Solving DEC-POMDPs by Expectation Maximization of Value Functions

Zhao Song, Xuejun Liao, Lawrence Carin
Duke University, Durham, NC 27708, USA
{zhao.song, xjliao, lcarin}@duke.edu

Abstract

We present a new algorithm called PIEM to approximately solve for the policy of an infinite-horizon decentralized partially observable Markov decision process (DEC-POMDP). The algorithm uses expectation maximization (EM) only in the step of policy improvement, with policy evaluation achieved by solving the Bellman's equation in terms of finite state controllers (FSCs). This marks a key distinction of PIEM from the previous EM algorithm of (Kumar and Zilberstein, 2010), i.e., PIEM directly operates on a DEC-POMDP without transforming it into a mixture of dynamic Bayes nets. Thus, PIEM precisely maximizes the value function, avoiding complicated forward/backward message passing and the corresponding computational and memory cost. To overcome local optima, we follow (Pajarinen and Peltonen, 2011) to solve the DEC-POMDP for a finite length horizon and use the resulting policy graph to initialize the FSCs. We solve the finite-horizon problem using a modified point-based policy generation (PBPG) algorithm, in which a closed-form solution is provided which was previously found by linear programming in the original PBPG. Experimental results on benchmark problems show that the proposed algorithms compare favorably to state-of-the-art methods.

Introduction

The decentralized partially observable Markov decision process (DEC-POMDP) is a concise and powerful model for multi-agent planning and has been used in a wide range of applications, including robot navigation and distributed sensor networks (Oliehoek 2012; Durfee and Zilberstein 2013). The joint policy of a DEC-POMDP is defined by a set of local policies, one for each agent, executed in a decentralized manner without direct communication among the agents. Thus, the action selection of each agent must be based on its own local observations and actions. The decentralization precludes the availability of a global belief state and hence the possibility of converting a DEC-POMDP into a belief-state representation. As a result, a DEC-POMDP is more challenging to solve than its centralized counterpart (Sondik 1971; Kaelbling, Littman, and Cassandra 1998).

Research on DEC-POMDPs in the artificial intelligence community has led to two categories of methods, respectively aiming to obtain finite-horizon or infinite-horizon policies. A finite-horizon policy is typically represented by a decision tree (Hansen, Bernstein, and Zilberstein 2004; Nair et al. 2003), with the tree often learned centrally and offline, for a given initial belief about the system state. Solving a DEC-POMDP for the optimal finite-horizon policy is an NEXP-complete problem (Bernstein et al. 2002), and MBDP (Seuken and Zilberstein 2007b) was the first algorithm to approximately solve the problem for large horizons. Though MBDP employs a point-based approach to limit the required memory, the number of policies it evaluates is still exponential in the number of observations. The subsequent work in (Seuken and Zilberstein 2007a; Wu, Zilberstein, and Chen 2010a; 2010b) tried to address this issue to make the method scalable to larger problems. Moreover, as shown in (Pajarinen and Peltonen 2011), the multiple policy trees returned by the memory-bounded approaches could be represented as a policy graph, where the number of layers and number of nodes in each layer correspond to the horizon length and the number of policy trees, respectively.

An infinite-horizon policy has typically been represented by finite state controllers (FSCs) (Sondik 1978; Hansen 1997; Poupart and Boutilier 2003), with parameters estimated by linear programming (LP) (Bernstein et al. 2009) or nonlinear programming (NLP) (Amato, Bernstein, and Zilberstein 2010), considering mainly two-agent cases. Expectation-maximization (EM) (Kumar and Zilberstein 2010; Pajarinen and Peltonen 2011) has been shown as a promising algorithm for scaling up to the number of agents. Recently, the state-of-the-art value iteration algorithms (MacDermed and Isbell 2013; Dibangoye, Buffet, and Charpillet 2014) have been proposed to transform the original DEC-POMDP problem into a POMDP problem, which is subsequently solved by the point-based methods. The values reported in (MacDermed and Isbell 2013; Dibangoye, Buffet, and Charpillet 2014) are shown to be higher than the previously best methods on the benchmark problems; however, the policy returned in (MacDermed and Isbell 2013; Dibangoye, Buffet, and Charpillet 2014) does not have a form as concise as an FSC.

The EM algorithm of (Kumar and Zilberstein 2010) is an

extension of the single-agent POMDP algorithm in (Toussaint, Harmeling, and Storkey 2006; Vlassis and Toussaint 2009; Toussaint, Storkey, and Harmeling 2011), which maximizes a value function indirectly by maximizing the likelihood of a dynamic Bayesian net (DBN). It employs the classic EM algorithm (Dempster, Laird, and Rubin 1977) to obtain the maximum-likelihood (ML) estimator. This method, referred to here as DBN-EM, transforms a DEC-POMDP into a mixture of DBNs in a pre-processing step. There are two major drawbacks of DBN-EM. First, it relies on transforming the value into a "reward likelihood", and therefore the quality of the resulting policy depends on the fidelity of the transformation. For an infinite-horizon DEC-POMDP, the DBN representation has an infinite number of mixture components, which in practice must be truncated to a cut-off time that is either fixed *a priori* or determined based on the likelihood accumulation. Second, DBN-EM manipulates Markovian sequences, which require complicated forward/backward message passing in the E-step, incurring considerable computational and memory cost.

In this paper we present a new algorithm for FSC parameter estimation, solving an infinite-horizon DEC-POMDP. The algorithm is based on policy iteration, like DBN-EM. However, unlike DBN-EM, which uses EM to both improve and evaluate the policy, the new algorithm evaluates the policy by solving Bellman's equation in FSC nodes[1], leaving only the step of Policy Improvement for the EM to implement; the algorithm is therefore termed PIEM. Note that the EM used by PIEM is not probabilistic and may better be interpreted as surrogate optimization (Lange, Hunter, and Yang 2000; Neal and Hinton 1998). To overcome local optima, we adopt the idea in (Pajarinen and Peltonen 2011) to initialize the FSCs using a policy graph obtained by solving the finite-horizon DEC-POMDP. The finite-horizon problem is solved using point-based policy generation (PBPG) modified by replacing linear programming with a closed-form solution in each iteration of PBPG, with the modification constituting another contribution of this paper.

The DEC-POMDP Model

A DEC-POMDP can be represented as a tuple, $\mathcal{M} = \langle \mathcal{I}, \mathcal{A}, \mathcal{O}, \mathcal{S}, b_0(s), \mathcal{T}, \Omega, \mathcal{R}, \gamma \rangle$ where

- $\mathcal{I} = \{1, \cdots, n\}$ indexes a finite set of agents;
- $\mathcal{A} = \otimes_i \mathcal{A}_i$ is the set of joint actions, with $\mathcal{A}_i$ available to agent i and $\vec{a} = (a_1, \cdots, a_n) \in \mathcal{A}$ denoting the joint action;
- $\mathcal{O} = \otimes_i \mathcal{O}_i$ is the set of joint observations, with $\mathcal{O}_i$ available to agent i and $\vec{o} = (o_1, \cdots, o_n) \in \mathcal{O}$ denoting the joint observation;
- $\mathcal{S}$ is a set of finite system states;
- $b_0(s)$ denotes the initial belief state;
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the state transition function with $p(s'|s, \vec{a})$ denoting the probability of transitioning to s' after taking joint action $\vec{a}$ in s;
- $\Omega : \mathcal{S} \times \mathcal{A} \to \mathcal{O}$ is the observation function with $p(\vec{o}|s', \vec{a})$ the probability of observing $\vec{o}$ after taking joint action $\vec{a}$ and arriving in state s';
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function with $R(s, \vec{a})$ the immediate reward received after taking joint action $\vec{a}$ in s.
- $\gamma \in (0, 1)$ is the discount factor defining the discount for future forwards in infinite-horizon problems.

Due to the lack of access to other agents' observations, each agent has a local policy π_i, defined as a mapping from the local observable history prior to time t to the action at t, where the local observable history includes the actions the agent has taken and the observations it has received. A joint policy consists of the local policies of all agents. For an infinite-horizon DEC-POMDP, the objective is to find a joint policy $\Pi = \otimes_i \pi_i$ that maximizes the value function, starting from an initial belief b_0 about the system state, with the value function expressed as $V^{\Pi}(b_0) = \mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t R(s_t, \vec{a}_t) | b_0, \Pi\big]$.

The PIEM algorithm

The policy of agent i is represented by an FSC, with $p(a_i|z_i)$ denoting the probability of selecting action a_i at node z_i, and $p(z_i'|z_i, o_i)$ representing the probability of transitioning from node z_i to node z_i' when o_i is observed. The stationary value, as a function of FSC nodes $\vec{z} = (z_1, \ldots, z_n)$ and system state s, satisfies the Bellman's equation (Amato, Bernstein, and Zilberstein 2007; Bernstein, Hansen, and Zilberstein 2005)

$$V(s, \vec{z}) = \sum_{\vec{a}} R(s, \vec{a}) \prod_i p(a_i|z_i) \; + \; \gamma \sum_{s', \vec{a}, \vec{o}, \vec{z}'} p(\vec{o}|s', \vec{a}) \; p(s'|s, \vec{a}) \, V(s', \vec{z}') \prod_i p(a_i|z_i) \, p(z_i'|z_i, o_i) \qquad (1)$$

where the factorized forms over the agents indicate that the policy of each agent is executed independently of other agents (no explicit communication).

Let $\Theta = \{\Theta_i\}_{i=1}^n$ where $\Theta_i = \{p(a_i|z_i), p(z_i'|z_i, o_i)\}$ collects the parameters of the FSC of agent i. The PIEM algorithm iterates between the following two steps, starting from an initial Θ.

- Policy evaluation: Solve (1) to find the stationary value function $V(s, \vec{z})$ for the most recent Θ.
- Policy improvement: Improve Θ by using EM to maximize the righthand side of (1), given the most recent $V(s, \vec{z})$.

The PIEM algorithm is summarized in Algorithm 1, where the details of policy improvement are described subsequently.

[1] Bellman's equation in belief state does not exist for DEC-POMDPs due to local observability. Bellman's equation for each agent is here expressed in terms of FSC nodes, as in the DEC-POMDP literature (Amato, Bernstein, and Zilberstein 2010).

Algorithm 1 PIEM for infinite-horizon DEC-POMDPs

Input: Initial FSCs $\left\{p^{(0)}(a_i|z_i), p^{(0)}(z_i'|z_i, o_i) : z_i = 1, \ldots, Z\right\}_{i=1}^{n}$.
while $\max_{\vec{z}} V(b_0, \vec{z})$ not converging **do**
  Policy evaluation by solving (1) to find $V(s, \vec{z})$.
  Policy improvement by doing the following.
  for $i = 1$ to n **do**
    for $z_i = 1$ to Z **do**
      Iteratively update $p(a_i|z_i)$ and $p(z_i'|z_i, o_i)$ for each instantiation of z_i using (7) and (8) until convergence.
    end for
  end for
end while

Policy improvement in PIEM

The objective function we wish to maximize is the value function for a given initial belief denoted as b_0, i.e.,

$$\begin{aligned} V(b_0, \vec{z}) &= \sum_s b_0(s)\, V(s, \vec{z}) \\ &= \sum_{\vec{a}} R(b_0, \vec{a}) \prod_i p(a_i|z_i) \;+\; \gamma \sum_{s', \vec{a}, \vec{o}, \vec{z}'} p(\vec{o}|s', \vec{a}) \\ &\quad p(s'|b_0, \vec{a}) V(s', \vec{z}') \prod_i p(a_i|z_i) p(z_i'|z_i, o_i) \end{aligned} \tag{2}$$

where

$$\begin{aligned} R(b_0, \vec{a}) &= \sum_s b_0(s)\, R(s, \vec{a}) \\ p(s'|b_0, \vec{a}) &= \sum_s b_0(s)\, p(s'|s, \vec{a}) \end{aligned}$$

The goal is to increase the value function $V(b_0, \vec{z})$ for any $\vec{z}$, by updating $\{p(a_i|z_i), p(z_i'|z_i, o_i)\}_{i=1}^{n}$, the parameters of the FSCs of all agents. Following the ideas employed in (Bernstein, Hansen, and Zilberstein 2005; Bernstein et al. 2009), we let the agents take turns in updating their FSCs, improving Θ_i while keeping $\{\Theta_j : j \neq i\}$ fixed. An appealing property of PIEM is that it can update $\Theta_i = \{p(a_i|z_i), p(z_i'|z_i, o_i)\}$ in parallel at different FSC nodes, i.e., different instantiations of z_i. Focusing on Θ_i for a particular instantiation of z_i, we rewrite (2) as

$$\begin{aligned} V(b_0, \vec{z}) = \sum_{a_i} p(a_i|z_i)\, \alpha(a_i, z_i) \;+\; \sum_{z_i', a_i, o_i} p(a_i|z_i) \\ p(z_i'|z_i, o_i)\, \beta(z_i, a_i, o_i, z_i') \end{aligned} \tag{3}$$

where

$$\alpha(a_i, z_i) = \sum_{\vec{a}_{-i}} R(b_0, \vec{a}) \prod_{j \neq i} p(a_j|z_j) \tag{4a}$$

$$\beta(z_i, a_i, o_i, z_i') = \sum_{\vec{z}'_{-i}, \vec{a}_{-i}, \vec{o}_{-i}, s'} \gamma\, p(\vec{o}|s', \vec{a})\, p(s'|b_0, \vec{a})$$

$$V(s', \vec{z}') \prod_{j \neq i} p(a_i|z_i)\, p(z_i'|z_i, o_i), \tag{4b}$$

and the subscript $-i$ is defined as $-i = \{j : j \neq i\}$, i.e., it denotes the indices of all agents except agent i.

Maximizing $V(b_0, \vec{z})$ is equivalent to maximizing $\ln V(b_0, \vec{z})$ and, for the latter, it holds that

$$\begin{aligned} &\ln\left[V(b_0, \vec{z})\right] \\ &= \ln\left\{ \sum_{a_i} \eta(a_i, z_i) \frac{p(a_i|z_i)\alpha(a_i, z_i)}{\eta(a_i, z_i)} + \sum_{z_i', a_i, o_i} p(a_i|z_i) \right. \\ &\quad \left. \cdot \frac{\rho(z_i', a_i, o_i, z_i)\, p(z_i'|z_i, o_i)\, \beta(z_i, a_i, o_i, z_i')}{\rho(z_i', a_i, o_i, z_i)} \right\} \\ &\geq \sum_{a_i} \eta(a_i, z_i) \ln\left[\frac{p(a_i|z_i)\alpha(a_i, z_i)}{\eta(a_i, z_i)}\right] + \sum_{z_i', a_i, o_i} \\ &\cdot \rho(z_i', a_i, o_i, z_i) \ln\left[\frac{p(a_i|z_i)\, p(z_i'|z_i, o_i)\beta(z_i, a_i, o_i, z_i')}{\rho(z_i', a_i, o_i, z_i)}\right] \end{aligned} \tag{5}$$

$$\begin{aligned} &= \sum_{a_i} \eta(a_i, z_i) \ln\left[p(a_i|z_i)\right] + \sum_{z_i', a_i, o_i} \rho(z_i', a_i, o_i, z_i) \\ &\quad \cdot \left\{ \ln\left[p(a_i|z_i)\right] + \ln\left[p(z_i'|z_i, o_i)\right] \right\} + \text{const} \end{aligned} \tag{6}$$

where const represents a term that is independent of $p(a_i|z_i)$ and $p(z_i'|z_i, a_i, o_i)$. The inequality in (5) holds for any $\eta(a_i, z_i) \geq 0$ and $\rho(z_i', a_i, o_i, z_i) \geq 0$ [2] that satisfy $\sum_{a_i} \eta(a_i, z_i) + \sum_{z_i', a_i, o_i} \rho(z_i', a_i, o_i, z_i) = 1$, based on Jensen's inequality. From (5) and (6), we can derive the following EM update equations used in the PIEM algorithm.

- E-step: Tighten the lower bound in (5) by setting

$$\eta(a_i, z_i) = \frac{p^{(l)}(a_i|z_i)\, \alpha(a_i, z_i)}{\xi(z_i)} \tag{7a}$$

$$\begin{aligned} \rho(z_i', a_i, o_i, z_i) = \Big[p^{(l)}(a_i|z_i)\, p^{(l)}(z_i'|z_i, o_i) \\ \cdot \beta(z_i, a_i, o_i, z_i') \Big] / \xi(z_i) \end{aligned} \tag{7b}$$

  where l denotes the iteration index and

$$\begin{aligned} \xi(z_i) = \sum_{a_i} p^{(l)}(a_i|z_i)\, \alpha(a_i, z_i) \;+\; \sum_{z_i', a_i, o_i} \\ \cdot p^{(l)}(a_i|z_i)\, p^{(l)}(z_i'|z_i, o_i)\, \beta(z_i, a_i, o_i, z_i') \end{aligned}$$

- M-step: Maximizes the right side of (6) with respect to $p(a_i|z_i)$ and $p(z_i'|z_i, o_i)$. The solution is provided in Theorem 1.

Theorem 1. *The maximizer of* (6) *in the M step is*

$$p^{(l+1)}(a_i|z_i) = \frac{\eta(a_i, z_i) + \sum_{z_i', o_i} \rho(z_i', a_i, o_i, z_i)}{\sum_{a_i} \left[\eta(a_i, z_i) + \sum_{z_i', o_i} \rho(z_i', a_i, o_i, z_i)\right]} \tag{8a}$$

[2] At the beginning of our algorithm, we normalize the immediate reward in the same way as (Kumar and Zilberstein 2010) to ensure η and ρ are always nonnegative.

$$p^{(l+1)}(z_i'|z_i, o_i) = \frac{\sum_{a_i} \rho(z_i', a_i, o_i, z_i)}{\sum_{z_i', a_i} \rho(z_i', a_i, o_i, z_i)} \tag{8b}$$

Proof. We use $f(\Theta_i)$ to represent the right side of (6). It follows that the M step corresponds to the following optimization problem:

$$\begin{aligned} \underset{\Theta_i}{\text{maximize}} \quad & f(\Theta_i) \\ \text{subject to} \quad & \sum_{a_i} p(a_i|z_i) = 1,\ \sum_{z_i'} p(z_i'|z_i, o_i) = 1,\ \forall o_i \\ & p(a_i|z_i) \geq 0, \forall a_i,\ p(z_i'|z_i, o_i) \geq 0,\ \forall o_i, \forall z_i' \end{aligned}$$

Since $f(\Theta_i)$ is a concave function of Θ_i, applying KKT condition (Boyd and Vandenberghe 2004) leads to the updates in (8). □

Theorem 2. *The proposed PIEM algorithm improves the objective function monotonically.*

Proof. We first rewrite the objective function $\ln [V(b_0, \vec{z})] = \mathcal{Q}(\Theta)$ and the corresponding lower bound in (5) as $\widehat{\mathcal{Q}}(\Theta, \Omega)$ where $\Omega = \{\eta,\ \rho\}$. For every two consecutive iterations t and $t+1$, we have

$$\begin{aligned} \mathcal{Q}(\Theta^{(t+1)}) &= \widehat{\mathcal{Q}}(\Theta^{(t+1)}, \Omega^{(t+1)}) && (9) \\ &\geq \widehat{\mathcal{Q}}(\Theta^{(t+1)}, \Omega^{(t)}) && (10) \\ &\geq \widehat{\mathcal{Q}}(\Theta^{(t)}, \Omega^{(t)}) && (11) \\ &= \mathcal{Q}(\Theta^{(t)}) && (12) \end{aligned}$$

where (9), (10), and (12) use the fact that Ω^t derived in the E-step tightens the lower bound $\widehat{\mathcal{Q}}(\Theta^{(t)}, \Omega)$. (11) holds because Θ^{t+1} provided in the M-step maximizes $\widehat{\mathcal{Q}}(\Theta, \Omega^{(t)})$. □

Time complexity

Let S be the number of system states. Let A, O, and Z respectively denote the maximum numbers of actions, observations, and FCS nodes, across all agents. The analysis below (for two agents) extends readily to the general case.

- Policy evaluation. Solving (1) for $V(s, \vec{z})$ requires time $O(Z^4 S^2 N_{\text{ite}})$, where N_{ite} is the number of iterations.
- Policy improvement. This part is dominated by the computation in (4b), which requires time of $O(Z^3 S^2 A^2 O^2)$.

As a comparison, DBN-EM requires $O(Z^4 S^2 A^2 O^2)$ for computing Markov transitions, $O(Z^4 S^2 T_{\max})$ for message propagation, and additional time for completing the M-step; here $T_{\max}$ is the cut-off time (Kumar and Zilberstein 2010). Therefore PIEM is generally computationally more efficient than DBN-EM, and the relative efficiency is more prominent when $N_{\text{ite}} \leq T_{\max}$, a condition that holds in our experiments.

Initialization of PIEM

We initialize the FSCs using a policy graph obtained by a modified version of point-based policy generation (PBPG) (Wu, Zilberstein, and Chen 2010a), a state-of-the-art algorithm for finite-horizon DEC-POMDPs. The PBPG significantly reduces the number of potential policies by using linear programing (LP) to learn a stochastic mapping from observations to policy candidates for each agent. Here we show that each sub-problem originally solved by LP has a closed-form solution, which we use in place of LP to yield a modified PBPG algorithm. Note that unlike the original PBPG, which returns the best policy tree, the modified PBPG outputs the policy graph.

Definition 1. *A policy graph $\mathcal{G}$ in the finite-horizon DEC-POMDP is a directed graph with T layers and K nodes in each layer, obtained from the memory bounded dynamic programming approaches.*

The policy execution in the policy graph is described as follows. Starting from a single node in the top level, every agent takes the action according to the node's corresponding action. The agents then receive their individual observation and move to the nodes at the next level, according to the current node's connection. The agents can take subsequent actions and repeat this process iteratively until reaching the bottom level. The policy in the policy graph can be learned by the MBDP approaches. Therefore, learning a policy graph reduces to determining the action for each node and its connections to the nodes at the next level, based on different observations.

The modified PBPG algorithm is summarized in Algorithm 2. The main step in the modified PBPG is to add nodes into the policy graph for every t and k, one for each agent, and decide the corresponding action and connections to the next level for every observation. In Algorithm 2, $\mathcal{B}$ is a belief portfolio containing the following beliefs [3]:

- The initial belief b_0.
- The uniform belief b_u with $b_u(s) = \frac{1}{S}$ for $s = 1, \ldots, S$.
- Heuristic beliefs which include both random policy beliefs and MDP policy beliefs.

Closed-form stochastic mappings

The value function for the joint policy of a finite-horizon DEC-POMDP can be computed recursively (Wu, Zilberstein, and Chen 2010a)

$$V^{t+1}(s, \vec{q}^{\,t+1}) = R(s, \vec{a}) + \sum_{s', \vec{o}} p(s'|s, \vec{a}) p(\vec{o}|s', \vec{a})\, V^t(s', \vec{q}_{\vec{o}}^{\,t}) \tag{13}$$

where $\vec{q}_{\vec{o}}^{\,t}$ denotes the subtree of $\vec{q}^{\,t+1}$ after observing $\vec{o}$. Note that although (13) was originally presented to compute the value in terms of policy tree, it can be extended to the policy graph immediately, by treating $\vec{q}^{\,t}$ as joint nodes in policy graph. In the remaining discussion, we use the terms "tree" and "graph node" interchangeably.

[3] For the problems with few states, we simply uniformly sample the belief space to obtain $\mathcal{B}$ since the heuristic beliefs would be very close to each other for some trials.

A straightforward way to construct $\vec{q}^{\,t+1}$ from $\vec{q}^{\,t}_{\vec{o}}$ is to consider all possible subtrees; however, this approach is exponential in the number of observations, and hence is intractable if the size of the observation set is large (Seuken and Zilberstein 2007a). This issue was overcome in (Wu, Zilberstein, and Chen 2010a) by using a stochastic mapping from observations to subtrees, i.e., $\pi(q_i^t|o_i)$, which represents the probability of subtree q_i^t given the observation o_i. Consequently, we can select the subtree based on this distribution rather than considering all possible subtrees; in this way, the policy tree is constructed more efficiently, even for a large observation set. With this stochastic mapping, we obtain from (13) the following value function given a belief b:

$$\hat{V}^{t+1}(b,\pi) = \sum_s b(s)\, R(s,\vec{a}) + \sum_{s',s,\vec{o},\vec{q}^{\,t}} p(s'|s,\vec{a})\, b(s) \; p(\vec{o}|s',\vec{a})\, V^t(s',\vec{q}^{\,t}) \prod_i \pi(q_i^t|o_i). \quad (14)$$

The goal is to find the stochastic mappings $\{\pi(q_i^t|o_i)\}_{i=1}^n$ such that the righthand side of (14) is maximized. The maximization is performed by taking turns to solve the mapping for one agent, fixing the mappings for other agents. The original PBPG solves each subproblem using linear programming, which requires an iterative procedure. Here we show that the solution for each subproblem can be expressed in closed-form, avoiding the cost of linear programming.

Since the first term on the right side of (14) does not involve $\pi(q_i^t|o_i)$, $\forall\, i$, we need only consider the second term, which can be rewritten as

$$\widetilde{V}(\vec{\pi}^{\,t},b) = \sum_{s',\vec{o},\vec{q}^{\,t},s} p(\vec{o}|s',\vec{a})\, p(s'|s,\vec{a})\, b(s)\, V(s',\vec{q}^{\,t}) \prod_i \pi(q_i^t|o_i) \quad (15)$$

Focusing on agent i and keeping $\{\pi(q_j^t|o_j) : j \neq i\}$ fixed, we obtain a simple function in $\pi(q_i^t|o_i)$ alone,

$$\begin{aligned}\widetilde{V}(\vec{\pi}^{\,t},b) &= \sum_{o_i,q_i^t} \pi(q_i^t|o_i) \sum_{s',o_{-i},q_{-i}^t} p(\vec{o}|s',\vec{a}) \; p(s'|\vec{a},b)\, V(s',\vec{q}^{\,t}) \prod_{j\neq i} \pi(q_j^t|o_j) \\ &= \sum_{o_i,q_i} \pi(q_i^t|o_i)\, W(q_i^t,o_i) \quad (16)\end{aligned}$$

where

$$W(q_i^t,o_i) = \sum_{s',o_{-i},q_{-i}^t} p(\vec{o}|s',\vec{a}) p(s'|\vec{a},b)\, V(s',\vec{q}^{\,t}) \prod_{j\neq i} \pi(q_j^t|o_j).$$

It follows from the rightmost side of (16) that the solution to the problem $\max_{\pi(q_i^t|o_i)} \widetilde{V}(\vec{\pi}^t,b)$ is given in closed-form by

$$\pi^{(t)}(q_i^t|o_i) = \begin{cases} 1, & \text{if } q_i^t = \arg\max_{\hat{q}_i^t} W(\hat{q}_i^t,o_i) \\ 0, & \text{otherwise.} \end{cases} \quad (17)$$

Algorithm 2 Modified PBPG for finite-horizon DEC-POMDPs

Initialization: Horizon T, maximum number of trees K, empty policy graph $\mathcal{G}$, initial belief b_0, belief portfolio $\mathcal{B}$.

Determine the policy graph at level 1 from OneStep(K, b_0, $\mathcal{B}$).
for $t = 1$ to $T-1$ **do**
  for $k = 1$ to K **do**
    while The optimal node q^* has appeared in $\mathcal{G}^{t+1}(1:k-1)$ **do**
      Generate belief b from $\mathcal{B}$. Set $v^* = -\infty$ and $q^* = \emptyset$
      for all $\vec{a}$ **do**
        Iteratively determine the mapping $\pi(q_i^t|o_i)$, $\forall\, i$, using (17).
        Build the local node q_i^{t+1} with a tuple $\langle a_i, o_i, \arg\max_{q_i^t} \pi(q_i^t|o_i)\rangle, \forall i$.
        Evaluate $\vec{q}^{\,t+1}$ with belief b to obtain v.
        if $v > v^*$ **then**
          Let $v^* = v$ and $q^* = \vec{q}^{\,t+1}$.
        end if
      end for
    end while
    Add the joint node to $\mathcal{G}$ as $\mathcal{G}^{t+1}(k) = q^*$.
  end for
end for
Return the policy graph $\mathcal{G}$.

// $\mathcal{G}^1$ = OneStep(K, b_0, $\mathcal{B}$)
for $k = 1$ to K **do**
  while The optimal node q^* has appeared in $\mathcal{G}^1(1:k-1)$ **do**
    Generate belief b from $\mathcal{B}$. Set $v^* = -\infty$ and $q^* = \emptyset$
    for all $\vec{a}$ **do**
      Build the local node q_i^1 with $a_i, \forall i$.
      Evaluate $\vec{q}^{\,1}$ with belief b to obtain v.
      if $v > v^*$ **then**
        Let $v^* = v$ and $q^* = \vec{q}^{\,1}$.
      end if
    end for
  end while
  Add the joint node to $\mathcal{G}$ as $\mathcal{G}^1(k) = q^*$.
end for

Given $\{\pi(q_j^t|o_j) : j \neq i\}$, the optimal $\pi(q_i^t|o_i)$ is given by (17). By cyclically employing (17) for $i = 1, \cdots, n$ and repeating the process, one obtains sequence $\{\pi^{(t)}(q_j^t|o_j)\}_{j=1}^n$ which converge to the optimal mappings. We also employ random restart to avoid local optimum, as used in (Wu, Zilberstein, and Chen 2010a) and (Pajarinen and Peltonen 2011). Note that the expression in (17) corresponds to a *deterministic* mapping since given o_i, the mapping at the next level q_i^t is unique. Compared with the approach in (Wu, Zilberstein, and Chen 2010a), the modified approach avoids the cost of linear programming and therefore is computationally more efficient.

Transforming the policy graph into an FSC

Good initialization speeds up an iterative algorithm and improves the solution quality. In PeriEM (Pajarinen and Peltonen 2011), a finite-horizon policy graph was first learned based on (Wu, Zilberstein, and Chen 2010a) where the linear programming step is replaced by direct search; the obtained policy graph is further improved and then transformed into a periodic FSC by connecting the last layer to the first deterministically; the periodic FSC was then again improved and subsequently employed as a good initialization for the DBN-EM algorithm. Adopting this idea, we transform the finite-horizon policy graph returned by the modified PBPG algorithm into FSCs, which are subsequently used to initialize the PIEM algorithm (see Algorithm 1). Our proposed approach differs from (Pajarinen and Peltonen 2011) in that we do not attempt to determine the connection from the last layer to the first layer deterministically and instead set it randomly.

To determine the node selection function, we treat each node $q_i^{\mathcal{T}}$ in the tree as a node $z_i^{\mathcal{F}}$ in the FSC and set

$$p(a_i^{\mathcal{T}} \mid z_i^{\mathcal{F}}) = 1.0 \quad (18a)$$

$$p(a_i \mid z_i^{\mathcal{F}}) = 0.0,\ \forall a_i \neq a_i^{\mathcal{T}} \quad (18b)$$

where $a_i^{\mathcal{T}}$ is the corresponding action at node $q_i^{\mathcal{T}}$ in the tree.

The node transition function of a policy tree above the bottom level is determined by following the flow of the tree. For example, if a node $q_j^{\mathcal{T}}$ follows observation o and a node $q_i^{\mathcal{T}}$ with action a in the policy tree, we set the corresponding node transition function in FSC as

$$p(z_j^{\mathcal{F}} \mid z_i^{\mathcal{F}}, o) = 1.0 \quad (19a)$$

$$p(z_j \mid z_i^{\mathcal{F}}, o) = 0.0,\ \forall z_j \neq z_j^{\mathcal{F}} \quad (19b)$$

The remaining node transition from the nodes at the bottom level to the nodes at the top level are set randomly from a uniform distribution.

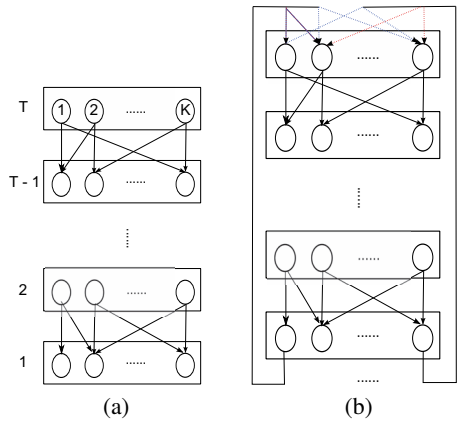

Figure 1: Policy graph. (a) Finite-horizon (b) infinite-horizon.

Fig. 1 shows an example of how to transform the finite-horizon policy graph into an FSC with T levels and K nodes in each level. After obtaining the policy graph (here we only draw the policy graph for one agent, for simplicity) by solving the finite-horizon DEC-POMDP problem, we transform the action nodes of the tree into the corresponding nodes of an FSC and then set the node selection function according to (18). We further set the node transition function in the FSC by using (19). Finally, we let each node at the lowest level make random transitions to the nodes at the top level (each color indicates the transitions upon receiving a distinct observation), as shown in Fig. 1(b). The resulting FSC is used as an initialization for the PIEM algorithm.

Recall that PeriEM uses DBN-EM to refine the initial FSC, while we use PIEM to refine the initial FSC. As shown in the complexity analysis, PIEM has lower asymptotic computational complexity than DBN-EM for FSCs with the same total number of node (i.e., Z). Therefore, when initializing the FSC with the same number of levels and the same number of nodes in each level, PIEM has lower complexity than PeriEM.

Experimental Results

We compare PIEM to state-of-the-art infinite-horizon algorithms for DEC-POMDPs. We consider a total of seven benchmark DEC-POMDP domains: Dec Tiger, Broadcast, Recycling Robot, Meeting in a 3×3 Grid, Box Pushing, Wireless Network, and Mars Rover. A more complete description of these benchmarks, except Wireless Network, is available at http://rbr.cs.umass.edu/camato/decpomdp/download.html. The model description for Wireless Network is available at http://masplan.org/problem_domains.

Both PBPG and modified PBPG require a portfolio of beliefs, denoted $\mathcal{B}$, for generating subtrees. The original PBPG generates a belief randomly with a 0.55 chance and by an MDP heuristic with a 0.45 chance. The modified PBPG employs a trial-based method to efficiently sample the beliefs (Wu, Zilberstein, and Chen 2010b) with the number of trials set to be 20. We make a small change to this belief-generation method, by placing two base beliefs, namely, the initial belief and the uniform belief, in $\mathcal{B}$ *a priori* before the random beliefs and MDP beliefs are generated. This change, though small, can significantly affect the diversity of the belief portfolio and thus influence the performance of the resulting policy, as seen shortly in the results.

Due to the stochasticity in sampling the random and MDP beliefs, and the initialization for the infinite-horizon policy graph, we perform 10 independent trials of each experiment and report the average results. We implemented modified PBPG and PIEM by `C++` and ran them on a Linux machine with Intel i5-4440 3.1 GHz Quad-Core CPU and 1 GB available memory.

We test our proposed PIEM algorithm on seven benchmark problems, of which Wireless Network has a discount factor $\gamma = 0.99$ and all others have a discount factor of $\gamma = 0.9$. In addition to the dynamic Bayesian net based EM algorithms, i.e., DBN-EM and PeriEM, we also compare our PIEM with the latest state-of-the-art methods: Peri (Pajarinen and Peltonen 2011), NLP (Amato, Bernstein, and Zilberstein 2007), Mealy-NLP (Amato, Bonet, and Zilberstein 2010), PBVI-BB (MacDermed and Isbell 2013), and FB-HSVI (Dibangoye, Buffet, and Charpillet

Table 1: A comparison of different methods, in terms of policy value and solver size, on seven infinite-horizon DEC-POMDP benchmark problems.

Algorithm	Value	Size
Dec Tiger (S = 2, A = 3, O = 2)		
Peri	$\mathbf{13.45}$	10×30
FB-HSVI	13.448	25
PBVI-BB	13.448	231
PIEM	12.97	10×100
PeriEM	9.42	7×10
Mealy-NLP	-1.49	4
DBN-EM	-16.30	6
Broadcast (S = 4, A = 2, O = 5)		
PIEM	$\mathbf{9.1}$	1×30
NLP	$\mathbf{9.1}$	1
DBN-EM	9.05	1
Recycling Robot (S = 4, A = 3, O = 2)		
PIEM	$\mathbf{31.929}$	6×100
FB-HSVI	$\mathbf{31.929}$	109
PBVI-BB	$\mathbf{31.929}$	37
Mealy-NLP	$\mathbf{31.928}$	1
Peri	31.84	6×30
PeriEM	31.80	6×10
DBN-EM	31.50	2
Wireless Network (S = 64, A = 2, O = 6) †		
PIEM	$\mathbf{-162.30}$	15×100
PBVI-BB	-167.10	374
DBN-EM	-175.40	3
Peri	-181.24	15×100
PeriEM	-218.90	2×10
Mealy-NLP	-296.50	1
Meeting in a 3×3 Grid (S = 81, A = 5, O = 9)		
PIEM	$\mathbf{5.82}$	8×100
FB-HSVI	5.802	108
Peri	4.64	20×70
Box Pushing (S = 100, A = 4, O = 5)		
FB-HSVI	$\mathbf{224.43}$	331
PBVI-BB	224.12	305
Peri	148.65	15×30
Mealy-NLP	143.14	4
PIEM	138.40	15×30
PeriEM	106.68	4×10
DBN-EM	43.33	6
Stochastic Mars Rover (S = 256, A = 6, O = 8)		
FB-HSVI	$\mathbf{26.94}$	136
Peri	24.13	10×30
PIEM	20.20	10×30
Mealy-NLP	19.67	3
PeriEM	18.13	3×10
DBN-EM	17.75	3

†: The discount used by FB-HSVI is $\gamma = 0.9$, as reported in (Dibangoye, Buffet, and Charpillet 2014).

2014) since these methods have been reported to achieve the highest policy values for the benchmark problems considered here, according to http://rbr.cs.umass.edu/camato/decpomdp/download.html. The results of DBN-EM, Peri, PeriEM, PBVI-BB, and FB-HSVI are cited from (Pajarinen and Peltonen 2011; MacDermed and Isbell 2013; Dibangoye, Buffet, and Charpillet 2014). Our focus here is to compare PIEM with DBN-EM and PeriEM, which both belong to the family of EM-based methods. Nonetheless, it is still interesting to check the performance gap between PIEM and other types of solvers.

Following (Pajarinen and Peltonen 2011), we set a time limit of two CPU hours for PIEM. The number of EM iterations in policy improvement for PIEM is set to 5. The length of the policy graph for modified PBPG is set to be $T = 100$ for Dec Tiger, Recycling Robot, Wireless Network, and Meeting in a 3×3 Grid problems, and $T = 30$ for the remaining problems. In terms of implementation, we employ the sparsity property in the transition probability, observation probability, and belief vector, with the purpose of improving code efficiency.

The results in terms of policy value and solver size [4], on the infinite-horizon DEC-POMDP benchmarks, are summarized in Table 1, which shows that PIEM achieves policy values higher than, or similar to, all of its competitors, on all of the benchmark problems except for Box Pushing and Stochastic Mars Rover. Furthermore, we observe that PIEM outperforms DBN-EM and PeriEM on all of the benchmarks, with obvious policy value gains. The reason why PIEM overall performs better than PeriEM is that given a time limit, PIEM could handle more FSC nodes than PeriEM which is consistent with (i) our time complexity analysis which shows that PIEM has lower computational complexity than DBN-EM (PeriEM uses DBN-EM to refine the initial FSC); (ii) the closed-form solution for the stochastic mapping makes modified PBPG (which is used to initialize PIEM) more efficient than the original PBPG (which is used to initialize PeriEM); (iii) our algorithm implementation employs the sparsity of model parameters to speed up computation.

For the last two problems, PIEM is outperformed by some of its competitors, with the value gap particularly significant in the Box Pushing problem when compared with FB-HSVI and PBVI-BB. This is likely due to the fact that a larger problem tends to have more local optima, which could make PIEM more easily converge to suboptimal policies. Furthermore, similar to PeriEM, PIEM cannot handle more nodes in these large problems, given a time limit. In contrast, the point-based FB-HSVI was shown to converge to a much better local optima, with bounded approximation errors. Considering that FB-HSVI outperforms all other methods here on the last two problems, it would be interesting to employ the heuristic strategy to obtain a better belief portfolio in our future work.

[4] Referred to the number of hyperplanes for PBVI-BB and FB-HSVI, and the number of FSC nodes for other methods.

Conclusions

We have proposed a new policy-iteration algorithm to learn the polices of infinite-horizon DEC-POMDP problems. The proposed approach is based on direct maximization of the value function, and hence avoids the drawbacks of the state-of-the-art DBN-EM algorithm. Moreover, we prove that PIEM can improve the objective value function monotonically. Motivated by PeriEM, the PIEM algorithm is initialized by FSCs converted from a finite-horizon policy graph, with the latter found by a modified PBPG algorithm, proposed here to speed up the original PBPG by using a closed-form solution in place of linear programming. We have also investigated the connection between policy graphs and FSCs, and show how to initialize the FSC with the policy graph. The experiments on benchmark problems show that the proposed algorithms achieve better or competitive performances in the infinite-horizon DEC-POMDP cases. Future work includes incorporation of heuristic search to create better belief portfolio, investigation of the methods to determine the optimal number of FSC nodes, and extending the algorithms here to when the DEC-POMDP model is not known *a priori*.

The technique presented here can be extended to the reinforcement learning (RL) setting where the DEC-POMDP is not given and the agents' policies are learned from their experiences. Such an extension can be made in multiple ways, one of which would be to express each agent policy explicitly in terms of local action-value functions and perform a distributed Q-learning based on a global action-value function composed of the local functions. The agents may still take turns to update their respective local action-value functions, with the update of each local function requiring the parameters of other agents' local functions. With an appropriate parametrization, the communication between agents can be implemented efficiently, We leave this to our next-step work.

Acknowledgements

This research was supported in part by ARO, DARPA, DOE, NGA, ONR and NSF.

References

Amato, C.; Bernstein, D. S.; and Zilberstein, S. 2007. Optimizing memory-bounded controllers for decentralized POMDPs. In *UAI*.

Amato, C.; Bernstein, D. S.; and Zilberstein, S. 2010. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *JAAMAS* 21(3).

Amato, C.; Bonet, B.; and Zilberstein, S. 2010. Finite-state controllers based on mealy machines for centralized and decentralized POMDPs. In *AAAI*.

Bernstein, D. S.; Givan, R.; Immerman, N.; and Zilberstein, S. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research* 27(4).

Bernstein, D. S.; Amato, C.; Hansen, E. A.; and Zilberstein, S. 2009. Policy iteration for decentralized control of markov decision processes. *JAIR* 34(1).

Bernstein, D. S.; Hansen, E. A.; and Zilberstein, S. 2005. Bounded policy iteration for decentralized POMDPs. In *IJCAI*.

Boyd, S., and Vandenberghe, L. 2004. *Convex optimization*. Cambridge university press.

Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum likelihood from incomplete data via the EM algorithm. *J. Roy. Statist. Soc. Ser. B* 39(1):1–38. with discussion.

Dibangoye, J. S.; Buffet, O.; and Charpillet, F. 2014. Error-bounded approximations for infinite-horizon discounted decentralized POMDPs. In *ECML/PKDD*.

Durfee, Z., and Zilberstein, S. 2013. Multiagent planning, control, and execution. In Weiss, G., ed., *Multiagent Systems*. MIT Press.

Hansen, E. A.; Bernstein, D. S.; and Zilberstein, S. 2004. Dynamic programming for partially observable stochastic games. In *AAAI*.

Hansen, E. A. 1997. An improved policy iteration algorithm for partially observable MDPs. In *NIPS*, volume 10.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1).

Kumar, A., and Zilberstein, S. 2010. Anytime planning for decentralized POMDPs using expectation maximization. In *UAI*.

Lange, K.; Hunter, D. R.; and Yang, I. 2000. Optimization Transfer Using Surrogate Objective Functions. *Journal of Computational and Graphical Statistics* 9(1).

MacDermed, L. C., and Isbell, C. 2013. Point based value iteration with optimal belief compression for Dec-POMDPs. In *NIPS*.

Nair, R.; Tambe, M.; Yokoo, M.; Pynadath, D.; and Marsella, S. 2003. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *IJCAI*.

Neal, R., and Hinton, G. 1998. A view of the EM algorithm that justifies incremental, sparse, and other variants. In Jordan, M. I., ed., *Learning in Graphical Models*. Kluwer Academic Publishers.

Oliehoek, F. A. 2012. Decentralized POMDPs. In *Reinforcement Learning*. Springer.

Pajarinen, J. K., and Peltonen, J. 2011. Periodic finite state controllers for efficient POMDP and DEC-POMDP planning. In *NIPS*.

Poupart, P., and Boutilier, C. 2003. Bounded finite state controllers. In *NIPS*.

Seuken, S., and Zilberstein, S. 2007a. Improved memory-bounded dynamic programming for decentralized POMDPs. In *UAI*.

Seuken, S., and Zilberstein, S. 2007b. Memory-bounded dynamic programming for DEC-POMDPs. In *IJCAI*.

Sondik, E. J. 1971. *The Optimal Control of Partially Observable Markov Processes*. Ph.D. Dissertation, Stanford University.

Sondik, E. J. 1978. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research* 26.

Toussaint, M.; Harmeling, S.; and Storkey, A. 2006. Probabilistic inference for solving (PO)MDPs. Technical Report EDIINF-RR-0934, School of Informatics, University of Edinburgh.

Toussaint, M.; Storkey, A.; and Harmeling, S. 2011. Expectation-maximization methods for solving (PO)MDPs and optimal control problems. In Chiappa, S., and Barber, D., eds., *Bayesian Time Series Models*. Cambridge University Press.

Vlassis, N., and Toussaint, M. 2009. Model-free reinforcement learning as mixture learning. In *ICML*.

Wu, F.; Zilberstein, S.; and Chen, X. 2010a. Point-based policy generation for decentralized POMDPs. In *AAMAS*.

Wu, F.; Zilberstein, S.; and Chen, X. 2010b. Trial-based dynamic programming for multi-agent planning. In *AAAI*.