Learning of Coordination: Exploiting Sparse Interactions in Multiagent Systems

Francisco S. Melo
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15217, USA
fmelo@cs.cmu.edu

Manuela Veloso
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15217, USA
veloso@cs.cmu.edu

ABSTRACT

Creating coordinated multiagent policies in environments with uncertainty is a challenging problem, which can be greatly simplified if the coordination needs are known to be limited to specific parts of the state space, as previous work has successfully shown. In this work, we assume that such needs are unknown and we investigate coordination learning in multiagent settings. We contribute a reinforcement learning based algorithm in which independent decision-makers/agents learn both individual policies and when and how to coordinate. We focus on problems in which the interaction between the agents is sparse, exploiting this property to minimize the coupling of the learning processes for the different agents. We introduce a two-layer extension of Q-learning, in which we augment the action space of each agent with a coordination action that uses information from other agents to decide the correct action. Our results show that our agents learn both to act coordinate and to act independently, in the different regions of the space where they need to, and need not to, coordinate, respectively.

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, multiagent coordination, local interaction

1. INTRODUCTION

Markov games provide a rich framework to model sequential decision-making in multiagent systems. However, a Markov game approach considers a tight coupling between the agents, and typical solution methods rely on strong joint-observability and joint-action assumptions that seldom hold in practice. Models that disregard joint-observability are in general too complex to be solved exactly: even in the benign case in which all agents share the same payoff function (*e.g.*, in Dec-MDPs), the problem of acting optimally is provably undecidable [2]. This complexity is largely due to the tight coupling between the actions of all agents and their effects on the state of the system and the reward received by each agent.

Interestingly, we observe that in many applications the interactions between the different agents coexisting in a common environment are not very frequent. Consider, for example, two robots moving in an office environment coordinating to deliver faxes. They may need to coordinate to distribute the task among them, but certainly for most of the time, each robot can move about the environment disregarding the position and actions of the other robot. Only occasionally – for example when both robots intend to cross the same doorway – should they consider the position/actions of the other robot. Similarly, in robot soccer, an attacker in front of the opponent's clear goal and in possession of the ball can act independently of its teammates to effectively score [15].

In this work, we investigate the problem of *learning* these situations in which the agents need to coordinate. Hence, our driving question is "to coordinate or not to coordinate". Our approach seeks to exploit *local interactions*, allowing the agents to act independently whenever possible.

Several authors have previously explored locality of interaction in different ways. Possible approaches make use of coordination graphs [8–11], hierarchical task decompositions [6] or decentralized execution of joint policies [15]. As discussed in Section 3, our approach is closer to those in [14, 16].

In this paper, we frame the problem of coordination learning as a reinforcement learning (RL) problem. Unlike several of the aforementioned works, we do not assume any prior knowledge on the structure of the problem, and each agent must learn both how to complete its individual task and when and how to coordinate. Our approach takes advantage of the assumed sparse interaction in domains where the multiple agents are loosely coupled. Concretely, each agent must *learn* from experience those situations in which coordination is beneficial. We introduce a *two-layer* extension of Q-learning, in which we augment the action space of each agent with a coordination action that attempts to use information from the other agents (gathered by means of active perception) to decide the correct action. Our results show that our agents learn to coordinate when necessary in terms of the received reward and they independently choose their actions, otherwise.

The paper is organized as follows. In Section 2 we review some background material on reinforcement learning in single and multiagent settings. We describe our approach in Section 3, outlining the main differences between our approach and those in related works. In Section 4 we illustrate the application of our algorithm in several problems of different complexities and conclude in Section 5 with some final remarks.

Cite as: Learning of Coordination: Exploiting Sparse Interactions in Multiagent Systems, Francisco S. Melo, Manuela Veloso, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 773–780

2. BACKGROUND

We review some background on Markov decision processes and Markov games that we later use in describing our framework and algorithm.

2.1 Markov decision processes

A *Markov decision problem* (MDP) describes a sequential decision problem in which an agent must choose the sequence of actions that maximizes some reward-based optimization criterion. Formally, an MDP is a tuple $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathsf{P}, r, \gamma)$, where $\mathcal{X}$ represents the state-space, $\mathcal{A}$ represents the action-space, $\mathsf{P}^a(x, y)$ represents the transition probabilities from state x to state y when action a is taken and $r(x, a)$ represents the expected reward for taking action a in state x. The scalar γ is a discount factor. The agent must choose its actions so as to maximize the functional

$$V(\{A(t)\}, x) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R\big(X(t), A(t)\big) \mid X(0) = x\right],$$

where $X(t)$ represent the state of the agent/world at time t, $A(t)$ the action taken by the agent at that time instant, and $R(x, a)$ the associated random reward.[1]

A *policy* is a mapping $\pi : \mathcal{X} \times \mathcal{A} \longrightarrow [0, 1]$ such that

$$\pi(x, a) = \mathbb{P}\left[A(t) = a \mid X(t) = x\right].$$

When the action sequence $\{A(t)\}$ is generated by a policy π, we write $V^\pi(x)$ instead of $V(\{A(t)\}, x)$. For any finite MDP, there is at least one policy π^* such that

$$V^{\pi^*}(x) \geq V^\pi(x)$$

for any policy π and state x. Such policy is an *optimal policy* and the corresponding value function is compactly denoted as V^*.

The optimal value function V^* verifies Bellman optimality equation,

$$V^*(x) = \max_{a \in \mathcal{A}} \left[r(x, a) + \gamma \sum_{y \in \mathcal{X}} \mathsf{P}^a(x, y) V^*(y)\right] \qquad (1)$$

from where we can define the *optimal Q-function* as

$$Q^*(x, a) = r(x, a) + \gamma \sum_{y \in \mathcal{X}} \mathsf{P}^a(x, y) V^*(y). \qquad (2)$$

2.2 Q-learning

From (1) and (2), it is possible to write Q^* recursively as

$$Q^*(x, a) = r(x, a) + \gamma \sum_{y \in \mathcal{X}} \mathsf{P}^a(x, y) \max_{b \in \mathcal{A}} Q^*(y, b).$$

and a standard fixed-point iteration can be used to compute Q^*. On the other hand, if P and/or r are unknown, Q^* can be estimated using the *Q-learning algorithm*. Q-learning allows Q^* to be estimated from empirical data obtained from the actual system, and is defined by the update rule

$$\begin{aligned} Q_{k+1}(x, a) = {} & (1 - \alpha_k) Q_k(x, a) \\ & + \alpha_k \big[R(x, a) + \gamma \max_{b \in \mathcal{A}} Q_k(X(x, a), b)\big], \end{aligned} \qquad (3)$$

where $Q_k(x, a)$ is the kth estimate of $Q^*(x, a)$, $X(x, a)$ is a $\mathcal{X}$-valued random variable obtained according to the probabilities defined by P and $\{\alpha_k\}$ is a step-size sequence. $R(x, a)$ and $X(x, a)$ can be obtained from a generative model or from the actual system, not requiring the knowledge of either P or r. Under suitable conditions, the estimates Q_k converge to Q^* w.p.1 [3].

[1] We have that $r(x, a) = \mathbb{E}\left[R(x, a)\right]$.

2.3 Markov games

A tuple $\Gamma = \big(N, \mathcal{X}, (\mathcal{A}_k), \mathsf{P}, (r_k), \gamma\big)$ defines a *N-agent Markov game* (MG), where $\mathcal{X}$, P and γ are as in MDPs, $\mathcal{A} = \times_{k=1}^{N} \mathcal{A}_k$ is the set of *joint actions* and r_k is the expected reward function for agent $k, k = 1, \ldots, N$. The main differences between an MG and an MDP lie on the distributed action selection and distributed reward function in the former. In other words, in MGs the transition probabilities/rewards depend on the actions of *all* agents and each agent has its own reward function.

In MGs one must distinguish between an *individual policy* – a mapping π_k defined over $\mathcal{X} \times \mathcal{A}_k$ – and a joint policy – a vector $\pi = (\pi_1, \ldots, \pi_N)$ of individual policies. In multiagent settings, we refer to a deterministic policy as being a *pure policy* and a *mixed policy* otherwise. We refer to π_{-k} as a *reduced policy*, obtained from π by removing the individual policy of agent k.

As in MDPs, the purpose of each agent k is maximize the functional

$$V_k(\{A(t)\}, x) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_k\big(X(t), A(t)\big) \mid X(0) = x\right],$$

where R_k is the random reward received by agent k. V_k now depends on the actions of *all* agents, which means that the concept of "optimal policy" must be replaced by that of *equilibrium policy*.

Several approaches have been proposed in the RL literature to extend Q-learning to multiagent domains. Early approaches treated multiagent learning as several decoupled learning problems. However, this approach was shown to yield poor results in most problems, since it failed to capture the interaction between the different agents [5, 17]. Subsequent approaches addressed the problem of learning equilibrium policies in multiagent domains (*e.g.*, [4, 7, 13, 18]). However, most such approaches rely on implicit joint-observability assumptions that seldom hold in practice. In particular, most such approaches assume that each agent is able to observe (*a posteriori*) the actions taken by other agents. However, this is seldom the case in practical problems. MG-based approaches also assume that each agent is able to observe the *full state* of the game, when in many situations the agent has only a *local perception* of the state of the game.

In this paper, we present a new algorithm for Markov games that alleviates such joint-observability assumptions, by exploiting the fact that, in many problems, an agent can actually act "optimally" requiring minimum information on the other agents.

3. EXPLOITING SPARSE INTERACTION

Consider a very simple problem in which two mobile robots must navigate a common environment, each trying to reach its own goal (*e.g.*, see Fig. 1). Each robot has 4 possible actions, "Move North," "Move South," "Move East" and "Move West." Each action moves the robot in the corresponding direction with a given success probability. Furthermore, the actions of one robot do not affect the movement of the other.

It is possible to break this problem in two independent problems: each robot focuses on learning its own optimal policy and completely disregards the existence of the other robot. This separation is possible if the tasks of both robots are independent (the dynamics and rewards for each robot are independent of the state/action of the other). Each robot can thus be modeled independently as an

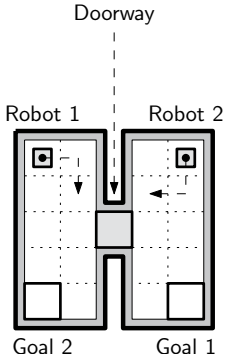


Figure 1: Simple environment where two agents need to coordinate only around the doorway.

MDP and there is no need to consider interaction between the two robots.

However, it may happen that, in some situations, the rewards of one robot do depend on the state/actions of the other robot. In our example, suppose that the doorway in Figure 1 is too narrow for both robots to pass simultaneously without some minimal damage. Therefore, a negative reward may be issued to both robots to discourage them from simultaneously trying to cross the doorway. In this situation, it would be desirable that the robots learned to coordinate. In the continuation we formalize problem such as the one just described and propose an extension of Q-learning that, using minimal extra information, is able to successfully learn how to coordinate.

3.1 Markov games with partial observability

Let $\Gamma = \big(N, \mathcal{X}, (\mathcal{A}_k), \mathsf{P}, (r_k), \gamma\big)$ be a Markov game with finite state-space $\mathcal{X}$. We assume that, at each time instant, each agent has only access to a local observation of the state of the system. In particular, we assume that the state-space $\mathcal{X}$ can be factored as $\mathcal{X} = \times_{k=1}^{N} \mathcal{X}_k$; the state at time instant t is a vector $X(t) = \big(X_1(t), \ldots, X_N(t)\big)$, where $X_k(t)$ corresponds to the "local state" as perceived by agent k. Although the actual state of the game can be inferred from the joint observations of all agents, each agent has *partial observability* in that its individual perception of the state is not enought to unambiguously determine the actual state of the game.

In line with our sparse interaction framework, we consider each r_k to be decomposable as

$$r_k(x, a) = r_k^I(x_k, a_k) + r^C(x, a),$$

where r_k^I is the *individual component* of the reward function, that depends only on the local state and action of agent k, and r^C is a *common component* of the reward function, that depends on the overall state of the game and on the actions of *all* agents. In a sense, the individual component encodes the individual goal of agent k while the common component determines the situations in which interaction must occur.[2] Still keeping in mind the sparse interaction framework, we also consider that the dynamics of the game can be decoupled in terms of the individual actions of the agents by assuming the kth component of $X(t+1)$ to depend only on $X_k(t)$ and $A_k(t)$.

[2]The common component, as considered here, is the same for *all* agents. In a more general setting, this component need not be the same for all agents, but we focus on the simpler case in which it is. We further discuss this topic in Section 5.

The Markov game thus defined is, in its essence, similar to an *interaction-driven Markov game* (IDMG) [16]. In the planning approach to IDMGs featured in [16] the states in which the agents are able to communicate/coordinate are included in the model definition. In those states, each agent shares its local state information that other agents can then use to coordinate. Since the IDMG model is assumed known, this means that each agent can consider the action that the other agent will take to plan accordingly.

Unlike the approach in [16], we do not assume any knowledge on the structure of the problem. Instead, we assume that the agents know nothing about the transition probabilities or about any of the reward functions r_k. Each agents must therefore *learn from experience* not only its "individual goal" but also in which states it should coordinate with the other agents. In [14] a somewhat similar idea was explored, in which the agents determine at execution time how to best employ communication to execute a centralized plan in a distributed fashion. Due to the lack of model knowledge in our approach, our algorithm does not rely on any previously computed plan but learns from experience those situations in which communication can actually lead to an improvement in performance.

Several other authors have explored locality of interaction in different ways. In [8, 9], the authors describe local interactions between the agents using *coordination graphs*. These coordination graphs take advantage of an additive decomposition of the joint reward function to allow the agents to act independently, whenever that does not imply a loss of optimality. In [10, 11], the authors further explore the use of coordination graphs, which are now learned from the interactions of the agents. In [6], the authors propose a hierarchical multiagent RL algorithm that takes advantage of *subtasks* that can be learned individually by the agents, allowing coordination to be addressed only at the level of the main task.

However, the approach followed here is fundamentally different from those described above in several aspects. First of all, several of the above works consider problems in which the *joint-state and joint-action* are fully observable. Although the above methods do explore locality of interaction, most still rely heavily on (often implicit) assumptions of joint-state and joint-action observability, even if only at a local level. In our case, we do not assume joint-action observability, although the algorithm does explore joint-state observability whenever possible.

A second difference is that, in general, the agents in our setting *do not share the same reward function*. This is a fundamental difference that drives the work in this paper away from standard frameworks addressing multiagent decision making in partially observable scenarios (such as Dec-MDPs or Dec-POMDPs). It also distinguishes the work in this paper from most of those surveyed above.

Finally, it is important to point out that in several of the aforementioned references, the situations in which the agents should interact/coordinate are assumed *predefined*. This is not the case in this paper, as one of the main goals of our algorithm is precisely to *learn when and how* the agents should coordinate.

3.2 Learning to coordinate

We start by considering the simple case of a 2-agent MG $\Gamma = \big(2, \mathcal{X}, (\mathcal{A}_k), \mathsf{P}, (r_k), \gamma\big)$ verifying the assumptions in the previous subsection. Notice that, if $r^C \equiv 0$, then each agent can use standard Q-learning to learn its optimal individual policy π_k^*, and the policy $\pi^* = (\pi_1^*, \pi_2^*)$ will be optimal for Γ. However, if the reward $r^C(x, a) \neq 0$ in some state x, the policy learned by each agent using Q-learning and discarding the existence of the other agent will, in general, be suboptimal. In fact, the optimal policy in that case

Algorithm 1 Learning algorithm for agent k

Initialize Q_k^* and Q_k^C;
Set $t = 0$;
while (FOREVER) **do**
  Choose $A_k(t)$ using π_e;
  if $A_k(t) =$ COORDINATE **then**
    if $ActivePercept =$ TRUE **then**
      $\hat{A}_k(t) = \pi_g(Q_k^C, X(t))$;
    else
      $\hat{A}_k(t) = \pi_g(Q_k^*, X_k(t))$;
    end if
    Sample $R_k(t)$ and $X_k(t+1)$;
    if $ActivePercept =$ TRUE **then**
      QLUpdate$\big(Q_k^C; X(t), \hat{A}_k(t), R_k(t), X_k(t+1), Q_k^*\big)$;
    end if
  else
    Sample $R_k(t)$ and $X_k(t+1)$;
  end if
  QLUpdate$\big(Q_k^*; X_k(t), A_k(t), R_k(t), X_k(t+1), Q_k^*\big)$;
  $t = t + 1$;
end while

must take into account whatever the other agent is doing.

As stated in the previous subsection, we are interested in addressing those situations in which the interaction between the two agents is *sparse*. In terms of reward function, this translates into having a sparse function r^C with a bounded influence over $\mathcal{X}$. This means that, in order to perform optimally, each agent needs only to coordinate "around" those states x for which $r(x, a) \neq 0$. This coordination will require each agent to have access to state/action information about the other agent.

To this purpose, we augment the individual action-space of each agent with one "pseudo-action," henceforth referred as the COORDINATE action. This pseudo-action consists of two steps:

- The first step of COORDINATE is an *active perception step*, in which the agent tries to determine the local state information of the other agent;
- The second step of COORDINATE is a *coordinating step*, in which the agent makes use of the local state information from the other agent (if available) to choose one of its primitive actions.

We emphasize that the active perception step of the COORDINATE action need not succeed; whether or not it actually succeeds is environment-dependent. Going back to our robot navigation example, the active perception step can consist for example on the use of a onboard camera to localize the other robot. In this case, the robot will be able to localize the other robot only when the latter is in the field-of-view of the camera (which will depend on the configuration of the environment, on the position of the other robot, etc). Another possibility consists on the use of explicit communication, in which one robot requires the other robot to divulge its location.

Going back to our algorithm, each agent k will now use Q-learning to estimate $Q_k^*(x_k, a_k)$ for all $x_k \in \mathcal{X}_k$ and all $a_k \in \mathcal{A}_k \cup \{\text{COORDINATE}\}$. It remains only to describe how the coordinating step of the COORDINATE action takes place. The role of the coordinating step is to use the local state information from the other agent (provided by the active perception step) to guide the actual choice of the actions in $\mathcal{A}_k$. To this purpose, each agent k keeps a *second Q-function* that we henceforth denote by Q_k^C. It is important to observe that Q_k^C is defined in terms of the immediate reward of agent k and the value of agent k's policy from the next instant on. However, since this policy is defined in terms of Q_k^*, this means that the values of Q_k^C at different state-action pairs are *independent*. This can be seen in the following relation

$$Q_k^C(x, a) = r_k(x, a_k) + \gamma \sum_{y_k \in \mathcal{X}_k} \mathsf{P}_k^a(x_k, y_k) \max_{b_k \in \mathcal{A}_k} Q_k^*(y_k, b_k).$$

The above relation can be used in a Q-learning-like update to estimates the value Q_k^C associated with each individual action $a_k \in \mathcal{A}_k$ in each joint state (x_1, x_2).

Our algorithm is summarized in Algorithm 1. We denoted by π_ℓ the learning policy (*i.e.*, a policy that ensures sufficient exploration, such as an ε-greedy policy) and by $\pi_g(Q, \cdot)$ the greedy policy with respect to function Q. The flag $ActivePercept$ is true if the active perception step is successful and the general instruction QLUpdate$(Q; x, a, r, y, Q')$ is equivalent to

$$Q(x, a) = (1 - \alpha)Q(x, a) + \alpha(r + \gamma \max_b Q'(y, b)).$$

Several important observations are in order. First of all, notice that the update of Q_k^C uses the estimates of Q_k^*. As seen before, this is because the individual action at the next step will depend on the values in Q_k^* and not on Q_k^C. In other words, the values in Q_k^C only determine the one-step behavior of the COORDINATE action, and therefore there is not a "direct" dependency among entries in Q_k^C corresponding to different states/actions (see Fig. 2 for an illustration). This is particularly useful since it implies that, unlike algorithms that learn directly on the joint state-action-space, our matrix Q_k^C will be *sparse* and require approximately the same computational sampling effort than that necessary to learn Q_k^*.

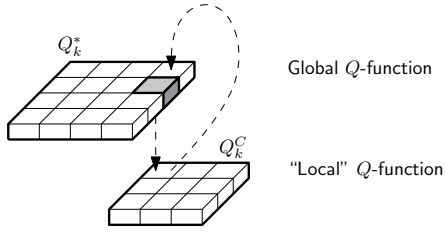


Figure 2: Illustration of the dependence between the "global" Q-function, Q_k^*, and the "local" Q-function, Q_k^C. The value of the COORDINATE action in Q_k^* at a state x depends on $Q_k^C(x)$. This, in turn depends on $Q_k^*(y)$ where y is a successor state of x. Notice that there is no inter-state dependency in Q_k^C.

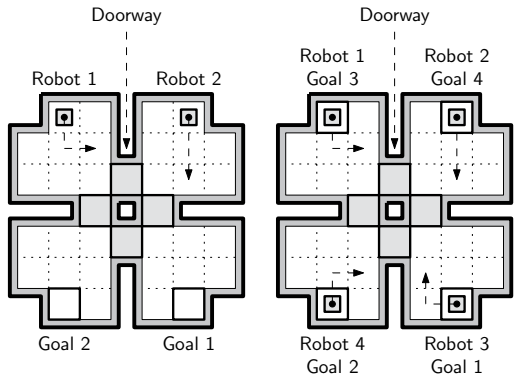
Secondly, notice that the COORDINATE action does not use any joint action information, only joint state information. This fact and the aforementioned independence of the components of Q_k^C associated with different x_k makes the learning of Q_k^C similar to a *generalized fictitious play process* [12].

Finally, we conclude by referring that, so far, we have described how to apply this algorithm for 2-agent MGs. In MGs with $N > 2$ agents and since the local state information from the other agents arises from an active perception step, we consider that each agent can only perceive the local state information concerning *one* other agent. This has two obvious implications: first of all, in problems for which coordination requires state information concerning more than two agents, the performance of our algorithm is expected to decrease. Interestingly, however, our experimental results show that the effect of this coupling in problems with more than two agents may be less noticeable than one would expect. Another implication is that we can apply Algorithm 1 *without modification*, by

disregarding the *identity* of the other agent (assuming the agent set to be homogeneous) and merely considering that the local information obtained by active perception concerns *some* other agent.[3]

4. RESULTS

To test our algorithm, we applied it in several test scenarios of different complexity (both in terms of size, number of agents and coordination requirements). The different scenarios are depicted in Figures 1, 3, and Figures 4.



(a) Env. 2 (2 robots) (b) Env. 2 (4 robots)

Figure 3: Scenarios used in the experiments.

In the first scenario, two robots must move in an environment consisting of 21 states. As depicted in Figure 1, each robot starts in one corner of the environment and must move to the opposite corner, receiving a reward of 10 for succeeding. If both robots simultaneously end up in the shaded state (the doorway) they both receive a penalty of -20 (a "miscoordination" penalty). Each robot can choose between 4 possible actions, each moving the robot in one of the 4 possible directions (North, South, East and West). These actions succeed with a 0.8 probability and fail with a 0.2 probability. The second test scenario is similar to the first test scenario, only considering a different environment (this time with 36 states). In this new environment there are 4 doorways granting a penalty to both agents if both simultaneously end up in the same doorway. In the third test scenario we placed 4 robots in the 36-state environment. In all scenarios, and in order to disencourage the excessive use of the COORDINATE action, each agent receives a penalty of -1 every time it chooses this action. Notice that the use of the "excessive" use of the COORDINATE action impacts the sparsity of Q_k^C, which is undesirable for large problems.

We ran our algorithm in each of the environments for 10^4 time steps. For comparison purposes, we also ran standard Q-learning with one robot at a time (which means that the robots never experience the miscoordination penalty) and with all robots simultaneously.[4]

[3]We remark, however, that if we consider more general active perception processes – *i.e.*, the agent is actually able to perceive the state of all other agents – the algorithm can be trivially modified to address coordination of any number of agents, with an obvious tradeoff in terms of the memory requirements of the algorithm.

[4]Due to the general lack of knowledge about the global state information, the agents in the two algorithms used for comparison are *independent learners* and not *joint action learners* in the sense of [5].

Table 1: Performance of the different algorithms in the 3 test scenarios by applying the learned policy during 100 time-step episodes. The values presented were averaged over all the robots.

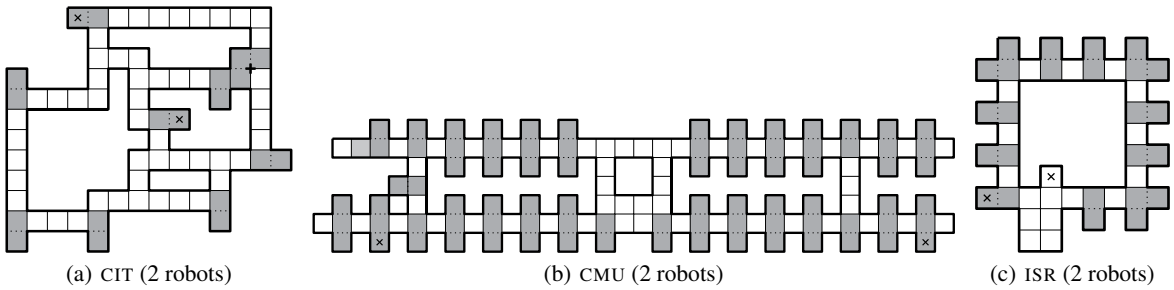
	Learning	Total disc. reward
Env. 1 (2R)	Indiv.	119.5 ± 12.5
	Non-coop.	119.3 ± 12.2
	Coop.	126.3 ± 10.1
Env. 2 (2R)	Indiv.	103.3 ± 26.3
	Non-coop.	102.3 ± 11.4
	Coop.	110.9 ± 9.6
Env. 2 (4R)	Indiv.	102.1 ± 13.6
	Non-coop.	104.1 ± 12.4
	Coop.	111.2 ± 9.7

We then evaluated each of the learned policies in the several environments, for 100 time step episodes. We point out that, for evaluation purposes, all learned policies (Coord., Indiv. and Uncoord.) were evaluated *in the original problem*, *i.e.*, with two/four robots moving simultaneously in the environment. For averaging purposes, we performed $1,000$ independent Monte-Carlo trials. Table 2 presents the results obtained for the single-robot Q-learning (Indiv.), the multi-robot Q-learning (Non-coop.) and our algorithm (Coop.). The results in Table 1 show that our algorithm outperforms both other methods. The superior performance of our algorithm is perceivable in two distinct ways: (i) the average performance observed in all environments is superior to that of the other algorithms; and (ii) the *reliability* of the observer performance is much superior in our algorithm. The latter aspect is observable by noticing the larger variance in the other algorithms. This means that the "ability" of the policies computed by the other methods to avoid miscoordinations greatly depends on chance (a lucky run may yield a large reward while an unlucky run may bring a large penalty).

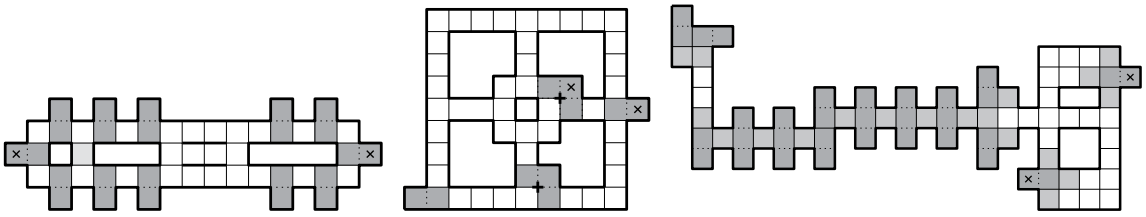
Table 2: Performance of the different algorithms in the 3 test scenarios, now in terms of time-to-goal and number of miscoordinations.

	Learning	Time to goal	Miscoord.
Env. 1 (2R)	Indiv.	10.02 ± 1.57	0.40 ± 0.52
	Non-coop.	10.02 ± 1.58	0.41 ± 0.54
	Coop.	9.94 ± 1.57	0.00 ± 0.00
Env. 2 (2R)	Indiv.	12.45 ± 1.68	0.12 ± 0.33
	Non-coop.	12.45 ± 1.77	0.12 ± 0.33
	Coop.	12.51 ± 1.72	0.00 ± 0.00
Env. 2 (4R)	Indiv.	12.46 ± 1.75	0.47 ± 0.59
	Non-coop.	12.49 ± 1.74	0.49 ± 0.59
	Coop.	12.49 ± 1.77	0.00 ± 0.00

To gain a clearer understanding of the results in Table 1 and understand how miscoordinations impact the performance of the robots, let us consider the time to goal and the average number of miscoordinations in each of the three test scenarios, found in Table 2). Noticeably, our algorithm exhibits a similar performance in terms of time to goal, which means that the inclusion of the COORDINATE action does not significantly affect the time that the robots take to reach the corresponding goal. Another interesting aspect to observe is that, in all environments, our algorithm exhibits *no mis-*



(a) CIT (2 robots) (b) CMU (2 robots) (c) ISR (2 robots)



(d) MIT (2 robots) (e) PENTAGON (2 robots) (f) SUNY (2 robots)

coordinations. This means that the robots are able to choose their path so as to completely avoid the coordination penalties, and this without significant cost in terms of the average time to goal.

Another interesting aspect to notice is that the existence of 4 robots in the 36-state scenario *does not seem to impact negatively the performance of the team*. This is an interesting indicator that, even if our algorithm only considers coordination between pairs of robots, this seems to be sufficient to ensure coordination at least in some problems.

We then tested the algorithm in several large navigation scenarios used as benchmarks in the POMDP literature. These scenarios have different sizes ranging from 43 states in the ISRenvironment (corresponding to $1,849$ total joint states) to 133 in the CMUenvironment (corresponding to a total of $17,689$ joints states). The shaded cells correspond to doorways, and the agents get a penalty if both stand in any of the shaded cells simultaneously. In each of this scenario, the crosses indicate the starting state for the two robots. The goal for each robot is to move to the starting state of the other robot. As before, each agent receives a penalty of -1 every time it chooses this action.

We ran our algorithm in each of the larger environments for 10^5 time steps. As before, for comparison purposes, we also ran standard Q-learning with one robot at a time and with all robots simultaneously. We then evaluated each of the learned policies in the several environments, for 100 time step episodes. We ran $1,000$ independent Monte-Carlo trials and present the average results in Table 3.

One fundamental difference between these larger environments and those in Figures 1 and 3 is that interaction in these environment is much sparser than that in the smaller environments. In other words, in some of the test scenarios (*e.g.*, CIT, ISR), the environment and the initial/goal positions for both robots are such that explicit coordination actions are not really necessary to avoid miscoodinations. This means that both the individualistic Q-learners and the non-cooperative Q-learners should be able to attain a good performance. On the other hand, these scenarios allow us to perceive whether, in situations where no coordination is necessary, the agents are indeed able to learn to act independently.

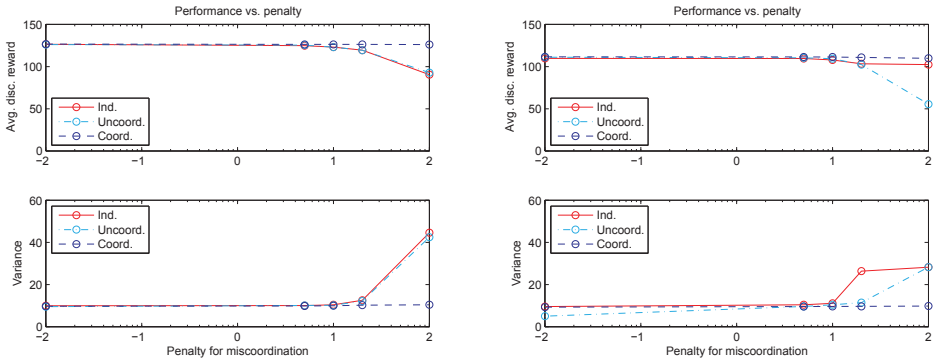
As seen in Table 3, all methods attain similar performance in all large environments. This is important because it indicates that no coordination actions are used by our algorithm (or, otherwise, the total discounted reward would be inferior to that of the other methods). This means that, as intended, the agents are able learn to coordinate *only when coordination is necessary to attain good performance*.

Table 3: Performance of the different algorithms in the large test scenarios, obtained with the learned policy during 100 time-step episodes. The values presented were averaged over all the robots.

	Learning	Total disc. reward
CIT (2R)	Indiv.	111.2 ± 9.9
	Non-coop.	111.1 ± 9.9
	Coop.	111.3 ± 9.9
CMU (2R)	Indiv.	27.4 ± 5.8
	Non-coop.	27.1 ± 4.6
	Coop.	27.5 ± 4.6
ISR (2R)	Indiv.	143.5 ± 10.0
	Non-coop.	143.4 ± 9.8
	Coop.	143.6 ± 9.9
MIT (2R)	Indiv.	66.7 ± 7.9
	Non-coop.	66.6 ± 7.9
	Coop.	66.7 ± 7.9
PENTAGON (2R)	Indiv.	161.1 ± 12.3
	Non-coop.	160.8 ± 12.6
	Coop.	163.0 ± 9.1
SUNY (2R)	Indiv.	111.2 ± 9.7
	Non-coop.	111.3 ± 9.8
	Coop.	111.4 ± 9.8

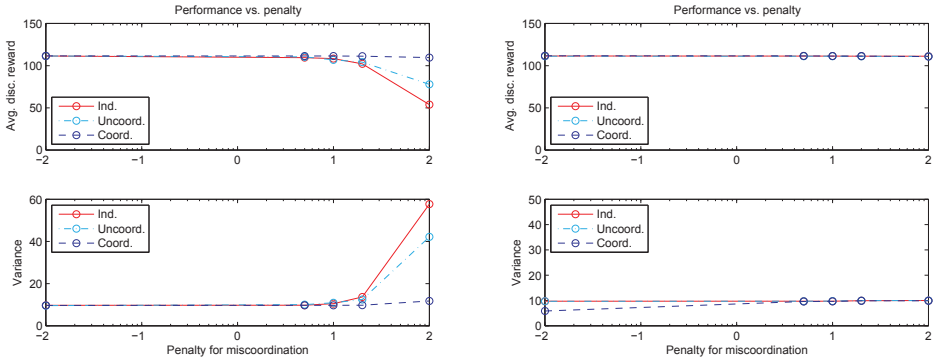
Finally, we also analyzed how the increasing penalties for miscoordination affect the performance of the robots in all 9 scenarios. To that purpose, we ran all three learning algorithms for different values of the miscoordination penalty and evaluated the obtained policies. Figure 4 depicts the results obtained as we vary the miscoordination penalty from 0 to -100.

The remarkable aspect to keep in mind is that, for a penalty of 0, *all methods are optimal*, since the tasks of the different robots are completely decoupled. Notice, however, that the performance



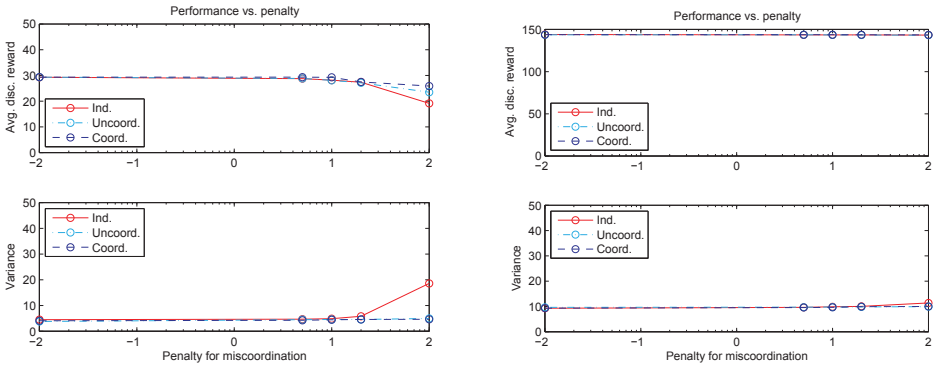
(a) Environment 1 (2 robots)

(b) Environment 2 (2 robots)



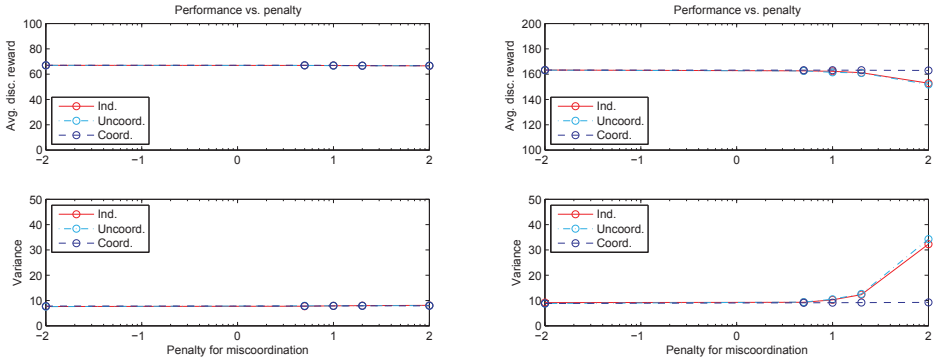
(c) Environment 2 (4 robots)

(d) CIT (2 robots)



(e) CMU (2 robots)

(f) ISR (2 robots)



(g) MIT (2 robots)

(h) PENTAGON (2 robots)

Figure 4: Evolution of the performance of the algorithm as the penalty for miscoordination increases, both in terms of average reward and variance, for several of the test scenarios. The numbers in the xx-axis correspond to powers of 10. Also, the value in the leftmost part of the plots corresponds to a penalty of 0 (and not 10^{-2}).

of our algorithm *remains approximately constant* as the miscoordination penalty is increased. This means that, in the above scenarios, our method is actually able to attain *optimal performance*, by using minimum information concerning the state of the other agents in the environment. Notice also that, in several environments, the *reliability* of the performance of the individualistic and non-coordinated policies (indicated by the variance) greatly decreases as the miscoordination penalty increases,

Notice also that, as expected, the non-cooperative Q-learning algorithm is somewhat able to surpass the individualistic Q-learning in several scenarios. This is particularly evident by observing the performance as the miscoordination penalty increases, and can be interpreted as having the non-cooperative agents to act with "increasing caution" due to the penalties experienced during learning.

5. DISCUSSION

In this paper we presented a learning algorithm that explores the sparse interaction observed in many multiagent domains. This is done by means of a "pseudo-action," the COORDINATE action, that allows each agent to use local state information from other agents to choose its actions so as to avoid miscoordination problems. Our experimental results show that our algorithm is successfully able to coordinate in several scenarios with multiple agents.

Our results also launch several interesting questions to be addressed in future work. One first interest issue to be explored is concerned with the dependence of the performance of the algorithm with the cost of the COORDINATE action. In fact, by increasing the cost of the COORDINATE action, the agents must learn a trade-off between the benefits arising from good coordination and the cost of that same coordination. This bares a close relation with the problem of exchanging reward by information arising in the POMDP literature [1] and it would be interesting to bring analyze how the problem addressed here extends to situations where further partial state observability is considered.

Another interesting aspect to be explored is related with the performance guarantees of the algorithm. As remarked in Section 3, the parallel learning process taking place when the agent executes the COORDINATE action during learning bares significant resemblances with a generalized fictitious play behavior. It would be interesting to perceive how the convergence guarantees of such processes can be translated to our particular algorithm. Another related issue that should be explored in the future is concerned with the limitations of the algorithm. In the particular scenarios used in this paper, the algorithm exhibited a sound performance. However, and since no formal guarantees exist at this point on the performance of the algorithm, it would be interesting to understand the main limitations associated with the algorithm. The scenarios used in this paper required only very *localized coordination*. We anticipate that problems in which coordination is required at a more global level may cause the performance of the algorithm to decrease.

Finally, in view of the completely "independent" way by which the agents learn, it is reasonable to expect that the algorithm should still be useful if the common reward function r^C is not the same for all agents, although it still depends on the actions of all agents. This means that the algorithm could be applied to a broader class of MGs than that explored here. Another interesting aspect to explore is the consideration of scenarios in which the agents are not completely independent (in terms of dynamics) but exhibit some weak coupling (for example in the states where interaction occurs).

Acknowledgements

This research was partially sponsored by the Portuguese Fundação para a Ciência e a Tecnologia under the Carnegie Mellon-Portugal Program and the Information and Communications Technologies Institute (ICTI), www.icti.cmu.edu. The views and conclusions contained in this document are those of the authors only.

6. REFERENCES

[1] D. Aberdeen. A (revised) survey of approximate methods for solving partially observable Markov decision processes. Technical report, National ICT Australia, Canberra, Australia, 2003.

[2] D. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.

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

[4] M. Bowling and M. Veloso. Rational and convergent learning in stochastic games. In *Proc. Int. Joint Conf. Artificial Intelligence*, pages 1021–1026, 2001.

[5] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proc. Nat. Conf. Artificial Intelligence*, pages 746–752, 1998.

[6] M. Ghavamzadeh, S. Mahadevan, and R. Makar. Hierarchical multi-agent reinforcement learning. *J. Autonomous Agents and Multi-Agent Systems*, 13(2):197–229, 2006.

[7] A. Greenwald and K. Hall. Correlated Q-learning. In *Proc. Int. Conf. Machine Learning*, pages 242–249, 2003.

[8] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored MDPs. In *Proc. Adv. Neural Information Proc. Systems 14*, pages 1523–1530, 2001.

[9] C. Guestrin, S. Venkataraman, and D. Koller. Context specific multiagent coordination and planning with factored MDPs. In *Proc. National Conf. Artificial Intelligence*, pages 253–259, 2002.

[10] J. Kok, P. Hoen, B. Bakker, and N. Vlassis. Utile coordination: Learning interdependencies among cooperative agents. In *Proc. Symp. on Computational Intelligence and Games*, pages 29–36, 2005.

[11] J. Kok and N. Vlassis. Sparse cooperative Q-learning. In *Proc. Int. Conf. Machine Learning*, pages 61–68, 2004.

[12] D. Leslie and E. Collins. Generalised weakened fictitious play. *Games and Economic Behavior*, 56:285–298, 2006.

[13] M. Littman. Value-function reinforcement learning in Markov games. *J. Cognitive Systems Research*, 2(1):55–66, 2001.

[14] M. Roth, R. Simmons, and M. Veloso. Reasoning about joint beliefs for execution-time communication decisions. In *Proc. Int. Joint Conf. Auton. Agents and Multi-Agent Systems*, pages 1098–1105, 2005.

[15] M. Roth, R. Simmons, and M. Veloso. Exploiting factored representations for decentralized execution in multi-agent teams. In *Proc. AAMAS*, pages 469–475, 2007.

[16] M. Spaan and F. Melo. Local interactions in decentralized multiagent planning under uncertainty. In *Proc. Int. Joint Conf. Auton. Agents and Multiagent Systems*, pages 525–532, 2008.

[17] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Readings in Agents*, pages 487–494, 1997.

[18] X. Wang and T. Sandholm. Reinforcement learning to play an optimal Nash equilibrium in team Markov games. In *Proc. NIPS 15*, pages 1571–1578, 2002.