Informed Initial Policies for Learning in Finite Horizon Dec-POMDPs

Landon Kraemer and Bikramjit Banerjee
School of Computing
University of Southern Mississippi
118 College Dr. #5106
Hattiesburg, MS 39406-0001
landon.kraemer@eagles.usm.edu, bikramjit.banerjee@usm.edu

ABSTRACT

Decentralized partially observable Markov decision processes (Dec-POMDPs) offer a formal model for planning in cooperative multiagent systems where agents operate with noisy sensors and actuators, and local information. Prevalent Dec-POMDP solution techniques have mostly been centralized and have assumed knowledge of the model. In real world scenarios, however, solving centrally may not be an option and model parameters may be unknown. To address this, we propose a distributed, model-free algorithm for learning Dec-POMDP policies, in which agents take turns learning, with each agent not currently learning following a static policy. For agents that have not yet learned a policy, this static policy must be *initialized*. We propose a principled method for learning such initial policies through interaction with the environment. We show that by using such *informed initial policies*, our alternate learning algorithm can find near-optimal policies for three benchmark problems.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent Systems*

General Terms

Algorithms, Experimentation

Keywords

Reinforcement learning, Decentralized POMDPs

1. INTRODUCTION

Decentralized partially observable Markov decision processes (Dec-POMDPs) offer a formal model for planning in cooperative multiagent systems where agents operate with noisy sensors and actuators, and local information. While many techniques have been developed for solving Dec-POMDPs exactly and approximately, they have been primarily centralized and reliant on full knowledge of the model parameters. But in real world scenarios, model parameters may not be known a priori, and centralized solvers fail to decentralize the policy computation.

Simple distributed reinforcement learning, particularly Q-learning [15], can address both of the above limitations. Q-learning agents can learn mappings from their own action-observation histories (H_i) to their own actions (A_i), via a quality function ($Q_i : H_i \times A_i \mapsto \mathbb{R}$) that evaluates the long-term effects of selecting an action after observing an individual action-observation history. Since agents cannot observe states or each others' actions directly, *independent learning* [5] is a more reasonable approach than joint learning. Independent learners ignore the actions and observations of the other learners and simply learn their own Q-functions directly.

We investigate two ways of applying independent Q-learning to solving finite-horizon Dec-POMDP problems: one in which agents learn concurrently (Q-Conc), and one in which agents take turns to learn the best responses to each other's policies (Q-Alt). In the latter method, each agent that has not yet taken a turn learning needs an initial policy to follow. There are simple ways to choose an arbitrary initial policy in constant time, e.g., choose a policy at random or follow a uniform stochastic policy, but such methods do not consider the transition, observation, or reward structure of the Dec-POMDP. As the main contribution of this paper, we propose a simple and principled approach to building a finite initial joint policy that is based upon the transition and reward (but not the observation) functions of a Dec-POMDP. To lay the groundwork, we first discuss how this initial joint policy can be *computed* in a centralized, model-based fashion. We then discuss how agents can *learn* such policies in a distributed, model-free manner, and then demonstrate for three benchmark problems that Q-Alt initialized with this *informed policy* (QIP-Alt) produces better joint policies than Q-Conc and Q-Alt initialized with uniform stochastic policies (QSto-Alt) and Q-Alt initialized with randomly-chosen pure policies (QRand-Alt).

This work extends the short paper [8], and the informed policy that we propose has also been used for initialization in another successful Monte Carlo based approach called MCQ-Alt [2].

2. BACKGROUND

In this section we introduce the POMDP and the Dec-POMDP formalisms, both of which will be used in our work.

2.1 POMDPs

We can define a partially-observable Markov decision process (POMDP) as a tuple $\langle S, A, P, R, \Omega, O \rangle$, where:

- S is a finite set of (unobservable) environment states.
- A is a finite set of actions.

- $P(s'|s,a)$ gives the probability of transitioning to state $s' \in S$ when action $a \in A$ is executed in state $s \in S$.
- $R(s,a)$ gives the reward the agent receives upon executing action $a \in A$ in state $s \in S$.
- Ω is a finite set of observations.
- $O(\omega|s',a)$ gives the probability of observing $\omega \in \Omega$ if the current state is $s' \in S$ and the previous action was $a \in A$.

POMDP is an extension of the MDP formalism, where only certain features of the environment may be directly observed by an agent. That is, the agent may not directly know the state and must instead use observations received from the environment and the transition and observation probabilities to maintain a *belief* over the environment states and act rationally under this uncertainty.

An agent can interact with a POMDP for a finite or infinite number of steps, or *horizon*. At each step, the agent chooses an action and receives reward based upon the environment state and the action executed, and it receives an observation based upon the action executed and the resulting state. At a given step, the agent knows all of the actions it has executed and all of the observations it has received up to that point, and the goal is then to find a policy which maps action-observation *histories* to actions, such that the reward the agent expects to receive is maximized. More formally, if the t step history of an agent is $h_t = \langle a_1, \omega_1, \ldots, a_t, \omega_t \rangle$, then its policy must map this action-observation history to some action that the agent will execute in the next step, a_{t+1}.

For a finite horizon, such a policy can be represented as a tree where each level of the tree corresponds to a step and every node is labeled with an action to be performed. If T represents the horizon, then each node of depth $d < T$ has $|\Omega|$ children, i.e. one child for each observation. An agent follows a tree policy by executing the action that labels the root node, and then by following the subtree policy along the branch corresponding to the observation it receives.

Figure 1 gives an example policy as both an action-observation history to action mapping (left) and a policy tree (right) for the single agent tiger problem introduced in [7] with T=3. In the tiger problem, $\Omega = \{HearLeft(HL), HearRight(HR)\}$ and $A = \{Listen(L), OpenLeft(OL), OpenRight(OR)\}$. An agent following this policy will always listen twice. If it hears the tiger behind the same door twice, it will open the opposite door. If the agent receives mixed observations, it will not open a door. Note that a single policy may not be able to map many histories to actions because they do not partake in the policy, e.g., histories beginning with the agent opening a door are not a part of the policy in Figure 1.

$$
\begin{array}{rcl}
() & \rightarrow & \text{L} \\
(L, HL) & \rightarrow & \text{L} \\
(L, HL, L, HL) & \rightarrow & \text{OR} \\
(L, HL, L, HR) & \rightarrow & \text{L} \\
(L, HR) & \rightarrow & \text{L} \\
(L, HR, L, HL) & \rightarrow & \text{L} \\
(L, HR, L, HR) & \rightarrow & \text{OL}
\end{array}
$$

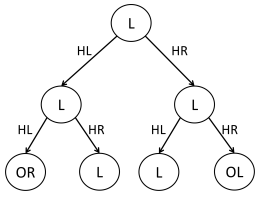

Figure 1: Two equivalent representations of an example policy for the single-agent tiger problem.

2.2 Decentralized POMDPs

The Decentralized POMDP (Dec-POMDP) formalism extends the POMDP formalism to accommodate multiple agents. 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 playing the game.
- 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 $\vec{a} \in A$.

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, for instance, 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 spread out to efficiently cover the area and also may need to travel deep underneath rubble, both of which could interfere with wireless communication.

Assuming agents cannot communicate observations and actions, each agent must choose actions based only upon his own actions and observations; however, since the transition, reward, and observation functions depend on *joint* actions, the quality of each agent's policy is dependent on the policies played by all agents or the *joint policy*. The goal of the Dec-POMDP problem, then, is to find the joint policy that maximizes the expected reward received by the agents. The problem of finding this optimal joint policy has been proven to be NEXP-complete [3].

3. Q-LEARNING FOR DEC-POMDPS

While many Dec-POMDP solution techniques have been devised, most of the prevalent Dec-POMDP solution techniques have been centralized and have required knowledge of the model. That is, a central solver determines which joint policy the agents should follow given P,R, and Ω, and then it distributes these policies to the agents. However, in real-world scenarios, the model may not be available, and it may not be feasible for the problem to be solved centrally. There is a need then, for a method by which agents can learn a Dec-POMDP policy in a distributed manner without prior knowledge of the model.

Reinforcement learning algorithms have previously been applied in infinite horizon POMDPs in both model-based [4, 9, 13] and model-free ways [10]. Model-based methods first learn a model of the environment, and then compute a policy based on that model, where model-free methods learn

a policy directly. In Dec-POMDPs, the actions and observations of the other agents are hidden, which makes model-based learning complex; therefore, we opt for the model-free approach. Recently, Zhang and Lesser [17] have applied reinforcement learning to a variant of the finite horizon Dec-POMDP problem, where agents are organized in a network, and agents' influences on one another are limited to cliques. While our goal is similar, we focus on less structured Dec-POMDPs that are inherently less-scalable.

Reinforcement learning algorithms for finite horizon often evaluate an action-quality value function Q, given by

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

where $\gamma \in [0,1]$ is a discount factor, $R(s,a)$ is the reward the agent receives for executing a in state S, and $P(s'|s,a)$ is the probability of transitioning from state s to state s' if action a is executed. $V^{\pi}(s', t+1)$ gives the expected sum of rewards received when executing policy π starting at step $t+1$ given that the state is s'.

Since state transitions are dependent on joint actions in Dec-POMDPs, the equivalent Q function would be $Q(s, \vec{a}, t)$, i.e. based upon joint actions; however, since each agent can only know his own actions and observations, we use an *independent learning* [5] approach where each agent is responsible for learning individual action quality values. Since the states are not visible to the agents, we use a Q function based upon the policy representation of Dec-POMDPs, i.e. each agent maps his individual action-observation histories to actions [17].

Agents can learn these Q-Values concurrently; however, this can lead to oscillation, and our empirical results show that allowing the agents to alternate learning can produce better policies. To elaborate, in alternate learning, the agents take turns learning, and while an agent is learning, all other agents play static policies. Nair et. al have previously proposed an algorithm named JESP [11] in which agents alternately *compute* best responses to each other's policies using knowledge of the model; however, as we do not assume knowledge of the model, we instead *learn* the best-response policies.

For the sake of clarity, we will assume only two agents in notation. Given the policy of the other agent π_-, the quality of learner i's action at a given level t individual action-observation history h_t is then given by

$$Q^*(h_t, a|\pi_-) = R^*(h_t, a|\pi_-) + \sum_{\omega \in \Omega_i} H^*(h_t, a, h_{t+1}|\pi_-) \cdot \max_{a' \in A_i} Q^*(h^*_{t+1}, a') \quad (2)$$

where h_{t+1} is a level-$t+1$ action-observation history produced by the concatenation of h_t and (a, ω), i.e. $h_{t+1} = (h_t, a, \omega)$. The best response policy of the learner, π_l, to the other agents' policy is given by

$$\pi_l(h_t) = \arg\max_{a \in A_i} Q^*(h_t, a|\pi_-) \quad (3)$$

The function R^* and H^* represent the expected immediate reward and history transition functions for the learner, given by

$$R^*(h_t, a|\pi_-) = \sum_{s, h_t} P(s|h_t, h_-) P(h_-|h_t, \pi_-) R(s,a) \quad (4)$$

$$H^*(h_t, a, h_{t+1}|\pi_-) = \sum_{s, s', h_-} P(s|h_t, h_-) P(h_-|h^t, \pi_-) \cdot \sum_{\omega_- \in \Omega_-} P(\vec{\omega}|s', \vec{a}) \quad (5)$$

where h_- is the history of action-observations encountered by the other agent, and $\vec{\omega} = \langle \omega, \omega_- \rangle$ and $\vec{a} = \langle a, a_- \rangle$ are the joint observation and joint action, respectively.

Since the agents cannot communicate and do not have prior knowledge of the model, a learning agent will not have knowledge of π_-, h_-, and a_- and must estimate R^* and H^* (and therefore Q^*) from its own experience of executing actions and receiving observations and rewards. We denote the estimation of Q^* as Q. An agent can learn $Q(h_t, a)$ by applying the following update each time it executes a when the current history is h_t

$$Q(h_t, a) = Q(h_t, a) + \alpha \left[r + \gamma \max_{a'} Q(h_{t+1}, a') - Q(h_t, a) \right] \quad (6)$$

where ω and r are the observation and reward received, respectively, after executing a, and $h_{t+1} = (h_t, a, \omega)$.

In our work, we compare both alternating and concurrent Q-learning methods, both of which use the Q update method in equation 6. We refer to these as Q-Alt and Q-Conc, respectively. For both learning methods, each agent follows Algorithm 1 during each episode. In Q-Conc, the *learning* flag is always true for every agent, but in Q-Alt, the *learning* flag can only be true for one agent at a time.

Because the observation and reward functions in Dec-POMDPs depend on joint actions, step 4 of algorithm 1 represents a synchronization point because an agent cannot receive r or ω until all agents have executed an action. This does not, however, imply that agents must execute actions simultaneously.

Algorithm 1 ExecuteEpisode(*learning*)

```
1: h_1 ← ∅
2: for t = 1 to T do
3:    a ← SelectAction(h_t, learning)
4:    Execute a and receive r, ω.
5:    h_{t+1} ← (h_t, a, ω)
6:    if learning then
7:       Q(h_t, a) ← Q(h_t, a) +
8:            α [r + γ max_{a'} Q(h_{t+1}, a') − Q(h_t, a)]
9:    end if
10: end for
```

Algorithm 2 SelectAction(h_t, *learning*) for agent i

```
1: if learning then
2:    if explore then
3:       return explored_action
4:    else
5:       return arg max_{a∈A_i} Q(h_t, a)
6:    end if
7: else
8:    return π(h_t), the current static policy for this agent.
9: end if
```

Algorithm 2 describes the method by which agents select actions to execute. If an agent is currently learning it will

either explore (*explore* and *explored_action* are calculated by a chosen exploration method) or choose an action based upon current Q-Values; however, if an agent is not currently learning - which will only occur in alternate learning - it will choose an action based upon some policy π. Now, if the agent has already learned, π will be the policy it learned previously; however, if the agent has not had a turn learning, π must be some *initial* policy.

There are multiple efficient ways to arbitrarily initialize π. For example, we could let π be a pure policy (i.e. one where each possible action-observation history maps to exactly one action) chosen at random (as is done in the JESP algorithm), or alternatively, we could let π be a uniform stochastic policy, where the agent always chooses an action at random from a uniform distribution. We note, however, that such methods do not consider the behavior of the model. That is, for problems with identical $|A|$ and $|\Omega|$ the policies would initialized the same way, regardless of how P, R, and Ω are defined. In the following section, we will present a principled method for choosing an initial policy that respects both P and R (but not Ω).

The main motivation for selecting initial policies in a more informed manner is the hope of getting in the zone of attraction of the optimal Nash equilibrium. Our approach of alternating best response learning can be essentially looked upon as a form of hill climbing where each step is taken by a different agent. Since hill climbing can only be guaranteed to converge to a local optimum, we expect alternating best response learning to eventually lead to a *locally optimal* Nash equilibrium policy. In order to overcome locality of the optimum, hill climbing approaches often resort to multiple (re)starts, which is unrealistic in our case since each hill climbing step is an entire best response learning phase with a non-trivial cost. Instead, we devote some effort in selecting the initial location of the agents in the joint policy space, hoping that this sets us close enough to the *globally optimal* Nash equilibrium so that it becomes the attractor to the alternating hill climbing trajectory.

4. INFORMED INITIAL POLICIES

In this section, we present a centralized, model-based method to compute an initial policy which is informed by the model. The high-level idea is to map a Dec-POMDP problem into a constrained POMDP problem, solve that POMDP problem, and then map the resulting optimal POMDP policy back into the space of Dec-POMDP joint policies.

To ground the intuition on existing literature, consider the Q_{POMDP} heuristic used in [16], that finds an upper-bound for a Dec-POMDP by mapping the problem into a POMDP where the action set is the set of all joint actions and the observation set is the set of all joint observations.

The Q_{POMDP} relaxation can be formalized as follows. Given a Dec-POMDP instance $D = \langle n, S, A, P, R, \Omega, O\rangle$, create and find the optimal policy value for a POMDP instance $D' = \langle S, A, P, R, \Omega, O\rangle$.

Essentially, Q_{POMDP} assumes the agents are allowed to share their individual observations, which, in turn, eliminates miscoordination because the agents know exactly what their teammates have observed and executed and can therefore choose the best action for each contingency. This is not the case in Dec-POMDP, where each agent must choose the best action based only upon its own action-observation history. In a policy found by Q_{POMDP}, an agent may have different actions associated with the same action-observation history, disambiguated by others' action-observation histories; however, agents do not have access to the observations of other agents during Dec-POMDP execution. Thus, the resulting POMDP policy cannot be (and was not intended to be) mapped to a Dec-POMDP policy in a meaningful way.

In this backdrop, we note that if there were only one possible observation that each agent could receive at each step, this ambiguity would disappear because the agents would be able to know the joint observation history at all times.

Unfortunately, most (if not all) Dec-POMDPs of interest will have more than one observation; however, if the agent policies *ignore* observations, a similar effect can be achieved.

Our proposed method is simple. Create and solve a POMDP where the action set is the set of all joint actions and the observation set is $\{\delta\}$ (a dummy observation). That is, for a Dec-POMDP $D = \langle n, S, A, P, R, \Omega, O\rangle$, create the POMDP $D' = \langle S, A, P, R, \{\delta\}, O\rangle$. Since δ is the only possible observation, the optimal POMDP policy can be thought of as a chain of joint actions $\pi_{opt} = \vec{a}^1, \vec{a}^2, \ldots, \vec{a}^T$, where T is the horizon. The informed initial Dec-POMDP policy can then be constructed for each agent i by setting all level t action nodes to a_i^t (agent i's contribution to a^t). That is, agent i will always execute a_i^t at level t, regardless of what it has previously observed. Figure 2 depicts this process for $T = 2$.

5. LEARNING INFORMED INITIAL POLICIES

In the previous section, we presented a centralized, model-based method for calculating an informed initial policy; however, if we are to use such initial policies for model-free, distributed algorithms (such as Q-Alt), the method by which we calculate these policies should be model-free and distributed as well.

It is generally assumed in Dec-POMDPs that the agents cannot observe each other's actions, so without extra assumptions, the agents must learn independently. One simple approach would be concurrent Q-learning, where each agent i tries to learn

$$Q' : H^{A_i} \times A_i \mapsto \mathbb{R} \tag{7}$$

for all individual action-only histories $h_t^{A_i} \in H^{A_i}$ and individual actions $a \in A_i$. To accomplish this, agents use a modified version of algorithm 1 where the true observation ω is replaced with the dummy observation γ. Since the agents learn concurrently, the *learning* flag will always be true for each agent.

Independent concurrent learning can lead to oscillation because each agent is learning in a dynamic environment (all other agents are treated as part of the environment) that changes in response to the agent's behavior.

Suppose, though, that each agent i can announce the random seed he will use for exploration and his action set A_i before learning begins. This would allow each agent to model the exploration of every other agent. Since observations are ignored when learning the initial policy and since the agents all receive identical rewards, if agents can model each other agent's exploration, then each agent can learn the function

$$Q' : H^A \times A \mapsto \mathbb{R} \tag{8}$$

directly, where A is the set of joint actions and H^A repre-

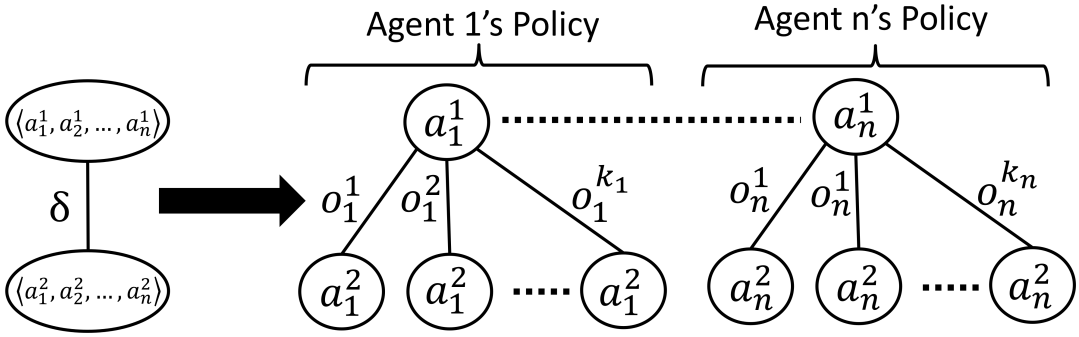

Figure 2: Conversion of the optimal policy of the constrained POMDP to a valid policy of the original Dec-POMDP for T=2.

sents the set of joint action-only histories. Note that such a technique can not be applied to learn $Q : H \times A \mapsto \mathbb{R}$, where H is the set of joint action-observation histories because observations are generated by nature, not the agents, and hence unsharable by announcing random seeds..

The agents can learn equation 8 by making a few modifications to algorithms 1 and 2. Specifically, all actions must be *joint* actions rather than individual actions, and the Q function should accept joint action-observation histories rather than individual histories. As before, ω should be replaced by the dummy observation γ, and the agents will learn concurrently (i.e. the *learning* flag will always be true).

It is important to note that because the set of joint actions A grows exponentially in the number of agents (i.e. $|A| = O(|A_{i^*}|^n)$, where A_{i^*} represents the largest individual action set), the potential memory requirement ($O((|A_{i^*}|^n)^T \cdot |A_{i^*}|)$) for the joint Q'-Value function also grows exponentially in the number of agents. However, for scenarios where this memory requirement is intractable, it is likely that the number of episodes required for convergence of the independent Q'-Value function is also intractably large.

6. INITIAL POLICIES FOR CONCURRENT LEARNERS

In allowing the agents to share random seeds for policy initialization, we are introducing extra information that Q-Conc is unable to exploit. For comparison purposes, we also would like to give Q-Conc access to this information as well. While our learned initial policy is readily usable for alternate learning where all agents except the currently learning agent follow static policies, applying our learned initial policy to concurrent learning where no agent follows a static policy is less straightforward.

In order to use our informed initial policy for concurrent learning, we use the Q'-Values learned as per Section 5 to initialize the agents' Q-Values as follows. When an agent i encounters a history $h_t = (a_1, o_1, a_2, o_2, \ldots, a_t, o_t)$ for the first time, if it learned independent Q'-Values (per equation 7), it will set $Q(h_t, a) = Q'(h_t^{A_i}, a) \forall a \in A_i$, where $h_t^{A_i} = (a_1, a_2, \ldots, a_t)$.

The process is less straightforward if agents learned joint Q'-Values (per equation 8). Recall from section 4 that an informed initial policy can be written as a single chain of joint actions. Suppose h_t^A/h_t is the joint action-only history in which all agents except for i are executing actions per the informed initial policy and agent i is executing $(a_1, a_2, \ldots, a_t)$. Also, suppose $\vec{a}_{t+1}/a$ is the action corresponding to step $t+1$ in the inital policy chain with agent i's action replaced with a. Then, agent i will initialize $Q(h_t, a) = Q'(h_t^A/h_t, \vec{a}_{t+1}/a) \forall a \in A_i$. Essentially, agent i is assuming that the other agents will continue still follow the initial policy, even though h_t or a may deviate from it.

7. EVALUATION

In this section, we evaluate the performance of the five different Q-learning settings: QIP-Alt, QIP-Conc, Q-Conc, QSto-Alt, and QRand-Alt for the BroadcastChannel [6], DecTiger [11], and MarsRover [1] benchmark problems. Note that we do not compare against state-of-the-art model-based Dec-POMDP solvers such as GMAA*-ICE [14] because such solvers rely on full knowledge of the model, which we do not assume is available.

7.1 Problem Domains

In the BroadcastChannel domain, two nodes broadcast messages over a channel that is limited to one message at a time. The nodes receive a reward for each message they successfully send; however, if they attempt to send a message at the same time, a collision occurs. Each node has a buffer that holds at most one message, and at every step this buffer becomes full with some probability.

In the Dec-Tiger domain, there are two doors. Behind one door is treasure, but behind the other door is a tiger. Opening the door concealing the treasure, yields a reward, and opening the door concealing the tiger, yields a penalty. The two agents do not know a priori which door the tiger is behind; however, they are allowed to listen for clues. Upon listening, each agent either hears the tiger behind the left door or the right door. The observations are noisy, however, and each agent has a fifteen percent chance of hearing the tiger behind the wrong door. This serves to make coordination more difficult, as an agent cannot be sure what the other has heard, and miscoordination is heavily penalized.

In the MarsRover domain, two rovers are tasked with conducting scientific experiments at four different test sites, arranged in a four-by-four grid. At each step the agents can choose to move north, south, east, or west, or they can sample or drill. Two of the sites are intended to be sampled, and thus drilling them incurs a large negative reward. Two of the sites must be drilled by both agents simultaneously. Each agent has full knowledge of his own location as well as whether or not an experiment has been performed at each site.

With only two states, two observations, and two actions, BroadcastChannel is the smallest of the three. It is also os-

tensibly the least complex, as it has been solved for a horizon of 900 [14]. DecTiger with two states, two observations, and three actions is larger and has only been solved for a horizon of six. MarsRover with 256 states, eight observations, and six actions is the largest of the three, and it has only been solved for a horizon of four [12].

7.2 Policy Computation and Evaluation

In order to compare the performance of the different settings, we must calculate the value of the policies they produce. One method for calculating a learned policy value is simulation-based, that simply allows the agents to interact with the environment for a sufficient number of episodes and averages the total reward received per episode; however, since the domains we use are known benchmark problems, we can exactly calculate the value of a policy using the model parameters. To evaluate the value of a joint policy $\vec{\pi}$, we find

$$V^{\vec{\pi}}(\vec{h}_0) = \sum_{s \in S} b_0(s) V^{\vec{\pi}}(\vec{h}_0, s) \quad (9)$$

where $b_0 \in \Delta(S)$ is the initial state distribution. $V^{\vec{\pi}}(\vec{h}_t, s)$ for a given joint history $\vec{h}_t$ and state s is given by

$$V^{\vec{\pi}}(\vec{h}_t, s) = R(s, \vec{\pi}(\vec{h}_t)) + \sum_{s' \in S} P(s'|s, \vec{\pi}(\vec{h}_t)) \cdot \sum_{\vec{\omega} \in \Omega} O(\vec{\omega}|s', \vec{\pi}(\vec{h}_t)) V^{\vec{\pi}}((\vec{h}_t, \vec{\pi}(\vec{h}_t), \vec{\omega}_t), s') \quad (10)$$

where $\vec{\pi}(\vec{h}_t) = \langle \pi_1(h_{t,1}), \ldots, \pi_n(h_{t,n}) \rangle$.

7.3 Experimental Results

For each domain, we chose eleven increasing values $k_1, \ldots, k_{11}$ and allowed each setting to learn for an $k = k_1, \ldots, k_{11}$ episodes. The k values were rounded to integers, and derived from experiments with another algorithm (reported in [2]).

k represents the *total* number of episodes agents are allowed to execute, including episodes required to learn initial policies and all alternations. Thus, in Q-Conc and QIP-Conc both agents learn for k episodes (with agents devoting k' of those episodes to learning the initial policy in QIP-Conc), in QSto-Alt and QRand-Alt each agent learns for $\frac{k}{n}$ episodes (where n is the number of agents), in QIP-Alt each agent learns for $k' + \frac{k-k'}{n}$ episodes. Note that since QIP-Alt and QIP-Conc allocate k' episodes for learning the initial policy, they are at a relative disadvantage if the initialization is not helpful because the other settings have k' extra episodes for learning.

We used $k' = 20000$ for DecTiger and $k' = \max(200000, \frac{k}{20})$ for both BroadcastChannel and MarsRover. For the alternating settings in Dectiger and BroadcastChannel, the agents were allowed one alternation each with $\frac{k-k'}{n}$ episodes per alternation ($k' = 0$ for QSto-Alt and QRand-alt), and in the MarsRover problem each agent was allowed we used epsilon-greedy exploration with $\epsilon = .05$. two alternations with $\frac{k-k'}{2n}$ episodes per alternation. In all runs, we used a learning rate $\alpha = .001$, and for our exploration method (lines 2-3 in algorithm 2), we used epsilon-greedy exploration with $\epsilon = .05$ [15]. All trials were run on an Intel Xeon 3.00 Ghz processor.

Tables 1, 2, and 3 give the relative error, $\frac{|v_{learned} - v_{opt}|}{v_{opt}}$ (where v_{opt} is the value of the known optimal policy), averaged over 50 runs for each setting, domain, horizon, and value of k. We have opted to report this data in tables because the range of values is wide some cases (e.g. [0, 16.494]), and such a coarse resolution makes it difficult to interpret relations between smaller values. If an error value reported in the table is bolded, this indicates that it is greater than the error for QIP-Alt for the same horizon and k. In addition to the tables, we report average relative error versus average runtime for QIP-Alt and Q-Conc for all the three domains in figures 3, 4, and 5. Note the the horizontal axes have been scaled logarithmically in these figures.

First note that for each domain and horizon, there exists a k beyond which QIP-Alt produces lower error than all other settings. In the case of BroadcastChannel, the advantage of QIP-Alt is less pronounced; however, since BroadcastChannel is the simplest of our domains, this is not entirely surprising. For MarsRover and DecTiger, the results are stronger, however. QIP-Alt is the only algorithm to have average relative error of zero (implying that all 50 policies found were optimal). In most cases, the nearest competitor in those two domains has error $> .2$ (and often it is much higher). As noted previously, T=4 is the highest horizon for which the MarsRover domain has been solved by a centralized algorithm with knowledge of the model. It is encouraging then that, being distributed and model-free, QIP-Alt is able to achieve near optimal solutions on average for that horizon.

In the case of QIP-Conc, it is unclear whether or not initializing the Q-Values as described in section 6 is beneficial. For Dectiger with $T = 3, k \geq 10^{5.02}$ and BroadcastChannel with $T = 3, k \geq 10^{5.73}$, it is clearly an improvement on Q-Conc, but for other domain-horizon combinations, it is not so clearly beneficial, especially in the MarsRover domain.

While these results suggest that the informed policy is beneficial if optimality is the only goal, they do not speak to the practicality of it. For instance, while QIP-Alt produces its lowest error for T=3 in MarsRover at $k = 10^{7.48}$, it takes on average 60000 seconds to produce that result, whereas $k = 10^{6.42}$ requires only 3000 seconds on average. In real-world scenarios, time may be more important than optimality. To this end, we provide figures 3, 4, and 5, which show average relative error vs average runtime for QIP-Alt and QIP-Conc for all three domains. Since these are benchmark problems, we have no reason to prefer time over optimality (or vice-versa), so it is difficult to make a qualitative judgement about them; however, had QIP-Alt required days or months to produce error below that produced by Q-Conc, the informed policy would likely be of less practical use.

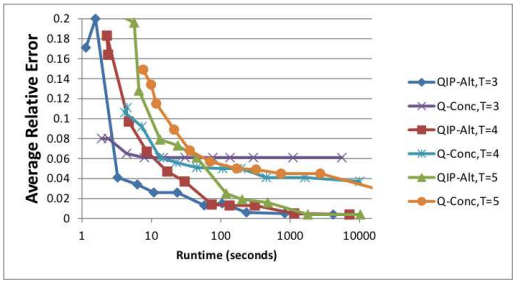

Figure 3: Average relative error vs runtime in seconds for QIP-Alt and QIP-Conc in the BroadcastChannel domain. Horizontal axis has been scaled logarithmically.

k	T=3					T=4					T=5				
	QIP-Alt	QIP-Conc	Q-Conc	QSto-Alt	QRand-Alt	QIP-Alt	QIP-Conc	Q-Conc	QSto-Alt	QRand-Alt	QIP-Alt	QIP-Conc	Q-Conc	QSto-Alt	QRand-Alt
$10^{5.34}$	0.171	**0.175**	0.080	0.085	0.112	0.164	**0.255**	0.111	**0.229**	**0.173**	0.201	**0.330**	0.149	**0.231**	**0.242**
$10^{5.46}$	0.200	0.094	0.080	0.089	0.128	0.183	0.141	0.106	**0.190**	0.166	0.196	0.186	0.134	**0.227**	**0.203**
$10^{5.73}$	0.041	**0.061**	**0.065**	**0.059**	**0.121**	0.097	0.078	0.092	**0.145**	**0.120**	0.128	0.117	0.115	**0.194**	**0.205**
$10^{6.00}$	0.034	**0.053**	**0.061**	**0.049**	**0.112**	0.067	**0.069**	0.061	**0.105**	**0.125**	0.079	**0.102**	**0.089**	**0.165**	**0.184**
$10^{6.27}$	0.026	**0.052**	**0.061**	**0.041**	**0.109**	0.047	**0.067**	**0.056**	**0.065**	**0.122**	0.073	**0.091**	0.068	**0.139**	**0.159**
$10^{6.57}$	0.026	**0.052**	**0.061**	**0.040**	**0.112**	0.037	**0.066**	**0.051**	**0.050**	**0.116**	0.061	**0.087**	0.057	**0.097**	**0.153**
$10^{6.97}$	0.013	**0.036**	**0.061**	**0.033**	**0.107**	0.014	**0.033**	**0.050**	**0.036**	**0.109**	0.025	**0.031**	**0.050**	**0.075**	**0.152**
$10^{7.24}$	0.015	**0.034**	**0.061**	**0.034**	**0.113**	0.013	**0.035**	**0.050**	**0.031**	**0.123**	0.019	**0.036**	**0.049**	**0.065**	**0.143**
$10^{7.60}$	0.006	**0.027**	**0.061**	**0.035**	**0.108**	0.013	**0.039**	**0.041**	**0.028**	**0.105**	0.016	**0.038**	**0.045**	**0.054**	**0.146**
$10^{8.16}$	0.005	**0.026**	**0.061**	**0.032**	**0.114**	0.005	**0.036**	**0.041**	**0.027**	**0.111**	0.004	**0.033**	**0.045**	**0.031**	**0.153**
$10^{8.84}$	0.004	**0.011**	**0.061**	**0.032**	**0.107**	0.004	**0.047**	**0.037**	**0.026**	**0.111**	0.004	**0.023**	**0.030**	**0.023**	**0.154**

Table 1: Average relative errors (compared to the optimal value) for the BroadcastChannel domain for all settings and horizons T=3,4,5. Bolded values are less than respective QIP-Alt values.

k	T=3					T=4					T=5				
	QIP-Alt	QIP-Conc	Q-Conc	QSto-Alt	QRand-Alt	QIP-Alt	QIP-Conc	Q-Conc	QSto-Alt	QRand-Alt	QIP-Alt	QIP-Conc	Q-Conc	QSto-Alt	QRand-Alt
$10^{4.40}$	0.402	**2.058**	**0.734**	**1.437**	**6.407**	1.014	**2.297**	0.726	**15.439**	**13.257**	1.807	**1.885**	1.135	**17.912**	**16.494**
$10^{4.63}$	0.263	**1.425**	**0.617**	**1.612**	**6.194**	0.537	**1.507**	**0.704**	**11.667**	**10.894**	1.221	**1.522**	1.122	**15.293**	**13.940**
$10^{4.81}$	0.084	**0.972**	**0.511**	**1.065**	**6.212**	0.395	**1.071**	**0.692**	**6.158**	**9.511**	1.177	**1.227**	1.089	**16.069**	**12.921**
$10^{5.02}$	0.032	**0.182**	**0.286**	**1.054**	**6.115**	0.197	**0.301**	**0.550**	**2.031**	**8.125**	1.208	0.802	1.034	**12.485**	**9.844**
$10^{5.37}$	0.000	**0.042**	**0.136**	**1.054**	**6.019**	0.000	**0.300**	**0.357**	**1.329**	**7.850**	1.156	0.751	0.941	**3.340**	**6.415**
$10^{5.56}$	0.000	**0.032**	**0.114**	**1.065**	**5.974**	0.000	**0.288**	**0.369**	**1.409**	**7.998**	1.153	0.705	0.816	**2.052**	**6.611**
$10^{5.74}$	0.000	**0.053**	**0.104**	**1.054**	**6.046**	0.005	**0.301**	**0.335**	**1.322**	**8.163**	1.159	0.691	0.754	**1.646**	**5.817**
$10^{6.13}$	0.000	**0.031**	**0.104**	**1.054**	**5.912**	0.005	**0.311**	**0.391**	**1.428**	**7.626**	0.078	**0.641**	**0.648**	**1.215**	**6.035**
$10^{6.42}$	0.011	**0.042**	**0.125**	**1.054**	**6.017**	0.000	**0.299**	**0.334**	**1.321**	**8.266**	0.005	**0.633**	**0.575**	**1.521**	**5.959**
$10^{6.82}$	0.000	**0.042**	**0.125**	**1.054**	**6.026**	0.005	**0.321**	**0.345**	**1.329**	**7.887**	0.030	**0.586**	**0.470**	**1.200**	**5.842**
$10^{7.48}$	0.000	**0.053**	**0.157**	**1.054**	**5.863**	0.000	**0.278**	**0.356**	**1.320**	**7.990**	0.029	**0.275**	**0.359**	**1.201**	**5.821**

Table 2: Average relative errors (compared to the optimal value) for the DecTiger domain for all settings and horizons T=3,4,5. Bolded values are less than respective QIP-Alt values.

k	T=2					T=3					T=4				
	QIP-Alt	QIP-Conc	Q-Conc	QSto-Alt	QRand-Alt	QIP-Alt	QIP-Conc	Q-Conc	QSto-Alt	QRand-Alt	QIP-Alt	QIP-Conc	Q-Conc	QSto-Alt	QRand-Alt
$10^{5.34}$	0.426	**0.862**	0.343	**0.529**	0.408	0.570	**0.682**	0.478	**0.592**	0.509	0.551	**0.672**	0.445	**0.561**	0.525
$10^{5.46}$	0.430	0.407	0.343	**0.476**	0.418	0.589	**0.666**	0.444	**0.601**	0.517	0.537	**0.664**	0.431	**0.586**	0.506
$10^{5.73}$	0.366	**0.381**	0.343	**0.434**	0.298	0.567	0.469	0.411	0.529	0.465	0.517	**0.525**	0.376	**0.537**	0.480
$10^{6.00}$	0.274	**0.381**	**0.343**	**0.415**	**0.291**	0.520	0.433	0.409	0.505	0.461	0.447	0.365	0.340	**0.499**	0.407
$10^{6.27}$	0.274	**0.381**	**0.343**	**0.382**	0.274	0.435	0.429	0.404	**0.461**	0.386	0.425	0.336	0.328	**0.432**	0.354
$10^{6.57}$	0.280	**0.381**	**0.343**	**0.380**	0.274	0.413	**0.427**	0.401	**0.465**	0.378	0.387	0.311	0.308	0.376	0.331
$10^{6.97}$	0.076	**0.381**	**0.343**	**0.377**	**0.288**	0.388	**0.426**	**0.399**	**0.473**	0.374	0.349	0.302	0.287	**0.394**	0.314
$10^{7.24}$	0.000	**0.366**	**0.343**	**0.376**	**0.288**	0.202	**0.427**	**0.398**	**0.451**	**0.369**	0.271	**0.301**	**0.282**	**0.383**	**0.327**
$10^{7.60}$	0.000	**0.381**	**0.343**	**0.375**	**0.301**	0.051	**0.423**	**0.398**	**0.444**	**0.383**	0.095	**0.299**	**0.277**	**0.366**	**0.297**
$10^{8.16}$	0.000	**0.380**	**0.343**	**0.374**	**0.281**	0.057	**0.422**	**0.396**	**0.444**	**0.358**	0.076	**0.302**	**0.272**	**0.363**	**0.292**
$10^{8.84}$	0.000	**0.381**	**0.343**	**0.374**	**0.263**	0.002	**0.421**	**0.393**	**0.436**	**0.342**	0.069	**0.297**	**0.296**	**0.301**	**0.270**

Table 3: Average relative errors (compared to the optimal value) for the MarsRover domain for all settings and horizons T=2,3,4. Bolded values are less than respective QIP-Alt values.

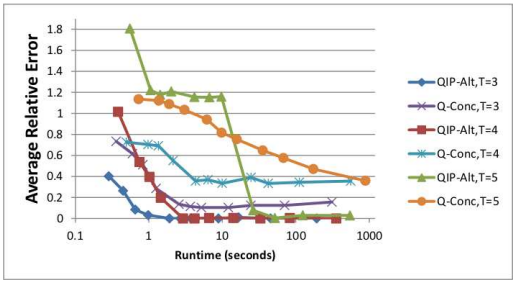

Figure 4: Average relative error vs runtime in seconds for QIP-Alt and QIP-Conc in the DecTiger domain. Horizontal axis has been scaled logarithmically.

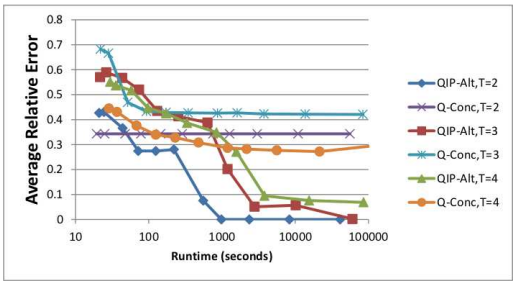

Figure 5: Average relative error vs runtime in seconds for QIP-Alt and QIP-Conc in the MarsRover domain. Horizontal axis has been scaled logarithmically.

8. CONCLUSION

We have presented a simple, principled technique to compute a valid Dec-POMDP policy for use as an initial policy in conjunction with reinforcement learning. Furthermore, we have discussed how such policies can be learned in a model-free manner, and we have shown for three benchmark problems that using such policies as initial policies (instead of stochastic policies or pure policies chosen at random) can improve the outcome of alternating Q-learning.

Acknowledgment: We thank the anonymous reviewers for helpful feedback. This work was supported in part by the US Army under grant #W911NF-11-1-0124.

9. REFERENCES

[1] C. Amato and S. Zilberstein. Achieving goals in decentralized POMDPs. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems*, pages 593–600, Budapest, Hungary, 2009.

[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*, Toronto, Canada, July 2012. To appear.

[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. Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Articial Intelligence*, pages 183–188, San Jose, CA, 1992. AAAI Press.

[5] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, AAAI '98/IAAI '98, pages 746–752, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[6] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the 19th National Conference on Artifical Intelligence*, pages 709–715, San Jose, CA, 2004.

[7] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artifical Intelligence*, 101:99–134, 1998.

[8] L. Kraemer and B. Banerjee. Informed initial policies for learning in dec-pomdps. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence Student Abstract and Poster Program*, Toronto, Canada, July 2012. To appear.

[9] A. K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Department of Computer Science, University of Rochester, 1995.

[10] N. Meuleau, L. Peshkin, K. Kim, and L. Kaelbling. Learning finite-state controllers for partially observable environments. In *Proc. UAI*, pages 427–436, 1999.

[11] R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella. Taming decentralized pomdps: Towards efficient policy computation for multiagent settings. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 705–711, Acapulco, Mexico, 2003.

[12] F. A. Oliehoek, S. Whiteson, and M. T. J. Spaan. Lossless clustering of histories in decentralized POMDPs. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-09)*, pages 577–584, Budapest, Hungary, 2009.

[13] G. Shani, R. Brafman, and S. Shimony. Model-based online learning of POMDPs. In *Proceedings of the European Conference on Machine Learning (ECML)*, volume Lecture Notes in Computer Science 3720, pages 353–364. Springer, 2005.

[14] 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.

[15] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, Mar. 1998.

[16] D. Szer, F. Charpillet, and S. Zilberstein. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 576–583, Edinburgh, Scotland, 2005.

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