Rehearsal Based Multi-agent Reinforcment Learning of Decentralized Plans

Landon Kraemer
The University of Southern Mississippi
118 College Dr. #5106
Hattiesburg, MS 39402
Landon.Kraemer@eagles.usm.edu

Bikramjit Banerjee
The University of Southern Mississippi
118 College Dr. #5106
Hattiesburg, MS 39402
Bikramjit.Banerjee@usm.edu

ABSTRACT

Decentralized partially-observable Markov decision processes (Dec-POMDPs) are a powerful tool for modeling multi-agent planning and decision-making under uncertainty. Prevalent Dec-POMDP solution techniques require centralized computation given full knowledge of the underlying model. Reinforcement learning (RL) based approaches have been recently proposed for distributed solution of Dec-POMDPs without full prior knowledge of the model, but these methods assume that conditions during learning and policy execution are identical. In practical scenarios this may not necessarily be the case, and agents may have difficulty learning under unnecessary constraints. We propose a novel RL approach in which agents are allowed to *rehearse* with information that will not be available during policy execution. The key is for the agents to learn policies that do not explicitly rely on this information. We show experimentally that incorporating such information can ease the difficulties faced by non-rehearsal-based learners, and demonstrate fast, (near) optimal performance on many existing benchmark Dec-POMDP problems. We also propose a new benchmark that is less abstract than existing problems and is designed to be particularly challenging to RL-based solvers, as a target for current and future research on RL solutions to Dec-POMDPs.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent Systems*; I.2.8 [**Problem Solving, Control Methods and Search**]:

General Terms

Algorithms, Experimentation, Performance

Keywords

Multi-agent reinforcement learning, Dec-POMDPs

INTRODUCTION

Decentralized partially observable Markov decision processes (Dec-POMDPs) offer a powerful model of decentralized decision making under uncertainty and incomplete knowledge. Many exact and approximate solution techniques have been developed for Dec-POMDPs [9, 5, 7], but these approaches are not scalable because the underlying problem is provably NEXP-complete [3].

Recently, reinforcement learning (RL) techniques, particularly multi-agent reinforcement learning (MARL), have been applied [10, 2] to overcome other limitations of the Dec-POMDP solvers, viz., that they are centralized and model-based. That is, these traditional Dec-POMDP solvers compute the set of prescribed behaviors for agents centrally, and assume that a comprehensive set of model parameters are already available. While RL distributes the policy computation problem among the agents themselves, it essentially solves a more difficult problem, because it does not assume the model parameters are known a-priori. The hardness of the underlying problem translates to significant sample complexity for RL solvers as well.

One common feature of the existing RL solvers is that the learning agents subject themselves to the same constraints that they would encounter when executing the learned policies. In particular, agents assume that the environment states are hidden and the other agents' actions are invisible, in addition to the other agents' observations being hidden too. We argue that in many practical scenarios, it may actually be easy to allow learning agents to observe some otherwise hidden information *while they are learning*. We view such learning as a *rehearsal* – a phase where agents are allowed to access information that will not be available when executing their learned policies. While this additional information can facilitate the learning during rehearsal, agents must learn policies that can indeed be executed in the Dec-POMDP (i.e., without relying on this additional information). Thus agents must ultimately wean their policies off of any reliance on this information. This creates a principled incentive for agents to explore actions that will help them achieve this goal. Based on these ideas, we present a new approach to RL for Dec-POMDPs – *R*einforcement *L*earning *a*s a *R*ehearsal or RLaR, including a new exploration strategy. Our experiments show that this new approach can nearly optimally solve most existing benchmark Dec-POMDP problems with a low sample complexity. We raise the bar for RL solvers by proposing a new benchmark problem that is particularly challenging to RL solvers – robot alignment. We hope this problem will spur research to address the difficulties that it poses, leading to more competent RL solvers of Dec-POMDPs in the future.

BACKGROUND

Appears in *The Eighth Annual Workshop on Multiagent Sequential Decision-Making Under Uncertainty* (*MSDM-2013*), held in conjunction with *AAMAS*, May 2013, St. Paul, Minnesota, USA.

Decentralized POMDPs

We can define a Dec-POMDP as a tuple $\langle n, S, A, P, R, \Omega, O\rangle$, where:

- n is the number of agents in the domain.
- S is a finite set of (unobservable) environment states.
- $A = \times_i A_i$ is a set of joint actions, where A_i is the set of individual actions that agent i can perform.
- $P(s'|s, \vec{a})$ gives the probability of transitioning to state $s' \in S$ when joint action $\vec{a} \in A$ is taken in state $s \in S$.
- $R(s, \vec{a})$ gives the immediate reward the agents receive upon executing action $\vec{a} \in A$ in state $s \in S$.
- $\Omega = \times_i \Omega_i$ is the set of joint observations, where Ω_i is the finite set of individual observations that agent i can receive from the environment.
- $O(\vec{\omega}|s', \vec{a})$ gives the probability of the agents jointly observing $\vec{\omega} \in \Omega$ if the current state is $s' \in S$ and the previous joint action was $\vec{a} \in A$.

Additionally, for finite horizon problems, a horizon T is given that specifies how many steps of interaction the agents are going to have with the environment and each other. The objective in such problems is to compute a set of decision functions or *policies* – one for each agent – that maps the history of action-observations of each agent to its best next action, such that the joint behavior (note the transition, reward, and observation functions depend on *joint* actions) over T steps optimizes the total reward obtained by the team.

In Dec-POMDPs, it is generally assumed that agents cannot communicate their observations and actions to each other. These constraints are often present in real world scenarios, where communication may be expensive or unreliable. Consider, a scenario in which a team of robots must coordinate to search a disaster area for survivors. In such a task, robots may need to disperse to efficiently cover the area and also may need to travel deep underneath rubble, both of which could interfere with wireless communication.

Reinforcement Learning for Dec-POMDPs

Reinforcement Learning (RL) is a family of techniques applied normally to MDPs [8] $\langle S, A, P, R\rangle$ (i.e., Dec-POMDPs with full observability and single agent). When states are visible, the task of an RL agent in a horizon T problem is to learn a *non-stationary* policy $\pi : S \times t \mapsto A$ that maximizes the sum of current and future rewards from any state s, given by,

$$V^\pi(s^0, t) = E_P[R(s^0, \pi(s^0, t)) + R(s^1, \pi(s^1, t+1)) + \ldots + R(s^{T-t}, \pi(s^{T-t}, T))]$$

where $s^0, s^1, \ldots s^{T-t}$ are successive samplings from the distribution P following the Markov chain with policy π. A popular RL algorithm for such problems is Q-learning, which maintains an action-quality value function Q given by

$$Q(s, t, a) = R(s, a) + \max_\pi \sum_{s'} P(s'|s, a) V^\pi(s', t+1)$$

This quality value stands for the sum of rewards obtained when the agent starts from state s at step t, executes action a, and follows the optimal policy thereafter. These Q-values

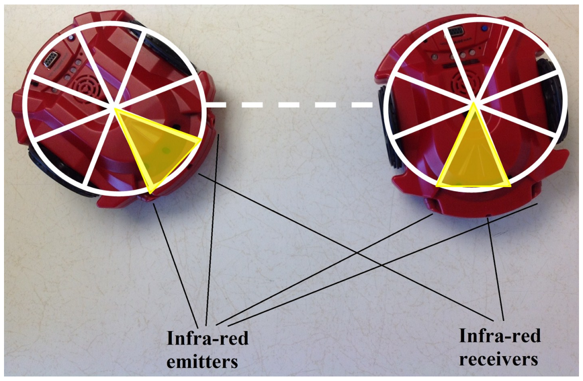

Figure 1: The Scribbler 2 robots featured in the robot alignment problem.

can be learned in model-free ways, where no prior knowledge of R, P is required.

Multi-agent reinforcement learning (MARL) has recently been applied to Dec-POMDPs [10, 2], where a quality function $Q(h, a)$ is learned, mapping the history of an agent's own past actions and observations, h, and next actions (a) to real values. While [10] assumes perfect communication among teammates in a special class of Dec-POMDPs (called ND-POMDPs), we assume indirect and partial communication among the agents, via a third party observer (but only while learning). Whereas the approach proposed in [2] has a large sample complexity, our results are all based on orders of magnitude fewer episodes in each setting. Furthermore, [2] proposes a turn taking learning approach where the non-learner's experience is wasted, whereas we propose a concurrent learning algorithm.

MOTIVATING DOMAIN

Many benchmark problems have been developed for evaluation of Dec-POMDP solvers in the past [6]. The majority of these problems were developed to illustrate concepts pertaining to methods which *compute* solutions given the full Dec-POMDP model. Furthermore, these benchmarks tend to be rather abstract. We introduce a new, more concrete, Dec-POMDP benchmark problem that illustrates the difficulties faced by *learning-based* Dec-POMDP solvers.

Figure 1 shows two Scribbler 2 robots, each possessing two infra-red (IR) emitters and one IR receiver on their front faces – marked by the shaded sector as shown. The robots can rotate by some angle, emit or receive IR signals, and can also locomote. The IR system can be used for communication between robots, but only if the robots are facing each other.

A task that is often required in robotics is for robots to get aligned, i.e., face each other. The S2 robots, for instance, need to be aligned to communicate with IR. Not only is it important that the robots become aligned, but they must also *know* that they are aligned, since alignment is usually a precursor to some other behavior. This problem is more concrete than most Dec-POMDP benchmark problems because a computed policy can be readily implemented on real S2 robots.

The Dec-POMDP Model

We model this problem using the Dec-POMDP framework as follows. The heading space of each agent is discretized into slices as shown in Figure 1. The state space of the Dec-POMDP model S is simply all possible combinations of joint headings (slices). It is assumed there is exactly one state, say a_0b_0 (when both the shaded sectors touch the dotted line in Figure 1) in which agents are aligned.

Each agent is capable of four actions: A_i={*done*, *no-op*, *emit-IR*, *turn-left*}. Observation sets are Ω_i={*IR*, *no-IR*}. An agent observes *IR* only if the other agent emitted IR and they are in state a_0b_0, otherwise it observes *no-IR*. However, if both agents emit IR in state a_0b_0, neither observes IR due to the destructive interference of the IR waves. The transition function encodes state transitions when at least one agent turns-left, otherwise the state remains unchanged. Agents get maximum reward, $R_{\max}$, if they jointly execute *done* in state a_0b_0. If agents execute the *done* action in any state other than a_0b_0, or if only one agent executes *done*, the agents receive $-R_{max}$. All other actions cost -1. The reward structure we place on the *done* action encourages agents to understand that they are aligned, and heavily penalizes incorrectly assuming they are aligned. Whenever an agent executes the *done* action, we assume that the state resets (i.e. a new state is chosen randomly from a uniform distribution).

Difficulties for RL

Because P, O are deterministic in the above problem, it is easily solved by traditional Dec-POMDP solvers, especially if the number of joint heading sectors is small. However, this problem embodies one of the most challenging scenarios for RL, viz., *combination lock*. A combination lock represents a scenario where an agent must execute a specific sequence of actions (a *combination*) to reach a desirable state or unlock high rewards, and the reward structure is such that the agent is not naturally guided to this state by exploring greedily. In a multi-agent scenario this is further compounded because the agents must execute a *joint* combination. With independent random exploration, it would be extremely difficult for them to hit the right joint combination even once, let alone learn good Q estimates by repeatedly reaching the goal.

In the robot alignment problem, the agents must *reach* a particular goal state (i.e. a_0b_0), *probe* to understand that they have reached the goal, and then *coordinate* to signal that they are aligned. This constitutes a combination lock of significant proportions. Contrast this with problems such as DecTiger and Box Pushing. In DecTiger, agents do not need to reach a particular state and are only tasked with probing to understand which door conceals the tiger, and coordinating to open a door. In Box Pushing, on the other hand, agents do not need to probe to receive observations and are tasked with navigating the state space and coordinating to push boxes.

One considerable source of difficulty in the robot alignment problem is that agents must coordinate in order to receive informative observations about the state. If neither agent emits IR or if both agents emit IR, the agents will not observe IR, regardless of the state. Thus, at least two steps are required for both agents to be able to determine if a particular state is the goal, with a different agent emitting IR in each step. Contrast this with DecTiger. In DecTiger, agents must also coordinate to gather information about the state in DecTiger; however, in DecTiger, to gain information both agents execute the same action, *listen*, and *both* receive information. This is not the case, in robot alignment as there are two different roles for information gathering, the IR-emitter and the receiver, with the emitter receiving no information. Also, the reward structure of DecTiger is such that agents will automatically prefer *listen* when they are unsure of the state because it is a relatively safe action. In robot alignment, however, agents are not guided by rewards to prefer any action involved in information gathering, much less to coordinate with the other agent.

Just as the immediate rewards do not guide the agents to engage in information gathering, they also do not induce a non-uniform preference over the state space. Agents will receive the same immediate reward executing *no-op* every step as they will executing *turn left* every step unless *done* is executed properly. Thus, in order to learn that becoming aligned is worthwhile, agents must first become aligned, be able to recognize that they are aligned, and then signal that they are aligned.

REINFORCEMENT LEARNING AS A REHEARSAL

RL has previously been applied in [10, 2] to enable agents to learn mappings from local action-observation histories to next actions. While their assumptions about local information differ - Banerjee et. al assume that agents only observe their own actions and observations, where Zhang and Lesser assume that agents observe those of their team mates as well - both methods assume that only local information is available during learning. That is, they both assume that the learning must occur under execution conditions. However, we argue this assumption is not always necessary and may make learning unnecessarily difficult.

There are many scenarios for which the training conditions need not be as strict as execution conditions. Consider, for example, the multi-robot rescue scenario mentioned previously. If those robots were trained in a simulator, rather than in the field, they could easily access information that would be hidden in a true rescue scenario. Consider also the robot alignment problem. Since the dynamics of the alignment problem do not depend on the environment, robots could easily be trained in a laboratory setting, where a computer connected to an overhead camera could relay states and others' actions to the robots. Note that providing robots with this same information in a typical execution environment would be unwieldy as some sort of third robot with a camera would have to follow the robots around, requiring more coordination than the original problem. Clearly, then, while constraints on execution conditions may be justified, these constraints need not always be applied to the training environment.

To this end, we explicitly break up a problem into distinct "learning" and "execution" phases. We treat the MARL problem as a rehearsal *before* a final stage performance, so to speak. The inspiration comes from real life; while actors train together to produce a coordinated performance on stage, they do not necessarily subject themselves to the same constraints during rehearsal. For instance, actors routinely take breaks, use prompters, receive feedback, practice separately, etc. during rehearsals, that are not available/doable during the final stage performance. Similarly in MARL, while the agents must learn distributed policies that can be executed in the target task (the stage of performance), they do not need to be constrained to the exact setting of the target task while learning.

In this paper, we assume that agents rehearse under the supervision of a third party observer that can convey to them the hidden state *when they are learning/rehearsing, but not when they are executing the learned policies.* This observer also tells the agents what the other agents' last actions were. However, this observer cannot observe the agents' observations and hence cannot cross-communicate them at any time. We call the extra information available to a learner during rehearsal, $(s \in S, a_-)$ - i.e, the hidden state and the others' actions - the *rehearsal features. The key challenge is that agents still must learn policies that will not rely on these external communications, because the rehearsal features will not be visible at the time of executing the policies.* Therefore, the policies returned by our approach must be in the same language as those returned by the Dec-POMDP solvers. We call our algorithm (described in the next section) *R*einforcement *L*earning *a*s a *R*ehearsal, or RLaR.

The RLaR Algorithm

An immediate benefit of providing agents with the state, s, and joint action, $\vec{a}$, is that agents can maintain their own estimates of the transition function $\hat{P}(s'|s,\vec{a})$, the reward function $\hat{R}(s,\vec{a})$, the initial distribution over states $b_0 \in \Delta S$, and an *individual* observation probability function $\hat{O}_i(\omega_i|s',\vec{a})$. That is, the agents can maintain internal models of the environment's dynamics.

More importantly, this enables the agents to treat the problem as the fully-observable MDP $\langle S, A, \hat{P}, \hat{R}\rangle$, and learn an MDP policy via action-quality values given by

$$Q(s,t,\vec{a}) = \hat{R}(s,\vec{a}) + \sum_{s'\in S} \hat{P}(s'|s,\vec{a}) \max_{\vec{a'}\in A} Q(s',t+1,\vec{a'}) \quad (1)$$

Obviously, this MDP policy will not be useful during policy execution as it assumes nearly full observability. Instead, this policy could be a useful starting point – an idea studied before as *transfer learning.* In the robot alignment problem, having learned the MDP policy, agents will have an understanding of the mechanics of aligning themselves; however, they will still have no understanding of how to coordinate with each other to detect alignment when the third party observer disappears. But the MDP policy tells the agents that jointly executing *done* in state $s = a_0b_0$ is desirable, and as a result, they will have the incentive to learn how to detect alignment when s is not available. In general, RLaR agents will have an incentive to learn to predict the rehearsal features (s, a_-), which creates a new, principled exploration strategy for such learners, that is unique to partially observable domains. We shall expand on this exploration strategy in the next section.

We break the rehearsal phase into 2 successive stages, where the MDP policy is learned as above in the first stage. In the second stage, agents learn joint action values given s and the current observable history h_t, i.e., $Q(s,h_t,\vec{a})$, while also learning the correlation between the (ultimately forbidden) rehearsal features and the observable history. The MDP Q values are used to initialize the $Q(s,h_t,\vec{a})$ values as

$$Q(s,h_t,\vec{a}) \leftarrow Q(s,t,\vec{a}).$$

Then the $Q(s,h_t,\vec{a})$ are learned as

$$Q(s,h_t,\vec{a}) = \hat{R}(s,\vec{a}) + \sum_{\omega\in\Omega_i} \tilde{P}(\omega|s,\vec{a}) \max_{a'\in A_i} Q(h_{t+1},a'), \quad (2)$$

where $\vec{a} = \langle a, a_-\rangle$ with a_- representing the other agents' actions, h_{t+1} is the concatenation of h_t with (a,ω), i.e. (h_t,a,ω), and $Q(h_{t+1},a')$ is explained later. $\tilde{P}(\omega|s,\vec{a})$ is calculated using the agent's internal model parameters via

$$\tilde{P}(\omega|s,\vec{a}) = \alpha \sum_{s'\in S} \hat{P}(s'|s,\vec{a})\hat{O}_i(\omega|s',\vec{a}) \quad (3)$$

where α is a normalization factor.

$Q(s,h_t,\vec{a})$ gives the immediate reward agents will receive when executing $\vec{a}$ in state s plus the future reward agents can expect when the next state s' and other agent's next action a'_- *cannot be observed.* This future reward is encapsulated in the quantity $Q(h_{t+1},a')$ used in the update rule of equation 2, and maintained by the learners alongside $Q(s,h_t,\vec{a})$ values. $Q(h_t,a)$ values are the real objective of RLaR since they are independent of rehearsal features and can be used during the execution phase.

As with RL algorithms used for Dec-POMDPs, $Q(h_t,a)$ gives the expected reward of executing individual action a having observed individual action-observation history, h_t. However, instead of learning these values, we estimate them based on the learned $Q(s,h_t,\vec{a})$ values as follows:

$$Q(h_t,a) = \sum_{s\in S}\sum_{a_-\in A_-} P(s,a_-|h_t)Q(s,h_t,\vec{a}). \quad (4)$$

$P(s,a_-|h_t)$ gives the probability (or the learner's *belief*) that after observing h_t, the state will be s and the other agent will execute a_-. This is essentially the agent's prediction of the rehearsal features based on observable history. Agents could estimate this probability directly via sampling; however, we note that $P(s,a_-|h_t) = P(a_-|h_t,s)P(s|h_t)$. Therefore, agents can estimate $P(a_-|h_t,s)$ by sampling, but *propagate* the belief $P(s|h_t)$ using

$$P(s'|h_t) = \sum_{s\in S}\sum_{a_-\in A_-} \hat{O}_i(\omega|s',a)\hat{P}(s'|s,\vec{a})P(s,a_-|h_{t-1}) \quad (5)$$

where $h_t = (h_{t-1},a,\omega)$. This allows an agent to incorporate its model estimate into the calculation, which can improve belief estimates because while $P(a_-|h_t,s)$ changes as the agents learn, the underlying internal model tracks stationary distributions.

The RLaR algorithm is outlined in Algorithm 1. Each agent executes a separate instance of RLaR, but concurrently with other agents. Since the Dec-POMDP framework assumes that actions are synchronized, calls to EXECUTEACTION represent synchronization points. EXECUTEACTION returns the next state s', the last joint action $\vec{a}$, the agent's own observation ω, and the reward r. This feedback includes rehearsal features, and comes from the third party observer. Upon receiving this feedback, each agent updates its model $\langle S, A, \hat{P}, \hat{R}, \Omega_i\rangle$ as well as its model of the other agent's action $P(a_-|h_t,s)$. In the first stage agents do not require a model of other agent's actions; however, we have found it useful to sample $P(a_-|t,s)$ to replace missing $P(a_-|h_t,s)$ values in stage 2. UPDATEQ$(s,t,\vec{a})$ implements equation 1, while UPDATEQ$(s,h,\vec{a})$ implements equations 3, 2, 4, 5 in that order.

Action Selection

One common method of action selection in reinforcement learning is the ϵ-greedy approach [8], in which an agent

$$Q^*_i(h_i, a_i) = \sum_{s \in S, a_{-i} \in A_{-i}, h_{-i} \in H_{-i}} I(h_{-i}, a_{-i}) P(s, h_{-i}|h_i) \cdot \left[R(s, \vec{a}) + \sum_{\omega_i \in \Omega_i} P(\omega_i|s, \vec{a}) \max_{b \in A_i} Q^*_i((h_i, a_i, \omega_i), b) \right] \tag{6}$$

$$P(s', (h_{-i}, a_{-i}, \omega_{-i})|(h_i, a_i, \omega_i)) = I(h_{-i}, a_{-i}) \sum_{s \in S} P(s', \omega_{-i}|s, \vec{a}, \omega_i) P(s, h_{-i}|h_i) \tag{7}$$

Algorithm 1 RLaR(*mdp_episodes*, *total_episodes*)

```
 1: for $m = 1 \dots total\_episodes$ do
 2:   $s \leftarrow$ GetInitialState()
 3:   $h \leftarrow \emptyset$
 4:   for $t = 1 \dots T$ do
 5:     if $m \leq mdp\_episodes$ then
 6:       $a \leftarrow$ SelectActionMDP()
 7:       $(s', \vec{a}, \omega, r) \leftarrow$ ExecuteAction($a$)
 8:       UpdateModel($s, \vec{a}, s', \omega, r$)
 9:       UpdateQ($s, t, \vec{a}$)
10:     else
11:       $a \leftarrow$ SelectAction()
12:       $(s', \vec{a}, \omega, r) \leftarrow$ ExecuteAction($a$)
13:       UpdateModel($s, \vec{a}, s', \omega, r$)
14:       UpdateQ($s, h, \vec{a}$)
15:     end if
16:     $s \leftarrow s'$
17:     $h \leftarrow (h, a, \omega)$
18:   end for
19: end for
```

chooses its action randomly with some probability ϵ (i.e., explores randomly), but otherwise greedily chooses the action which maximizes the expected value, i.e., $\arg\max_a Q(h_t, a)$. However, since the learners intend to output policies that are independent of the rehearsal features, there is a principled incentive to explore actions that help predict the rehearsal features. Thus, we propose an additional exploration criterion, beyond ϵ-greedy, that values information gain rather than expected reward.

Having observed history h_t, a learner can pick action $a_{explore}$ with some probability, given by

$$a_{explore} = \arg\max_{a \in A_i} \sum_{\omega \in \Omega_i} P(\omega|h_t, a) E(h_t, a, \omega) \tag{8}$$

where $P(\omega|h_t, a) = \sum_{s,a_-} \tilde{P}(\omega|s, \vec{a}).P(s, a_-|h_t)$, calculated from equation 3 and its prediction of the rehearsal features. While $E(h_t, a, \omega)$ measures the entropy of the prediction distribution that would result if action a is executed and observation ω is received, equation 8 measures the *expected* entropy accounting for the fact that action a has not been executed yet, and observation ω has not actually been received. $E(h_t, a, \omega)$ is estimated as

$$E(h_t, a, \omega) = -\sum_{s \in S} \sum_{a_- \in A_-} P(s, a_-|h_t, a, \omega) \log P(s, a_-|h_t, a, \omega)$$

where $P(s, a_-|h_t, a, \omega)$ is estimated as $P(s, a_-|h_t, a, \omega) = \alpha \tilde{P}(\omega|s, \vec{a}) P(s, a_-|h_t)$, where α is a normalization factor.

One of the difficulties the robot alignment problem poses is that agents have no incentive to gather information until they have gathered information frequently enough to learn that it is worthwhile. Therefore, we expect this exploration criterion – called *entropy based exploration* – to be particularly beneficial to robot alignment. However, given the general need of a RLaR learner to achieve independence from rehearsal features, we expect it to be valuable in all Dec-POMDP problems.

In our experiments, we use an upper-confidence bound (UCB) [1] approach for SelectActionMDP; however, other action selection approaches might be applied successfully too. Under the UCB approach, agents select joint actions via

$$\arg\max_{\vec{a} \in A} Q(s, t, \vec{a}) + (R_{max} - R_{min}) C \sqrt{\left(\frac{2 \log n_{s,t}}{n_{s,t,\vec{a}}} \right)}, \tag{9}$$

for some constant C, where $n_{s,t}$ is the number of times the state s has been encountered at step t, and $n_{s,t,\vec{a}}$ is the number of times $\vec{a}$ has been executed in state s at step t. The term added to $Q(s, t, \vec{a})$ is an exploration bonus that balances exploration with exploitation with optimal asymptotic behavior.

Ideal Solution

In this work, we only evaluate the convergence of RLaR empirically by comparing an agents learned Q-Values, $Q_i(h_i, a)$, against a set of target (optimal) Q-Values, $Q^*_i(h_i, a^*)$. In principle, we can calculate these Q^*-values by solving a constraint optimization problem with real variables $Q^*_i(h_i, a)$, $P(s, h_{-i}|h_i)$, and binary variables $I(h_i, a)$ for every agent i. The $I(h_i, a)$ variables essentially describe the agents' policies, i.e. $I(h_i, a) = 1$ implies that agent i will execute action a after observing history h_i. The $I(h_i, a)$ are constrained to be consistent with the Q^*-values so that $\arg\max_{b \in A_i} Q^*_i(h_i, b) \neq a_i \implies I(h_i, a_i) = 0$, and they are further constrained so that exactly one action must be chosen for each history, i.e. $\sum_{a_i \in A_i} I(h_i, a) = 1$.

The $P(s, h_{-i}|h_i)$ variables are constrained by equation 7. A given $P(s, h_{-i}|h_i)$ describes the probability that agent $-i$ encountered h_{-i} and the state is s after agent i encountered h_i. Note the value of a given $P(s, h_{-i}|h_{-i})$ depends on the policy of agent $-i$ but not agent i's policy.

Finally, the $Q^*_i(h_i, a)$ values are constrained by equation 6, and the objective of the target optimization problem is given by $\max \sum_{i=1}^{n} \sum_{a \in A_i} I(h_\emptyset, a) Q^*_i(h_\emptyset, a)$, where $h_\emptyset$ is the empty history.

Solving this optimization problem will yield an optimal policy via the I variables as well as the optimal Q-values corresponding to that policy. Unfortunately, this cannot be any easier than solving the Dec-POMDP, and furthermore, this optimization problem may have multiple solution sets corresponding to multiple optimal policies. The important thing to note is that while there may be multiple optimal policies, there is *exactly one set of Q^* values for a given optimal policy.*

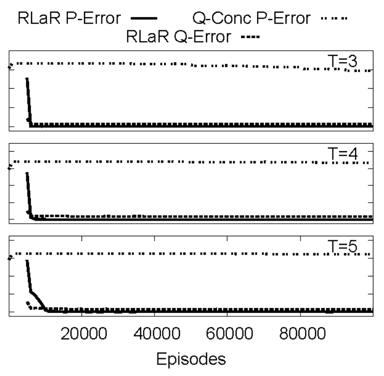

Figure 4: GridSmall

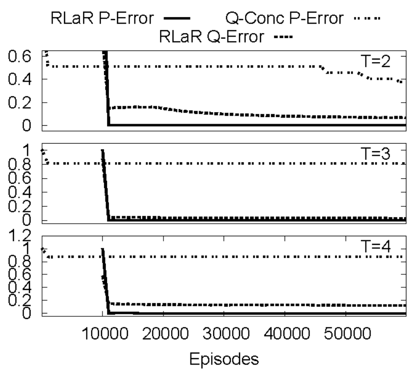

Figure 5: Box Pushing

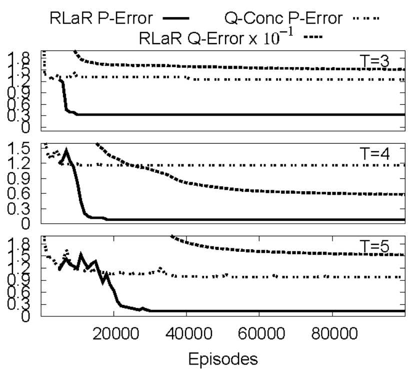

Figure 6: Robot Alignment

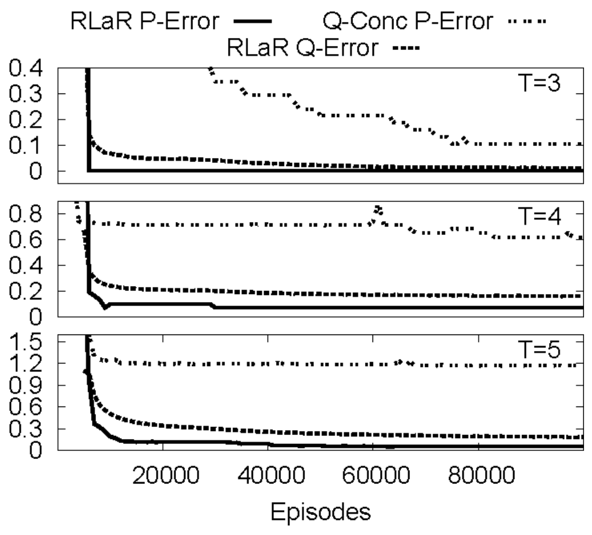

Figure 2: Dectiger

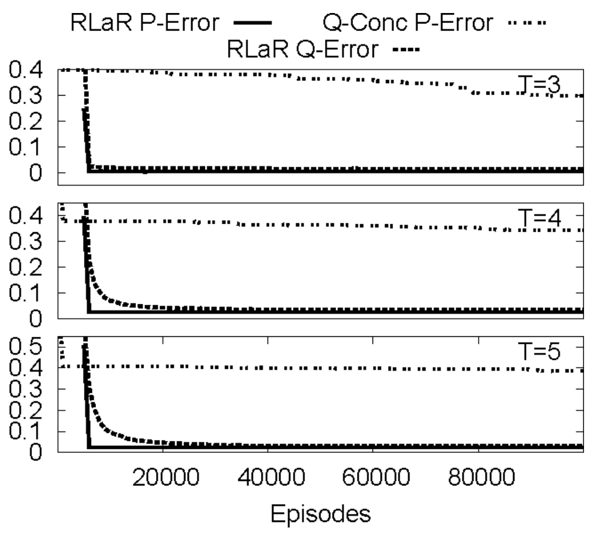

Figure 3: Recycling

EXPERIMENTS

We evaluated the performance of RLaR for the robot alignment problem as well as four well-known benchmark problems: Dectiger, Meeting on a 2x2 Grid (GridSmall), Recycling Robots, and Box Pushing [6].

In the particular variant of the robot alignment problem we used, agent 1's heading space is discretized into two slices and agent 2's heading space discretized into four slices. While noise could be added to transition and observation distributions, we assumed no noise in either, and set Rmax to 100.

For RLaR we used a combination of entropy-based exploration (equation 8) with a probability of 0.25 and ϵ-greedy exploration with $\epsilon = 0.005$.

For each domain and horizon value T, we evaluated the performance of RLaR over 20 runs of 100000 episodes each. For the box pushing domain we allocated 10000 of those episodes to stage 1, and for the others we used 5000 stage 1 episodes. For comparison purposes, we also report the performance of concurrent Q-Learning without rehearsal, i.e. agents that learn $Q(h_t, a)$ using only local information. This setting is labeled as "Q-Conc". No stage 1 episodes were allocated for Q-Conc because an MDP policy cannot be learned without access to the rehearsal features. Furthermore, while an initialization phase for Q-Conc was studied in [4], it was unclear that such an initialization improved results. We use ϵ-greedy exploration for Q-Conc with $\epsilon = 0.05$ which gave the best results for Q-conc.

As noted previously, we only study the convergence of RLaR empirically by comparing the learned Q-Values $Q_i(h_i, a)$ to the ideal $Q_i^*(h, a)$ values described in equation 6. Despite the existence of multiple solution sets, if we already have an optimal policy, π, we can easily find a unique set of Q^* values by setting the I to be consistent with π. Note that when calculating $Q*$ this way, if a given history h_i is inconsistent with π, $Q_i^*(h_i, \cdot)$ will be undefined. Thus, we compare the set of Q-values only for those histories which are consistent with the known optimal policy. Some domains, such as the robot alignment and GridSmall have multiple optimal policies, which necessarily have different sets of consistent histories, and so, in order to measure convergence more accurately, when possible, we compared to the particular optimal policy that each run converged to. For runs that did not converge to an optimal policy we chose an optimal comparison policy arbitrarily. In the plots, we report this value as $Q\text{-}Error = \sum_{h \in H^C} \frac{|Q_i^*(h,a^*) - Q_i(h,a^*)|}{|H^C|}$, where H^C is the set of histories consistent with the comparison policy and a^* for a given history h is $\arg\max_{a \in A_i} Q_i^*(h, a)$.

*Q-Error*s only indicate the quality of the learner's value function for the histories in H^C. However, a learner potentially learns Q-values for a much larger set of histories, H^L, and poor learning on $H^L - H^C$ can produce poor policies even when *Q-Error*s are low. Therefore, it is important to also evaluate the quality of the policies actually produced by RLaR. In order to measure the policy quality for RLaR and Q-Conc after each episode, we found the error of agents' current policy relative to the known optimal policy value (the comparison policy), i.e. $\frac{|v_{pol} - v_{opt}|}{|v_{opt}|}$. We refer to this measure in our plots as the "policy error" or "P-Error".

From Figures 2–6 we can see that RLaR converges rapidly (usually in fewer than 20000 episodes) to (near) optimal policies (< 0.1 policy error) on all domains and horizons except for robot alignment $T = 3, 5$. Q-Conc performs much worse than RLaR in all domains, which highlights the impact of incorporating non-local information into the learning process, i.e., the efficacy of rehearsal based learning. Also note that, Q-Conc performs substantially worse on the robot alignment problem than it does in the other domains, which reinforces our claim that robot alignment is a difficult problem for learning-based approaches.

The Q-Error was relatively small in all domains except Robot Alignment, suggesting that the learned Q-Values indeed converged to values close to the ideal Q^* values. Note that in robot alignment, multiple optimal policies exist for each horizon, and many runs did not converge to a clear optimal policy, so the arbitrary comparison policy (as mentioned before) contributed to large Q-Error. Furthermore, due to the "combination lock" nature of the robot alignment problem, when agents fail to converge to the optimal solution, the histories in H^C tend to be unexplored, so the Q-Error is especially high for those runs.

CONCLUSIONS

We have presented a novel reinforcement learning approach, RLaR with a new exploration strategy suitable for partially observable domains, for learning Dec-POMDP policies when agents are able to rehearse using information that will not be available during execution. We have shown that RLaR can learn near optimal policies for several existing benchmark problems, with a low sample complexity.

We have also introduced a new benchmark problem, robot alignment, which is easy for centralized solvers that *compute* policies given the full Dec-POMDP model, yet quite difficult for reinforcement-learning based Dec-POMDP solvers. Our hope is that this benchmark problem will help motivate more sophisticated RL-based Dec-POMDP solution techniques in the future.

REFERENCES

[1] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

[2] B. Banerjee, J. Lyle, L. Kraemer, and R. Yellamraju. Sample bounded distributed reinforcement learning for decentralized pomdps. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI-12)*, pages 1256–1262, Toronto, Canada, July 2012.

[3] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of Operations Research*, 27:819–840, 2002.

[4] L. Kraemer and B. Banerjee. Informed initial policies for learning in dec-pomdps. In *Proceedings of the AAMAS-12 Workshop on Adaptive Learning Agents (ALA-12)*, pages 135–143, Valencia, Spain, June 2012.

[5] F. A. Oliehoek, M. T. J. Spaan, J. S. Dibangoye, and C. Amato. Heuristic search for identical payoff bayesian games. In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-10)*, pages 1115–1122, Toronto, Canada, 2010.

[6] M. Spaan. Dec-POMDP problem domains and format. `http://users.isr.ist.utl.pt/~mtjspaan/decpomdp/`.

[7] M. T. J. Spaan, F. A. Oliehoek, and C. Amato. Scaling up optimal heuristic search in Dec-POMDPs via incremental expansion. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI-11)*, pages 2027–2032, Barcelona, Spain, 2011.

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

[9] D. Szer and F. Charpillet. Point-based dynamic programming for dec-pomdps. In *Proceedings of the 21st National Conference on Artificial Intelligence*, pages 1233–1238, Boston, MA, 2006.

[10] C. Zhang and V. Lesser. Coordinated multi-agent reinforcement learning in networked distributed POMDPs. In *Proc. AAAI-11*, San Francisco, CA, 2011.