Policy Graph Pruning and Optimization in Monte Carlo Value Iteration for Continuous-State POMDPs

Weisheng Qian, Quan Liu, Zongzhang Zhang, Zhiyuan Pan and Shan Zhong
School of Computer Science and Technology, Soochow University, Suzhou 215006, P.R. China
Email: qianweisheng@hotmail.com, {quanliu, zzzhang}@suda.edu.cn, owenpzy@gmail.com, sunshine-620@163.com

***Abstract*—Nowadays, Partially Observation Markov Decision Processes (POMDPs) provide a principled mathematical framework for solving some realistic problems with continuous spaces. The recently introduced Monte Carlo Value Iteration (MCVI) can tackle such problems with continuous state spaces. It uses a policy graph implicitly to represent the value function, instead of using a set of α-functions explicitly. However, the size of its graph would grow over time and it doesn't take any measure to optimize the graph. This makes it not applicable for the devices with limited resources such as wearable watches. This paper introduces three novel techniques to prune and optimize the policy graph obtained by MCVI. First, we optimize the internal structure of a policy graph G whenever a new node is added into the policy graph. Second, we evaluate the value of each node in G and prune the nodes dominated by others. Third, we prune the redundant nodes, meaning that they are not reachable from the initial action node in any optimal policy graph. Empirical results show that, on the corridor and musical chairs problems, our pruning and optimization methods are useful for constructing more compact policy graphs with comparable qualities.**

I. Introduction

Partially Observable Markov Decision Processes (POMDPs) provide a natural framework for planning under uncertainty [1]. In POMDP planning, the robot's possible states are represented probabilistically as a *belief* and we can systematically reason over the belief space in order to derive a policy with imperfect state information. There are two major computational challenges in POMDP planning. The first is the "curse of dimensionality": the size of belief space grows exponentially. For example, a robot navigates in a planar environment modeled as a 5×5 grid. The resulting belief space is 25-dimensional! The second obstacle is the "curse of history". In POMDP planning tasks, a robot often takes many actions before reaching its goal which results in a long planning horizon. The complexity of the planning task grows very fast with the horizon.

It is notoriously intractable to solve POMDP problems exactly because of highly computational complexity [2]. However, tremendous progress has been made in recent years to develop approximate POMDP solutions [3]–[5] including a number of point-based POMDP algorithms [6], [7]. HSVI2 [8] and SARSOP [9] are the well-known POMDP algorithms for discrete models. Over the last decades, the POMDP model has gained a great success for robotic applications such as robot navigation [10]–[13], active sensing [14], object grasping [15], target exploration [8] and spoken dialogue systems [16]. Efficient POMDP algorithms [17]–[21] today assume discrete models, in which a robot's states, actions and observations are all discrete. However, continuous models are often much more natural. Monte Carlo Value Iteration (MCVI) [22] can tackle the POMDP planning problems with very large discrete state spaces or continuous state spaces, but the size of its policy graph grows over time. This makes the graph inapplicable for the mobile and embedded devices with limited resources, such as smart phones and wearable systems. We are motivated to develop a new method to improve the policy graph and make it much more compact.

In this paper, we describe three novel techniques to prune and optimize the policy graph gained from the MCVI algorithm. The first way is to optimize the internal structure of a policy graph G when MCVI updates the graph at a belief b. We evaluate each new node which would be added into G. Some new nodes do not need to be added into G if we could optimize the internal structure of G. The second one is to evaluate the value of each node in G and prunes the nodes if they are dominated by some other ones. During this process, we can get a new policy graph G' with a value equal to or better than the value of the original one. The third one is to prune the redundant nodes in G. We can prune the nodes if they are not reachable from the initial action node when we obtain an optimal policy graph. We show the ways of constructing more compact policy graphs with comparable qualities.

This paper is structured as follows. Section II introduces the foundations about POMDPs and policy graphs. We also simply review the MCVI algorithm. Section III explains three steps to improve the MCVI algorithm. At each step, we can prune some nodes and get a more compact policy graph. Section IV reports experiments with two benchmark problems. Section V reports related works about algorithms for continuous-state POMDPs. Section VI concludes this paper and lists some issues that we will study in the future.

II. Background

The continuous-state POMDP framework is a rich mathematical model for single robot's sequential decision making. Its goal is to maximize the *Average Discounted Reward* (ADR) in a partially observable and stochastic environment. It can be defined as a tuple $\langle S, A, Z, T, \Omega, R, \gamma \rangle$. In this tuple, S is a set of states, A is a set of actions, Z is a set of observations, T is a transition function defined as T: $S \times A \times S \to [0, 1]$, Ω is an observation function defined as Ω: $A \times S \times Z \to [0, 1]$, R is a reward function defined as R: $S \times A \times S \to \mathbb{R}$ and $\gamma \in [0, 1)$

is a discount factor which makes the total reward finite and the problem well defined. At each time step t, the robot takes some action $a \in A$ from a start state s to an end state s' and then it would receive an observation $z \in Z$ and a reward R. The ADR is given by $\mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$, where s_t and a_t denote the robot's state and action at time t, respectively. In this paper, we only consider POMDPs with discrete action and observation spaces.

A *belief state* (or *belief*) b is a sufficient statistic for the history of actions and observations. A belief state space $\mathcal{B}$ is a set of all possible beliefs. The robot will arrive at a new belief $b^{a,z}$ $(= \tau(b, a, z))$ when it takes action a and receives observation z according to Bayes' theorem:

$$b^{a,z}(s') = \frac{\Omega(a, s', z) \int_{s \in S} T(s, a, s')\, b(s)}{\Pr(z|a, b)}, \tag{1}$$

where $\Pr(z|a, b) = \int_{s' \in S} \Omega(a, s', z) \int_{s \in S} T(s, a, s')\, b(s)$.

The goal of solving a POMDP planning problem is to find an optimal policy π that maximizes the ADR. A POMDP policy π: $\mathcal{B} \to A$ maps a belief $b \in \mathcal{B}$ to a prescribed action $a \in A$. In other words, π indicates which action a to be taken at belief b. The value function $V^{\pi}(b)$ specifies the ADR of executing π starting from b:

$$V^{\pi}(b) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(b_t, \pi(b_t)) \,|b_0 = b\right], \tag{2}$$

where $R(b_t, \pi(b_t)) = \int_{s \in S} R(s, \pi(b_t))\, b_t(s)$.

When a policy π maximizes the value function V^{π}, it is the optimal policy and denoted as π^*. π^* specifies the optimal action to take at the current step for each b if the robot acts optimally at future time steps. The value of an optimal policy π^* is defined by the optimal value function V^*. It satisfies the Bellman optimality equation:

$$V^* = H_{\text{POMDP}} V^*, \tag{3}$$

where H_{POMDP} is the Bellman backup operator for POMDPs, defined as:

$$V^*(b) = \max_{a \in A}\left[R(b_t, a) + \gamma \sum_{z \in Z} p(z|b, a) V^*(b^{a,z})\right]. \tag{4}$$

We ensure the solution is optimal when Eq. (4) holds for every $b \in \mathcal{B}$. V^* can be approximated arbitrary well by a piecewise linear and convex (PWLC) value function. The value function also can be defined as:

$$V(b) = \max_{\alpha \in \Gamma} \int_{s \in S} \alpha(s) b(s)\, ds, \tag{5}$$

where $\alpha \in \Gamma$ is a function over S. We commonly call it α-function. The gradient of the value function at b is given by the function $\alpha^b = \arg\max_{\alpha \in \Gamma} \int_{s \in S} \alpha(s) b(s)\, ds$.

A policy graph G could be seen as a Finite State Controller (FSC). We use a tuple $\langle \mathcal{N}, \phi, \psi \rangle$ to represent the graph, which is defined by a set $\mathcal{N}$ of nodes n, a mapping ϕ: $\mathcal{N} \to A$ indicating which action a to be taken at each node n and a mapping $\psi : \mathcal{N} \times Z \to \mathcal{N}$ indicating that the edge rooted at

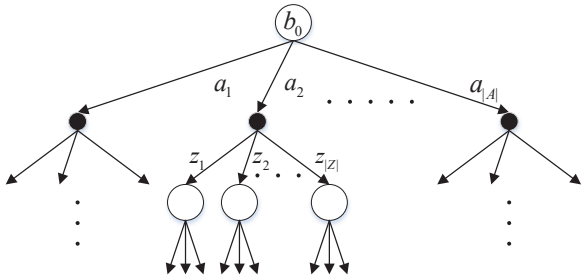


Fig. 1. The belief tree $T_{\mathcal{R}}$ with the initial state b_0.

n and labeled by z should point to n'. The value α_n of the graph when the robot starts from n is computed as follows:

$$\alpha_n(s) = R(s, \phi(n)) + \gamma \int_{s' \in S} \Pr(s'|s, a) \sum_{z \in Z} \Pr(z|s', a) \alpha_{\psi(n,z)}(s') \quad \forall n, s. \tag{6}$$

Here, $\alpha_n(s)$ is similar to the α-function.

Given a belief state b, the value rooted at node n for continuous state space is $\alpha_n(b) = \int_{s \in S} b(s)\, \alpha_n(s)$. The optimal value of G can be computed as:

$$\alpha_G(b) = \max_{n \in \mathcal{N}} \alpha_n(b), \tag{7}$$

where $\alpha_G(b)$ is piecewise-linear and convex since $\alpha_n(b)$ is linear with respect to the belief state.

Value iteration (VI) is a common way to compute the optimal POMDP value function V^*. A *backup* is an iteration of the value function and constructs a new value V_{t+1} from the current value V_t with the backup operator H_{POMDP}. V_t converges to the unique optimal value function V^* after a sufficient large number of iterations. There are many benefits to represent a value function as a set of α-functions, but it is difficult to store and compute α-functions over high-dimensional, continuous spaces. In the MCVI algorithm, it represents a value function implicitly as a policy graph instead of a set of α-functions explicitly. Let V_G denote the value function of the current policy graph G, we get a new value function $V_{G'}$ of the new graph G' after performing a *backup*:

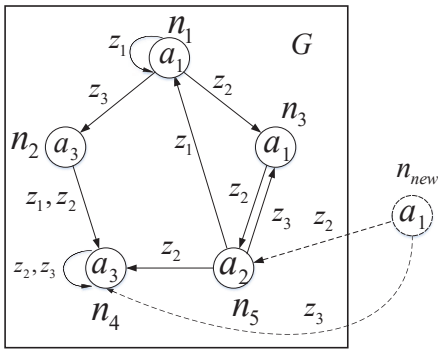
$$V_{G'}(b) = H_{\text{POMDP}} V_G(b) = \max_{a \in A}\left[\int_{s \in S} R(s, a) b(s)\, ds + \gamma \sum_{z \in Z} p(z|b, a) \max_{n \in \mathcal{N}} \int_{s \in S} \alpha_n(s) b^{a,z}(s)\, ds\right]. \tag{8}$$

The entire right-hand side of Eq. (8) can be evaluated via sampling and MC simulations. This process is called MC-backup. The MCVI algorithm uses MC-backup and particles filtering to handle continuous state spaces. It computes an approximate optimal policy by updating the policy graph. $\mathcal{R} \subseteq \mathcal{B}$ is defined as a subset of reachable beliefs from a given initial belief $b_0 \in \mathcal{B}$. The MCVI algorithm samples a

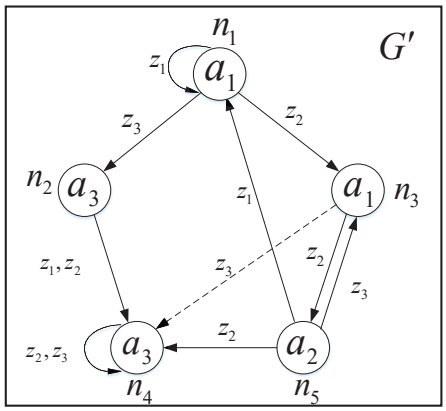
Algorithm 1 Optimize the policy graph G when adding a new node n_{new}

OPTIMAL-BACKUP(G, n_{new})

Initialize a new graph $G' \leftarrow \emptyset$.
$\mathcal{N}_{\phi(n_{new})} \leftarrow \emptyset$.
$c \leftarrow 0$.
for all $n \in \mathcal{N}$ **do**
  if $\phi(n) = \phi(n_{new})$ **then**
    $\mathcal{N}_{\phi(n_{new})} \leftarrow \mathcal{N}_{\phi(n_{new})} \cup \{n\}$.
  end if
end for
for all $n \in \mathcal{N}_{\phi(n_{new})}$ **do**
  for all $z_n \in \bar{Z}_n$ **and** $z_{n_{new}} \in \bar{Z}_{n_{new}}$ **do**
    if $z_n = z_{n_{new}}$ **and** $\psi(n, z_n) \neq \psi(n_{new}, z_{n_{new}})$ **then**
      $c \leftarrow c + 1$.
      break.
    end if
    Choose $z \in \bar{Z}_{n_{new}}$ and $z \notin \bar{Z}_n$.
    $\bar{Z}_n \leftarrow \bar{Z}_n \cup \{z\}$.
    $\psi(n, z) \leftarrow \psi(n_{new}, z)$.
    $G' \leftarrow optimizeGraph(G)$.
  end for
end for
if $c = |\mathcal{N}_{\phi(n_{new})}|$ **then**
  $\mathcal{N} \leftarrow \mathcal{N} \cup \{n_{new}\}$.
  $G' \leftarrow optimizeGraph(G)$.
end if
return G'.



(a)



(b)

Fig. 2. (a) Backup a policy graph G; (b) new policy graph G' by optimizing G. The dashed lines indicate the new node and edges. $|A| = 3$, $|Z| = 3$.

set of beliefs from the reachable space $\mathcal{R}$ rather than $\mathcal{B}$ for computational efficiency. All the sampled beliefs form a tree $T_{\mathcal{R}}$ (Fig. 1). The algorithm updates $T_{\mathcal{R}}$ by sampling in $\mathcal{R}$. To sample new belief points, MCVI sets a target gap size ϵ between the upper and lower bounds at the root b_0 of $T_{\mathcal{R}}$. It traverses a single path down $T_{\mathcal{R}}$ by choosing at each node the action with the highest upper bound and the observation that makes the largest contribution to the gap at the root of $T_{\mathcal{R}}$. The sampled path terminates when it reaches a node whose gap between the upper and lower bounds is smaller than $\gamma^{-l}\epsilon$, where l is the depth of the node in $T_{\mathcal{R}}$. We then go back to b_0 and perform backup along the way to improve the upper and lower bound estimates. The sampling and backup procedures will be repeated until the gap between the upper and lower bounds at b_0 is smaller than ϵ, and we finally get an approximate optimal policy graph.

III. Policy Graph Improvement

The policy graph in MCVI grows rapidly over time. We now describe three techniques to prune and optimize the policy graph generated by MCVI.

A. Optimize the Backup of Policy Graph

The MCVI algorithm uses MC-backup to back up the policy graph G at a belief b during the sampling and backup procedures. The beliefs are represented with particle filters. MC-backup draws N samples to estimate the value of b. It picks $s_i \in S$ with probability $b(s_i)$ and generates the new state s_i' and resulting observation z_i with probability $T(s_i, a, s_i')$ and $\Omega(a, s_i', z_i)$, respectively. The robot has a higher probability to know where it is located over time and samples s_i near its real state. The observations whose probabilities are 0 would not be sampled. At this moment, the resulting observations of each belief consist of a subset $\bar{Z} \subseteq Z$. After performing a MC-backup, the number of observation edges at the new node n_{new} has a high probability to be less than $|Z|$. The square part in Fig. 2(a) shows a policy graph G after several MC-backups. In G, nodes n_1 and n_5 own all possible observation edges, nodes n_2 and n_4 own two, and node n_3 only owns one, i.e., z_1, with the probability one of observing z_1.

MC-backup generates a new policy graph G' by adding a new node n_{new} into G. This makes the policy graph grow rapidly over time. However, some new nodes could be pruned if we can optimize the internal structure of G. For example, the new node n_{new} in Fig. 2(a) has the same action a_1 with n_3 in G. They both have the observation z_2 which directs to the same next node n_5. If we add a new observation edge z_3 at n_3 and make it direct to n_4, it is not necessary to add n_{new} into G. Fig. 2(b) shows the new graph G' by this method.

Conceptually, Algorithm 1 considers all possible nodes

n_{new} when it updates the policy graph G. G' is the new graph after updating the graph G. $\mathcal{N}$ is a set of all the nodes in G. $\mathcal{N}_{\phi(n_{new})}$ is a set of the nodes who own the same action as n_{new}. c is a counter. The nodes in G could not be optimized when c equals to $|\mathcal{N}_{\phi(n_{new})}|$. The loop in line 4 selects the nodes $n \in \mathcal{N}$ that own the same action as n_{new}. The loop in line 10 optimized the node n if each n and n_{new}'s same observation edge points to the same next node. Line 11 ignores the node n if the observation edges rooted at n and n_{new} are same but they have different next nodes. Lines 16$\sim$19 generate a new graph G' if we can optimize the node n and avoid adding n_{new} into G. If we cannot optimize any node in G, n_{new} would be added into G (line 22).

Theorem 1: Let G' be the policy graph without optimizing the nodes and G'' the optimized one. α' and α'' are the values of G' and G'', respectively. Then, α'' is always equal to α'.

Proof: If there is no node in G could be optimized, the new node n_{new} would be added into G. This means G' and G'' are identical. Obviously, they have the same value. If the node n_i in G could be optimized, the new node n_{new} would not be added into G''. But it does not affect the structure of G'. G' and G'' are similar to Fig. 2(a) and 2(b). In order to prove that α' and α'' are equal. We only need to prove $\alpha'_{n_{new}}(b) = \alpha''_{n_i}(b)$. If the edges rooted at $\alpha'_{n_{new}}$ and α''_{n_i} have the same observations, they would direct to the same next node. The probability of observations at b won't be affect by the policy graph. Although the observation edges rooted at n_i might be more than n_{new}. The probability of the different edges is 0 according to $T_{\mathcal{R}}$ (Fig. 1). So we have $\alpha'_{n_{new}}(b) = \alpha''_{n_i}(b)$ (Eq. (6)) and α'' is always equal to α'. ■

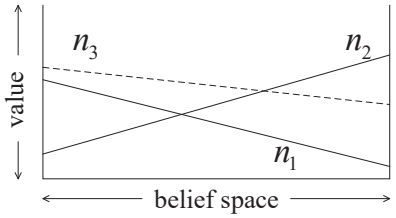
B. Prune the Dominated Nodes

The policy graph often contains some dominated nodes. We describe a technique to prune these nodes while ensuring the value of G does not decrease (Algorithm 2). The way is to prune the nodes that are *pointwise dominated* [23]. A node is pointwise dominated when its value is less than that of some other node at all belief states (e.g., n_1 is pointwise dominated by n_3 in Fig. 3(a)). Fig. 3(b) shows the original policy graph and 3(c) shows the new graph once the dominated node n_1 is pruned.

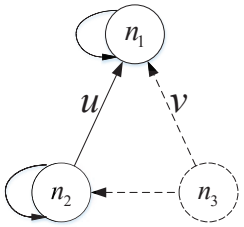
Algorithm 2 shows how to prune the dominated nodes in G. $S_{samples} \subseteq S$ is a set of all the sampled beliefs. The loop in line 3 samples J states with probability distribution b_0 and gains the value at each node by simulation. α is the real value when J and K is big enough. The loop in line 10 prunes the dominated nodes. Their inward edges are redirected to the dominating nodes. The complexity of Algorithm 2 is acceptable with the sparsity [23].

C. Prune the Redundant Nodes

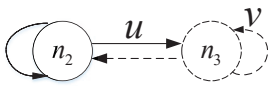
The policy graph G is initialized with a simple default policy (e.g., the robot always performs a single fixed action). The node n_0 in Fig. 4 represents the default policy and directs to itself. Several nodes would be added into G in search procedures [24], such as n_1, n_2 and n_3 in Fig. 4. The subscribe



(a)



(b)



(c)

Fig. 3. (a) Value function of G, n_3 is the dominating node; (b) original policy graph; (c) new policy graph once the dominated node n_1 is removed and its inward edges u, v are redirected to n_3. The dashed lines indicate the dominating node and edges.

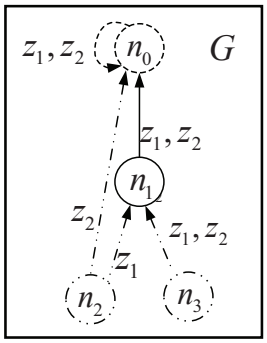


Fig. 4. The optimal policy graph G.

of n_i indicates an sequence when a node is added into G. For example, we add n_1 into G earlier than n_3. Each layer in Fig. 4 represents a search procedure with MC-backup. The edges rooted at node n_i in G always direct to the nodes with smaller subscribe. If the policy graph G in Fig. 4 is optimal after several searches and n_3 is the initial action node at b_0, the robot would not reach n_2 forever. At this point, we can remove n_2 without any loss. Nodes like n_2 are called redundant ones. Algorithm 3 describes how to prune these nodes. We demonstrate that the new graph G' have the same performance with the old one.

Theorem 2: The procedure in Algorithm 3 returns a policy graph with a value equal to the value of the original one.

Proof: Suppose that G' is the updated graph in Algorithm 3 and G is the original one. G' and G have the same initial

$$\alpha^{G'}_{n_{opt}}(s) \leftarrow R(s,\phi(n_{opt})) + \gamma \int_{s'\in S} \Pr(s'|s,a) \sum_{z\in Z} \Pr(z|s',a)\alpha^{G'}_{\psi^{G'}(n_{opt},z)}(s') \quad \forall s, \tag{9}$$

$$\alpha^{G}_{n_{opt}}(s) \leftarrow R(s,\phi(n_{opt})) + \gamma \int_{s'\in S} \Pr(s'|s,a) \sum_{z\in Z} \Pr(z|s',a)\alpha^{G}_{\psi^{G}(n_{opt},z)}(s') \quad \forall s. \tag{10}$$

Algorithm 2 Prune the dominated nodes in G at the initial belief b_0 with J initial states and K samples

PRUNE-DOM(G, b_0, J, K)

repeat
  $S_{samples} \leftarrow \emptyset$.
  for $i = 1$ **to** J **do**
    Sample a state s_i with probability distribution b_0.
    $S_{samples} \leftarrow S_{samples} \cup \{s_i\}$.
    for each $n \in \mathcal{N}$ **do**
      Set $\alpha_n(s_i)$ to be the ADR of K simulations and each simulation starts at n from the initial state s_i.
    end for
  end for
  for each $n_1 \in \mathcal{N}$ **do**
    for each $n_2 \in \mathcal{N} \setminus \{n_1\}$ **do**
      if $\alpha_{n_1}(s) \geq \alpha_{n_2}(s)\ \forall s \in S_{samples}$ **then**
        $\mathcal{N} \leftarrow \mathcal{N} \setminus \{n_2\}$.
        for all $n \in \mathcal{N}$ **and** $z \in \bar{Z}_n$ **do**
          if $\psi(n,z) = n_2$ **then**
            $\psi(n,z) \leftarrow n_1$.
          end if
        end for
      end if
      if $\alpha_{n_1}(s) \leq \alpha_{n_2}(s)\ \forall s \in S_{samples}$ **then**
        $\mathcal{N} \leftarrow \mathcal{N} \setminus \{n_1\}$.
        for all $n \in \mathcal{N}$ **and** $z \in \bar{Z}_n$ **do**
          if $\psi(n,z) = n_1$ **then**
            $\psi(n,z) \leftarrow n_2$.
          end if
        end for
        break.
      end if
    end for
  end for
until G doesn't change
return G

node n_{opt}. We have $\alpha^{G'}_{n_0}(s) = \alpha^{G}_{n_0}(s)\ \forall s$ since the robot always takes the same action at n_0. $\alpha^{G'}_{n_{opt}}$ and $\alpha^{G}_{n_{opt}}$ are their value function. We want to prove $\alpha^{G'}_{n_{opt}}(s) = \alpha^{G}_{n_{opt}}(s)\ \forall s$. The value of G' and G is computed as Eqs. (9) and (10).

G' shares the same transition function $T(s,a,s')$ and observation function $\Omega(a,s',z)$ with G. Algorithm 3 does not change the next node for the same observation at n. Thus, the following equation holds for any n and z:

Algorithm 3 Prune the redundant nodes in an optimal graph G with the initial node n

PRUNE-REDUNDANT(G, n)

for each $n \in \mathcal{N}$ **do**
  if n isn't reachable from n **then**
    $\mathcal{N} \leftarrow \mathcal{N} \setminus \{n\}$
    $G' \leftarrow optimizeGraph(G)$
  end if
end for

$$n' = \psi^{G'}(n,z) = \psi^{G}(n,z) \quad \forall n, z. \tag{11}$$

Here, n' is the next node with observation z at n. Both G' and G share the same observation edges at node n. So, we only need to prove $\alpha^{G'}_{n'}(s) = \alpha^{G}_{n'}(s)\ \forall s$. Like the demonstration of the value at n_{opt}, we just need to demonstrate $\alpha^{G'}_{n''}(s) = \alpha^{G}_{n''}(s)\ \forall s$. With this analogy, we just need to demonstrate $\alpha^{G'}_{n_0}(s) = \alpha^{G}_{n_0}(s)\ \forall s$, and this equation has been clearly established at the beginning. ■

IV. Experiments

We used the mcvi-0.2 software package [22] to implement our pruning and optimization algorithms. The experimental platform is Intel Xeon 4-Core Processor 3.4GHz, 8GB memory. The operation system is Ubuntu Linux 14.04. In Algorithm 2, we estimated the α-functions by running 10000∼100000 simulations, and used the ADR as the value of the nodes. To reduce the variation caused by the randomness in MCVI, each experiment was repeated 10 times.

A. Corridor

We tested our algorithms on the corridor problem [22], [25] and compared the performance with the unimproved one. In this task, a robot can move along the 1-D corridor with four doors. Its target is to enter in the second door from the left. There are three actions that the robot can take: *move-left*, *move-right* and *enter*. Due to the uncertainty, the robot does not know where it is exactly located and only receives four observations: *left-end*, *right-end*, *door* and *corridor*. The robot gets a positive reward when it takes the action of *entering* into the correct door. Otherwise it receives a punishment. If the robot tries to move out of the corridor, it would be punished. Here, we used the same model and parameters as MCVI [22].

We ran Algorithm 1 with 600 particles and 400 samples for each MC-backup. We compared the nodes before and after optimization at some given time points. In Table I, Gap indicates the difference between the upper and lower bounds

TABLE I
THE NUMBER OF NODES BEFORE AND AFTER OPTIMIZING THE BACKUP PROCEDURE

Time (s)	Gap	$\|B\|$	$\mathcal{N}_{ori}$	$\mathcal{N}_{opt}$
10	2.14	14	41	37
100	1.55	47	120	100
200	1.24	70	172	140
400	1.14	99	248	200
600	1.10	124	314	249
800	1.04	146	375	289
1000	1.00	164	424	321
1200	0.96	181	486	365
3600	0.79	317	913	645
7200	0.66	441	1300	892

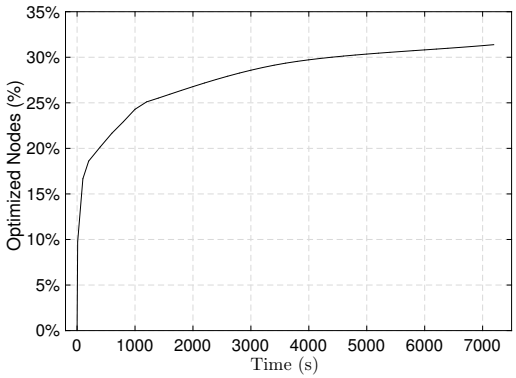


Fig. 5. The percentage of optimized nodes in Algorithm 1 on the corridor problem.

at the initial belief b_0; $|B|$ is the number of beliefs has been sampled; $\mathcal{N}_{ori}$ and $\mathcal{N}_{opt}$ are the number of the nodes in the policy graph before and after optimization, respectively. In comparison of $\mathcal{N}_{ori}$ to $\mathcal{N}_{opt}$ in Table I, we found that Algorithm 1 could remove optimizable nodes in the policy graph efficiently and avoided the graph to grow rapidly.

Fig. 5 shows the percentage of optimized nodes, meaning that they can be removed from the current graph without losing its accuracy, with Algorithm 1. At the beginning, the curve increases fast because there is a large probability of finding optimizable nodes by using a graph with simpler structure. But as time goes on, the probability of finding the optimizable nodes becomes smaller, and therefore, the gradient of the curve becomes smaller and approaches to 0. After two hours, Algorithm 1 has pruned about a third of optimizable nodes.

In Table II, we compared Algorithm 2 with MCVI in different parameter settings. Some dominated nodes have been pruned by Algorithm 2 without sacrificing the graph quality.

To avoid removing useful nodes, Algorithm 3 uses a near optimal policy graph as an input. Both the number of particles and the number of samples are set to be 100. We obtained such a graph with $Gap = 0.22$ by running MCVI in 18000 seconds. There are 1354 belief nodes in the belief tree and 2646 nodes in the graph. After pruning the redundant nodes with Algorithm 3, we got a new policy with 770 nodes. It is amazing that we have pruned about 70 percent of nodes

TABLE II
THE NUMBER OF NODES BEFORE AND AFTER OPTIMIZING THE DOMINATED NODES

Particles and Samples	Time (s)	MCVI Graph		Optimized Graph	
		Nodes	Value	Nodes	Value
$N = 100$, $M = 100$	3600	1198	1.21190	1192	1.21833
	18000	2655	1.36347	2638	1.37628
$N = 400$, $M = 600$	3600	562	1.32464	534	1.32466
	18000	1213	1.50492	1195	1.50691

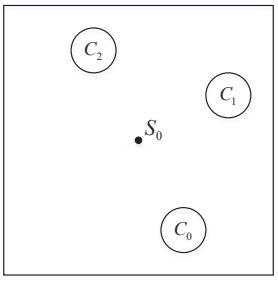


Fig. 6. Musical chairs.

in the original policy graph. Generating such a simpler graph without losing solution quality is attractive when considering to use them into some devices with limited resources.

B. Musical Chairs

The second tested problem is the so-called musical chairs task (see Fig. 6). There are three chairs (C_0, C_1 and C_2) in the classroom. A robot and two people play this game. Obviously, the robot could always find an empty chair. The distance from each chair to the robot is the same. At the beginning, each person randomly chooses whether or not to select a chair to sit down. As a result, there will be one to three chairs empty. The robot's target is to find one empty chair to sit down as fast as it can. There are eight actions the robot can take: *move-east*, *move-north*, *move-west*, *move-south*, *check*C_0, *check*C_1, *check*C_2 and *sit-down*. The robot spends no cost when it moves around the classroom or checks the chairs. When the robot takes the action *sit-down*, it receives a reward $+10$ when it is very close to an empty chair. In other cases, it gets a penalty -10 and the game is over. If the robot moves out of the room, it will receive a penalty -100 and the game is over. Each motion action is stochastic. The real position of the robot is a two-dimension Gaussian distribution around the real position of the deterministic one.

The observations of the robot are *good* and *bad*. Only when the robot takes the action $checkC_i$ and the checked chair is not empty, it would receive the observation *bad*. Otherwise, it receives *good*. The observation of a chair is independent of other chairs (it only depends on the robot position). Because of the noise of the sensors, the robot may receive the wrong perception when it checks a chair. The probability that the robot receives the correct perception is related to the distance from the robot to the chair. Due to the independence of the observations, the complexity of computing $\Pr(z|b,a)$ is

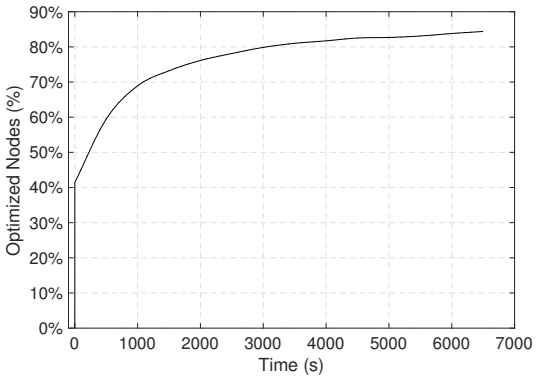


Fig. 7. The percentage of optimized nodes in Algorithm 1 on musical chairs.

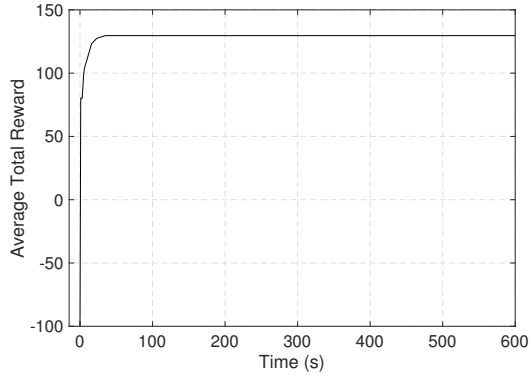


Fig. 8. The average total reward computed by Algorithm 1 on musical chairs.

greatly reduced. The computation of $\Pr(z|b, checkC_i)$ reduces to: $\Pr(z|b, checkC_i) = \Pr(accurate|x_p, checkC_i) \cdot \Pr(x_i^C = z) + (1 - \Pr(accurate|x_p, checkC_i) \cdot (1 - \Pr(x_i^C = z)))$. $\Pr(accurate|x_p, checkC_i) = \frac{1+\eta(x_p,i)}{2}$ is the accuracy of the probability that the robot receives the true observation about chair C_i, where $\eta(x_p, i) = e^{-(d(x_p,i)\cdot\ln(2))/d_0}$. $d(x_p, i)$ is the Euclidean distance between position x_p and the position of chair C_i, and d_0 is a constant which specifies the *half efficiency distance*. $\Pr(x_i^C = z)$ is easily obtained from the probability whether the chair C_i is empty or not. The discount factor γ is 0.95 and the *half efficiency distance* d_0 is 30. The deviation of each motion action is 0.01 in the Gaussian distribution without covariance. The robot starts from the position S_0. If C_i is empty, the robot can sit in C_i when it is in the circle of C_i.

Fig. 7 and 8 show the percentage of the optimized nodes and the average total reward of musical chairs obtained by Algorithm 1, respectively. Here, we used 350 particles and 350 samples. We found that the percentage of optimizable nodes is about 40% initially, because the number of observations were at most two and there are not many nodes in the graph at the beginning stage. We obtained an optimal policy graph within 100 seconds with high accuracy sensors (Fig. 8) and the number of pruned nodes due to its redundance accounts for 96% of the whole nodes in the original graph.

To summarize, by using our pruning and optimization techniques in the MCVI algorithm, more compact policy graphs have been empirically constructed with comparable qualities on the two tested continuous-state POMDP problems.

V. Related Work

Most real world POMDP problems are naturally modeled by continuous-state spaces. In this case, the dimensionality of the belief space is infinite; expected values over states are defined by integrals, for which in general no closed form exists. Thus, simple generalizations of known approaches and complexity results for discrete-state models to continuous-state domains are not appropriate or applicable. To find approximate solutions to continuous-state POMDPs, existing approaches first approximately represent beliefs in the infinite-dimensional space by a low-dimensional discrete representation. In [25]–[27], sufficient statistics are used to reduce the dimension of the belief space automatically. Porta et al. [25] represented the beliefs using Gaussian mixtures or particle sets to guarantee that the belief update and Bellman backup can be computed in closed form. Brooks et al. [26] used finite vectors of sufficient statistics to approximate the belief space. Zhou et al. [27] approximated the belief space by a parameterized density and solved the approximate belief MDP on the parameter space. These approaches share the idea to represent beliefs in the infinite-dimensional continuous belief space by a parametric form and solve the approximate POMDP on the parametric space.

The algorithm recently proposed in Brechtel et al. [28] automatically learn a low-dimensional, discrete representation of the continuous-state space during the process of solving. The insight exploited in [28] is that states which are close in the continuous-state space will usually lead to similar outcomes and thus have similar α-vectors in most problems. They used decision trees to represent the space partitioning. Bai et al. [21], [22] avoided the discretization of the continuous spaces and computed an approximation to an optimal policy by updating a policy graph. Backups were performed on the policy graph, which was a finite-state controller with continuous input and discrete output.

Note that some points in our pruning and optimization techniques are inspired by the work of Grześ et al. [29]. One main difference between the two is that our techniques are designed for problems with continuous state spaces.

VI. Conclusion and Future Work

As a state-of-the-art continuous-state POMDP algorithm, MCVI computes an approximate solution represented as a policy graph by combining Monte Carlo sampling with dynamic programming. However, no pruning and optimization techniques have been used into the policy graph obtained by MCVI. As a result, its graph size grows rapidly over time and has no upper bound. This makes the graph inapplicable for mobile and embedded devices with limited resources. In this paper, we present three techniques to reduce the graph size with few sacrifice in the solution quality, and thus make the graph representation much more compact. Specially, we

optimize the internal structure at the time of adding a new node into the policy graph G; evaluate the value of each node in G and prune the dominated nodes; and prune the nodes that are not reachable from the initial action node through any optimal policy graph. Our experiments are performed in both the corridor task, a classic continuous-state POMDP benchmark problem, and the so-called musical chairs task, a more challenging task designed to test our proposed techniques. Preliminary experimental results show that these new pruning and optimization techniques appear to be useful in constructing more compact policy graphs with comparable qualities through pruning lots of dominated and abundant nodes.

We leave the following problems as future topics. First, we would like to explore ways of reducing the computational time spent in sampling states and simulating with few accuracy loss of α-functions when pruning dominated nodes. Second, we are interested in the possible ways (e.g., linear programming, bounded policy iteration [30]) of pruning the jointly dominated nodes [29] to further reduce the policy graph size. Finally, we would like to extend our method to continuous observation spaces [31] and use macro-actions to further improve its efficiency in handling long planning horizon problems [32].

ACKNOWLEDGEMENTS

This paper was partially supported by National Natural Science Foundation of China (61502323, 61502329, 61272005, 61303108, 61373094, 61472262), Natural Science Foundation of Jiangsu (BK2012616), High School Natural Foundation of Jiangsu (13KJB520020, 16KJB520041), and Suzhou Industrial Application of Basic Research Program Part (SYG201422).

REFERENCES

[1] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, no. 1, pp. 99–134, 1998.

[2] C. Lusena, J. Goldsmith, and M. Mundhenk, "Nonapproximability results for partially observable Markov decision processes," *Journal of Artificial Intelligence Research*, vol. 14, pp. 83–103, 2001.

[3] B. Bonet and H. Geffner, "Solving POMDPs: RTDP-Bel vs. point-based algorithms," in *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 2009, pp. 1641–1646.

[4] G. Shani, J. Pineau, and R. Kaplow, "A survey of point-based POMDP solvers," *Autonomous Agents and Multi-Agent Systems*, vol. 27, no. 1, pp. 1–51, 2013.

[5] T. Smith, "Probabilistic planning for robotic exploration," Ph.D. dissertation, Massachusetts Institute of Technology, 2007.

[6] J. Pineau, G. Gordon, S. Thrun *et al.*, "Point-based value iteration: An anytime algorithm for POMDPs," in *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 2003, pp. 1025–1032.

[7] J. Pineau, G. Gordon, and S. Thrun, "Anytime point-based approximations for large POMDPs," *Journal of Artificial Intelligence Research*, vol. 27, pp. 335–380, 2006.

[8] T. Smith and R. Simmons, "Point-based POMDP algorithms: Improved analysis and implementation," in *Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence (UAI)*, 2005, pp. 542–547.

[9] H. Kurniawati, D. Hsu, and W. S. Lee, "SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces," in *Robotics: Science and Systems*, 2008.

[10] H. Bai, S. Cai, N. Ye, D. Hsu, and W. S. Lee, "Intention-aware online POMDP planning for autonomous driving in a crowd," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 454–460.

[11] A. Foka and P. Trahanias, "Real-time hierarchical POMDPs for autonomous robot navigation," *Robotics and Autonomous Systems*, vol. 55, no. 7, pp. 561–571, 2007.

[12] N. Roy, G. J. Gordon, and S. Thrun, "Finding approximate POMDP solutions through belief compression," *Journal Artificial Intelligence Research*, vol. 23, pp. 1–40, 2005.

[13] M. T. J. Spaan and N. A. Vlassis, "A point-based POMDP algorithm for robot planning," in *2004 IEEE International Conference on Robotics and Automation (ICRA)*, 2004, pp. 2399–2404.

[14] M. T. J. Spaan, T. S. Veiga, and P. U. Lima, "Active cooperative perception in network robot systems using POMDPs," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010, pp. 4800–4805.

[15] K. Hsiao, L. P. Kaelbling, and T. Lozano-Perez, "Grasping POMDPs," in *2007 IEEE International Conference on Robotics and Automation (ICRA)*, 2007, pp. 4685–4692.

[16] J. D. Williams and S. Young, "Partially observable Markov decision processes for spoken dialog systems," *Computer Speech and Language*, vol. 21, no. 2, pp. 393–422, 2007.

[17] Z. Zhang, D. Hsu, and W. S. Lee, "Covering number for efficient heuristic-based POMDP planning," in *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 2014, pp. 26–34.

[18] Z. Zhang, D. Hsu, W. S. Lee, Z. W. Lim, and A. Bai, "PLEASE: Plam leaf search for POMDPs with large observation spaces," in *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, 2015, pp. 249–257.

[19] D. Silver and J. Veness, "Monte-Carlo planning in large POMDPs," in *Proceedings of the 24th Annual Conference on Neural Information Processing Systems (NIPS)*, 2010, pp. 2164–2172.

[20] A. Somani, N. Ye, D. Hsu, and W. S. Lee, "DESPOT: Online POMDP planning with regularization," in *Proceedings of the 27th Annual Conference on Neural Information Processing Systems (NIPS)*, 2013, pp. 1772–1780.

[21] A. Bai, F. Wu, Z. Zhang, and X. Chen, "Thompson sampling based Monte-Carlo planning in POMDPs," in *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS)*, 2014, pp. 28–36.

[22] H. Bai, D. Hsu, W. S. Lee, and V. A. Ngo, "Monte Carlo value iteration for continuous-state POMDPs," in *Algorithmic foundations of robotics IX*, 2010, pp. 175–191.

[23] E. A. Hansen, "An improved policy iteration algorithm for partially observable MDPs," in *Advances in Neural Information Processing Systems (NIPS)*, 1998, pp. 1015–1021.

[24] H. Bai, D. Hsu, M. J. Kochenderfer, and W. S. Lee, "Unmanned aircraft collision avoidance using continuous-state POMDPs," *Robotics: Science and Systems VII*, vol. 1, 2012.

[25] J. M. Porta, N. Vlassis, M. T. J. Spaan, and P. Poupart, "Point-based value iteration for continuous POMDPs," *Journal of Machine Learning Research*, vol. 7, pp. 2329–2367, 2006.

[26] A. Brooks, A. Makarenko, S. Williams, and H. Durrant-Whyte, "Parametric POMDPs for planning in continuous state spaces," *Robotics and Autonomous Systems*, vol. 54, no. 11, pp. 887–897, 2006.

[27] E. Zhou, M. C. Fu, and S. I. Marcus, "Solving continuous-state POMDPs via density projection," *IEEE Transactions on Automatic Control*, vol. 55, no. 5, pp. 1101–1116, 2010.

[28] S. Brechtel, T. Gindele, and R. Dillmann, "Solving continuous POMDPs: Value iteration with incremental learning of an efficient space representation," in *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013, pp. 370–378.

[29] M. Grześ, P. Poupart, X. Yang, and J. Hoey, "Energy efficient execution of POMDP policies," *IEEE Transactions on Cybernetics*, vol. 45, no. 11, pp. 2484–2497, 2015.

[30] P. Poupart and C. Boutilier, "Bounded finite state controllers," in *Advances in Neural Information Processing Systems (NIPS)*, 2003, pp. 823–830.

[31] H. Bai, D. Hsu, and W. S. Lee, "Integrated perception and planning in the continuous space: A POMDP approach," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1288–1302, 2014.

[32] Z. Lim, L. Sun, and D. Hsu, "Monte Carlo value iteration with macro-actions," in *Advances in Neural Information Processing Systems (NIPS)*, 2011, pp. 1287–1295.