the difference between the different local states of the other agents for its own local state. This principle is represented in Figure 3. Agent 1 starts sampling the rewards until termination of the episode in local state x based on the local state information y', y'' and y'' of Agent 2. If enough samples have been collected and if a significant difference is detected among these
We distinguish two cases for updating the Q-values:

1. An agent is in a state in which it used the global state information to select an action. In this situation the following update rule is used:

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

   where \( Q_k \) stands for the Q-table containing the local states, and \( Q_k^j \) contains the joint states using global information \( js \). Note that this second Q-table is initially empty. The Q-values of the local states of an agent are used to bootstrap the Q-values of the states that were augmented with global state information.

2. An agent is in a state in which it selected an action using only its local state information. In this case the Q-learning rule of Equation 3 is used with only local state information.

We do not consider the case where we use the Q-table with joint states to bootstrap in our update scheme since at timestep \( t \) an agent can not know at that time that it will be in a state where coordination will be necessary at timestep \( t+1 \) as this will also depend on the actions of other agents.

**Algorithm 1 FCQ-Learning algorithm for agent \( k \)**

1: Initialise \( Q_k^1 \) to \( Q_k^j \) to zero, and list of safe states to \{\};
2: while true do
3: \( s_k \) \textbf{if} Agents \( k \), state \( s_k \) of Agent \( k \) is a safe state \textbf{then}
4: \hspace{1em} Select \( a_k \) for Agent \( k \) from \( Q_k^1 \)
5: \hspace{1em} \textbf{else}
6: \hspace{2em} Select \( a_k \) for Agent \( k \) from \( Q_k^j \)
7: \hspace{1em} \textbf{end if}
8: \( s_k \) \textbf{if} KS-test fails to reject the hypothesis that the Q-values of \( Q_k^1 \) are the same as \( Q_k^j \) \textbf{then}
9: \hspace{1em} Mark state \( s_k \) as a sample state
10: \hspace{2em} if \( s_k \) is a sample state \textbf{then}
11: \hspace{3em} Store the state information of other agents, and collect the rewards until termination of the episode
12: \hspace{2em} \textbf{if} enough samples have been collected \textbf{then}
13: \hspace{3em} perform Friedmann test on the samples for the state information of the other agents. If the test indicates a significant difference, augment \( s_k \) to include state information of the other agents
14: \hspace{3em} \textbf{end if}
15: \hspace{1em} \textbf{end if}
16: \hspace{1em} \textbf{end if}
17: \textbf{end if}
18: \textbf{if} \( s_k \) is an augmented state for Agent \( k \) \textbf{then}
19: \hspace{1em} Update \( Q_k^1(js) \leftarrow (1 - \alpha_t)Q_k^1(js) + \alpha_t[r(js, a_k) + \gamma \max_{a_k} Q_k(s', a_k)] \)
20: \hspace{1em} \textbf{end if}
21: \textbf{else}
22: \hspace{1em} Update \( Q_k^j(s) \leftarrow (1 - \alpha_t)Q_k^j(s) + \alpha_t[r(js, a_k) + \gamma \max_{a'_k} Q_k(s', a'_k)] \)
23: \hspace{1em} \textbf{end if}
24: \textbf{end while}

For every augmented state a confidence value is maintained which indicates how certain the algorithm is that this is indeed a state in which coordination might be beneficial. This value is updated at every visit of the local state. If when this local state is visited, the state information about the other agents corresponds to the augmented state, the confidence value is increased, otherwise it is decreased. This ensures that states, where an agent would request state information about another agent, but where this state information does not correspond to the augmented state, are reduced again to states where agents only consider local state information.
By reducing this value less than we increase it, we built some fault tolerance against too quickly reducing states again. The algorithm is more formally described in Algorithm 1.

### 3.2 FCQ-learning with uninitialised agents

Having initialised agents beforehand who have learned the correct Q-values to complete their task is an ideal situation, since agents can transfer the knowledge they learned in a single agent setting to a multi-agent setting, adapting only their policy when they have to. This is of course not always possible. This is why we propose a simple variant of FCQ-learning. In the original algorithm, the initialised Q-values are being used for the KS-test which will detect in which states the agent should start sampling rewards. As such, all changes in state information and state information about the other agents in those states where this is not necessary, since it allows an agent to only sample in those states that are being visited by the current policy and in which a change has been detected. If this compact set of states in which coordination problems should be explored cannot be obtained, it is possible to collect samples for every state-action pair at every timestep. This results in a lot more data to run statistical tests on, most of which will be irrelevant. The changes in Algorithm 1 for this variant are to remove the lines 8 to 10 and 15 to 16.

### 3.3 Discussion of the algorithms

The main idea of our approach is to detect coordination problems, several timesteps ahead of the actual occurrence of the problem. In FCQ-learning such coordination problems are then solved by expanding the state information an agent can use to select an action. We would like to emphasize that for some problems, this approach might not yield in the wanted result, since this information might still be insufficient, or because selecting the actions independently is not sufficient and agents need to coordinate. Other possibilities are to communicate with the other agent, or even use domain knowledge about the task at hand, to get a more descriptive representation of the problem or to rely on joint-action techniques such as joint-action learners [2]. Vice versa, problem situations can also be used to update or refine this domain knowledge.

### 4. EXPERIMENTAL RESULTS

The testbed for our algorithms is a set of two and three-agent gridworld games with varying difficulty in terms of size complexity. 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 with LoC (described in Section 2.3). The environments we used are depicted in Figure 4. The initial position of the agents is marked by an x, the goal is indicated with a dot. If the agents have different goals, like in environment d, there goal is marked in the same colour as their initial position.

To create more complex coordination problems in these environments, agents can not only collide with each other in every cell, but in environments a, b and c the agents also have to enter the goal location in a specific order. In environment d it is clear that if agents adopt the shortest path to the goal, they collide in the middle of the corridor.

All experiments were run for 20,000 episodes (an episode was completed when all agents were in the goal state) using a learning rate of 0.1 with a time limit of 500,000 steps per episode. Exploration was regulated using a fixed ε-greedy policy with ε = 0.1. If agents collided they remained in their respective original locations and receive a penalty for colliding. On all other occasions, transitions and rewards were deterministic. The results described in the remainder of this paragraph are the running averages over the last 50 episodes taken over 50 independent runs. In LoC we could not implement a form of virtual sensory input to detect when coordination was necessary for the active perception step. The reason for this is that a sensor can not determine the need for interaction in the future. To circumvent this issue, we used a list of joint states. In each of these joint states, coordination with the other agent would be better than to play independent.

As such this implementation could be seen as incorporating domain knowledge in the algorithm. If this knowledge however is not available, an active perception function that always returns true, might be a good option.

<table>
<thead>
<tr>
<th>Environment</th>
<th>Grid Game 2</th>
<th>TunnelToGoal_3</th>
<th>TunnelToGoal</th>
<th>Bottleneck</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td><img src="a" alt="Grid Game 2" /></td>
<td><img src="b" alt="TunnelToGoal_3" /></td>
<td><img src="c" alt="TunnelToGoal" /></td>
<td><img src="d" alt="Bottleneck" /></td>
</tr>
</tbody>
</table>

Figure 4: The different games used throughout the experiments. An x marks the initial position of an agent, a dot marks the goal position. If there are different goals, the start state and goal state are those where the colors match. In environments a to c the agents had to reach the goal in a certain order to obtain the maximum reward.

We will begin by describing the results in terms of the reward obtained per episode, the number of steps needed to reach the goal and the number of collisions that occurred each episode. These results are shown in Figure 5. Each row contains the results of one environment, using the same order as in Figure 4, Grid_game_2 at the top, followed by TunnelToGoal_3, TunnelToGoal on the third row and finally Bottleneck at the bottom. The first column represents the rewards that were obtained during an episode. Note that the maximum reachable value is 20 if agents did not collide with each other or with a wall. The second column contains
the graphs for the number of steps both agents needed to complete the episode, i.e. both agents reached their goal. The third column displays how often agents collided with each other per episode.

In Grid game, the independent learners do not manage to find a collision free policy. This is to be expected since in their initial state they have to choose between bumping into a wall, or taking the action that will get them closer to the goal, but also results in a penalty for colliding with the other agent. They still manage to reach the goal eventually due to the randomness of the action selection strategy. Joint state learners quickly learned a good policy without much problems. The size of the joint state space of this environment contains after all only 81 states. Agents using the LoC algorithm did not learn to use their COORDINATE action in their initial state and hence did not manage to act without colliding or reach the goal in the correct order. Both variants of FCQ-learning however did find a collision free policy and reached the goal in order as soon as they managed to collect enough samples. Before this, their behaviour was similar to the agents using only their local state information. The results for TunnelToGoal, depicted in the third row, are very similar.

When three agents are present in the environment, as in the TunnelToGoal environment, we quickly see a decay in the performance of the joint state learners, who need a lot more time to find a collision free policy. Note that we show the first 10,000 episodes for this environment, compared to the first 3,000 for Grid game and TunnelToGoal. We also see a decrease in the performance for the FCQ-learning algorithms. This is caused by their coordination strategy. As explained in Section 3.3, using joint-state information for selecting an action, while bootstrapping with the Q-values of the independent states is not the best choice, since the values of the independent states might not represent the correct value of the next joint state. A more advanced coordination strategy, that also includes future joint states or that is even based on communication might give better results.

For the Bottleneck environment we cannot present any result for the independent learners. This is because the $\epsilon$-greedy exploration strategy they use, does not provide enough exploration to find a policy to the goal, since this is blocked by the penalties it receives by entering the tunnel in the middle. Contrary to TunnelToGoal, 1 move is not enough to avoid collisions here so independent learners cannot escape collisions due to an exploratory move. Coordination must occur before both agents are in the corridor. In this environment we clearly see that FCQ-learning finds a shorter path and obtains higher average rewards than all other approaches. LoC does not manage to learn anything useful in this environment since this algorithm is steered by the immediate reward to learn for which states coordination is necessary. But at the moment the collision occurs, and the immediate reward reflects that a bad action was selected, the agents might already be inside the corridor and can not learn to exit it again to let one agent pass.

In general we can conclude that FCQ-learning performs very similar or slightly better than joint state learners for relative small environments. When playing in larger environments such as TunnelToGoal which contains 166,375 possible joint states, FCQ-learning outperforms the other algorithms in terms of number of episodes needed to converge and quality of the solution.

Furthermore we also compared the number of times the algorithms selected an action using joint state information. For the independent learners this number is always 0, whereas for the joint state learners this number equals the number of steps needed to finish the episode. For the remaining algorithms the results are shown in Figure 6.

**Figure 6: Number of times an algorithm used joint state information to select an action for environments** (a) Grid game, (b) TunnelToGoal, (c) TunnelToGoal and (d) Bottleneck.

All algorithms learn a compact set of states in which they use state information about other agents to select their actions, except for LoC in the Bottleneck environment. The reason for this is the same as why it needed a large number of steps to complete an episode. The immediate reward does not reflect the coordination problems, so agents will learn to coordinate in the wrong states, and still receive negative rewards. As mentioned earlier in the discussion of the algorithms (Section 3.3), using a different coordination technique for LoC than just selecting an action in the joint state space might be a good idea.

5. CONCLUSION

In this paper we presented an algorithm that learns in which states of the state space an agent needs to include knowledge
Figure 5: The rows represent the different gridworlds in which we tested the algorithm, Grid_game_2 on top, followed by TunnelToGoal_3, TunnelToGoal and Bottleneck. In the columns we show (a) the rewards per episode, (b) the number of steps per episode and (c) the collisions per episode.
or state information about other agents in order to avoid coordination problems that might occur in the future. Situations in which such problems occur are for instance when multiple autonomous robots are required to go through a small corridor where they can only pass one at a time. By means of statistical tests on the obtained rewards and the local state information of other agents, FCQ-learning is capable of learning in which states it has to augment its state information in order to select actions using this augmented state information. We have shown two variants on this algorithm which perform similar in terms of the quality of the found solution, but have a different computational complexity in terms of processing power and memory usage, due to the number of samples collected and on which statistical tests have to be performed.

To the best of our knowledge, our technique is the first one to use sparse interactions with other agents to solve delayed coordination problems. Using sparse interactions has already been proven to have many advantages in recent literature. When solving problems in which delayed coordination problems occur, sparse interactions also prove to be beneficial. The biggest improvement could be seen in our experiments using three agents. The learning process of agents who always use the joint state space was very slow compared to our approach based on sparse interactions.

We would like to emphasize that our algorithm can be seen in a broader way as a technique of detecting when the current policy fails due to the interference of other agents and in which situations this interference takes place. As such it can be put in the wider context of robocup, where a team of agents can evaluate its strategy and learn a set of preconditions about the other team to detect when their strategy fails. This is an interesting application to explore in future research.

Another interesting avenue for future research is exploring the possibilities for detecting situations where coordination among multiple agents is necessary, such as intersections. On the other hand, detecting these situations is only half the work done. These conflicts also have to be solved, which results in exploring different coordination techniques than merely selecting actions using more state information.

6. REFERENCES