APPEARED IN
Int. Workshop on the Algorithmic Foundations of Robotics, 2010

Monte Carlo Value Iteration for Continuous-State POMDPs

Haoyu Bai, David Hsu, Wee Sun Lee, and Vien A. Ngo

Abstract Partially observable Markov decision processes (POMDPs) have been successfully applied to various robot motion planning tasks under uncertainty. However, most existing POMDP algorithms assume a discrete state space, while the natural state space of a robot is often continuous. This paper presents *Monte Carlo Value Iteration* (MCVI) for continuous-state POMDPs. MCVI samples both a robot's state space and the corresponding belief space, and avoids inefficient a priori discretization of the state space as a grid. Both theoretical results and preliminary experimental results indicate that MCVI is a promising new approach for robot motion planning under uncertainty.

1 Introduction

A challenge in robot motion planning and control is the uncertainty inherent in robots' actuators and sensors. Incorporating uncertainty into planning leads to much more reliable robot operation.

Partially observable Markov decision processes (POMDPs) provide a principled general framework for modeling uncertainty and planning under uncertainty. Although POMDPs are computationally intractable in the worst case [13], point-based approximation algorithms have drastically improved the speed of POMDP planning in recent years [12, 14, 19, 20]. Today, the fastest POMDP algorithms, such as HSVI [19] and SARSOP [12], can solve moderately complex POMDPs with hundreds of thousands states in reasonable time. POMDPs have been used successfully to model a variety of robotic tasks, including navigation [3, 18], grasping [8], target tracking [10, 14], and exploration [19]. Most of the existing point-based POMDP algorithms, however, assume a discrete state space, while the natural state space for a robot is often continuous. Our primary goal is to develop a principled and practical POMDP algorithm for robot motion planning in continuous state spaces.

Haoyu Bai, David Hsu, Wee Sun Lee, and Vien A. Ngo
Department of Computer Science, National University of Singapore, Singapore
e-mail: {haoyu,dyhsu,leews,ngoav}@comp.nus.edu.sg

If the state space S is continuous, one common way of using existing POMDP algorithms would be to place a regular grid over S and construct a discrete POMDP model first. The difficulty with this approach is that the number of states grow exponentially with the robot's degrees of freedom (DoFs), resulting in the "curse of dimensionality" well known in geometric motion planning (without uncertainty). The effect of a large number of states is in fact aggravated in POMDP planning. Due to uncertainty, the robot's state is not known exactly and is modeled as a *belief*, which can be represented as a probability distribution over S. We plan in the *belief space* $\mathcal{B}$, which consists of all possible beliefs. The result of POMDP planning is a *policy*, which tells the robot how to act at any belief $b \in \mathcal{B}$. A standard belief representation is a vector b, in which an entry $b(s)$ specifies the probability of the robot being in the discretized state $s \in S$. The *dimensionality* of $\mathcal{B}$ is then equal to the number of states in the discrete POMDP model.

Probabilistic sampling is a powerful idea for attacking the curse of dimensionality [23]. In geometric motion planning, the idea of probabilistically sampling a robot's configuration space led to tremendous progress in the last two decades [5]. Similarly, in POMDP planning, a key idea of point-based algorithms is to sample a small set of points from the belief space $\mathcal{B}$ as an approximate representation of $\mathcal{B}$ rather than represent $\mathcal{B}$ exactly. However, this is not enough, if the robot's state space S is continuous. To compute a policy, we need to evaluate the effect of executing a sequence of actions with an initial belief b. Conceptually, we apply the sequence of actions to *each* state $s \in S$ and average the execution results with probabilistic weights $b(s)$. It is clearly impossible to perform this computation exactly in finite time, as there are infinitely many states in a continuous state space S.

In this paper, we propose *Monte Carlo Value Iteration* (MCVI) for continuous state POMDPs. MCVI samples both a robot's state space S and the corresponding belief space $\mathcal{B}$, and avoids inefficient a priori discretization of the state space as a grid. The main technical innovation of MCVI is to use Monte Carlo sampling in conjunction with dynamic programming to compute a policy represented as a finite state controller. We show that, under suitable conditions, the computed policy approximates the optimal policy with a guaranteed error bound. We also show preliminary results of the algorithm applied to several distinct robotic tasks, including navigation, grasping, and exploration.

In the following, we start with some preliminaries on POMDPs and related work (Section 2). Next, we describe the main idea of MCVI and the algorithmic details (Section 3). We then present experimental results (Section 4). Finally, we conclude with some remarks on future research directions.

2 Background

2.1 Preliminaries on POMDPs

A POMDP models an agent taking a sequence of actions under uncertainty to maximize its total reward. In each time step, the agent takes an action $a \in A$ and moves from a state $s \in S$ to $s' \in S$, where S and A are the agent's state space and action

space, respectively. Due to the uncertainty in actions, the end state s' is represented as a conditional probability function $T(s, a, s') = p(s'|s, a)$, which gives the probability that the agent lies in s', after taking action a in state s. The agent then takes an observation $o \in O$, where O is the observation space. Due to the uncertainty in observations, the observation result is also represented as a conditional probability function $Z(s', a, o) = p(o|s', a)$ for $s' \in S$ and $a \in A$. To elicit desirable agent behavior, we define a reward function $R(s, a)$. In each time step, the agent receives a real-valued reward $R(s, a)$, if it is in state $s \in S$ and takes action $a \in A$. The agent's goal is to maximize its expected total reward by choosing a suitable sequence of actions. When the sequence of actions has infinite length, we typically specify a discount factor $\gamma \in (0, 1)$ so that the total reward is finite and the problem is well defined. In this case, the expected total reward is given by $\mathrm{E}\big(\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\big)$, where s_t and a_t denote the agent's state and action at time t, respectively.

The goal of POMDP planning is to compute an *optimal policy* π^* that maximizes the agent's expected total reward. In the more familiar case where the agent's state is fully observable, a policy prescribes an action, given the agent's current state. However, a POMDP agent's state is partially observable and modeled as a belief, i.e., a probability distribution over S. A POMDP policy $\pi\colon \mathcal{B} \to A$ maps a belief $b \in \mathcal{B}$ to the prescribed action $a \in A$.

A policy π induces a *value function* $V_\pi\colon \mathcal{B} \to \mathrm{R}$. The *value* of b with respect to π is the agent's expected total reward of executing π with initial belief b:

$$V_\pi(b) = \mathrm{E}\Big(\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \;\Big|\; \pi, b\Big). \tag{1}$$

If the action space and the observation spaces of a POMDP are discrete, then the optimal value function V^* can be approximated arbitrarily closely by a piecewise-linear, convex function [15]:

$$V(b) = \max_{\alpha \in \Gamma} \int_{s \in S} \alpha(s) b(s)\, ds, \tag{2}$$

where each $\alpha \in \Gamma$ is a function over S and commonly called an α-function. If the state space is also discrete, we can represent beliefs and α-functions as vectors and replace the integral in (2) by a sum. For each fixed α, $h(b) = \sum_{s \in S} \alpha(s) b(s)$ then defines a hyperplane over $\mathcal{B}$, and $V(b)$ is the maximum over a finite set of hyperplanes at b. In this case, it is clear why $V(b)$ is piecewise-linear and convex.

POMDP policy computation is usually performed offline, because of its high computational cost. Given a policy π, the control of the agent's actions is performed online in real time. It repeatedly executes two steps. The first step is action selection. If the agent's current belief is b, it takes the action $a = \pi(b)$, according to the given policy π. The second step is belief update. After taking an action a and receiving an observation o, the agent updates its belief:

$$b'(s') = \tau(b, a, o) = \eta Z(s', a, o) \int_{s \in S} T(s, a, s') b(s)\, ds, \tag{3}$$

where η is a normalizing constant.

More information on POMDPs is available in [11, 22].

2.2 Related Work

POMDPs provide a principled general framework for planning under uncertainty, but they are often avoided in robotics, because of their high computational complexity. In recent years, point-based POMDP algorithms made significant progress in computing approximate solutions to discrete POMDPs [12, 14, 19, 20]. Their success hinges on two main ideas. First, they sample a small set of points from the belief space $\mathcal{B}$ and use it as an approximate representation of $\mathcal{B}$. Second, they approximate the optimal value function as a set of α-vectors. The α-vectors allow partial policies computed at one belief point to be used for other parts of $\mathcal{B}$ when appropriate, thus bringing substantial gain in computational efficiency.

In comparison, progress on continuous POMDPs has been much more limited, partly due to the difficulty of representing beliefs and value functions for POMDPs when high-dimensional, continuous state spaces are involved. As mentioned earlier, discretizing the state space with a regular grid often results in an unacceptably large number of states. One idea is to restrict beliefs and value functions to a particular parametric form, e.g., a Gaussian [3, 16] or a linear combination of Gaussians [4, 15]. For robots in complex geometric environments with many obstacles, uni-modal distributions, such as the Gaussian, are often inadequate. In theory, a linear combination of Gaussians can partially address this inadequacy. However, when the environment geometry contains many "discontinuities" due to obstacles, the number of Gaussian components required often grows too fast for the approach to be effective in practice. Other algorithms, such as MC-POMDP [21] and Perseus [15], use particle filters to represent beliefs. Perseus still uses a linear combination of Gaussians for value function representation and thus suffers some of the same shortcomings mentioned above. MC-POMDP represents a value function by storing its values at the sampled belief points and interpolating over them using Gaussians as kernel functions and KL divergence as the distance function. Interpolation in a belief space is not easy. KL divergence does not satisfy the metric properties, making it difficult to understand the interpolation error. Furthermore, choosing suitable parameter values for the Gaussian kernels involves some of the same difficulties as those in choosing an a priori discretization of the state space.

MCVI also uses the particle-based belief representation, but it exploits one key successful idea of point-based discrete POMDP algorithms: the α-vectors. It captures the α-functions implicitly as a policy graph [6, 11] and retains their main benefits by paying a computational cost. To construct the policy graph, MCVI makes use of approximate dynamic programming by sampling the state space and performing Monte Carlo (MC) simulations. Approximate dynamic programming has also been used in policy search for Markov decision processes (MDPs) and POMDPs without exploiting the benefits of α-functions [1].

MCVI takes the approach of offline policy computation. An alternative is to perform online search [17, 7]. These two approaches are complementary and can be combined to deal with challenging planning tasks with long time horizons.

3 Monte Carlo Value Iteration

In this paper, we focus on the main issue of continuous state spaces and make the simplifying assumption of discrete action and observation spaces.

3.1 Policy Graphs

One way of representing a policy is a *policy graph* G, which is a directed graph with labeled nodes and edges. Each node of G is labeled with an action $a \in A$, and each edge of G is labeled with an observation $o \in O$. To execute a policy π_G represented this way, we use a finite state controller whose states are the nodes of G. The controller starts in a suitable node v of G, and a robot, with initial belief b, performs the associated action a_v. If the robot then receives an observation o, the controller transitions from v to a new node v' by following the edge (v, v') with label o. The process then repeats. The finite state controller does not maintain the robot's belief explicitly, as in (3). It encodes the belief implicitly in the controller state based on the robot's initial belief b and the sequence of observations received.

For each node v of G, we may define an α-function α_v. Let $\pi_{G,v}$ denote a partial policy represented by G, when the controller always starts in node v of G. The value $\alpha_v(s)$ is the expected total reward of executing $\pi_{G,v}$ with initial robot state s:

$$\alpha_v(s) = \mathrm{E}\big(\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\big) = R(s, a_v) + \mathrm{E}\big(\sum_{t=1}^{\infty} \gamma^t R(s_t, a_t)\big) \tag{4}$$

Putting (4) together with (1) and (2), we define the value of b with respect to π_G as

$$V_G(b) = \max_{v \in G} \int_{s \in S} \alpha_v(s) b(s) ds. \tag{5}$$

So V_G is completely determined by the α-functions associated with the nodes of G.

3.2 MC-Backup

The optimal POMDP value function V^* can be computed with *value iteration* (VI), which is based on the idea of dynamic programming [2]. An iteration of VI is commonly called a *backup*. The backup operator H constructs a new value function V_{t+1} from the current value function V_t:

$$V_{t+1}(b) = HV_t(b) = \max_{a \in A} \Big\{ R(b, a) + \gamma \sum_{o \in O} p(o|b, a) V_t(b') \Big\}, \tag{6}$$

where $R(b, a) = \int_{s \in S} R(s, a) b(s) ds$ is the robot's expected immediate reward and $b' = \tau(b, a, o)$ is the robot's next belief after it takes action a and receives observation o. At every $b \in \mathcal{B}$, the backup operator H looks ahead one step and chooses the action that maximizes the sum of the expected immediate reward and the expected total reward at the next belief. Under fairly general conditions, V_t converges to the unique optimal value function V^*.

Representing a value function as a set of α-functions has many benefits, but storing and computing α-functions over high-dimensional, continuous state spaces is

difficult (Section 2.2). We do not represent a value function explicitly as a set of α-functions, but instead represent it implicitly as a policy graph. Let V_G denote the value function for the current policy graph G. Substituting (5) into (6), we get

$$HV_G(b) = \max_{a\in A}\Big\{\int_{s\in S} R(s,a)b(s)ds + \gamma \sum_{o\in O} p(o|b,a) \max_{v\in G} \int_{s\in S} \alpha_v(s)b'(s)ds\Big\}. \tag{7}$$

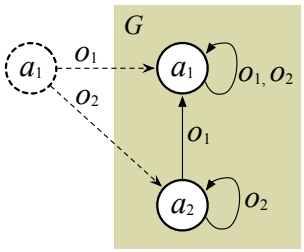


Fig. 1 Backup a policy graph G. The dashed lines indicate the new node and edges.

Let us first evaluate (7) at a particular point $b \in \mathcal{B}$ and construct the resulting new policy graph G', which contains a new node u and a new edge from u for each $o \in O$ (Fig. 1). Since we do not maintain α-functions explicitly, it seems difficult to compute the integral $\int_{s\in S} \alpha_v(s)b'(s)ds$. However, the definition of α_v in (4) suggests computing the integral by MC simulation: repeatedly sample a state s with probability $b'(s)$ and then simulate the policy $\pi_{G,v}$. Pushing further on this idea, we can in fact evaluate the entire right-hand side of (7) via sampling and MC simulation, and construct the new policy graph G'. We call this MC-backup of G at b (Algorithm 1).

Conceptually, Algorithm 1 considers all possible ways of generating G'. The new node u in G' has $|A|$ possible labels, and each outgoing edge from u has $|G|$ possible end nodes in G, where $|G|$ denotes the number of nodes in G (Fig. 1). Thus, there are $|A||G|^{|O|}$ candidates for G'. Each candidate graph G' defines a new policy $\pi_{G',u}$. We draw N samples to estimate the value of b with respect each candidate $\pi_{G',u}$. For each sample, we pick s from the state space S with probability $b(s)$. We run an MC simulation under $\pi_{G',u}$, starting from s, for L steps and calculate the total reward $\sum_{t=0}^{L} \gamma^t R(s_t, a_t)$. The simulation length L is selected to be is sufficiently large so that the error due to the finite simulation steps is small after discounting. We then choose the candidate graph with the highest average total reward. Unfortunately, this naive procedure requires an exponential number of samples.

Algorithm 1 computes the same result, but is more efficient, using only $N|A||G|$ samples. The loop in line 3 matches the maximization over actions $a \in A$ in (7). The loop in line 4 matches the first integral over states $s \in S$ and the sum over observations $o \in O$. The loop in line 8 matches the maximization over nodes $v \in G$. The three nested loops generate the simulation results and store them in $V_{a,o,v}$ for $a \in A, o \in O,$ and $v \in G$. Using $V_{a,o,v}$, one can compare the values at b with respect to any candidate policy graphs and choose the best one (lines 11–14).

Interestingly, a relatively small number of samples are sufficient for MC-backup to be effective. Let $\hat{H}_b V_G$ denote the value function for the improved policy graph resulting from MC-backup of G at b. With high probability, $\hat{H}_b V_G$ approximates HV_G well at b, with error decreasing at the rate $\mathcal{O}(1/\sqrt{N})$. For simplicity, we assume in our analysis below that the simulation length L is infinite. Taking the finite

Algorithm 1 Backup a policy graph G at a belief b with N samples.

MC-BACKUP(G, b, N)

For each action $a \in A$, $R_a \leftarrow 0$.
For each action $a \in A$, each observation $o \in O$, and each node $v \in G$, $V_{a,o,v} \leftarrow 0$.
for each action $a \in A$ **do**
  for $i = 1$ to N **do**
    Sample a state s_i with probability distribution $b(s_i)$.
    Simulate taking action a in state s_i. Generate the new state s'_i by sampling from the distribution $T(s_i, a, s'_i) = p(s'_i|s_i, a)$. Generate the resulting observation o_i by sampling from the distribution $Z(s'_i, a, o_i) = p(o_i|s'_i, a)$.
    $R_a \leftarrow R_a + R(s_i, a)$.
    for each node $v \in G$ **do**
      Set V' to be the expected total reward of simulating the policy represented by G, with initial controller state v and initial robot state s'_i.
      $V_{a,o_i,v} \leftarrow V_{a,o_i,v} + V'$.
  for each observation $o \in O$ **do**
    $V_{a,o} \leftarrow \max_{v \in G} V_{a,o,v}, \quad v_{a,o} \leftarrow \arg\max_{v \in G} V_{a,o,v}$.
  $V_a \leftarrow (R_a + \gamma \sum_{o \in O} V_{a,o})/N$.
$V^* \leftarrow \max_{a \in A} V_a, \quad a^* \leftarrow \arg\max_{a \in A} V_a$.
Create a new policy graph G' by adding a new node u to G. Label u with a^*. For each $o \in O$, add the edge $(u, v_{a^*,o})$ and label it with o.
return G'.

simulation length into account adds another error term that decreases exponentially with L.

Theorem 1. *Let $R_{\max}$ be an upper bound on the magnitude of $R(s, a)$ over $s \in S$ and $a \in A$. Given a policy graph G and a point $b \in \mathcal{B}$,* MC-BACKUP(G, b, N) *produces an improved policy graph such that for any $\tau \in (0, 1)$,*

$$\left|HV_G(b) - \hat{H}_b V_G(b)\right| \leq \frac{2R_{\max}}{1-\gamma} \sqrt{\frac{2\big(|O| \ln |G| + \ln(2|A|) + \ln(1/\tau)\big)}{N}},$$

with probability at least $1 - \tau$.

Proof. There are $|A||G|^{|O|}$ candidates for the improved policy graph. Effectively MC-BACKUP uses N samples to estimate the value at b with respect to each candidate and chooses the best one.

First, we calculate the probability that all the estimates have small errors. Let σ_i be a random variable representing the total reward of the ith simulation under a candidate policy. Define $\sigma = \sum_{i=1}^{N} \sigma_i / N$. Using Hoeffding's inequality, we have $p(|\sigma - \mathrm{E}(\sigma)| \geq \epsilon) \leq 2\mathrm{e}^{-N\epsilon^2/2C^2}$, where $C = R_{\max}/(1-\gamma)$ and ϵ is a small positive constant. Let E_i denote the event that the estimate for the ith candidate policy has error greater than ϵ. Applying the union bound $p(\bigcup_i E_i) \leq \sum_i p(E_i)$, we conclude that the estimate for any of the candidate policy graphs has error greater than ϵ with probability at most $\tau = 2|A||G|^{|O|}\mathrm{e}^{-N\epsilon^2/2C^2}$. So we set $\epsilon = C\sqrt{\frac{2(|O| \ln |G| + \ln(2|A|) + \ln(1/\tau)}{N}}$.

Next, let G^* denote the best candidate policy graph and G^*_{MC} denote the candidate graph chosen by MC-BACKUP. Let σ^* and σ^*_{MC} be the corresponding estimates of

the value at b in MC-BACKUP. Then,

$$\begin{aligned} HV_G(b) - \hat{H}_b V_G(b) &= \mathrm{E}(\sigma^*) - \mathrm{E}(\sigma^*_{\text{MC}}) \\ &= \mathrm{E}(\sigma^*) - \sigma^* + \sigma^* - \mathrm{E}(\sigma^*_{\text{MC}}) \\ &\leq \mathrm{E}(\sigma^*) - \sigma^* + \sigma^*_{\text{MC}} - \mathrm{E}(\sigma^*_{\text{MC}}), \end{aligned}$$

The inequality in the last line follows, as MC-BACKUP always chooses the candidate policy graph with the highest estimate. Thus $\sigma^* \leq \sigma^*_{\text{MC}}$. Finally, the result in the previous paragraph implies that $|\sigma^* - \mathrm{E}(\sigma^*)| \leq \epsilon$ and $|\sigma^*_{\text{MC}} - \mathrm{E}(\sigma^*_{\text{MC}})| \leq \epsilon$, with probability at least $1 - \tau$. Hence, $|HV_G(b) - \hat{H}_b V_G(b)| \leq 2\epsilon$, and the conclusion follows. □

Now we combine MC-backup, which samples the state space S, and point-based POMDP planning, which samples the belief space $\mathcal{B}$. Point-based POMDP algorithms use a set B of points sampled from $\mathcal{B}$ as an approximate representation of $\mathcal{B}$. Let $\delta_B = \sup_{b \in \mathcal{B}} \min_{b' \in B} \|b - b'\|_1$ be the maximum L_1 distance from any point in $\mathcal{B}$ to the closest point in B. We say that B covers $\mathcal{B}$ well if δ_B is small. Suppose that we are given such a set B. In contrast to the standard VI backup operator H, which performs backup at every point in $\mathcal{B}$, the operator $\hat{H}_B$ applies MC-BACKUP(G, b, N) on a policy graph G at every point in B. Each invocation of MC-BACKUP(G, b, N) returns a policy graph with one additional node added to G. We take a union of the policy graphs from all the invocations over $b \in B$ and construct a new policy graph G'. Let V_0 be value function for some initial policy graph and $V_{t+1} = \hat{H}_B V_t$.

The theorem below bounds the approximation error between V_t and the optimal value function V^*.

Theorem 2. *For every $b \in \mathcal{B}$ and every $\tau \in (0, 1)$,*

$$\begin{aligned} |V^*(b) - V_t(b)| \leq \frac{2R_{\max}}{(1-\gamma)^2} \sqrt{\frac{2\big((|O|+1)\ln(|B|t) + \ln(2|A|) + \ln(1/\tau)\big)}{N}} \\ + \frac{2R_{\max}}{(1-\gamma)^2}\delta_B + \frac{2\gamma^t R_{\max}}{(1-\gamma)}, \end{aligned}$$

with probability at least $1 - \tau$.

To keep the proof simple, the bound is not tight. The objective here is to identify the main sources of approximation error and quantify their effects. The bound consists of three terms. The first term depends on how well MC-backup samples S (Algorithm 1, line 5). It decays at the rate $\mathcal{O}(1/\sqrt{N})$. We can reduce this error by taking a suitably large number of samples from S. The second term, which contains δ_B, depends on how well B covers $\mathcal{B}$. We can reduce δ_B by sampling a sufficiently large set B to cover $\mathcal{B}$ well. The last term arises from a finite number t of MC-backup iterations and decays exponentially with t. Note that although MC-backup is performed over points in B, the error bound holds for every $b \in \mathcal{B}$.

To prove the theorem, we need a Lipschitz condition on value functions:

Lemma 1. *Suppose that a POMDP value function V can be represented as or approximated arbitrarily closely by a set of α-functions. For any $b, b' \in \mathcal{B}$, if $\|b - b'\|_1 \leq \delta$, then $|V(b) - V(b')| \leq \frac{R_{\max}}{1-\gamma}\delta$.*

We omit the proof of Lemma 1, as it is similar to an earlier proof [9] for the special case $V = V^*$. We are now ready to prove Theorem 2.

Proof (Theorem 2). Let $\epsilon_t = \max_{b \in B} |V^*(b) - V_t(b)|$ be the maximum error of $V_t(b)$ over $b \in B$. First, we bound the maximum error of $V_t(b)$ over any $b \in \mathcal{B}$ in terms of ϵ_t. For any point $b \in \mathcal{B}$, let b' be the closest point in B to b. Then

$$|V^*(b) - V_t(b)| \leq |V^*(b) - V^*(b')| + |V^*(b') - V_t(b')| + |V_t(b') - V_t(b)|$$

Applying Lemma 1 twice to V^* and V_t, respectively, and using $|V^*(b') - V_t(b')| \leq \epsilon_t$, we get

$$|V^*(b) - V_t(b)| \leq \frac{2R_{\max}}{1-\gamma}\delta_B + \epsilon_t. \tag{8}$$

Next, we bound the error ϵ_t. For any $b' \in B$,

$$\begin{aligned} |V^*(b') - V_t(b')| &\leq |HV^*(b') - \hat{H}_{b'}V_{t-1}(b')| \\ &\leq |HV^*(b') - HV_{t-1}(b')| + |HV_{t-1}(b') - \hat{H}_{b'}V_{t-1}(b')|, \end{aligned} \tag{9}$$

The inequality in the first line holds, because by definition, $V^*(b') = HV^*(b')$, $V^*(b') \geq V_t(b')$, and $V_t(b') \geq \hat{H}_{b'}V_{t-1}(b')$. It is well known that the operator H is a contraction: $\|HV - HV'\|_\infty \leq \gamma\|V - V'\|_\infty$ for any value functions V and V', where $\|\cdot\|_\infty$ denotes the L_∞ norm. The contraction property and (8) imply

$$|HV^*(b') - HV_{t-1}(b')| \leq \gamma\Big(\frac{2R_{\max}}{1-\gamma}\delta_B + \epsilon_{t-1}\Big). \tag{10}$$

Theorem 1 guarantees small approximation error with high probability for a single MC-backup operation. To obtain V_t, we apply $\hat{H}_B$ for t times and thus have $|B|t$ MC-backup operations in total. Suppose that each MC-backup fails to achieve small error with probability at most $\tau/|B|t$. Applying the union bound together with Theorem 1, every backup operation $\hat{H}_{b'}$ achieves

$$|HV_{t-1}(b') - \hat{H}_{b'}V_{t-1}(b')| \leq \frac{2R_{\max}}{1-\gamma}\sqrt{\frac{2\big(|O|\ln(|B|t) + \ln(2|A|) + \ln(|B|t/\tau)\big)}{N}}. \tag{11}$$

with probability at least $1 - \tau$. We then substitute the inequalities (9–11) into the definition of ϵ_t and derive a recurrence relation for ϵ_t. For any initial policy graph, the error ϵ_0 can be bounded by $2R_{\max}/(1-\gamma)$. Solving the recurrence relation for ϵ_t and substituting it into (8) gives us the final result. □

3.3 Algorithm

Theorem 2 suggests that by performing MC-backup over a set B of suitably sampled beliefs, we can approximate the optimal value function with a bounded error. To complete the algorithm, we need to resolve a few remaining issues. First, we

need a method for sampling from the belief space and obtaining B. Next, $\hat{H}_B$ performs backup at every point in B, but for computational efficiency, we want to perform backup only at beliefs that lead to significant improvement in the value function approximation. Both issues occur in discrete POMDP algorithms as well and have been addressed in earlier work. Finally, we use particle filters [22] to represent beliefs over continuous state spaces. Particle filtering can be implemented very efficiently and has been used with great success in important robotic tasks, such as localization and SLAM [22].

We now give a short description of the algorithm. It shares the same basic structure with our SARSOP algorithm [12] for discrete POMDPs; however, it uses MC-backup and particle filtering to handle continuous state spaces.

Overview. The algorithm computes an approximation to an optimal policy by updating a policy graph G. To improve G, it samples beliefs incrementally and performs backup at selected sampled beliefs.

Let $\mathcal{R} \subseteq \mathcal{B}$ be a subset of points reachable from a given initial belief $b_0 \in \mathcal{B}$ under arbitrary sequences of actions and observations. Following the recent point-based POMDP planning approach, our algorithm samples a set of beliefs from this reachable space $\mathcal{R}$ rather than $\mathcal{B}$ for computational efficiency, as $\mathcal{R}$ is often much smaller than $\mathcal{B}$. The sampled beliefs form a tree $T_{\mathcal{R}}$. Each node of $T_{\mathcal{R}}$ represents a sampled belief $b \in \mathcal{R}$, and the root of $T_{\mathcal{R}}$ is the initial belief b_0. If b is a node of $T_{\mathcal{R}}$ and b' is a child of b in $T_{\mathcal{R}}$, then $b' = \tau(b, a, o)$ for some $a \in A$ and $o \in O$. By definition, the belief associated with every node in $T_{\mathcal{R}}$ lies in $\mathcal{R}$.

To sample new beliefs, our algorithm updates $T_{\mathcal{R}}$ by performing a search in $\mathcal{R}$. At each node b of $T_{\mathcal{R}}$, it maintains both upper and lower bounds on $V^*(b)$. We start from the root of $T_{\mathcal{R}}$ and traverse a single path down until reaching a leaf of $T_{\mathcal{R}}$. At a node b along the path, we choose action a that leads to the child node with the highest upper bound and choose observation o that leads to the child node making the largest contribution to the gap between the upper and lower bounds at the root of $T_{\mathcal{R}}$. These heuristics are designed to bias sampling towards regions that likely lead to improvement in value function approximation. If b is a leaf node, then we use the same criteria to choose a belief b' among all beliefs reachable from b with an action $a \in A$ and an observation $o \in O$. We compute $b' = \tau(b, a, o)$ using particle filtering and create a new node for b' in $T_{\mathcal{R}}$ as a child of b. The sampling path terminates when it reaches a sufficient depth to improve the bounds at the root of $T_{\mathcal{R}}$. We then go back up this path to the root and perform backup at each node along the way to update the policy graph as well as to improve the upper and lower bound estimates. We repeat the sampling and backup procedures until the gap between the upper and lower bounds at the root of $T_{\mathcal{R}}$ is smaller than a pre-specified value.

Policy and lower bound backup. The lower bound at a tree node b is computed from the policy graph G. As G always represents a valid policy, $V_G(b)$ is a lower bound of $V^*(b)$. We initialize G with a simple default policy, e.g., always performing a single fixed action. To update the lower bound at b, we perform MC-backup on G at b. As a result, we obtain an updated policy graph G' and an MC estimate of the value at b with respect to G' as an improved lower bound.

Upper bound backup. To obtain the initial upper bound at a node b, one general approach is to apply standard relaxation techniques. Assuming that a robot's state variables are all fully observable, we can solve a corresponding MDP, whose value function provides an upper bound on the POMDP value function. By assuming that a robot's actions are deterministic, we can further relax to a deterministic planning problem. To update the upper bound at b, we use the standard backup operator.

The upper and lower bounds in our algorithm are obtained via sampling and MC simulations, and are thus approximate. The approximation errors decrease with the number of samples and simulations. Since the bounds are only used to guide belief space sampling, the approximation errors do not pose serious difficulties.

For lack of space, our algorithm description is quite brief. Some additional details that improve computational efficiency are available in [12], but they are independent of the use of MC-backup and particle filtering to deal with continuous state spaces.

4 Experiments

We implemented MCVI in C++ and evaluated it on three distinct robot motion planning tasks: navigation, grasping, and exploration. In each test, we used MCVI to compute a policy. We estimated the expected total reward of a policy by running a sufficiently large number of simulations and averaging the total rewards, and used the estimate as a measure of the quality of the computed policy. As MCVI is a randomized algorithm, we repeated each test 10 times and recorded the average results. All the computation was performed on a computer with a 2.66 GHz Intel processor under the Linux operating system.

4.1 Navigation

This 1-D navigation problem first appeared in the work on Perseus [15], an earlier algorithm for continuous POMDPs. A robot travels along a corridor with four doors (Fig. 2*a*). The robot's goal is to enter the third door from the left. The robot has three actions: MOVE-LEFT, MOVE-RIGHT, and ENTER. The robot does not know its exact location, but can gather information from four observations: LEFT-END, RIGHT-END, DOOR, and CORRIDOR, which indicate different locations along the corridor. Both the actions and observations are noisy. The robot receives a positive reward if it enters the correct door, and a negative reward otherwise.

For comparison with Perseus, we use the same model as that in [15]. Perseus requires that all the transition functions, observation functions, and reward functions are modeled as a combination of Gaussians. See Fig. 2*b* for an illustration of the observation function for the observation CORRIDOR. Details of the model are available in [15]. It is important to note that representing the entire model with Gaussians imposes a severe restriction. Doing so for more complex tasks, such as grasping and obstructed rock sample in the following subsections, is impractical.

We ran MCVI with 600 particles for belief representation and $N = 400$ for MC-BACKUP. We also ran Perseus using the original authors' Matlab program, with parameter settings suggested in [15]. There are two versions of Perseus using

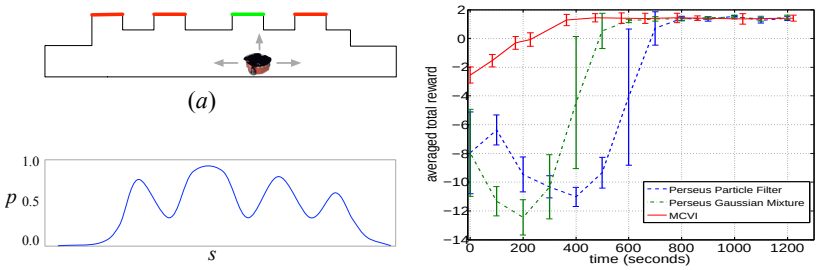


Fig. 2 Navigation in a corridor. (*a*) The environment. (*b*) The observation function for the observation CORRIDOR. (*c*) Estimated expected total rewards of computed policies.

different belief representations. One version uses Gaussian mixture, and the other one uses particle filtering. The results are plotted in Fig. 2*c*. The horizontal axis indicates the time required for policy computation. The vertical axis indicates the average total reward of a computed policy. Each data point is the average over 10 runs of each algorithm. The error bars indicate the 95% confidence intervals.

Since MCVI is implemented in C++ and Perseus is implemented in Matlab, the running times are not directly comparable. However, the plot indicates that MCVI reaches the same performance level as Perseus, even though MCVI does not require a Gaussian model and does not take advantage of it. Also, the smaller error bars for MCVI indicate that it is more robust, especially when the planning time is short.

The main purpose of this test is to compare with Perseus, a well-known earlier algorithm for continuous POMDPs. As one would expect, the task is relatively simple. We can construct a discrete POMDP model for it and compute a policy efficiently using discrete POMDP algorithms.

4.2 Grasping

In this simplified grasping problem [8], a robot hand with two fingers tries to grasp a rectangular block on a table and lift it up (Fig. 3). The fingers have contact sensors at the tip and on each side. Thus, each observation consists of outputs from all the contact sensors. The observations are noisy. Each contact sensor has a 20% probability of failing to detect contact, when there is contact, but 0% probability of mistakenly detecting contact, when there is none. Initially, the robot is positioned randomly above the block. Its movement is restricted to a 2-D vertical plane containing both the hand and the block. The robot's actions include four compliant guarded moves: MOVE-LEFT, MOVE-RIGHT, MOVE-UP, and MOVE-DOWN. Each action moves the robot hand until a contact change is detected. The robot also has OPEN and CLOSE actions to open and close the fingers as well as a LIFT action to lift up the block. If the robot performs LIFT with the block correctly grasped, it is considered a success, and the robot receives a positive reward. Otherwise, the robot receives a negative reward. In this problem, uncertainty comes from the unknown initial position of the robot hand and noisy observations.

We ran MCVI with 150 particles for belief representation and $N = 500$ for MC-BACKUP. On the average, the planning time is 160 seconds, and the computed

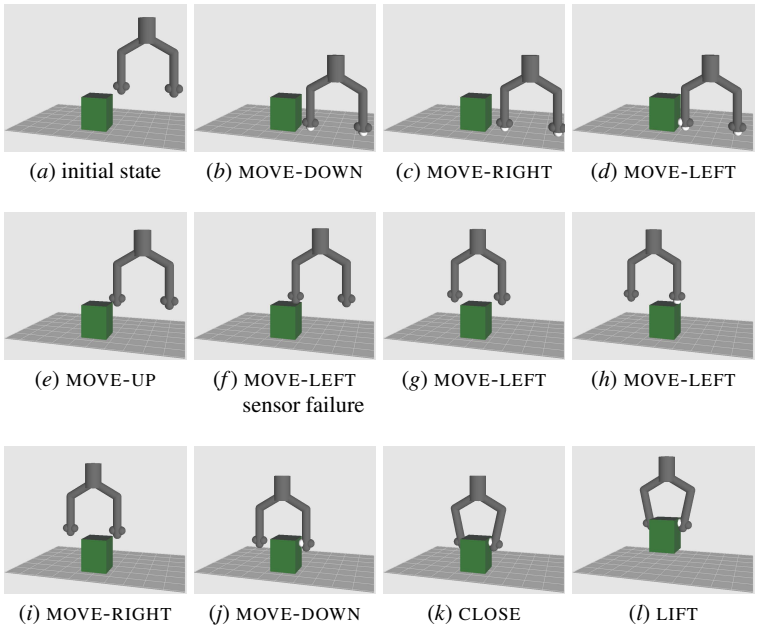


Fig. 3 A simulation run of the simplified grasping task. The spheres at the tip and the sides of the fingers indicate contact sensors. They turn white when contact is detected.

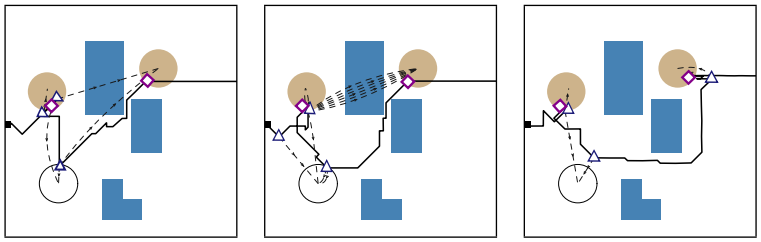
policy has a success rate of 99.7%. For comparison, we manually constructed a open-loop policy: MOVE-LEFT → MOVE-DOWN → MOVE-RIGHT → MOVE-UP → MOVE-RIGHT → MOVE-DOWN → CLOSE → LIFT. The success rate of this policy is only 77.2%. Many of the failures occur because the manually constructed policy does not adequately reason about noisy observations.

Fig. 3 shows a simulation run of the computed policy. In one MOVE-LEFT action (Fig. 3*f*), the tip contact sensor of the left finger fails to detect the top surface of the block. At a result, the robot does not end the MOVE-LEFT action in the proper position, but it recovers from the failure when the tip contact sensor of the right finger correctly detects contact (Fig. 3*h*).

The grasping problem can also be modeled as a discrete POMDP [8]. However, this requires considerable efforts in analyzing the transition, observation, and reward functions. Although the resulting discrete POMDP model is typically more compact than the corresponding continuous POMDP model, the discretization process may be difficult to carry out, especially in complex geometric environments. In contrast, MCVI operates on continuous state spaces directly and is much easier to use.

4.3 Obstructed Rock Sample

The original Rock Sample problem [19] is a benchmark for new discrete POMDP algorithms. In this problem, a planetary rover explores an area and searches for rocks with scientific value. The rover always knows its own position exactly, as well as those of the rocks. However, it does not know which rocks are valuable. It uses the



(*a*) (*b*) (*c*)

Fig. 4 Simulations runs for three ORS models: (*a*) low noise in sensing and movements, (*b*) higher sensor noise, and (*c*) higher movement noise. Shaded polygons indicate obstructed regions. Shaded and white discs indicate the regions in which the rover may perform the SAMPLE action. The rocks are located at the center of the discs. Shaded discs represent valuable rocks, and white discs represent bad rocks. Solid black curves indicates the rover's trajectories. Each "$\diamond$" marks a location where the rover performs a SAMPLE action. Each "$\triangle$" marks a location where the rover performs a SENSE action, and the corresponding dashed line indicates the rock being sensed.

SENSE action to take noisy long-range sensor readings on the rocks. The accuracy of the readings depends on the distance between the rover and the rocks. The rover can also apply the SAMPLE action on a rock in the immediate vicinity and receive a positive or negative reward, depending on whether the sampled rock is actually valuable. The robot's goal is to find as many valuable rocks as possible quickly and then move to the right boundary of the environment to report the results.

We extended Rock Sample in several ways to make it more realistic. We introduced obstructed regions, which the rover cannot travel through. Furthermore, the rover's movement is now noisy. In each time step, the rover can choose to move in any of eight equally spaced directions with two speed settings. Finally, the rover does not always know its own location exactly. It can only localize in the immediate vicinity of a rock, which serves as a landmark. We call this extended version *Obstructed Rock Sample* (ORS).

We created three models of ORS by varying the noise levels for the rover's movements and long-range rock sensor. We ran MCVI on each model. The average planning time ranges from 5 minutes for the low-noise model to a maximum of 2 hours.

Fig. 4 shows a simulation run for each computed policy. For the low-noise model (Fig. 4*a*), the rover first moves towards the top-left rock. It senses the rock and decides to sample it. It also senses the lower-left rock, but cannot determine whether the rock is valuable, because the rock is far away and the sensor reading is too noisy. The rover then approaches the lower-left rock and senses it again. Together the two sensor readings indicate that the rock is likely bad. So the rover does not sample it. Along the way, the rover also senses the top-right rock twice and decides that the rock is likely valuable. As the movement noise is low, the rover chooses to go through the narrow space between two obstacles to reach the rock and sample it. It then takes a shortest path to the right boundary. We do not have a good way of determining how well the computed policy approximates an optimal one. In this simulation run, the jaggedness in the rover's path indicates some amount of suboptimality. However, the rover's overall behavior is reasonable. When the sensor noise in the model is increased (Fig. 4*b*), the rover maintains roughly the same

behavior, but it must perform many more sensing actions to determine whether a rock is valuable. When the movement noise is increased (Fig. 4c), the rover decides that it is too risky to pass through the narrow space between obstacles and takes an alternative safer path.

A standard discrete POMDP model of Rock Sample uses a grid map of the environment. Typically discrete POMDP algorithms can handle a 10×10 grid in reasonable time. This is inadequate for complex geometric environments. The environment shown in Fig. 4, which consists of relatively simple geometry, requires a grid of roughly 50×50, due to closely spaced obstacles. A discrete POMDP model of this size requires about 4 GB of memory before any computation is performed. MCVI avoids this difficulty completely.

4.4 Discussion

While the experimental results are preliminary, the three different examples indicate that MCVI is flexible and relatively easy to use. It does not require artificial discretization of a continuous state space as a grid. It also does not impose restriction on the parametric form of the model.

Our current implementation of MCVI uses fixed values for the number of particles, M, for belief representation and the parameter N in MC-BACKUP. Our experimental results show that MC-BACKUP typically takes around 99% of the total running time and is the dominating factor. To improve efficiency, we may use the sample variance of the simulations to set N adaptively and stop the simulations as soon as the variance becomes sufficiently small. We may over-estimate M, as this does not affect the total running time significantly.

5 Conclusion

POMDPs have been successfully applied to various robot motion planning tasks under uncertainty. However, most existing POMDP algorithms assume a discrete state space, while the natural state space of a robot is often continuous. This paper presents Monte Carlo Value Iteration for continuous-state POMDPs. MCVI samples both a robot's state space and the corresponding belief space, and computes a POMDP policy represented as a finite state controller. The use of Monte Carlo sampling enables MCVI to avoid the difficulty of artificially discretizing a continuous state space and make it much easier to model robot motion planning tasks under uncertainty using POMDPs. Both theoretical and experimental results indicate that MCVI is a promising new approach for robot motion planning under uncertainty.

We are currently exploring several issues to improve MCVI. First, the running time of MCVI is dominated by MC simulations in MC-backup. We may group similar states together and avoid repeated MC simulations from similar states. We may also parallelize the simulations. Parallelization is easy here, because all the simulations are independent. Second, the size of a policy graph in MCVI grows over time. We plan to prune the policy graph to make it more compact [6]. Finally, an

important issue is to deal with not only continuous state spaces, but also continuous observation and action spaces.

Acknowledgments. This work is supported in part by MoE AcRF grant R-252-000-327-112 and MDA GAMBIT grant R-252-000-398-490.

References

1. J.A. Bagnell, S. Kakade, A. Ng, and J. Schneider. Policy search by dynamic programming. In *Advances in Neural Information Processing Systems (NIPS)*, volume 16. 2003.
2. R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, 1957.
3. A. Brooks, A. Makarendo, S. Williams, and H. Durrant-Whyte. Parametric POMDPs for planning in continuous state spaces. *Robotics & Autonomous Systems*, 54(11):887–897, 2006.
4. E. Brunskill, L. Kaelbling, T. Lozano-Perez, and N. Roy. Continuous-state POMDPs with hybrid dynamics. In *Int. Symp. on Artificial Intelligence & Mathematics*, 2008.
5. H. Choset, K.M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L.E. Kavraki, and S. Thrun. *Principles of Robot Motion : Theory, Algorithms, and Implementations*, chapter 7. The MIT Press, 2005.
6. E.A. Hansen. Solving POMDPs by searching in policy space. In *Proc. AAAI Conf. on Artificial Intelligence*, pages 211–219, 1998.
7. R. He, E. Brunskill, and N. Roy. PUMA: Planning under uncertainty with macro-actions. In *Proc. AAAI Conf. on Artificial Intelligence*, 2010.
8. K. Hsiao, L.P. Kaelbling, and T. Lozano-Pérez. Grasping POMDPs. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 4485–4692, 2007.
9. D. Hsu, W.S. Lee, and N. Rong. What makes some POMDP problems easy to approximate? In *Advances in Neural Information Processing Systems (NIPS)*, 2007.
10. D. Hsu, W.S. Lee, and N. Rong. A point-based POMDP planner for target tracking. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 2644–2650, 2008.
11. L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.
12. H. Kurniawati, D. Hsu, and W.S. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proc. Robotics: Science and Systems*, 2008.
13. C. Papadimitriou and J.N. Tsisiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.
14. J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proc. Int. Jnt. Conf. on Artificial Intelligence*, pages 477–484, 2003.
15. J.M. Porta, N. Vlassis, M.T.J. Spaan, and P. Poupart. Point-based value iteration for continuous POMDPs. *J. Machine Learning Research*, 7:2329–2367, 2006.
16. S. Prentice and N. Roy. The belief roadmap: Efficient planning in linear pomdps by factoring the covariance. In *Proc. Int. Symp. on Robotics Research*, 2007.
17. S. Ross, J. Pineau, S. Paquet, and B. Chaib-Draa. Online planning algorithms for POMDPs. *J. Artificial Intelligence Research*, 32(1):663–704, 2008.
18. N. Roy and S. Thrun. Coastal navigation with mobile robots. In *Advances in Neural Information Processing Systems (NIPS)*, volume 12, pages 1043–1049. 1999.
19. T. Smith and R. Simmons. Point-based POMDP algorithms: Improved analysis and implementation. In *Proc. Uncertainty in Artificial Intelligence*, 2005.
20. M.T.J. Spaan and N. Vlassis. A point-based POMDP algorithm for robot planning. In *Proc. IEEE Int. Conf. on Robotics & Automation*, 2004.
21. S. Thrun. Monte carlo POMDPs. In *Advances in Neural Information Processing Systems (NIPS)*. The MIT Press, 2000.
22. S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, 2005.
23. J.F. Traub and A.G. Werschulz. *Complexity and Information*. Cambridge University Press, 1998.