Stochastic Multi-Agent Planning with Partial State Models

Feng Wu
University of Science and Technology of China
Hefei, Anhui, China
wufeng02@ustc.edu.cn

Shlomo Zilberstein
University of Massachusetts Amherst
Amherst, MA, USA
shlomo@cs.umass.edu

Nicholas R. Jennings
Imperial College London
London, UK
n.jennings@imperial.ac.uk

ABSTRACT

People who observe a multi-agent team can often provide valuable information to the agents based on their superior cognitive abilities to interpret sequences of observations and assess the overall situation. The knowledge they possess is often difficult to be fully represent using a formal model such as DEC-POMDP. To deal with this, we propose an extension of the DEC-POMDP that allows states to be partially specified and benefit from expert knowledge, while preserving the partial observability and decentralized operation of the agents. In particular, we present an algorithm for computing policies based on history samples that include human labeled data in the form of reward reshaping. We also consider ways to minimize the burden on human experts during the labeling phase. The results offer the first approach to incorporating human knowledge in such complex multi-agent settings. We demonstrate the benefits of our approach using a disaster recovery scenario, comparing it to several baseline approaches.

CCS CONCEPTS

• **Computing methodologies** → **Multi-agent systems**; **Cooperation and coordination**; **Multi-agent planning**.

KEYWORDS

Decentralized POMDPs, Partial State Models

ACM Reference Format:

Feng Wu, Shlomo Zilberstein, and Nicholas R. Jennings. 2019. Stochastic Multi-Agent Planning with Partial State Models. In *DAI-19: First International Conference on Distributed Artificial Intelligence, October 13–15, 2019, Beijing, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3356464.3357699

1 INTRODUCTION

Decentralized Partially Observable Markov Decision Processes (DEC-POMDPs) provide a powerful model for multi-agent planning under uncertainty [3]. Once a model is obtained, it can be solved using various existing optimal or approximate methods. However, the specification of model parameters (i.e., the transition and observation probabilities and the reward values) can be non-trivial as the complexity of the domain grows. Specifically, to specify a model, every factor that is relevant to the decisions of the agents must be modeled using state variables. Unfortunately, the overall state space grows exponentially with the number of state variables, and the complexity of planning for DEC-POMDPs is double-exponential in the size of the state space. This makes it hard to develop complete models and to solve decentralized planning problems — a phenomenon also known as the "curse of dimensionality". Furthermore, in complex domains, there is often relevant knowledge that is difficult to capture and encode within the DEC-POMDP model. For example, in the Fukushima nuclear disaster, it is very hard to model the experts' knowledge for the tasks on where to sample contaminated water and how to install a water gauge in the basement of the reactor buildings [11]. Therefore, the need to fully specify such tasks within the DEC-POMDP model limits the applicability of these solution techniques.

To address this, we propose PS-DEC-POMDP — an extension of DEC-POMDPs with partial state models. Specifically, we allow parts of the state variables not to be explicitly modeled. In other words, they are hidden state variables. This is very convenient because in many applications some state variables are often very difficult to be represented and modeled by DEC-POMDPs. However, without a complete state model, the transition and observation functions cannot be specified in their closed forms because they are functions of the states. Therefore, we need additional assumptions about the model. Firstly, we assume that there is a *realistic simulator*[1] for the domain. It allows us to test the agents policies and record histories (action-observation sequences of the agents during execution). In practice, it is often easier to devise a simulator than build a complete DEC-POMDP. Another challenge is to specify the reward model since in DEC-POMDPs it is also a function of the states. Fortunately, many domains have very sparse rewards for the agents (e.g., the agents are only rewarded when a task is completed). Therefore, our second assumption is that there is a *compact reward function* regarding hidden state variables, which can be aggregated by labels so that the reward function can be compactly represented using the state labels.

We also present novel algorithms to solve PS-DEC-POMDPs and address the following challenges: (a) only a simulator instead of a complete model is available; (b) there is no prior knowledge about how a hidden state maps to a state label. To address (a), we borrow ideas from *sample-based planning* [20] where: we first sample a set of histories based on an initial policy; then, we optimize the policies based on the history samples; after that, we use the current policies to re-sample the histories and compute a new policy. We iteratively repeat the sampling and optimization processes until no

DAI-19, October 13–15, 2019, Beijing, China
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-7656-3/19/10…$15.0
https://doi.org/10.1145/3356464.3357699

[1]Here, the simulator is not limited to softwares but includes physical test sites such as the ones for disaster training exercises.

improvement is possible in all agents' policies. During optimization, we must know the reward values of a given history. Therefore, for (b), we ask domain experts to select a state label for each time step of a history based on the information that it provides. Given a set of labeled histories, the reward values can be obtained using the compact reward function. When it is costly to query the experts, a method based on the *value of querying* that we propose can minimize the number of queries while bounding the error resulting from not obtaining experts' labels for some histories. The key observation is that the rewards are sparse and therefore state labels often do not change frequently (e.g., it takes time for the agents to complete a task and get the reward). We also present a method for learning a mapping from histories to state labels when a history is frequently queried. This paper advances the state-of-the-art of multi-agent planning as follow: 1) We propose novel algorithms that benefit from experts' knowledge for problems where only a simulator is available and the reward function is compactly represented. 2) We devise two methods that can reduce the the burden on human experts. One is based on the *value of querying* and the other is based on an online learning method. We tested our algorithms on a disaster recovery domain and demonstrated the advantages of our method when solving complex problems.

2 THE PS-DEC-POMDP MODEL

2.1 Formal Model

A *decentralized partially observable Markov decision process with partial state models* (PS-DEC-POMDP) is defined by tuple $\mathcal{M} = \langle I, S \times \Theta, \{A_i\}, \{\Omega_i\}, P, O, R, T\rangle$, where:

- I is a set of n agents where each agent $i \in I$.
- $S \times \Theta$ is a set of states where S denotes the known state variables and Θ denotes the hidden state variables.
- A_i is the action set for agent i where we denote $\vec{a}$ a joint action and $\vec{A} = \times_{i \in I} A_i$ the set of joint actions.
- Ω_i is the observation set for agent i where $\vec{o}$ is a joint observation and $\vec{\Omega} = \times_{i \in I} \Omega_i$ is the joint set.
- $P : S \times \Theta \times \vec{A} \times S \times \Theta \rightarrow [0, 1]$ is the transition function and $P((s', \theta')|(s, \theta), \vec{a})$ is the probability of the next state (s', θ') when the agents take $\vec{a}$ in state (s, θ).
- $O : S \times \Theta \times \vec{A} \times \vec{\Omega} \rightarrow [0, 1]$ is the observation function and $O(\vec{o}|(s', \theta'), \vec{a})$ is the probability of observing $\vec{o}$ after taking joint action $\vec{a}$ with outcome state (s', θ').
- $R : S \times \Theta \times \vec{A} \rightarrow \Re$ is the reward function and $R((s, \theta), \vec{a})$ is the reward when all the agents take $\vec{a}$ in state (s, θ).
- T is the planning horizon.

We use finite controllers to represent the policies, as is common in standard DEC-POMDPs [1, 16, 20]. Specifically, we define agent i's local policy as $\delta_i = \langle Q_i, \pi, \lambda\rangle$, where:

- Q_i is a finite set of controller nodes for agent i.
- π is an action selection function where $\pi(a_i|q_i)$ is the probability of selecting action a_i in $q_i \in Q_i$.
- λ is a node transition function and $\lambda(q_i'|q_i, o_i)$ is the probability of transiting from q_i to q_i' given o_i.

Similar to [20], each policy δ_i has T layers, each of which has fixed number of nodes, and nodes in a layer only transit to the nodes of the next layer. When executing δ_i, agent i first selects an action $a_i \sim \pi(\cdot|q_i)$ where q_i is its current node and transits to a next

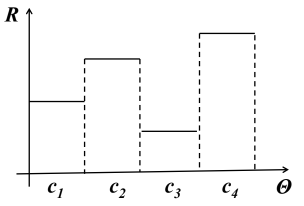


Figure 1: Compressed Reward Function

node $q_i' \sim \lambda(\cdot|q_i, o_i)$ based on its o_i. This process repeats until T is reached.

Here, A joint policy $\vec{\delta} = \langle \delta_1, \delta_2, \cdots, \delta_n\rangle$ is a collection of local policies, one for each agent. The goal of solving our model is to find a joint policy $\vec{\delta}^*$ that maximizes the expected value given the initial state distribution b^0:

$$\vec{\delta}^* = \arg\max_{\vec{\delta}} \mathbb{E}\left[\sum_{t=0}^{T-1} R((s^t, \theta^t), \vec{a}^t) \middle| b^0, \vec{\delta}\right] \tag{1}$$

2.2 Basic Assumptions

As aforementioned, we focus on problems where some state variables are hidden. To solve the problems, we make the following two assumptions:

- There is a realistic simulator that is available for testing agents' policies and recording histories.
- The rewards for the agents are sparse regarding hidden state variables and can be compactly represented.

In real-world applications, sophisticated simulators are often available for testing the system. Based on our first assumption, the outcome of the simulator is a joint history $\vec{h} = \langle h_1, h_2, \cdots, h_n\rangle$ where $h_i = (a_i^1, o_i^1, a_i^2, o_i^2, \cdots, a_i^t, o_i^t)$ is a local history for agent i. For the second assumption, we introduce the *compact reward function* defined as $R' : S \times C \times \vec{A} \rightarrow \Re$ where C is a set of state labels. Intuitively, C is a compact representation of the hidden state variables Θ. An example of state labels C is illustrated in Figure 1. It is easier to represent the compact reward function R' with state labels C than the original reward function R given hidden state variables Θ because we often have $|C| \ll |\Theta|$. For example, the problem considered in our experiments has huge hidden state space (i.e., $|\Theta| \approx 10^7$) but the rewards can be represented with a small number of state labels (i.e., $|C|$=4).

3 SOLVING PS-DEC-POMDPS

Two main challenges are inherent in PS-DEC-POMDPs. Firstly, the complete model is not available. We address this by computing policies with history samples drawing from a simulator. Secondly, we only have a compact reward function where we do not know which state label should be used for a given history. This is a very hard problem without knowing a complete model. Fortunately, we can query domain experts and ask them to choose a state label based on the underlying information from a given history.

The overall process is outlined in Algorithm 1. Given an initial joint policy, we sample histories by running the joint policy in the simulator. At each time step, a joint action is selected by the agents based on their component of the joint policy. Then, the joint action

Algorithm 1: Solving PS-DEC-POMDPs

```
Input: The model $\mathcal{M}$, an initial joint policy $\vec{\delta}$.
repeat
    $\vec{H} \leftarrow$ Sample joint histories from $\mathcal{M}$ with $\vec{\delta}$
    // Labeling joint histories
    foreach joint history $\vec{h} \in \vec{H}$ do
        if the predictor $\Phi(\vec{h})$ is sufficiently trained then
            $c \leftarrow$ Label $\vec{h}$ by the predictor $\Phi(\vec{h})$
        else
            $c \leftarrow$ Label $\vec{h}$ by the human experts
            $\Phi(\vec{h}) \leftarrow$ Train the predictor with label $c$
        $\vec{H} \leftarrow$ Update history $\vec{h}$ with label $c$
    // Planning with histories
    $\vec{\delta} \leftarrow$ Improve agents' policies with histories in $\vec{H}$
until joint policy $\vec{\delta}$ is converged.
return joint policy $\vec{\delta}$
```

is executed and each agent receives their own local observation from the environment. This repeats until the horizon is reached. During the process, we record the action and observation sequence of each agent. Then, we use them for labeling and planning as shown in the following sections with more detail.

3.1 Labeling Histories by Human Experts

To label a history, we ask domain experts to inspect the data (action and observation) at each time step of the history and select a state label based on their knowledge. For example, in robot systems, the history consists of a sequence of sensor reading (e.g., camera images) in time logged by the robots during task execution. Given this, the experts can replay the process using visualization tools and evaluate the performance of the robots. After that, a label is selected to reward (e.g., the robots complete a task) or penalize (e.g., the robots get trapped by an obstacle) them. Although the state labels can be arbitrary, we assume they are meaningful to the experts and relevant to the agents' performance.

Querying the experts may be costly. One useful observation is that we may not need to ask the experts for labeling every time step of some histories. For example, if a history is about the agents doing nothing, its label will remain unchanged for all steps because the rewards are not changing overtime. Thus, it is not worth querying the experts for every time step given these kinds of histories. To evaluate the benefit of asking the experts for labeling a history, we define *value of querying* (VoQ) as follows.

Let $\vec{c} = (c^0, c^1, \cdots, c^{T-1})$ be a sequence of labels and $\vec{C}$ the set of all possible such sequences. Our first type of VoQ is defined as: $VoQ_\infty(\vec{h}) \equiv V_{\max}(\vec{h}) - V_{\min}(\vec{h})$ where $V_{\max}(\vec{h}) = \max_{\vec{c}\in\vec{C}} V_{\vec{c}}(\vec{h})$, $V_{\min}(\vec{h}) = \min_{\vec{c}\in\vec{C}} V_{\vec{c}}(\vec{h})$. Thus, if $VoQ_\infty(\vec{h}) \le \varepsilon$ for history $\vec{h}$ where ε is a small number, we do not need to ask the experts for the label sequence $\vec{c}^*$ because for any label sequence $\forall \vec{c} \in \vec{C}$, we have:

$$\left\| V_{\vec{c}^*}(\vec{h}) - V_{\vec{c}}(\vec{h}) \right\| \le VoQ_\infty(\vec{h}) \le \varepsilon \tag{2}$$

Moreover, we may be able to query the label $\vec{c}_k$ from the experts every k steps and apply the same label within the interval. Given this, the VoQ for $\vec{c}_k$ is defined as:

$$VoQ_k(\vec{h}) \equiv \max\left\{ V_{\max}(\vec{h}) - V_{\vec{c}_k}(\vec{h}), V_{\vec{c}_k}(\vec{h}) - V_{\min}(\vec{h}) \right\} \tag{3}$$

and the cost of not querying the experts for the labels $\vec{c}^*$ at every step is bounded by:

$$\left\| V_{\vec{c}^*}(\vec{h}) - V_{\vec{c}_k}(\vec{h}) \right\| \le VoQ_k(\vec{h}) \le \varepsilon \tag{4}$$

if $VoQ_k(\vec{h})$ is no larger than ε. Now, the question is how to determine the proper interval k for a given history. Intuitively, if k were too small, there would not be a significant reduction in the number of querying. On the other hand, if k were too large, the error would be unbounded.

To address this, we propose a *divide-and-conquer* mechanism that can assist the experts to more efficiently label histories. Specifically, given a history $\vec{h}$, we start with computing the value of $VoQ_\infty(\vec{h})$. If $VoQ_\infty(\vec{h}) \le \varepsilon$, we remove $\vec{h}$ from the histories. Otherwise, we ask the experts to select a label c for the the whole history $\vec{h}$. Then, we generate a label sequence $\vec{c}_k$ by repeating c for the whole history where $k = |\vec{h}|$ and compute the value of $VoQ_k(\vec{h})$. If $VoQ_k(\vec{h}) \le \varepsilon$, we return $\vec{c}_k$ because it has already bounded the error. Otherwise, we select a time step t within history $\vec{h}$ and divide $\vec{h}$ into two sub-histories. We recursively repeat the above process for each sub-history. Finally, we combine the label sequences for each sub-history and return the result for $\vec{h}$.

Notice that the total number of queries initiated by our method is at most $|\vec{h}|$ but can be much smaller than $|\vec{h}|$ depending on the specific problem. Intuitively, our method is useful when some labels require no changes during a period of time. Here, we observed that the experts may need to go backwards a few steps to better understand the scenario but reviewing the entire history is often unnecessary. The outcome of our algorithm is a label sequence $\vec{c}$ made up of several subsequences, i.e., $\vec{c} = (\vec{c}_{k_1}, \vec{c}_{k_2}, \cdots, \vec{c}_{k_m})$, where each $\vec{c}_{k_j}$ consists of repeated instances of label c_{k_j}.

Theorem 1. *The error of the label sequence returned by our algorithm $\vec{c} = (\vec{c}_{k_1}, \vec{c}_{k_2}, \cdots, \vec{c}_{k_m})$ is bounded by $m\varepsilon$ where m is the total number of the subsequences.*

The proof of Theorem 3 can be found in the supplementary material. This is useful because it exploits the sparsity of the rewards in histories (labels are not changed frequently) and reduces the burden on the experts for labeling tasks.

3.2 Learning Predictors for State Labels

Although the hidden state variables Θ are unknown, we are still be able to train a predictor Φ using the state labels that we have already obtained. Then, we can use Φ to select labels during labeling phase. Specifically, the input of Φ is a joint history $\vec{h} \in \vec{H}$ and the output is a label $c \in C$. Formally, the learning problem is to train a predictor $\Phi : \vec{H} \to C$ given a set of labeled data $\langle(\vec{h}_1, c_1), (\vec{h}_2, c_2), \cdots, (\vec{h}_m, c_m)\rangle$.

In this paper, we use the *randomized weighted majority* (RWM) algorithm [4] to compute $\Phi(\vec{h})$ for joint history $\vec{h}$. It is an online learning method that can be easily integrated with our algorithm to train the predictor in parallel with the labeling process and output the prediction when required. Specifically, our adaption of RWM is as follows:

(1) Initialize the weights $w_1, w_2, \cdots, w_{|C|}$ to 1 where w_i is the weight for state label $c_i \in C$.
(2) Output c_i as a prediction of the state label given $\Phi(\vec{h})$ with the probability $p_i = w_i/W$ where $W = \sum_{i=1}^{|C|} w_i$.
(3) When the "correct" state label c^* is received from the experts for $\vec{h}$, penalize each "mistaken" $c_i \in C \backslash \{c^*\}$ by multiplying its weight by ρ, i.e., $\forall w_i, w_i \leftarrow \rho w_i$.

In RWM, a set of weights is maintained for $\vec{h}$, one for each state label $c \in C$. In the training phase, the weights are updated according to the choices of the experts for $\vec{h}$. In the predicting phase, the weights are used to compute a distribution over state labels C and select a label based on the distribution. Now, the key question is that when $\Phi(\vec{h})$ is sufficiently good for predicting the state label of $\vec{h}$.

Lemma 1 (Blum [4]). *The expected number of mistakes M made by the RWM algorithm satisfies:*

$$M \leq \frac{m \ln(1/\rho) + \ln|C|}{1-\rho} \tag{5}$$

where $0 < \rho < 1$ and m is the number of mistakes made by the most "correct" label $c \in C$ so far.

To compute m, we count the number of penalties applied to each label $c \in C$ and return the one with the minimum number. Here, the expected number of mistakes M can serve as a basis of the measurement on the quality of the prediction $\Phi(\vec{h})$ currently learned. Intuitively, it is less likely for the predictor to make mistakes if M is small. Let V_{error} be the maximum value gap for any two labels and K be the total number of training data points for $\vec{h}$.

Theorem 2. *The error in the expected value obtained by the predictor $\Phi(\vec{h})$ is bounded by:*

$$\left\| V(\vec{h}, \Theta) - V(\vec{h}, \Phi) \right\| \leq \frac{m \ln(1/\rho) + \ln|C|}{K(1-\rho)} V_{error} \tag{6}$$

Corollary 1. *Given an expected error bound ε, the sufficient size of the training data for $\vec{h}$ is:*

$$K \geq \frac{m \ln(1/\rho) + \ln|C|}{\varepsilon(1-\rho)} V_{error} \tag{7}$$

The proofs of Theorem 4 and Corollary 2 can be found in the supplementary material. In particular, Equation 22 can be used to decide when $\Phi(\vec{h})$ is sufficiently good for predicting labels for $\vec{h}$. Specifically, we set a *lower bound* $\mathcal{K}$ for the total number of training data K and start to use the predictor to predict the label when $K \geq \mathcal{K}$ as the predictor is considered as sufficiently trained.

3.3 Improving Policies with Labeled Histories

To improve the policies, we must find the best policy parameters π and λ for each agent that maximize the value function. Given a state distribution $b(s, \theta)$, the value function of a joint controller node $\vec{q}$ is defined by the Bellman equation:

$$\begin{aligned} V(b, \vec{q}) = \sum_{s,\theta} b(s,\theta) \sum_{\vec{a}} \pi(\vec{a}|\vec{q})[R((s,\theta),\vec{a}) + \\ \sum_{s',\theta',\vec{o},\vec{q}'} P((s',\theta')|(s,\theta),\vec{a}) O(\vec{o}|(s',\theta'),\vec{a}) \\ \lambda(\vec{q}'|\vec{q},\vec{o}) V(s',\theta',\vec{q}')] \end{aligned} \tag{8}$$

where $\pi(\vec{a}|\vec{q}) = \prod_i \pi(a_i|q_i), \lambda(\vec{q}'|\vec{q},\vec{o}) = \prod_i \lambda(q_i'|q_i,o_i)$. Let $\alpha(\vec{q},\vec{a}) \equiv \sum_{s,\theta} b(s,\theta) R((s,\theta),\vec{a})$ and $\beta(\vec{q},\vec{a},\vec{o},\vec{q}') \equiv \sum_{s,\theta} b(s,\theta) \sum_{s',\theta'} P((s',\theta') |(s,\theta),\vec{a}) O(\vec{o}|(s',\theta'),\vec{a}) V(s',\theta',\vec{q}')$. Equation 8 can be rewritten in a short form as follow:

$$\begin{aligned} V(b,\vec{q}) = \sum_{\vec{a}} \alpha(\vec{q},\vec{a}) \pi(\vec{a}|\vec{q}) + \sum_{\vec{a},\vec{o},\vec{q}'} \beta(\vec{q},\vec{a},\vec{o},\vec{q}') \cdot \\ \pi(\vec{a}|\vec{q}) \lambda(\vec{q}'|\vec{q},\vec{o}) \end{aligned} \tag{9}$$

If the values of α and β were computed, the policy parameters π and λ can be optimized given a joint node $\vec{q}$:

$$\begin{array}{ll} \max_{\pi,\lambda} & V(b,\vec{q}) \\ \text{s.t.} & \forall i \in I : \sum_{a_i \in A_i} \pi(a_i|q_i) = 1 \\ & \forall i \in I, o_i \in \Omega_i : \sum_{q_i' \in Q_i} \lambda(q_i'|q_i,o_i) = 1 \end{array} \tag{10}$$

where $V(b,\vec{q})$ is defined by Equation 9 and the constraints ensure that the parameters π and λ are probabilities. Note that this is a standard *nonlinear program* (NLP) that can be solved by standard NLP solvers (e.g., we use IPOPT[2]).

Unfortunately, in our problem, the complete model is not available. Hence, we cannot compute the values of α and β by the model according to their definitions. Instead, we estimate the values of α and β using the labeled histories. Specifically, for each joint history $\vec{h}$, we check whether $\vec{q}$ and $\vec{a}$ have been visited in $\vec{h}$. If so, the reward value is added to $\alpha(\vec{q},\vec{a})$. In more detail, we approximate α as:

$$\alpha(\vec{q},\vec{a}) \approx \frac{1}{|\vec{H}|} \sum_{\vec{h} \in \vec{H}} \sum_{t=1}^{T} r^t \cdot \phi(\vec{q},\vec{a}|\vec{h}^t) \tag{11}$$

where r^t is the reward value given the label at time t, $\phi(\vec{q},\vec{a}|\vec{h}^t) = 1$ if both $\vec{q},\vec{a}$ are visited in $\vec{h}$ at time t and 0 otherwise. Similarly, we approximate the values of β as:

$$\beta(\vec{q},\vec{a},\vec{o},\vec{q}') \approx \frac{1}{|\vec{H}|} \sum_{\vec{h} \in \vec{H}} \sum_{t=1}^{T} r^t \cdot \phi(\vec{q},\vec{a},\vec{o},\vec{q}'|\vec{h}^t) \tag{12}$$

where r^t is the reward value at time t, $\phi(\vec{q},\vec{a},\vec{o},\vec{q}'|\vec{h}^t) = 1$ if all $\vec{q},\vec{a},\vec{o},\vec{q}'$ are visited in $\vec{h}$ at time t and 0 otherwise.

The overall procedure is as follows. For every policy node $\vec{q}$, we compute the approximate values of $\forall\vec{a}, \alpha(\vec{a},\vec{a})$ and $\forall\vec{a},\vec{o},\vec{q}', \beta(\vec{q},\vec{a},\vec{o},\vec{q}')$ using the labeled histories according to Equations 11 and 12 respectively. Then, we compute a new set of parameters by solving the NLP as shown in Equation 10. In turn, we optimize the policy parameters of each node until no improvements in policy values are possible for all nodes. The final joint policy is returned as the result of policy improvement in Algorithm 1.

[2]`http://www.coin-or.org/Ipopt/`

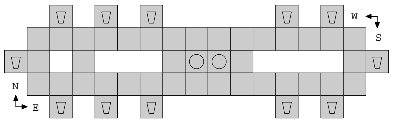


Figure 2: Power plant with 12 nuclear reactors.

Table 1: Rewards for the state labels.

Joint Actions		State Labels			
Robot 1	Robot 2	c_1	c_2	c_3	c_4
`repair-a`	`repair-a`	100	0	0	0
`repair-a`	`repair-b`	0	100	0	0
`repair-b`	`repair-a`	0	0	100	0
`repair-b`	`repair-b`	0	0	0	100

4 EMPIRICAL EVALUATION

4.1 Problem Setup

We tested our algorithms on a disaster recovery problem as shown in Figure 2. Specifically, we simulate a building of a power plant with 2 entrances in the middle and 12 nuclear reactors along the four sides. After a disaster, two mobile robots equipped with sensors and actuators are sent to repair the reactors. Specifically, each robot can navigate in the building with 4 actions (i.e., `stay`, `turn-left`, `turn-right`, and `move-forward`) and 3 observations indicating the object in front (i.e., `wall`, `other-robot`, and `free-space`). Each cell can only be occupied by one robot except for the cells with reactors. Apart from the actions and observations for navigation, each robot has 2 additional actions for repairing the reactors (i.e., `repair-a`, `repair-b`) and also 2 additional observations for the reactors. Each robot only receives these observations when it is in the same cell as the reactor.

We modeled this problem as a PS-DEC-POMDP. The known state variables are the robots' locations and orientations. Overall, there are 37,824 robot states. Each reactor has 4 states and there are 4^{12} = 16,777,216 reactor states. Note that the reactor states are hidden variables and not modeled. It is worth noting that in standard DEC-POMDPs the total number of states is $37{,}824\times16{,}777{,}216\approx6\times10^{11}$. Given the huge state space, it is very difficult to specify the model and even harder to solve it. However, it is straightforward to implement a simulator for this problem with two variables for the robot states and the other 12 variables for the reactor states.

Instead of specifying a reward function for 6×10^{11} states, we implemented 4 state labels that map to the 4 joint repairing actions. Accordingly, the compact reward function was specified in Table 1 where the reward is 100 only when the "correct" joint action is executed given the label and 0 otherwise. Intuitively, the mapping from the labels to the reactor states represents the domain knowledge of the experts and their ability to interpret the observation sequences, which could be arbitrarily complex in real-world settings. On one hand, which joint action should be rewarded depends on the label selected by the experts. On the other hand, how to select a label is based on the experts' domain knowledge (e.g., the reactors must be repaired in a specific order) and estimate of the reactor states.

4.2 Experimental Results

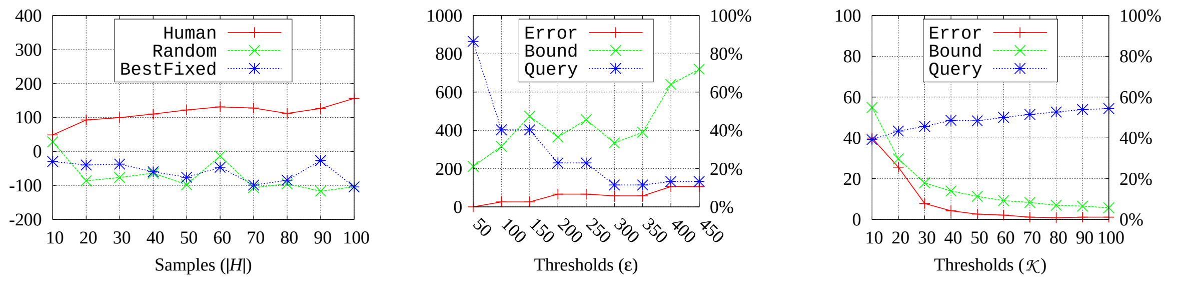
To test the algorithms, we first let the robots move around in the building and collect a set of histories using an initial policy. Next, we invited 30 people to label the histories with the instruction on which task should be performed by the robots in the tests (e.g., the reactors should be fixed in a specific order). A visualization tool is developed for them to replay the histories and view the robots' behavior. After that, we ran our algorithms to compute new policies for the robots with the labeled histories from the participants. We repeated this process with the new policies until it converged and output the final policies. To evaluate the performance of our algorithms, we executed the computed policies for 1000 runs and reported the average values.

Figure 3 summarizes our results in this domain. The first set of experiments tested our planning algorithm with different sizes of history samples (x-axis). In Figure 3(a), we compared the policy values (y-axis) computed using the participants' labels (i.e., `Human`) with two baselines (no human knowledge) in which the histories are randomly labeled (i.e., `Random`) or the label is fixed with the values of the best fixed label reported (i.e., `BestFixed`). As shown in the figure, the policies with human labeled data achieved better values than the other two baselines and the value increases with the sample size. This confirmed the importance of incorporating human's knowledge in such domains and thus the significance of our model and algorithm.

The second set of experiments tested the usefulness of the *value of querying* (VoQ) and the learned predicting model for labeling (RWM). In Figure 3(b), we reported the actual error (i.e. `Error`) between the labels produced by VoQ and the histories entirely labeled by participants (the left side of the y-axis) and the error bound (i.e., `Bound`) introduced by Theorem 3 (i.e., $m\varepsilon$) with different thresholds controlled by ε (x-axis). We also reported the percentage (i.e., `Query`) of the actual queries comparing to the total possible queries (the right side of the y-axis). As we can see from the figure, the actual error increases with ε and is bounded by $m\varepsilon$ while the number of queries drops very quickly (e.g., about 10% at 300) as ε increases. It is worth noting that the error bound is loose, which means that the actual performance is much better than the bound. Similarly in Figure 3(c), we reported the actual error (i.e., `Error`) of the labels produced by RWM, the error bound (i.e., `Bound`) defined by Theorem 4, and the percentage (i.e., `Query`) of the actual queries with different thresholds controlled by $\mathcal{K}$ (x-axis). We can see from the figure that the error decreases with $\mathcal{K}$ and is bounded as in Theorem 4. As expected, the number of queries increases since the algorithm needs more data for training the predictors. In contrast, the error bound here is tight, which indicates that Corollary 2 is useful to determine $\mathcal{K}$. These two results confirmed the usefulness of our algorithms for reducing the number of queries while bounding the errors.

5 RELATED WORK

Multi-agent systems are being increasingly deployed in real-world applications where people take an active part in the decision making and planning process [5, 7]. Agents in these settings need to be able to learn to perform new tasks, adapt to novel situations, and understand what their human users want. For instance, the MAPGEN system, used for daily planning for Mars exploration rovers, allows



(a) Reward Values (b) VoQ: Errors vs. Queries (c) RWM: Errors vs. Queries

Figure 3: Experimental results.

human tactical activity planners to use it in an interactive mode by selecting and placing new activities on timelines [5]. The HAC system, devised for coordinating the activities of first responders in disaster scenarios, allows them to make modifications or replan based on their preferences [7]. The key challenge of these systems is to efficiently transfer the experts' knowledge or preferences to agents and guide their decisions. In the single-agent case, work has been done using *inverse reinforcement learning* [13] to extract rewards from the behavior of an expert. Most recently, the TAMER framework has been proposed to train an agent via *interactive shaping* [8, 9] with human-generated rewards. However, it is very difficult to computed decentralized policies by the aforementioned methods due to the partial observability of the agents and huge policy space of DEC-POMDPs.

Given a complete model of DEC-POMDPs, it can be solved either optimally or approximately by many existing methods [1, 2, 6, 10, 12, 14, 17, 18, 20]. Among them, our planning is mostly related to [20] where they also compute a decentralized policies using samples drawing from a simulator. However, they assume that the states are fully observable and can be reset at any time by the simulator. Therefore, our problem cannot be solved by their method. In terms of alternatively optimization using nonlinear programs, our work is related to [1]. Again, they require a complete model with the transition, observation, and reward functions fully specified. This makes it unsuitable for our settings since the model is partially specified. Regarding the compact reward function, our work is related to [15, 19] where they also exploit the compact representation of DEC-POMDPs. Unlike their work, we make no assumption on the agents' interaction and use a small set of state labels to represent the reward function. Moveover, the decision on how to select a state label for a given history is made by human experts during planning.

6 CONCLUSIONS

We address settings where additional state variables or domain knowledge is possessed by people, but is difficult to incorporate into the domain model. The solution we propose is to directly include the human experts in the loop and allow them to interactively train the agents' policies during planning time. Specifically, the experts are asked to select labels given sampled histories and then map the selected labels to the reward signals for the histories. We propose a compact reward model to consider the labels and presented algorithms to compute joint policies using the sampled histories. To minimize the number of queries to the experts, a divided-and-conquer method is presented to interact with the experts based on the *value of querying*. Additionally, we propose an online learning method to learn the mapping from histories to labels in parallel to the planning process. We tested our algorithms on a disaster recovery domain where the repair of nuclear reactors can only be carried out with the knowledge of the experts. Experimental results with different settings confirm the benefits of our model and the proposed algorithms. In the future, we plan to test our algorithms on real-world applications and investigate more human factors with crowdsourcing platforms.

REFERENCES

[1] Christopher Amato, Daniel S Bernstein, and Shlomo Zilberstein. 2010. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *Autonomous Agents and Multi-Agent Systems* 21, 3 (2010), 293–320.
[2] Chistopher Amato, Jilles Steeve Dibangoye, and Shlomo Zilberstein. 2009. Incremental Policy Generation for Finite-Horizon DEC-POMDPs. In *Proc. of the 19th Int'l Conf. on Automated Planning and Scheduling*. 2–9.
[3] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research* 27, 4 (2002), 819–840.
[4] Avrim Blum. 1998. *On-line algorithms in machine learning*. Springer.
[5] John L Bresina, Ari K Jónsson, Paul H Morris, and Kanna Rajan. 2005. Activity Planning for the Mars Exploration Rovers.. In *Proc. of the 15th Int'l Conf. on Automated Planning and Scheduling*. 40–49.
[6] Jilles Steeve Dibangoye, Christopher Amato, Olivier Buffet, and François Charpillet. 2013. Optimally Solving Dec-POMDPs as Continuous-State MDPs. In *Proc. of the 23d Int'l Joint Conf. on Artificial Intelligence*.
[7] Nicholas R Jennings, Luc Moreau, David Nicholson, Sarvapali D Ramchurn, Stephen J Roberts, T Rodden, and Alex Rogers. 2014. On human-agent collectives. *Commun. ACM* (2014).
[8] W Bradley Knox and Peter Stone. 2009. Interactively shaping agents via human reinforcement: The TAMER framework. In *Proc. of the 5th Int'l Conf. on Knowledge Capture*. 9–16.
[9] W Bradley Knox and Peter Stone. 2010. Combining manual feedback with subsequent MDP reward signals for reinforcement learning. In *Proc. of the 9th Int'l Conf. on Autonomous Agents and Multiagent Systems*. 5–12.
[10] Akshat Kumar and Shlomo Zilberstein. 2010. Point-Based Backup for Decentralized POMDPs: Complexity and New Algorithms. In *Proc. of the 9th Int'l Conf. on Autonomous Agents and Multiagent Systems*. 1315–1322.
[11] Keiji Nagatani, Seiga Kiribayashi, Yoshito Okada, Satoshi Tadokoro, Takeshi Nishimura, Tomoaki Yoshida, Eiji Koyanagi, and Yasushi Hada. 2011. Redesign of rescue mobile robot Quince. In *Proc. of the 2011 IEEE Int'l Symposium on Safety, Security, and Rescue Robotics*. 13–18.
[12] Ranjit Nair, Milind Tambe, Makoto Yokoo, David V. Pynadath, and Stacy Marsella. 2003. Taming Decentralized POMDPs: Towards Efficient Policy Computation for Multiagent Settings. In *Proc. of the 18th Int'l Joint Conf. on Artificial Intelligence*. 705–711.
[13] Andrew Y Ng and Stuart J Russell. 2000. Algorithms for inverse reinforcement learning.. In *Proc. of the 17th Int'l Conf. on Machine Learning*. 663–670.

[14] Frans A Oliehoek, Matthijs TJ Spaan, Christopher Amato, and Shimon Whiteson. 2013. Incremental Clustering and Expansion for Faster Optimal Planning in Decentralized POMDPs. *Journal of Artificial Intelligence Research* 46 (2013), 449–509.

[15] Frans A Oliehoek, Matthijs TJ Spaan, Shimon Whiteson, and Nikos Vlassis. 2008. Exploiting locality of interaction in factored Dec-POMDPs. In *Proc. of the 7th Int'l Joint Conf. on Autonomous Agents and Multiagent Systems*. 517–524.

[16] Joni Pajarinen and Jaakko Peltonen. 2011. Periodic Finite State Controllers for Efficient POMDP and DEC-POMDP Planning. In *Proc. of the 25th Conf. on Neural Information Processing Systems*. 2636–2644.

[17] Sven Seuken and Shlomo Zilberstein. 2007. Memory-Bounded Dynamic Programming for DEC-POMDPs. In *Proc. of the 20th Int'l Joint Conf. on Artificial Intelligence*. 2009–2015.

[18] Matthijs T. Spaan, Frans A. Oliehoek, and Christopher Amato. 2011. Scaling up optimal heuristic search in Dec-POMDPs via incremental expansion. In *Proc. of the 22nd Int'l Joint Conf. on Artificial Intelligence*. 2027–2032.

[19] Stefan J Witwicki and Edmund H Durfee. 2011. Towards a unifying characterization for quantifying weak coupling in Dec-POMDPs. In *Proc. of the 10th Int'l Conf. on Autonomous Agents and Multiagent Systems*. 29–36.

[20] Feng Wu, Shlomo Zilberstein, and Xiaoping Chen. 2010. Rollout sampling policy iteration for decentralized POMDPs. In *Proc. of the 26th Conf. on Uncertainty in Artificial Intelligence*. 666–673.

A PROOFS OF THEOREMS

Theorem 3. *The error of the label sequence returned by our algorithm $\vec{c} = (\vec{c}_{k_1}, \vec{c}_{k_2}, \cdots, \vec{c}_{k_m})$ is bounded by $m\varepsilon$ where m is the total number of the subsequences.*

Proof. Let $\vec{c}$ be the label sequence returned by our algorithm for $\vec{h}$ and $\vec{c}^*$ be the "optimal" sequence obtained by querying the experts for every time step in $\vec{h}$. Apparently, in order to obtain $\vec{c}^*$, we must query the experts for $|\vec{h}|$ times. The goal of our algorithm is to approximate $\vec{c}^*$ with $\vec{c}$ where the number of querying for computing $\vec{c}$ can be much less than the total querying of $\vec{c}^*$, i.e., $m \ll |\vec{h}|$, while bound their difference (error) in terms of the expected values:

$$V_{error}(\vec{c}, \vec{h}) = \left\| V_{\vec{c}^*}(\vec{h}) - V_{\vec{c}}(\vec{h}) \right\| \tag{13}$$

If the algorithm terminates with $VoQ_\infty(\vec{h}) \le \varepsilon$, according to the definition, the difference in values between any sequences (including the "optimal" sequence from the experts) is less than ε. In this case, we arbitrarily select a single label c for the whole $\vec{h}$. Thus, the error is bounded by ε as:

$$\left\| V_{\vec{c}^*}(\vec{h}) - V_{\vec{c}}(\vec{h}) \right\| \le VoQ_\infty(\vec{h}) \le \varepsilon \tag{14}$$

with $m = 1$ where $\vec{c}$ is a sequence that repeats c for all the steps. Otherwise, a procedure is called in the algorithm to check the value of $VoQ_k(\vec{h})$ with different interval k.

To achieve this, we first query the experts for a label c at the last step of the joint history and compute VoQ_k assuming that c is used for the entire joint history. If $VoQ_k(\vec{h}) \le \varepsilon$, we return with a repeat of c. Otherwise, we recursively divide the joint history into two segments and compute VoQ_k for reach segments. Finally, we merge all the results and return the whole sequence $\vec{c} = (\vec{c}_{k_1}, \vec{c}_{k_2}, \cdots, \vec{c}_{k_m})$. Notice that we cache the querying results so each time step of the joint history is at most queried once.

Let $\vec{c}_k \subseteq \vec{c}$ be a subsequence of $\vec{c}$ that consists of repeated instances of one label c_k and m is the total number of such subsequences in $\vec{c}$. The expected value of $\vec{c}$ is written as:

$$\begin{aligned} V_{\vec{c}}(\vec{h}) &= V_{\vec{c}_{k_1}}(\vec{h}_{k_1}) + V_{\vec{c}_{k_2}}(\vec{h}_{k_2}) + \cdots + V_{\vec{c}_{k_m}}(\vec{h}_{k_m}) \\ &= \sum_{\vec{c}_k \subseteq \vec{c}} V_{\vec{c}_k}(\vec{h}_k) \end{aligned} \tag{15}$$

The error in terms of expected values between $\vec{c}$ and $\vec{c}^*$ is:

$$\begin{aligned} \|V_{\vec{c}^*}(\vec{h}) - V_{\vec{c}}(\vec{h})\| &= \|V_{\vec{c}^*}(\vec{h}) - \sum_{\vec{c}_k \subseteq \vec{c}} V_{\vec{c}_k}(\vec{h}_k)\| \\ &= \|\sum_{\vec{c}_k \subseteq \vec{c}} [V_{\vec{c}^*_k}(\vec{h}_k) - V_{\vec{c}_k}(\vec{h}_k)]\| \\ &\le \sum_{\vec{c}_k \subseteq \vec{c}} \varepsilon = m\varepsilon \end{aligned} \tag{16}$$

where $\vec{c}^*_k$ is a subsequence of $\vec{c}^*$ with the same starting and ending time as $\vec{c}_k$. Thus, comparing to the "optimal" sequence from the experts, the error is bounded by $m\varepsilon$, i.e.,

$$V_{error}(\vec{c}, \vec{h}) \le m\varepsilon \tag{17}$$

for the label sequence $\vec{c}$ computed by our algorithm given the joint history $\vec{h}$. □

Lemma 2 (Blum [4]). *The expected number of mistakes M made by the RWM algorithm satisfies:*

$$M \le \frac{m \ln(1/\rho) + \ln|C|}{1-\rho} \tag{18}$$

where $0 < \rho < 1$ and m is the number of mistakes made by the most "correct" task $c \in C$ so far.

Please refer to Blum [4] for the detail proof.

Theorem 4. *The error in the expected value obtained by the predictor $\Phi(\vec{h})$ is bounded by:*

$$\left\| V(\vec{h}, \Theta) - V(\vec{h}, \Phi) \right\| \le \frac{m \ln(1/\rho) + \ln|C|}{K(1-\rho)} V_{error} \tag{19}$$

where $V_{error} = \max_{s,c} R(s, c, \vec{a}) - \min_{s,c} R(s, c, \vec{a})$ is the maximum gap in value for any two labels and K is the total number of training data obtained so far for $\vec{h}$.

Proof. Let M be the expected number of mistakes made by the predictor Φ, K be the total number of training data obtained so far for $\vec{h}$, and m be the number of mistakes made by the most "correct" label so far. In other words, we train the predictor with K examples. Each example is a label c for $\vec{h}$ labeled by the experts. In the algorithm, given an example, we penalize the labels that are different from the example. Let P_c be the total number of penalties applied to label c given the K examples. The number of mistakes made by the most "correct" label so far can be computed as $m = \min_{c \in C} P_c$.

Since M is the expected number of mistakes made by the predictor in the K trials, the error rate is M/K. Given that the error for any two labels is less than or equal to V_{error}, we have:

$$\|V(\vec{h}, \Theta) - V(\vec{h}, \Phi)\| \le \frac{M}{K} V_{error} \tag{20}$$

Thus, Equation 19 holds by applying Lemma 2:

$$\begin{aligned} ||V(\vec{h}, \Theta) - V(\vec{h}, \Phi)|| &\leq \frac{M}{K} V_{error} \\ &\leq \frac{m \ln(1/\rho) + \ln |C|}{K(1-\rho)} V_{error} \end{aligned} \tag{21}$$

Note that V_{error} is a constant given $\vec{h}$. Therefore, the error bound on the right-hand side is determined by m and K. This concludes our proof that the predictor learned by RWM is error-bounded. □

Corollary 2. *Given an expected error bound ε, the sufficient size of the training data for $\vec{h}$ is:*

$$K \geq \frac{m \ln(1/\rho) + \ln |C|}{\varepsilon(1-\rho)} V_{error} \tag{22}$$

PROOF. Given an expected error bound ε, the goal is to guarantee that:

$$||V(\vec{h}, \Theta) - V(\vec{h}, \Phi)|| \leq \varepsilon \tag{23}$$

According to Theorem 4, we can have:

$$\left\| V(\vec{h}, \Theta) - V(\vec{h}, \Phi) \right\| \leq \frac{m \ln(1/\rho) + \ln |C|}{K(1-\rho)} V_{error} \leq \varepsilon \tag{24}$$

Thus, we can guarantee the error bound if the size of training data:

$$K \geq \frac{m \ln(1/\rho) + \ln |C|}{\varepsilon(1-\rho)} V_{error} \tag{25}$$

This concludes our proof. □