Monte-Carlo Expectation Maximization for Decentralized POMDPs

Feng Wu[†] **Shlomo Zilberstein**[‡] **Nicholas R. Jennings**[†]

[†] School of Electronics and Computer Science, University of Southampton, UK
[‡] School of Computer Science, University of Massachusetts Amherst, USA

Abstract

We address two significant drawbacks of state-of-the-art solvers of decentralized POMDPs (DEC-POMDPs): the reliance on complete knowledge of the model and limited scalability as the complexity of the domain grows. We extend a recently proposed approach for solving DEC-POMDPs via a reduction to the maximum likelihood problem, which in turn can be solved using EM. We introduce a model-free version of this approach that employs Monte-Carlo EM (MCEM). While a naïve implementation of MCEM is inadequate in multi-agent settings, we introduce several improvements in sampling that produce high-quality results on a variety of DEC-POMDP benchmarks, including large problems with thousands of agents.

1 Introduction

Decentralized partially observable Markov decision processes (DEC-POMDPs) offer a powerful model for multi-agent coordination and decision making under uncertainty. A rich set of techniques have been developed to solve DEC-POMDPs optimally [Hansen *et al.*, 2004; Szer *et al.*, 2005; Aras *et al.*, 2007; Spaan *et al.*, 2011] or approximately [Nair *et al.*, 2003; Seuken and Zilberstein, 2007; Bernstein *et al.*, 2009; Wu *et al.*, 2010; Dibangoye *et al.*, 2011; Pajarinen and Peltonen, 2011b]. However, most existing methods assume complete prior knowledge of the model. That is, a typical DEC-POMDP solver takes as input the *full* model parameters, including transition, observation and reward functions. Additionally, the high computational complexity of DEC-POMDPs further limits the scalability of solution methods that rely on a complete model description. These drawbacks are particularly critical when applying DEC-POMDP solvers to complex domains, such as space exploration, disaster response, and weather monitoring, where exact models are extremely large and hard to obtain.

Recently, several *model-free* algorithms have been proposed to learn decentralized policies for *finite-horizon* DEC-POMDPs. Specifically, Zhang and Lesser [2011] proposed a scalable distributed learning method for ND-POMDPs [Nair *et al.*, 2005], a special class of DEC-POMDPs where agents are organized in a network structure. Banerjee *et al.* [2012] proposed a distributed learning method for the general case of finite-horizon DEC-POMDPs, where agents take turns to learn best responses to each other's policies. However, both methods rely on reinforcement learning techniques that produce a Q-value function that grows exponentially in the problem horizon. Thus, these approaches are unsuitable for DEC-POMDPs with large or infinite horizons.

In this paper, we focus on *infinite-horizon* DEC-POMDPs. Our approach is inspired by recent advances in planning by probabilistic inference [Toussaint and Storkey, 2006; Toussaint *et al.*, 2008], where the planning problem is reformulated as likelihood maximization in a mixture of dynamic Bayesian networks and solved by Expectation-Maximization (EM) algorithms [Dempster *et al.*, 1977]. Most recently, this concept has been successfully applied to solve *infinite-horizon* DEC-POMDPs [Kumar and Zilberstein, 2010]. However, like other *model-based* DEC-POMDP algorithms, this approach requires complete knowledge of the model. Moreover, even with a full model, the EM algorithms do not scale well to problems with many agents. Other EM-based approaches usually rely on additional problem structure to improve scalability [Kumar *et al.*, 2011; Pajarinen and Peltonen, 2011a].

In contrast to these methods, our approach is *model-free* and does not require any additional structure of the models. Specifically, we use the Monte-Carlo EM (MCEM) [Wei and Tanner, 1990] where the costly E-step (the main bottleneck of the model-based EM algorithms for DEC-POMDPs) is performed by sampling from the model with importance weights corresponding to the sampled rewards. Similar ideas have been used to learn policies in single-agent (PO)MDPs [Vlassis and Toussaint, 2009; Vlassis *et al.*, 2009]. However, a direct application of these methods results in inefficient sampling for DEC-POMDPs given the huge joint policy space. The main contribution of our approach is the development of several efficient sampling techniques together with MCEM to learn the policies of DEC-POMDPs with thousands of agents. Furthermore, we show that our algorithm can be parallelized using the MapReduce paradigm [Dean and Ghemawat, 2008] and thereby can scale up well to large problems where the sampling process is expensive and time-consuming. The experimental results on common DEC-POMDP benchmarks and large multi-agent coordination problems (with 100 and 2500 agents) confirm the effectiveness of our algorithm.

The reminder of the paper is organized as follows. In Section 2, we briefly introduce the necessary background. In Section 3, we propose our main algorithm and analyze its properties. In Section 4, we present experimental results on several benchmark problems. We conclude with a summary of the contributions and future work.

2 Background

2.1 The DEC-POMDP Model

Formally, a *Decentralized Partially Observable Markov Decision Process* (DEC-POMDP) [Bernstein *et al.*, 2002] is defined as a tuple $\langle I, S, \{A_i\}, \{\Omega_i\}, P, O, R, b^0, \gamma\rangle$, where:

- I is a collection of n agents $i \in I$.
- S is a finite set of states for the system $s \in S$.
- A_i is a set of actions for agent i. We denote by $\vec{a} = \langle a_1, a_2, \cdots, a_n\rangle$ a joint action where $a_i \in A_i$ and $\vec{A} = \times_{i\in I} A_i$ is the joint action set.
- Ω_i is a set of observations for agent i. Similarly, $\vec{o} = \langle o_1, o_2, \cdots, o_n\rangle$ is a joint observation where $o_i \in \Omega_i$ and $\vec{\Omega} = \times_{i\in I}\Omega_i$ is the joint observation set.
- $P : S \times \vec{A} \times S \to [0, 1]$ is the transition function and $P(s'|s, \vec{a})$ denotes the probability of the next state s' where the agents take joint action $\vec{a}$ in state s.
- $O : S \times \vec{A} \times \vec{\Omega} \to [0, 1]$ is the observation function and $O(\vec{o}|s', \vec{a})$ denotes the probability of observing $\vec{o}$ after taking joint action $\vec{a}$ with outcome state s'.
- $R : S \times \vec{A} \to \Re$ is the reward function.
- $b^0 \in \Delta(S)$ is the initial state distribution.
- $\gamma \in (0, 1]$ is the discount factor.

For infinite-horizon DEC-POMDPs, the joint policy $\theta = \langle\theta_1, \theta_2, \cdots, \theta_n\rangle$ is usually represented as a set of *finite state controllers* (FSCs), one for each agent. Each FSC can be defined as a tuple $\theta_i = \langle Q_i, \nu_{q_i}, \pi_{a_i q_i}, \lambda_{q_i' q_i o_i}\rangle$, where:

- Q_i is a set of controller nodes $q_i \in Q_i$.
- $\nu_{q_i} \in \Delta(Q_i)$ is the initial distribution over nodes.
- $\pi_{a_i q_i}$ is the probability of executing action a_i in node q_i.
- $\lambda_{q_i' q_i o_i}$ is the probability of the transition from current node q_i to next node q_i' when observing o_i.

The goal is to find a joint policy θ^* that maximizes the expected value $V(\theta^*) = \mathbb{E}[\sum_{t=1}^{\infty}\gamma^t R^t|b^0, \theta^*]$.

2.2 The EM Algorithm

Recently, the *Expectation Maximization* (EM) algorithm has been proposed to efficiently solve infinite-horizon DEC-POMDPs [Kumar and Zilberstein, 2010]. To apply the EM approach, the reward function is scaled into the range $[0, 1]$ so that it can be treated as a probability: $\hat{R}(r = 1|s, \vec{a}) = [R(s, \vec{a}) - R_{min}]/(R_{max} - R_{min})$ where R_{min} and R_{max} are the minimum and maximum reward values and $\hat{R}(r = 1|s, \vec{a})$ is interpreted as the conditional probability of the binary reward variable r being 1. The FSC parameters θ are then optimized by maximizing the reward likelihood with respect to θ: $L(\theta) = \sum_{T=0}^{\infty} P(T)P(\hat{R}|T; \theta)$, where $P(T) = (1 - \gamma)\gamma^T$. This is equivalent to maximizing the expected discounted reward $V(\theta)$ in the DEC-POMDP.

At each iteration, the EM approach consists of two main steps: E and M. In the E-step, alpha messages $\hat{\alpha}(\vec{q}, s)$ and beta messages $\hat{\beta}(\vec{q}, s)$ are computed. Specifically, the alpha message $\hat{\alpha}(\vec{q}, s)$ can be computed by starting from the initial node and state distributions and projecting it T times forward: $\hat{\alpha}(\vec{q}, s) = \sum_{t=0}^{T}\gamma^t(1 - \gamma)\alpha_t(\vec{q}, s)$, where $\alpha_t(\vec{q}, s)$ is the probability of being in state s and nodes $\vec{q}$ at time t. The message $\hat{\beta}(\vec{q}, s)$ can be computed similarly: $\hat{\beta}(\vec{q}, s) = \sum_{t=0}^{T}\gamma^t(1 - \gamma)\beta_t(\vec{q}, s)$, where $\beta_t(\vec{q}, s)$ is the reward probability for being in state s and nodes $\vec{q}$ when projecting t time steps backwards. In the M-step, the FSC parameters are updated using the method of *Largrange* multipliers.

2.3 Related Work

To date, several approaches have been proposed to learn policies using sampling methods. Specifically, Zhang *et al.* [2007] represent policies as conditional random fields (CRFs) and use tree MCMC to sample CRFs. They assume that agents can communicate instantaneously and know the interaction structures. The approach that is closest to ours is Stochastic Approximation EM (SAEM) [Vlassis and Toussaint, 2009; Vlassis *et al.*, 2009] for (PO)MDPs. In that work, a new mixture model is proposed for the value functions and a Q-value function is learned by maximizing the likelihood of the new mixture model. In doing so, the (PO)MDP policies (a mapping from (belief) states to actions) can be straightforwardly induced from the learned Q function. In contrast, we want to learn the parameters of FSCs for infinite-horizon DEC-POMDPs. Specifically, our work addresses the questions: (1) how to pick up the initial controller node for each agent (ν_{q_i}); (2) how to choose an action for a node of the agent ($\pi_{a_i q_i}$); and (3) how to transition to a next node given a current node and an observation of the agent ($\lambda_{q_i' q_i o_i}$). Furthermore, given the fact that the policy space of DEC-POMDPs is substantially larger than the policy space of (PO)MDPs, direct application of sampling techniques is very inefficient. Thus, we propose and test several methods that improve sampling performance.

3 Monte-Carlo Expectation Maximization

As stated above, the goal of solving an infinite-horizon DEC-POMDP is to find a joint FSC θ^* that maximizes the discounted expected value $V(\theta^*)$. In standard algorithms, this value can be iteratively optimized by the Bellman equation. Alternatively, Toussaint and Storkey [2006] show that this optimization problem can be translated into a problem of likelihood maximization in a mixture of finite-length processes where $L(\theta) \propto (1-\gamma)V(\theta)$. Thus, maximizing the likelihood $L(\theta)$ is equivalent to maximizing $V(\theta)$ in the policy space. When the full model is given, this problem can be solved by computing backward and forward messages and optimizing the policy parameters with the EM algorithm. In model-free settings, we can use trajectory samples drawn from DEC-POMDPs to estimate the likelihood value $L(\theta)$.

In more detail, the sampling process starts with the system picking an initial state and each agent i selects an initial controller node $q_i^0 \sim \nu_{q_i}$ from its FSC θ_i. At time step t, each

agent i chooses an action $a_i^t \sim \pi_{a_i q_i}$ and executes it (or simulates its execution). Given the joint action $\vec{a}^t$, the system transitions from the current state to a new state and outputs the immediate reward r^t. Then, each agent i receives its own observation o_i^t and moves from its current node q_i^t to a new node $q_i^{t+1} \sim \lambda_{q_i' q_i o_i}$ based on the observation o_i^t. This process repeats until the problem horizon T is reached. In the sampling process, we record the data $\langle \vec{a}^t, r^t, \vec{o}^t, \vec{q}^t \rangle$ for every time step t. Samples can be generated either from simulators or real-world settings. The following section describes how to optimize the policy parameters using these samples.

3.1 Policy Optimization

Let $\xi = (x^0, x^1, x^2, \cdots, x^T)$ be a length-T trajectory where $x^t = \langle \vec{a}^t, r^t, \vec{o}^t, \vec{q}^t \rangle$, and $\mathcal{X}^T = \{\xi : |\xi| = T\}$ be the space of T-length trajectories. Given a joint policy θ, the joint distribution of a T-length trajectory ξ is denoted by $P(\hat{R}, \xi, T; \theta)$. In the EM algorithm, we assume $\hat{R}$ is observed and we want to find parameters θ of the joint policy that maximize the likelihood $P(\hat{R}; \theta) = \sum_{T,\xi} P(\hat{R}, \xi, T; \theta)$, where T, ξ are latent (hidden) variables. Let $q(\xi, T)$ be a distribution over these latent variables. The energy function that we want to maximize in the EM algorithm becomes:

$$\begin{aligned} F(\theta, q) &= \log P(\hat{R}; \theta) - D_{KL}(q(\xi, T) || P(\xi, T | \hat{R}; \theta)) \\ &= \sum_{T,\xi} q(\xi, T) \log P(\hat{R}, \xi, T; \theta) + H(q) \end{aligned} \tag{1}$$

where D_{KL} is the Kullback-Leibler divergence between $q(\xi, T)$ and $P(\xi, T | \hat{R}; \theta)$, and $H(q)$ is the entropy of q, which is independent of the distribution of θ.

Starting with an initial guess of θ, the EM algorithm iteratively finds a q that maximizes F for a fixed θ in the E-step, and then it finds a new θ that maximizes F for a fixed q in the M-step. This procedure converges to a local maximum of F. In the E-step, we can see that the optimal distribution q^* that maximizes F is the Bayes posterior computed with parameters θ^{old} produced by the last M-step:

$$q^*(\xi, T) = P(\xi, T | \hat{R}; \theta^{old}) \tag{2}$$

because the KL-divergence measure is always nonnegative and becomes zero only when the two distributions are equal. In the M-step, we maximize F over θ using the optimal q^*. By ignoring terms that do not depend on θ, the energy function can be expressed by:

$$\begin{aligned} F(\theta, q^*) &= \mathbb{E}_{P(\xi, T | \hat{R}; \theta^{old})}[\log P(\hat{R}, \xi, T; \theta)] \\ &= \sum_{T,\xi} P(\xi, T | \hat{R}; \theta^{old}) \log P(\hat{R}, \xi, T; \theta) \end{aligned} \tag{3}$$

In Monte-Carlo EM (MCEM) [Wei and Tanner, 1990], the expectation in the E-step is estimated by drawing samples from the conditional distribution of the latent variables $P(\xi, T | \hat{R}; \theta^{old})$ given the observed data $\hat{R}$ and θ. More specifically, if we obtain m trajectories $\xi_1, \xi_2, \cdots, \xi_m$ the expectation can be estimated by the Monte-Carlo sum:

$$\tilde{F}(\theta, q^*) = \frac{1}{m} \sum_{k=1}^{m} \log P(\hat{R}, \xi_k, T; \theta) \tag{4}$$

By the law of large numbers, the estimator $\tilde{F}$ converges to the theoretical expectation F when m is sufficiently large.

To sample a trajectory ξ from the conditional distribution $P(\xi, T | \hat{R}; \theta^{old}) \propto P(T) P(\xi | T; \theta^{old}) P(\hat{R} | \xi, T; \theta^{old})$, we first sample a maximal length T from $P(T) = (1-\gamma)\gamma^T$, then sample a T-length trajectory $\xi \sim P(\xi | T; \theta^{old})$ from the DEC-POMDP model using the joint policy θ^{old}, and then sample the binary reward $r = 1$ from the distribution $\hat{R}(r = 1 | s^T, \vec{a}^T)$ using the last state s^T and the last joint action $\vec{a}^T$ in the trajectory ξ. In fact, the process of sampling a T-length trajectory is equivalent to simulating the run of joint policy θ^{old} in a T-step DEC-POMDP.

In the M-step, we compute new parameters θ that maximize the estimated energy function:

$$\begin{aligned} \tilde{F}(\theta, q^*) = \frac{1}{m} \sum_{k=1}^{m} \log P(\hat{R}, \xi_k, T; \theta) = \cdots \\ + \sum_{i \in I} \frac{1}{m} \sum_{k=1}^{m} \xi_k^0(q_i) \log \nu_{q_i} \\ + \sum_{i \in I} \frac{1}{m} \sum_{k=1}^{m} \sum_{t=0}^{T-1} \xi_k^t(a_i, q_i) \log \pi_{a_i q_i} \\ + \sum_{i \in I} \frac{1}{m} \sum_{k=1}^{m} \sum_{t=0}^{T-1} \xi_k^t(q_i', q_i, o_i) \log \lambda_{q_i' q_i o_i} \end{aligned} \tag{5}$$

where the notation $\cdots$ represents the terms that are independent of the parameters θ, and $\xi_k^t(\cdot)$ is equal to 1 if the arguments $(\cdot)$ are the t-th elements of trajectory ξ_k and 0 otherwise. Then, we can apply the method of *Lagrange* multipliers to find the new parameters $\theta = \arg\max_{\theta'} \tilde{F}(\theta', q^*)$:

$$\begin{aligned} &\nabla_{\nu_{q_i}} [\tilde{F}(\theta, q^*) - C_i (\sum_{q_i \in Q_i} \nu_{q_i} - 1)] = 0 \Rightarrow \\ &\nu_{q_i} = \frac{1}{C_i} [\frac{1}{m} \sum_{k=1}^{m} \xi_k^0(q_i)] \end{aligned} \tag{6}$$

$$\begin{aligned} &\nabla_{\pi_{a_i q_i}} [\tilde{F}(\theta, q^*) - C_{q_i} (\sum_{a_i \in A_i} \pi_{a_i q_i} - 1)] = 0 \Rightarrow \\ &\pi_{a_i q_i} = \frac{1}{C_{q_i}} [\frac{1}{m} \sum_{k=1}^{m} \xi_k^0(a_i, q_i)] \end{aligned} \tag{7}$$

$$\begin{aligned} &\nabla_{\lambda_{q_i' q_i o_i}} [\tilde{F}(\theta, q^*) - C_{q_i o_i} (\sum_{q_i' \in Q_i} \lambda_{q_i' q_i o_i} - 1)] = 0 \Rightarrow \\ &\lambda_{q_i' q_i o_i} = \frac{1}{C_{q_i o_i}} [\frac{1}{m} \sum_{k=1}^{m} \xi_k^0(q_i', q_i, o_i)] \end{aligned} \tag{8}$$

where $-C_i, -C_{q_i}, -C_{q_i o_i}$ are the Lagrange multipliers and $\sum_{q_i \in Q_i} \nu_{q_i} = 1, \sum_{a_i \in A_i} \pi_{a_i q_i} = 1, \sum_{q_i' \in Q_i} \lambda_{q_i' q_i o_i} = 1$ are probability constraints for $\nu_{q_i}, \pi_{a_i q_i}, \lambda_{q_i' q_i o_i}$ respectively.

3.2 Importance Sampling

The performance of MCEM depends on the sample of trajectories $\xi_1, \xi_2, \cdots, \xi_m$ in the E-step obtained from a simulator using the previous joint policy θ^{old}. However, drawing a trajectory sample at each iteration could be prohibitively costly, particularly for large problems where the transition from the current state to a next state involves simulating a complex dynamical system (e.g., the movement of multiple robots in

a 3D environment). Furthermore, the EM algorithm may require an unrealistically large data sample for producing meaningful results. Fortunately, this computational expense can be substantially improved via *importance sampling*.

At each iteration, rather than directly obtaining a new sample from $P(\xi, T|\hat{R}; \theta^{old})$ with the most recent parameter θ^{old}, we can set importance weights to the original sample through the updated distribution $P(\xi, T|\hat{R}; \theta^{old})$. Then, the energy function becomes:

$$\tilde{F}(\theta, q^*) = \sum_{k=1}^{m} \omega_k \log P(\hat{R}, \xi_k, T; \theta) / \sum_{k=1}^{m} \omega_k \qquad (9)$$

where the original sample ξ_k obtained from the distribution $q(\xi_k)$ is corrected for the distribution we have at the current iteration (i.e., $p(\xi_k)$) through the weights:

$$\omega_k = \frac{p(\xi_k)}{q(\xi_k)}, \text{ where } p(\xi_k) = P(\xi_k, T|\hat{R}; \theta^{old}) \qquad (10)$$

The cost of obtaining the weight ω_k is less than the cost of a new sample because the weights only depend on the known distributions p and q, but not on the unknown likelihood $L(\theta)$.

Note that at each iteration we want to draw m trajectories from the conditional distribution:

$$P(\xi, T|\hat{R}; \theta^{old}) \propto P(T)P(\xi|T; \theta^{old})P(\hat{R}|\xi, T; \theta^{old}) \qquad (11)$$

The term $P(\hat{R}|\xi, T; \theta^{old})$ on the right-hand side merely depends on the last state and joint action in the trajectory ξ_k, i.e., $P(\hat{R}|\xi, T; \theta^{old}) = \hat{R}(r = 1|s^T_{\xi_k}, \vec{a}^{\,T}_{\xi_k})$. Instead of sampling the binary reward for each trajectory, the probabilities of the reward distribution can be used as the weights:

$$\omega_k = \frac{p(\xi_k)}{q(\xi_k)} = (1-\gamma)\gamma^T \hat{R}(r = 1|s^T_{\xi_k}, \vec{a}^{\,T}_{\xi_k}) \qquad (12)$$

In this case, we set $q(\xi_k) = P(\xi_k|T; \theta^{old})$ and generate a sample without sampling the binary rewards.

The above methods of importance sampling can be extended to every sub-trajectory of a given length-T trajectory ξ_k. Let ξ^τ_k be a length-τ sub-trajectory of ξ_k, i.e., a length-τ prefix of ξ_k, where $\tau \leq T$. According to Eq. (11), we have:

$$p(\xi^\tau_k) = q(\xi^\tau_k)P(\tau)P(\hat{R}|\xi^\tau_k, \tau; \theta^{old}) \qquad (13)$$

where $q(\xi^\tau_k) = P(\xi^\tau_k|\tau; \theta^{old})$. Therefore, the weight of the sub-trajectory ξ^τ_k can be computed as follows:

$$\omega^\tau_k = \frac{p(\xi^\tau_k)}{q(\xi^\tau_k)} = (1-\gamma)\gamma^\tau \hat{R}(r = 1|s^\tau_{\xi_k}, \vec{a}^{\,\tau}_{\xi_k}) \qquad (14)$$

where $s^\tau_{\xi_k}$ and $\vec{a}^{\,\tau}_{\xi_k}$ are the last state and joint action of ξ^τ_k. Intuitively, $q(\xi^\tau_k)$ is the probability of drawing a trajectory ξ^τ_k using the joint policy θ^{old} with the fixed length τ.

Applying the above results to Eq. (9), we compute the new energy function as follows:

$$\tilde{F}(\theta, q^*) = \eta \sum_{k=1}^{m} \sum_{\tau=1}^{T} \omega^\tau_k \log P(\hat{R}, \xi_k, \tau; \theta) \qquad (15)$$

where η is the normalizing factor. Thus, once a length-T trajectory ξ_k is sampled from the simulator, we can use the reward signal at each step of ξ_k to estimate the energy function instead of merely the last-step reward.

3.3 Policy Exploration

In standard MCEM, samples are randomly drawn from a simulator using the previous joint policy θ^{old}. However, entirely random draws often do not explore the policy space well. For instance, if the initial policy is extremely conservative (e.g., all agents always do nothing), the policy space will not be explored at all. On the other hand, if the initial policy is nearly optimal, we want to concentrate on it to assure fast convergence. This is known as the exploration-exploitation dilemma in reinforcement learning [Sutton and Barto, 1998] and has led to the development of a variety of heuristic strategies for obtaining a better spread of sampled trajectories.

For DEC-POMDPs, the sample space we want to explore is the joint policy space θ. The policy of each agent θ_i specifies how an action a_i is selected given its inner state (controller node) q_i. The inner state transitions depend on the observation signal o_i received from the environment. In standard MCEM, the action a_i and controller node q_i are randomly drawn from θ^{old}. Thus, the exploration depends on the quality of θ^{old}. Ideally, if we knew the optimal policy θ^*, we could use it to sample the trajectories and the policy parameters would quickly converge to their optimal values. While that is unrealistic, we can approximate θ^* by relaxing some of the assumptions of DEC-POMDPs. For example, we can assume the system state is fully observable during the learning phase. In that case, θ^* corresponds to the underlying MDP policy that can be computed more easily. For the exploration purpose, even random policies can be helpful. A more sophisticate approach is to combine multiple methods using a heuristic portfolio [Seuken and Zilberstein, 2007].

To explore the policy space, the simplest technique is using the ϵ-greedy strategy [Sutton and Barto, 1998]. Given the heuristic portfolio, we first select a heuristic from the portfolio. With a probability ϵ, we choose an action according to the heuristic and, with probability $1 - \epsilon$, we select an action by following the previous policy θ^{old}. Given such a trajectory ξ, the importance weight ω can be corrected as:

$$\omega = \frac{p(\xi)}{q(\xi)} = \omega' \prod_{(q_i, a_i) \in H_A(\xi)} \pi_{a_i q_i} \qquad (16)$$

where ω' is the original weight and $H_A(\xi)$ is the set of node-action pairs (q_i, a_i) in trajectory ξ where the action a_i has been selected by the heuristics. Notice that $\pi_{a_i q_i}$ is a probability of θ^{old} and is known to the agent. Similarly, we can explore the policy space by selecting the next controller node heuristically. Given a fixed number of controller nodes, we want each of them to be visited constantly during the sampling. Thus, inspired by the the research of multi-armed bandit problems, we count the visiting frequency of each node and use a UCB [Auer *et al.*, 2002] style heuristic to select the next controller node:

$$q^*_i = \arg\max_{q'_i \in Q_i} [\lambda_{q'_i q_i o_i} + c\sqrt{\frac{2 \ln N(q_i, o_i)}{N(q_i, o_i, q'_i)}}] \qquad (17)$$

where c is the rate, $N(q_i, o_i, q'_i)$ is the number of times that q'_i has been visited from q_i given o_i, and $N(q_i, o_i)$ is total number of times that the pair (q_i, o_i) has been visited so far. Intuitively, if $N(q_i, o_i, q'_i)$ is quite small, we should visit

q_i' more frequently from q_i given o_i. After applying this heuristic, the importance weight can be corrected by:

$$\omega = \frac{p(\xi)}{q(\xi)} = \omega' \prod_{(q_i, o_i, q_i') \in H_Q(\xi)} \lambda_{q_i' q_i o_i} \quad (18)$$

where $H_Q(\xi)$ is the set of node-observation tuples (q_i, o_i, q_i') in trajectory ξ where the next controller node q_i' has been selected by the heuristic. Notice that the action heuristics and node heuristics can be applied synchronously for multiple agents and the new importance weight becomes:

$$\omega = \omega' [\prod_{(q_i, a_i) \in H_A(\xi)} \pi_{a_i q_i}] \cdot [\prod_{(q_i, o_i, q_i') \in H_Q(\xi)} \lambda_{q_i' q_i o_i}] \quad (19)$$

3.4 Analysis and Discussion

As a variant of MCEM, our algorithm converges to a local optima in the limit. Additional convergence results can be found in a recent survey article [Neath, 2012]. Although it only finds local optima, our algorithm offers significant advantages over learning approaches [Banerjee *et al.*, 2012] based on the best-response strategy because EM is more stringent and satisfies the Karush-Kuhn-Tucker conditions [Bertsekas, 1999] (the norm in nonlinear optimization) while the best-response strategy provides no such guarantees. For the sample size, intuitively, we should start with a small size and increase the size as EM progresses [Wei and Tanner, 1990]. We can draw on a rich literature to decide how to increase the sample size and when to stop the learning process (see [Neath, 2012]).

As for the computational complexity, our algorithm is *linear* w.r.t. the sample size. Hence the runtime of our algorithm depends on the sampling process, more precisely, the simulator for drawing samples. If the simulations are expensive (e.g., a real-world 3D environment), we would like to run them concurrently on several machines. One advantage of MCEM is that it can run in parallel on a large number of computational nodes using the MapReduce framework [Dean and Ghemawat, 2008]. Specifically, in the *Map* step, each worker node reads the current policy and runs the simulator to sample trajectories. For every trajectory ξ_k^τ, the worker computes the weight ω_k^τ according to Eq. 14, or Eq. 19 if the heuristics are used, and emits the following key-value pairs:

$$(\nu_{q_i} \rightarrow \omega_k^\tau),\ (\pi_{a_i q_i} \rightarrow \omega_k^\tau),\ (\lambda_{q_i' q_i o_i} \rightarrow \omega_k^\tau) \quad (20)$$

where q_i is any agent i's controller node in ξ_k^τ, a_i is the action taken at node q_i, o_i is the received observation and q_i' is the next node. If q_i' is not in ξ_k^τ but in $\xi_k^{\tau+1}$, we emit $\omega_k^{\tau+1}$ for $\lambda_{q_i' q_i o_i}$ where $\omega_k^{\tau+1}$ is the weight of $\xi_k^{\tau+1}$. Note that Mappers are not assigned to agents, but offer a distributed offline concurrent mechanism to perform the E-step. In the *Reduce* step, each worker collects a list of weights for ν_{q_i}, $\pi_{a_i q_i}$ and $\lambda_{q_i' q_i o_i}$. These workers compute the averaged values of these weights and emit the new policy parameters ν_{q_i}, $\pi_{a_i q_i}$ and $\lambda_{q_i' q_i o_i}$ according to Eqs. 6, 7 and 8.

4 Experiments

We tested our algorithm on several common benchmark problems in the DEC-POMDP literature to evaluate the solution quality and runtime performance. Additionally, a traffic control domain with thousands of agents was used to test the scalability of our algorithm. For each problem, a simulator was implemented to simulate the DEC-POMDP with the discount factor γ=0.9. For MCEM, the sample size was 1000 for all problems. Although a domain-specified ϵ would be useful for the ϵ-greedy strategy, we used ϵ=0.1 for all the problems. The number of iterations was 300 as most of the tested problems could converge to a stable solution after 300 iterations. The policy parameters were initialized randomly with $|Q_i|$=3, as do most existing model-based approaches. Because EM converges to a local optimum that depends on the initial policies, we restarted the algorithm 10 times with different (random) policies and selected the best solution. To evaluate solution quality, we execute each policy on the simulator (each with T=1000 as γ^T is negligible at that point) and report the average value (total discounted rewards) over 200 runs (statistically significant as the policy value). All the algorithms were implemented in Java 1.6 and run on a machine with a 2.66GHZ Intel Core 2 Duo CPU and 4GB RAM.

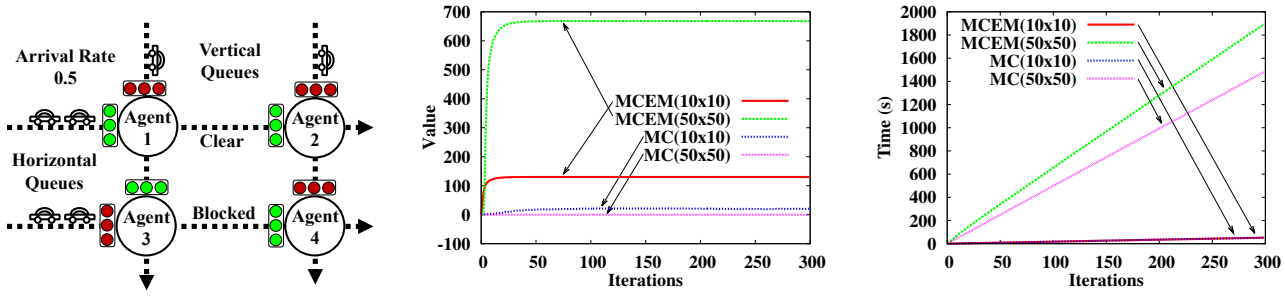
Table 1: Results for Common DEC-POMDP Benchmarks

Problems	MCEM		EM	
	Value	Time	Value	Time
Broadcast Channel	9.75	17s	9.05	$<$ 1s
Multi-Agent Tiger	-10.86	18s	-19.99	$<$ 1s
Recycling Robots	59.45	18s	62.17	5s
Meeting on a Grid	7.34	26s	7.42	42s
Box Pushing	59.76	32s	39.83	1320s
Mars Rovers	7.65	187s	9.96	9450s

4.1 Common Benchmark Problems

Our algorithm was tested on 6 common benchmark problems for infinite-horizon DEC-POMDPs, namely: broadcast channel ($|S|$=4, $|A_i|$=2, $|\Omega_i|$=5), multi-agent tiger($|S|$=2, $|A_i|$=3, $|\Omega_i|$=2), recycling robots($|S|$=3, $|A_i|$=3, $|\Omega_i|$=2), meeting in a 2$\times$2 grid ($|S|$=16, $|A_i|$=5, $|\Omega_i|$=4), box pushing ($|S|$=100, $|A_i|$=4, $|\Omega_i|$=5), and Mars rovers ($|S|$=256, $|A_i|$=6, $|\Omega_i|$=8). We used the underlying MDP policy (obtained by value iteration) as the heuristic for policy exploration. Because none of the existing algorithms can solve infinite-horizon DEC-POMDPs without knowing the complete model, a direct comparison is impossible. Nevertheless, we did compare the solution quality and runtime performance of our algorithm with the existing model-based EM approach [Kumar and Zilberstein, 2010], which uses full knowledge of the models. Currently, this is the leading approach in the literature for solving infinite-horizon DEC-POMDPs.

Table 1 shows the value and runtime results of our MCEM algorithm and the model-based EM approach. In these experiments, our MCEM algorithm consistently produces solutions that are competitive with the model-based EM solutions, despite the fact that it relies on samples. In fact, for some problems, our algorithm produces better solutions than the model-based EM thanks to the use of the MDP heuristics. For small problems (e.g., broadcast channel and multi-agent tiger), the model-based EM algorithm can converge very quickly, while our algorithm needs more time for sampling. However, the runtime of the model-based EM grows dramatically for larger



(a) Example on a 2×2 Grid (b) Value vs. Iterations (c) Runtime vs. Iterations

Figure 1: Illustration of the Traffic Control Problem on an n×n Grid and Results

problems (e.g., box pushing and Mars rovers) because it relies on the α and β messages that need to consider every state. In contrast, the runtime of our algorithm grows moderately with the number of states, as the simulators need more time to simulate the process. Overall, our algorithm is more scalable because it is usually cheaper to simulate a large process than considering every state.

4.2 Traffic Control on an n×n Grid

This is an abstract traffic domain first introduced by [Zhang *et al.*, 2007]. In this domain, as shown in Figure 1(a), traffic flows on an n×n grid from one side to the other without turning. Each intersection is controlled by an agent that only allows traffic to flow vertically or horizontally at each time step. When a path is clear, all awaiting traffic for that path propagates through instantly and each unit of traffic contributes $+1.0$ to the team reward. There are n+n traffic queues on the top and left boundaries. Traffic units arrive with probability 0.5 per time step per queue. As the queues fill up, they drop traffic if the maximum capability (10 traffic units per queue) is reached. As any misaligned gates will block traffic on that path, agents must coordinate with each other to maximize the overall rewards. This problem has $n \times n$ agents and 10^{n+n} states. Each agent has 2 actions (aligning vertically or horizontally) and 10×10 observations, indicating the number of traffic units waiting in the vertical and horizontal queues. In the original problem, agents can communicate with each other instantaneously and know the grid structure. Our DEC-POMDP version of the problem does not rely on such assumptions and is therefore fundamentally harder.

Figures 1(b) and 1(c) show the value and runtime results of this domain with 10×10 (100 agents and 10^{20} states) and 50×50 (2500 agents and 10^{100} states) grids. These two instances are prohibitively hard for the existing DEC-POMDP algorithms. We compared our results with a naïve implementation of MCEM (our algorithm without importance sampling and policy exploration described in Sections 3.2 and 3.3) to show the benefit of our sampling and exploration techniques. This baseline approach is called MC. Our algorithm used a hand-coded MDP heuristic for policy exploration. Figure 1(b) shows that our algorithm can converge to a stable solution in less than 20 iterations for both instances. In fact, the value matches what is achieved by the hand-coded MDP policy (assuming all agents observe the full state). Thus, using our approach, the agents do coordinate effectively. In contrast, the naïve MC method converged very slowly for the smaller (10×10) instance and the policies produced no improvement (always 0.0) for the larger (50×50) instance. The reason is that the joint policy space is huge ($2^{n \times n}$ joint actions, $100^{n \times n}$ joint observations) so the naïve MC method has a very small chance to gain positive reward signals. Both methods are linear with respect to the number of iterations. For the smaller instance, our algorithm took almost the same time as the naïve MC method. For the larger instance, the runtime of our algorithm increased because it took more time to compute the hand-coded heuristic.

5 Conclusions

We present a new MCEM-based algorithm for solving infinite-horizon DEC-POMDPs, without full prior knowledge of the model parameters. This is achieved by drawing trajectory samples and improving the joint policy using the EM algorithm. We show that a straightforward application of Monte-Carlo methods is very inefficient due to the extremely large joint-policy space, particularly with a large number of agents. To address that, we implement two sampling strategies that better utilize the samples and explore the policy space. Although these methods are well-known in statistics and reinforcement learning, combining them with MCEM in a multi-agent setting is nontrivial. The results on a range of DEC-POMDP benchmarks confirm the benefits of the new algorithm—particularly when scalability with respect to the number of agents is considered.

The algorithm produces good results for six common DEC-POMDP benchmarks. More importantly, it can effectively tackle much larger problems such as the traffic control domains with up to 2500 agents—an intractable problem for state-of-the-art DEC-POMDP solvers. In the future, we plan to extend our work to domains with continual state and action spaces and test our algorithm on additional complex real-world applications such as robot soccer and rescue, where fully functional simulators are readily available. Hence, this work lays the foundation for the application of DEC-POMDP techniques in such challenging domains.

Acknowledgments

Feng Wu and Nicholas R. Jennings were supported in part by the ORCHID project (EPSRC Reference: EP/I011587/1). Shlomo Zilberstein was supported in part by the US National Science Foundation under Grant IIS-1116917.

References

[Aras *et al.*, 2007] Raghav Aras, Alain Dutech, and François Charpillet. Mixed integer linear programming for exact finite-horizon planning in decentralized POMDPs. In *Proc. of the 17th Int'l Conf. on Automated Planning and Scheduling*, pages 18–25, 2007.

[Auer *et al.*, 2002] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

[Banerjee *et al.*, 2012] Bikramjit Banerjee, Jeremy Lyle, Landon Kraemer, and Rajesh Yellamraju. Sample bounded distributed reinforcement learning for decentralized POMDPs. In *Proc. of the 26th AAAI Conf. on Artificial Intelligence*, pages 1256–1262, 2012.

[Bernstein *et al.*, 2002] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, 2002.

[Bernstein *et al.*, 2009] Daniel S. Bernstein, Christopher Amato, Eric A. Hansen, and Shlomo Zilberstein. Policy iteration for decentralized control of Markov decision processes. *Journal of Artificial Intelligence Research*, 34:89–132, 2009.

[Bertsekas, 1999] Dimitri P. Bertsekas. *Nonlinear programming*. Athena Scientific, 1999.

[Dean and Ghemawat, 2008] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[Dempster *et al.*, 1977] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B*, pages 1–38, 1977.

[Dibangoye *et al.*, 2011] Jilles S. Dibangoye, Abdel-Illah Mouaddib, and Brahim Chaib-draa. Toward error-bounded algorithms for infinite-horizon DEC-POMDPs. In *Proc. of the 10th Int'l Conf. on Autonomous Agents and Multiagent Systems*, pages 947–954, 2011.

[Hansen *et al.*, 2004] Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *Proc. of the 19th National Conf. on Artificial Intelligence*, pages 709–715, 2004.

[Kumar and Zilberstein, 2010] Akshat Kumar and Shlomo Zilberstein. Anytime planning for decentralized POMDPs using expectation maximization. In *Proc. of the 26th Conf. on Uncertainty in Artificial Intelligence*, pages 294–301, 2010.

[Kumar *et al.*, 2011] Akshat Kumar, Shlomo Zilberstein, and Marc Toussaint. Scalable multiagent planning using probabilistic inference. In *Proc. of the 22nd Int'l Joint Conf. on Artificial Intelligence*, pages 2140–2146, 2011.

[Nair *et al.*, 2003] Ranjit Nair, Milind Tambe, Makoto Yokoo, David V. Pynadath, and Stacy Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proc. of the 18th Int'l Joint Conf. on Artificial Intelligence*, pages 705–711, 2003.

[Nair *et al.*, 2005] Ranjit Nair, Pradeep Varakantham, Milind Tambe, and Makoto Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proc. of the 20th National Conf. on Artificial Intelligence*, pages 133–139, 2005.

[Neath, 2012] Ronald C. Neath. On convergence properties of the Monte Carlo EM algorithm. *arXiv preprint arXiv:1206.4768*, 2012.

[Pajarinen and Peltonen, 2011a] Joni Pajarinen and Jaakko Peltonen. Efficient planning for factored infinite-horizon DEC-POMDPs. In *Proc. of the 22nd Int'l Joint Conf. on Artificial Intelligence*, pages 325–331, 2011.

[Pajarinen and Peltonen, 2011b] Joni Pajarinen and Jaakko Peltonen. Periodic finite state controllers for efficient POMDP and DEC-POMDP planning. In *Proc. of the 25th Annual Conf. on Neural Information Processing Systems*, pages 2636–2644, 2011.

[Seuken and Zilberstein, 2007] Sven Seuken and Shlomo Zilberstein. Memory-bounded dynamic programming for DEC-POMDPs. In *Proc. of the 20th Int'l Joint Conf. on Artificial Intelligence*, pages 2009–2015, 2007.

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

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998.

[Szer *et al.*, 2005] Daniel Szer, François Charpillet, and Shlomo Zilberstein. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proc. of the 21st Conf. on Uncertainty in Artificial Intelligence*, pages 576–590, 2005.

[Toussaint and Storkey, 2006] Marc Toussaint and Amos Storkey. Probabilistic inference for solving discrete and continuous state Markov decision processes. In *Proc. of the 23rd Int'l Conf. on Machine learning*, pages 945–952, 2006.

[Toussaint *et al.*, 2008] Marc Toussaint, Laurent Charlin, and Pascal Poupart. Hierarchical POMDP controller optimization by likelihood maximization. In *Proc. of the 24th Conf. on Uncertainty in Artificial Intelligence*, pages 562–570, 2008.

[Vlassis and Toussaint, 2009] Nikos Vlassis and Marc Toussaint. Model-free reinforcement learning as mixture learning. In *Proc. of the 26th Int'l Conf. on Machine Learning*, pages 1081–1088, 2009.

[Vlassis *et al.*, 2009] Nikos Vlassis, Marc Toussaint, Georgios Kontes, and Savas Piperidis. Learning model-free robot control by a Monte Carlo EM algorithm. *Autonomous Robots*, 27(2):123–130, 2009.

[Wei and Tanner, 1990] Greg C. G. Wei and Martin A. Tanner. A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms. *Journal of the American Statistical Association*, 85(411):699–704, 1990.

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

[Zhang and Lesser, 2011] Chongjie Zhang and Victor Lesser. Coordinated multi-agent reinforcement learning in networked distributed POMDPs. In *Proc. of the 25th AAAI Conf. on Artificial Intelligence*, pages 764–770, 2011.

[Zhang *et al.*, 2007] Xinhua Zhang, Douglas Aberdeen, and S.V.N. Vishwanathan. Conditional random fields for multi-agent reinforcement learning. In *Proc. of the 24th Int'l Conf. on Machine learning*, pages 1143–1150, 2007.