Multilevel Monte-Carlo for Solving POMDPs Online

Marcus Hoerger[1], Hanna Kurniawati[1], and Alberto Elfes[2]

[1] Research School of Computer Science, Australian National University, ACT, Australia,
{marcus.hoerger, hanna.kurniawati}@anu.edu.au
[2] Robotics and Autonomous Systems Group, Data61, CSIRO,
Alberto.Elfes@data61.csiro.au

Abstract. Planning under partial obervability is essential for autonomous robots. A principled way to address such planning problems is the Partially Observable Markov Decision Process (POMDP). Although solving POMDPs is computationally intractable, substantial advancements have been achieved in developing approximate POMDP solvers in the past two decades. However, computing robust solutions for systems with complex dynamics remain challenging. Most on-line solvers rely on a large number of forward-simulations and standard Monte-Carlo methods to compute the expected outcomes of actions the robot can perform. For systems with complex dynamics, e.g., those with non-linear dynamics that admit no closed form solution, even a single forward simulation can be prohibitively expensive. Of course, this issue exacerbates for problems with long planning horizons. This paper aims to alleviate the above difficulty. To this end, we propose a new on-line POMDP solver, called Multilevel POMDP Planner (MLPP), that combines the commonly known Monte-Carlo-Tree-Search with the concept of Multilevel Monte-Carlo to speed-up our capability in generating approximately optimal solutions for POMDPs with complex dynamics. Experiments on four different problems of POMDP-based torque control, navigation and grasping indicate that MLPP substantially outperforms state-of-the-art POMDP solvers.

Keywords: Partially Observable Markov Decision Process, POMDP, Monte-Carlo

1 Introduction

Planning under partial observability is both challenging and essential for autonomous robots. To operate reliably, an autonomous robot must act strategically to accomplish its tasks, despite being subject to various types of uncertainties, such as motion and sensing uncertainty, and uncertainty regarding the environment the robot operates in. Due to these uncertainties, the robot does

not have full observability on the state of the robot and/or its operating environment. The Partially Observable Markov Decision Processes (POMDP)[31] is a mathematically principled way to solve such planning problems.

Although solving a POMDP exactly is computationally intractable[23], the past two decades have seen tremendous progress in developing approximately optimal solvers that trade optimality for computational tractability. Various solvers have been proposed for POMDPs with large state spaces[20,19,21,24,27,29,30,33], large observation spaces[6,13], large or continuous actions spaces[26,35] and long planning horizons[1,10,18], enabling POMDPs to start to become practical for various robotics planning problems[5,14,15].

Most state-of-the-art on-line solvers, such as POMCP[27], DESPOT[30], and ABT[20] rely on a large number of forward simulations of the system and standard Monte-Carlo to estimate the expected values of different sequences of actions. While this strategy has substantially improved state-of-the-art solvers, their performance degrades for problems with complex non-linear dynamics where even a one-step forward simulation requires expensive numerical integrations. Aside from complex dynamics, long planning horizon problems —that is, problems that require more than 10 look-ahead steps before a good solution can be found— remain challenging for on-line solvers. In such problems, even when the computational cost for a one-step forward simulation is cheap, the solver must evaluate long sequences of actions before a good solution is found.

Although complex dynamics and long planning horizons seem like separate issues, both can be alleviated via simplified dynamics. For instance, simplifying the dynamics to reduce the cost of a one-step forward simulation would alleviate the first issue, while simplifying the dynamics, so as to reduce the amount of control inputs switching, could reduce the effective planning horizon. Simplified dynamics models are widely used in deterministic planning and control, albeit less so in solving POMDPs.

In this paper we propose a sampling-based on-line POMDP solver, called Multilevel POMDP Planner (MLPP), that uses multiple levels of approximation to the system's dynamics to reduce the number and complexity of forward simulations needed to compute a near-optimal policy. MLPP combines the commonly used Monte-Carlo-Tree-Search[17] with a relatively recent concept in Monte-Carlo, called Multilevel Monte-Carlo (MLMC)[9,11]. MLMC is a variance reduction technique that uses cheap and coarse approximations of the system to carry out the majority of the simulations and combines them with a small number of accurate but expensive simulations to maintain correctness. By constructing a set of correlated samples from a sequence of approximations of the original system's dynamics, in conjunction with applying Multilevel Monte-Carlo estimation to compute the expected value of sequences of actions, MLPP is able to compute near-optimal policies substantially faster than two of the fastest today's on-line solvers on four challenging robotic planning tasks under uncertainty. Two of these scenarios are articulated robots with POMDP-based torque control, while the other two have a required planning horizon of more

than 10 steps. We also show that under certain conditions, MLPP converges asymptotically to the optimal solution.

2 Background

2.1 Partially Observable Markov Decision Process (POMDP)

Formally a POMDP is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, \gamma >$, where $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{O}$ are the state, action and observation spaces of the robot. T and Z model the uncertainty in the effect of taking actions and perceiving observations as conditional probability functions $T(s, a, s') = p(s'|s, a)$ and $Z(s', a, o) = p(o|s', a)$, where $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$ and $o \in \mathcal{O}$. $R(s, a)$ models the reward the robot receives when performing action a from s and $0 < \gamma < 1$ is a discount factor. Due to uncertainties in the effect of executing actions and perceiving observations, the true state of the robot is only partially observable. Thus, given a *history* $h_t = \{a_0, o_0, ..., a_t, o_t\}$ of previous actions and observations, the robot maintains a *belief* $b(s, h_t)$, a probability distribution over states, conditioned on history h_t, and selects actions according to a *policy* $\pi(h_t)$, a mapping from histories to actions. The value of a policy π is the expected discounted future reward the robot receives when following π given h, i.e. $V_\pi(h) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[r_t|h, \pi]$, where r_t is the immediate reward received at time t. The solution of a POMDP is then an optimal policy π^* such that $\pi^* = \arg\max_\pi V_\pi(h)$.

2.2 Multilevel Monte-Carlo

Since its introduction in 2001, MLMC has been used to significantly reduce the computational effort on applications that involve computing expectations from expensive simulations[8,7,2]. Here, we provide a brief overview of the underlying concept of MLMC. An extensive overview is available at[9].

Suppose we have a random variable X and we wish to compute its expectation $\mathbb{E}[X]$. A simple Monte-Carlo (MC) estimator for $\mathbb{E}[X]$ is $\mathbb{E}[X] \approx \frac{1}{N} \sum_{i=1}^{N} X^{(i)}$, where $X^{(i)}$ are iid. samples drawn from X. In many applications sampling from X directly is expensive, causing the MC-estimator to converge slowly.

The idea of MLMC is to use the linearity of expectation property to reduce the cost of sampling. Suppose, $X_0, X_1, X_2, \ldots, X_L$ is a sequence of approximations to X, where $\lim_{L\to\infty} X_L = X$ and the approximation increases in accuracy and sampling cost as the index increases. Using the linearity of expectation, we have the simple identity:

$$\mathbb{E}[X] = \mathbb{E}[X_0] + \sum_{l=1}^{L} \mathbb{E}[X_l - X_{l-1}] \tag{1}$$

and can design the unbiased estimator:

$$\mathbb{E}[X] \approx \frac{1}{N_0} \sum_{i=1}^{N_0} X_0^{(i)} + \sum_{l=1}^{L} \frac{1}{N_l} \sum_{i=1}^{N_l} (X_l^{(i)} - X_{l-1}^{(i)}) \tag{2}$$

with independent samples at each level. The key here is that even though the samples at each level are independent, the individual samples $X_l^{(i)}$ and $X_{l-1}^{(i)}$ at level l are correlated, such that their differences have a small variance. Of course, the aim is to be able to sample only from the first few approximations while still computing a relatively good approximation of $\mathbb{E}[X]$. It turns out, if we define the sequence of approximations appropriately[9], the variance $\mathbb{V}[X_l - X_{l-1}]$ becomes smaller for increasing level l, and therefore we require fewer and fewer samples to accurately estimate the expected differences. This means we can take the majority of the samples at the coarser levels, where sampling is cheap, and only a few samples are required on the finer levels, thereby leading to a substantial reduction of the cost to estimate $\mathbb{E}[X]$ accurately.

3 Multilevel POMDP Planner (MLPP)

MLPP is an anytime on-line POMDP solver. Starting from the current history h_t, MLPP computes an approximation to the optimal policy by iteratively constructing and evaluating a search tree $\mathcal{T}$, a tree whose nodes are histories and edges represent a pair of action-observation. From hereafter, we use the term *nodes* and the *histories* they represent interchangeably. A history h' is a child node of h via edge (a, o) if $h' = hao$. The root of $\mathcal{T}$ corresponds to an empty history h_0. The policy of MLPP is embedded in $\mathcal{T}$ via $\pi(h) = \arg\max_{a \in \mathcal{A}} \widehat{Q}(h, a)$, where $\widehat{Q}(h, a)$ is an approximation of $Q(h, a) = R(h, a) + \gamma \mathbb{E}_{o \in \mathcal{O}}[V_{\pi^*}(hao)]$, i.e. the expected value of executing a from h and continuing optimally afterwards.

To compute $\widehat{Q}$, MLPP constructs $\mathcal{T}$ using a framework similar to POMCP[27] and ABT[20]: Given the current history h_t, MLPP repeatedly samples *episodes* starting from h_t. An episode e is a sequence of (s, a, o, r)-quadruples, where the state $s \in \mathcal{S}$ of the first quadruple is distributed according to the current belief b_t – we approximate beliefs by sets of particles – and the states of all subsequent quadruples are sampled from the transition function T, given the state and action of the previous quadruple. The observations $o \in \mathcal{O}$ are sampled from the observation function Z, while the reward $r = R(s, a)$ is generated by the simulation process. Each episode corresponds to a path in $\mathcal{T}$. Details on how the episodes are sampled are given in Section 3.1.

Key to MLPP is the adoption of the MLMC concept: Episodes are sampled using multiple levels of approximations of the transition function. Suppose T is the transition function of the POMDP problem. MLPP first defines a sequence of increasingly accurate approximations of the transition function $T_0, T_1, ..., T_L$ with $T_L = T$, and uses the less accurate but cheaper transition functions for the majority of the episode samples, to approximate the Q-value function fast. Note that to ensure asymptotic convergence of MLPP, we slightly modify MLMC such that L is finite and T_L is the most refined level MLPP samples from.

Let $V_k(e)$ be the total discounted reward of an episode starting from the k-th quadruple. For a node h of depth k, MLPP approximates $Q(h,a)$ according to:

$$\begin{aligned}\widehat{Q}(h,a) &= \widehat{Q}_0(h,a) + \sum_{l=1}^{L}(\widehat{Q}_l(h,a) - \widehat{Q}_{l-1}(h,a)) \\ &= \frac{1}{N_0(h,a)} \sum_{i=1}^{N_0(h,a)} V_k(e_0^{(i)}) + \sum_{l=1}^{L} \frac{1}{N_l(h,a)} \sum_{i=1}^{N_l(h,a)} (V_k(e_l^{(i)}) - V_k(e_{l-1}^{(i)})) \quad (3)\end{aligned}$$

where an episode e_l on level l is sampled using T_l, and $N_l(h,a)$ is the number of all episodes on level l that start from h_0, pass through h and execute a from h. Similar to eq.(2), the key here is that even though we draw independent samples on each level, the episode samples for the value differences $V_k(e_l^{(i)}) - V_k(e_{l-1}^{(i)})$ are *correlated*. The question is, *how do we correlate the sampled episodes?*

We adopt the concepts of *determinization*[30] and common random numbers [22], a popular variance reduction technique: To sample states and observations for an episodes on level l, we use a deterministic simulative model, i.e. a function $f_l : \mathcal{S} \times \mathcal{A} \times [0,1] \mapsto \mathcal{S} \times \mathcal{O}$ such that, given a random variable ψ uniformly distributed in $[0,1]$, $(s', o) = f_l(s, a, \psi)$ is distributed according to $T_l(s, a, s')O(s', a, o)$. For an initial state $s_0 \sim b_t$ and a sequence of actions, the states and observations of an episode on level l are then *deterministically* generated from f_l using a sequence $\Psi = (\psi_0, \psi_1, ...)$ of iid. random numbers. Now, to sample a correlated episode on level $l-1$, we use the same initial state s_0, the same sequence of actions and the same random sample Ψ used for the episode on level l, but generate next states and observations from the model f_{l-1} corresponding to T_{l-1}, such that for a given s and a, $(s', o) = f_{l-1}(s, a, \psi)$ is distributed according to $T_{l-1}(s, a, s')O(s', a, o)$. Using the same initial state, action sequence and random sample Ψ results in two closely correlated episodes, reducing the variance of $V_k(e_l^{(i)}) - V_k(e_{l-1}^{(i)})$.

To incorporate the above sampling strategy to the construction of $\mathcal{T}$, MLPP computes the estimator eq.(3) in two subsequent stages: In the first stage, MLPP samples episodes using the coarsest approximation T_0 of the transition function to compute the first term in eq.(3). In the second stage, MLPP samples correlated pairs of episodes to compute the value difference terms in eq.(3). These two stages are detailed in the next two subsections. An overview of MLPP is shown in Algorithm 1, procedure RunMLPP. We start by initializing $\mathcal{T}$, containing the empty history h_0 as the root, and setting the current belief to be the initial belief (line 1). Then, in each planning loop iteration (line 3-7) we first sample an episode using T_0 (line 4), followed by sampling two correlated episodes (line 6). Once the planning time for the current step is over, MLPP executes the action that satisfies $\arg\max_{a \in \mathcal{A}} \widehat{Q}(h_0, a)$. Based on the executed action a and perceived observation o, we update the belief using a SIR particle filter[3] (line 11) and continue planning from the updated history h_0ao. This process repeats until a maximum number of steps is reached, or the system enters a terminal state (we assume that we know when the system enters a terminal state).

Algorithm 1 MLPP

RunMLPP

$\mathcal{T}$ = initializeTree(); $b = b_0$; h = Root of $\mathcal{T}$; terminal = False; $t = 1$
while terminal is False and $t < t_{max}$ **do**
  while planning time not over **do**
    (e, Ψ) = SampleEpisode($\mathcal{T}$, b, h, 0)
    backupEpisode($\mathcal{T}$, e)
    SampleCorrelatedEpisodes($\mathcal{T}$, b, h)
  end while
  a = get best action in $\mathcal{T}$ from h
  terminal = Execute a
  o = get observation
  $b = \tau(b, a, o)$; $h = hao$
  $t = t + 1$
end while

SampleEpisode(Search tree $\mathcal{T}$, Belief b, History node h, level l)

s = sample a state from b
e = init episode; Ψ = init random number sequence; unvisitedAction = False
while unvisitedAction is False and s not terminal **do**
  (a, unvisitedAction) = UCB1(h, l) ▷ For $l = 0$ we select actions from $\mathcal{A}$, for $l > 0$ from $\mathcal{A}'(h)$
  if a is $\varnothing$ **then** break **end if**
  $\psi \sim [0, 1]$
  $(s', o) = f_l(s, a, \psi)$ ▷ Generate (s', o) such that $(s', o) \sim T_l(s, a, s')Z(s', a, o)$
  $r = R(s, a)$; insert (s, a, o, r) to e and ψ to Ψ
  $s = s'$; h = child node of h via edge (a, o). If no such child exists, create one
end while
$r = 0$
if unvisitedAction is True **then** r = calculateHeuristic(s, h) **end if**
insert $(s, -, -, r)$ to e
return (e, Ψ)

SampleCorrelatedEpisodes(Search tree $\mathcal{T}$, Belief b, History node h)

$l \sim 2^{-l}$ ▷ Sample a level l proportional to 2^{-l}
$(e_l, \Psi) = \text{sampleEpisode}(\mathcal{T}, b, h, l)$
e_{l-1} = init episode
$s = e_l[1].s$ ▷ State of the first quadruple of e_l
for $i = 1$ to $|h_l|$ **do**
  $a = e_l[i].a$ ▷ Action of the i-th quadruple of e_l
  $(s', o) = f_{l-1}(s, a, \Psi[i])$ ▷ s' is generated according to T_{l-1}
  $r = R(s, a)$; insert (s, a, o, r) to e_{l-1}
  $s = s'$; h = child node of h via edge (a, o). If no such child exists, create one
  if s' is terminal **then** break **end if**
end for
$r = 0$
if i is $|e_l|$ **then**
  r =calculateHeuristic(s, h)
end if
insert $(s, -, -, r)$ to e_{l-1} and backupRewardDifference($\mathcal{T}$, e_l, e_{l-1})

3.1 Sampling the episodes using T_0

To sample an episode using T_0, starting from the current history h, we first sample a state from the current belief which will then correspond to the state of the first quadruple of the episode (line 1 in Algorithm 1, procedure SampleEpisode). To sample a next state and observation, we first need to select an action from h (line 4). The action-selection strategy is similar to the strategy used in POMCP and ABT. Consider the set of actions $\mathcal{A}'(h) \subseteq \mathcal{A}$ that have already been selected from h. If $\mathcal{A}'(h) = \mathcal{A}$, i.e. all actions have been selected from h at least once, we formulate the problem of which action to select as a Multi-Arm-Bandit problem (MAB)[34]. MABs are a class of reinforcement learning problems where an agent has to select a sequence of actions to maximise the total reward, but the rewards of selecting actions is not known in advance. One of the most successful algorithms to solve MAB problems is Upper Confidence Bounds1 (UCB1)[4]. UCB1 selects an action according to $a = \arg\max_{a \in \mathcal{A}} \left(\widehat{Q}(h,a) + c_0 \sqrt{\frac{\log(N_0(h))}{N_0(h,a)}} \right)$, where $N_0(h)$ is the number of episodes that were sampled using T_0 that pass through h, $N_0(h,a)$ is the number of episodes that were sampled using T_0, pass through h and select action a from h and c_0 is an exploration constant. In case there are actions that haven't been selected from h, we use a rollout strategy that selects one of these actions uniformly at random.

We then sample a random number $\psi \sim [0,1]$ (line 6) and, based on ψ and the selected action, generate a next state and observation (line 7) from the model f_0 using T_0, an immediate reward (line 8) and add the quadruple to the episode. Additionally we set h to the child node that is connected to h via the selected action and sampled observation. If this child node doesn't exist yet, we add it to $\mathcal{T}$ (line 9). Note that selecting a previously unselected action always results in a new node. To get a good estimate of $\widehat{Q}_0(h,a)$ for a newly selected action, MLPP computes a problem dependent heuristic estimate (line 12) in its rollout strategy using the last state of the episode. Computing a heuristic estimate of $\widehat{Q}_0(h,a)$ helps MLPP to quickly focus its search on more promising parts of $\mathcal{T}$.

Once we have sampled the episodes, we backup the expected discounted reward of the episode all the way back to the current history (line 5 in procedure RunMLPP) to update the $\widehat{Q}_0$-values along the selected action sequence.

3.2 Sampling the correlated episodes

Once MLPP has sampled an episode using the coarsest approximation of T, it samples two correlated episodes, via procedure SampleCorrelatedEpisodes in Algorithm 1. For this we first sample a level l proportional to 2^{-l} (line 1), with $l \geq 1$. This is motivated by the idea that as we increase the level, fewer and fewer samples are needed to get a good estimate of the expected value difference. The idea of randomizing the level is motivated by[25]. Based on the sampled level l, we first sample an episode using the finer transition function T_l (line 2). Sampling this episode is similar to the coarsest level, with some notable differences in the action-selection strategy: At each node h, we only consider

actions from the set $\mathcal{A}'(h) \subseteq \mathcal{A}$ that have been selected at least once during sampling of the coarsest episodes. This is because actions that haven't been selected on the coarsest level yet, don't have an estimate for the first component $\widehat{Q}_0(h,a)$ of eq.(3), therefore we wouldn't be able to update the Q-value estimates in a meaningful way. Additionally, for each level, we maintain separate visitation counts $N_l(h)$ and $N_l(h,a)$, which allows us to use UCB1 as the action selection strategy, i.e. $a = \arg\max_{a \in \mathcal{A}'(h)} \left(\widehat{Q}(h,a) + c_l \sqrt{\frac{\log(N_l(h))}{N_l(h,a)}} \right)$. In case we end up in a node where $\mathcal{A}'(h)$ is empty, we stop the sampling of the episode.

To sample a correlated episode on the coarser level $l-1$, we use the model f_{l-1} corresponding to T_{l-1}, but the same initial state (line 4), the same action sequence (line 6) and the same random number sequence (line 7) that was used for the episode on level l. After we have obtained two correlated episodes on level l and $l-1$, we backpropagate the discounted reward difference between the two episodes along the action sequence all the way to the current history (line 16), to update the expected Q-value difference between level l and $l-1$, i.e. $\widehat{Q}_l(h,a) - \widehat{Q}_{l-1}(h,a)$ for each action in the sequence. Note that even though we use the same action sequence for both episodes, the sequence of visited nodes in $\mathcal{T}$ might be different due to different observations, or because the coarse episode terminates earlier than fine episode. If this is the case, we backup both episodes individually until we arrive at an action edge that is the same for both episodes (there is always at least one common action edge, which is the outgoing action of the current history). The actual Q-value estimates $\widehat{Q}(h,a)$ along the common action sequence are then updated according to

$$\widehat{Q}(h,a) = \widehat{Q}_0(h,a) + \sum_{l=1}^{K} w_l(h,a) \left(\widehat{Q}_l(h,a) - \widehat{Q}_{l-1}(h,a) \right) \tag{4}$$

During the early stages of planning, when only a few discounted reward differences have been sampled, the estimator $\widehat{Q}_l(h,a) - \widehat{Q}_{l-1}(h,a)$ might have a large variance, causing it to "overcorrect" the policy. To alleviate this issue, we use a weighting function w_l defined as $w_l(h,a) = \left(1 + \frac{\widehat{\mathbb{V}}[Q_l(h,a) - Q_{l-1}(h,a)]}{N_l}\right)^{-1}$, where $\widehat{\mathbb{V}}[\cdot]$ is an estimate of the variance of the Q-value difference $Q_l(h,a) - Q_{l-1}(h,a)$, obtained from the history samples, and N_l is the number of samples used to estimate $Q_l(h,a) - Q_{l-1}(h,a)$. As the number of samples on level l and $l-1$ increases, $w_l(h,a)$ converges towards 1, hence the limit of eq.(4) is the actual MLMC-estimator of $\widehat{Q}(h,a)$ defined in eq.(3).

4 Convergence of MLPP

We now discuss under which conditions MLPP converges to the optimal policy.

Suppose we have an action sequence $(a_1, a_2, a_3, ..., a_K)$ and an initial state $s_0 \sim b_t$. Applying the action sequence to s_0 results in a *trajectory* $(s_0, a_1, s_1, o_1, a_2, s_2, o_2, ...)$ which is distributed according to $\prod_{i=1}^{K} T(s_{i-1}, a_i, s_i) Z(s_i, a_i, o_i)$.

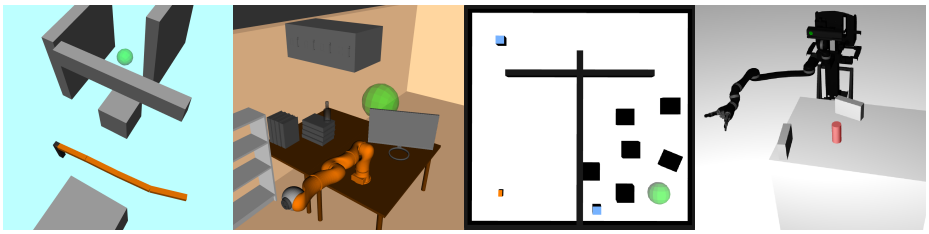
Now suppose we have a sequence of approximations of the transition function $T_0, T_1, ...T_L$ with $T_L = T$.

Assumption 1 *Given a POMDP P, with transition function T and a sequence of approximations of the transition function $T_0, T_1, ...T_L$ with $T_L = T$, then for any action sequence $(a_1, a_2, a_3, ..., a_K)$, $\prod_{i=1}^{K} T(s_{i-1}, a_i, s_i)Z(s_i, a_i, o_i) > 0$ implies $\prod_{i=1}^{K} T_l(s_{i-1}, a_i, s_i)Z(s_i, a_i, o_i) > 0$ for $0 \leq l \leq L$.*

Intuitively, under this assumption, any node in $\mathcal{T}$ than can be reached by episodes that are sampled using the original transition function T can also be reached by episodes that are sampled using T_l. Given this assumption, and the fact that we select actions according to UCB1 on each level independently, the estimator $\widehat{Q}(h, a)$ in 4 converges to $Q(h, a)$ in probability as the number of episodes that pass through h and execute a from h increases on each level. This is based on the analysis in[27,17]. Therefore MLPP's policy converges to the optimal policy in probability, too. Assumption 1 is quite strong and might be too restrictive for some problems. Relaxing this assumption is subject to future work. Nevertheless, problems whose transition and observation functions for all stat–action pairs are represented as distributions with infinite support (e.g., Gaussian) satisfy the assumption above.

5 Experiments and Results

MLPP is tested on two motion-planning problems under uncertainty with expensive non-linear transition dynamics and two problems with long-planning horizon. The scenarios are shown in Figure 1 and described below.



(a) (b) (c) (d)

Fig. 1. Test scenarios used to evaluate MLPP. (a) `4DOF-Factory` (b) `KukaOffice` (c) `CarNavigation` (d) `MovoGrasping`

5.1 Problem scenarios with expensive transition dynamics

4DOF-Factory A torque-controlled manipulator with 4 revolute joints must move from an initial state to a state where the end-effector lies inside a goal region (colored green in Figure 1(a)), without colliding with any of the obstacles. The state space is the joint product of joint-angles and joint-velocities. The control inputs are the joint-torques. To keep the action space small, the action space is set to be the maximum and minimum possible joint torque, resulting

in 16 discrete actions. We assume the input torque at each joint is affected by zero-mean additive Gaussian noise. The dynamics of the manipulators are defined using the well-known Newton-Euler formalism[32]. We assume that each torque input is applied for $\Delta t = 0.1$s. The robot has two sensors: One measures the position of the end-effector inside the robot's workspace, while the other measures the joint velocities. Both measurements are disturbed by zero-mean additive Gaussian noise. The initial state is known exactly, which is when the joint angles and velocities are zero.

The robot enters a terminal state and receives a reward of 1,000 upon reaching the goal. To encourage the robot to reach the goal area quickly, it receives a small penalty of -1 for every other action. Collision causes the robot to enter a terminal state and receive a penalty of -500. The discount factor is 0.98 and the maximum number of planning steps is limited to 50.

KukaOffice The scenario is very similar to the `4DOF-Factory` scenario. However, the robot and environment (illustrated in Figure 1(b)) are different. The robot is a Kuka iiwa with 7 revolute joints. We set the POMDP model to be similar to that of the `4DOF-Factory` scenario, but of course expanded to handle 7DOFs. For instance, the action space remains the maximum and minimum possible joint torque for each joint. However, due to the increase in DOFs, the POMDP model of this scenario has 128 discrete actions. The sensors and errors in both actions and sensing are the same as the `4DOF-Factory` scenario. Similar to the above scenario, we assume each torque input is applied for $\Delta t = 0.1$s. The initial state in this scenario is also similar to the above scenario: The initial joint-velocities are all zero and almost all joint-angles are zero too, except for the second joint, where it is $-1.5rad$.

5.2 Problem scenarios with long planning-horizons

CarNavigation A nonholonomic car-like robot drives on a flat xy-plane inside a 3D environment (shown in Figure 1(c)), populated by obstacles. The robot must drive from a known start state to a position inside the goal region (marked as a green sphere) without colliding with the obstacles. The state of the robot is a 4D-vector consisting of the position of the robot on the xy-plane, its orientation θ around the z-axis, and the forward velocity v. The control input is a $2D$-vector consisting of the acceleration α and the steering-wheel angle ϕ_t. The robots dynamics is subject to control noise $v_t = (\tilde{\alpha}_t, \tilde{\phi}_t) \sim N(0, \Sigma_v)$. The transition model of the robot is defined as $s_{t+1} = [x_t + \Delta t v_t \cos\theta_t;\ \ y_t + \Delta t v_t \sin\theta_t;\ \theta_t + \Delta t \tan(\phi_t + \tilde{\phi}_t)/0.11;\ \ v_t + \Delta t(\alpha_t + \tilde{\alpha}_t)]^T$, where $\Delta t = 0.05s$ is a fixed parameter that represents the duration of a time-step and the value 0.11 is the distance between the front and rear axles of the wheels. The robot is equipped with two sensors: The first one is a localization sensor that receives a signal from one of two beacons located in the environment (blue squares in Figure 1(c)), with probability proportional to the inverse euclidean distance to the beacons. The second sensor is a velocity sensor mounted on the car. With these two sensors the observation model is $o_t = \left[((x_t - \hat{x})^2 + (y_t - \hat{y})^2 + 1)^{-1}, v_t\right]^T + w_t$, where $(\hat{x}, \hat{y})$ is the location of the beacon the robot receives a signal from, v_t is the velocity and w_t is an error vector drawn from a zero-mean multivariate Gaussian

distribution. The robot starts from a state where it is located in the lower-left corner of the map. The robot receives a penalty of -500 when it collides with an obstacle, a reward of 10,000 when reaching the goal area (in both cases it enters a terminal state) and a small penalty of -1 for every step. The discount factor is 0.99 and we allow a maximum of 500 planning steps.

For this problem sampling from the transition function is cheap, thanks to the closed-form transition dynamics. However, the robot must perform a large number of steps (around 200) to reach the goal area from its initial state.

MovoGrasp A 6-DOF Movo manipulator equipped with a gripper must grasp a cylindrical object placed on a table in front of the robot while avoiding collisions with the table and the static obstacles on the table. The environment is shown in Figure 1(d). The state space of the manipulator is defined as $\mathcal{S} = \Theta \times GripperStates \times GraspStates \times \Phi_{obj}$, where $\Theta = (-3.14rad, 3.14rad)^6$ are the joint angles of the arm, $GripperStates = \{gripperOpen, gripperClosed\}$ indicates whether the gripper is open or closed, $GraspStates = \{grasp, noGrasp\}$ indicates whether the robot is grasping the object or not, and $\Phi_{obj} \subseteq \mathbb{R}^6$ is the set of poses of the object in the robot's workspace. The action space is defined as $\mathcal{A} = \mathcal{A}_\theta \times \{openGripper, closeGripper\}$ where $\mathcal{A}_\theta \subseteq \mathbb{R}^6$ is the set of fixed joint angle increments/decrements for each joint, and $openGripper, closeGripper$ are actions to open/close the gripper, resulting in 66 actions. When executing a joint angle increment/decrement action $\hat{\theta}$, the joint angles evolve linearly according to $\theta_{t+1} = \theta_t + \Delta t\hat{\theta} + v_t$, where $\Delta t = 0.25$ and v_t is a multivariate zero-mean Gaussian control error. We assume that the *openGripper* and *closeGripper* are deterministic.

Here the robot has access to two sensors: A joint-encoder that measures the joint angles of the robot and a grasp detector that indicates whether the robot grasps the object or not. For the joint-encoder, we assume that the encoder readings are disturbed by a small additive error drawn from a uniform distribution $[-0.05, 0.05]$. For the grasp detector we assume that we get a correct reading 90% of the time. The robot starts from an initial belief where the gripper is open, the joint angles of the robot are $(0.8, -0.2, 0.8, -0.03, 0.0, 0.7)rad$ and the object is placed on the table such that the x and y positions of the object are uniformly distributed according to $[0.86m \pm 0.01m, 0.2 \pm 0.01m]$. When the robot collides with the environment or the object, it enters a terminal state and receives a penalty of -250. In case the robot closes the gripper but doesn't grasp the object, it receives a penalty of -100. Additionally, when the gripper is closed and a grasp is not established, the robot receives a penalty of -700 if it doesn't execute the *openGripper* action. Each motion also incurs a small penalty of -3. When the robot successfully grasps the object, it receives a reward of 1,750 and enters a terminal state. The discount factor is 0.99 and we allow a maximum of 200 planning steps.

Similarly to the CarNavigation problem, the difficulty for this problem is the large number of steps that are required for the robot to complete its task (around 100). Additionally, the robot must act strategically when approaching the object to ensure a successful grasp.

5.3 Experimental setup

All four test scenarios and the solvers are implemented in C++ within the OPPT framework[12], ensuring that all solvers use the same problem implementation. For ABT we used the implementation provided by the authors[16]. For POMCP we used the implementation provided by https://github.com/AdaCompNUS/despot. Note that all three solvers rely on heuristic estimates of the action values in their rollout strategy. For a fair comparison, we use the same heuristic function for all three solvers, where we use methods from motion-planning, assuming the problem is deterministic.

All simulations were run single-threaded on an Intel Xeon Silver 4110 CPU with 2.1GHz and 128GB of memory. For the `4DOF-Factory` and `KukaOffice` problem, we use the ODE physics engine[28] to simulate the transition dynamics. The levels l used by MLPP in these scenarios are associated with the "discretization" (i.e., δt) used by the numerical integration of ODE. In particular, $\delta t = C_1 \cdot 2^{-C_2 l}$. For the scenarios `CarNavigation` and `MovoGrasp`, since the dynamics of these problems are simple, MLPP associates the levels l to the time-step, i.e., $\Delta t = C_1 \cdot 2^{-C_2 l}$. The exact parameters (i.e., C_1, C_2, and the number of levels L) were determined via systematic preliminary trials. As a result of these trials, we set the parameters used by MLPP for `4DOF-Factory` and `KukaOffice` to be C_1=0.0128, C_2=1, L=7, for `CarNavigation` to be C_1=0.4, C_2=1, L=3, and for `MovoGrasping` to be C_1=1, C_2=0.5, L=4.

The purpose of our experiments are three folds. First is to test whether our particular choice for the multiple levels of approximation of the transition functions results in a reduction of the variance of the Q-value difference terms in eq.(4). This ensures that, as we increase the level, fewer and fewer episode samples are required to accurately estimate the difference terms. To do this, we ran MLPP on each problem scenario for 10 runs with a planning time of 20s per step. Then, at each step, after planning time is over, we use the computed policy π and sample 50,000 additional episodes from the current history h on each level l to compute the variance $\mathbb{V}[Q_l(h,a)]$ and 50,000 correlated episodes on each level l to compute $\mathbb{V}[Q_l(h,a) - Q_{l-1}(h,a)]$, where a is the action performed from h according to $\pi(h)$. Taking the average of these variances over all steps and all simulation runs then gives us an indication how the variance of the Q-value difference terms in eq.(4) behaves as we increase the level of approximation of the transition function.

Second is to compare MLPP with two state-of-the-art POMDP solvers ABT [20] and POMCP[27]. For this purpose, we used a fixed planning-time per step for each solver, where we used 1s for the `4DOF-Factory`, `CarNavigation` and `MovoGrasp` problem, and 5s for the `KukaOffice` problem. For each problem scenario we tested ABT and POMCP using different levels of approximations of T for planning, to see whether using a single approximation of T helps to speed-up computing a good policy, compared to MLPP that uses all levels of approximations of T for planning.

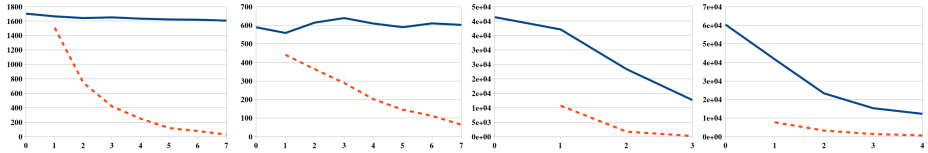
DESPOT[30] is not used as a comparator because for the type of problems we try to address, DESPOT's strategy of expanding each belief with ev-

ery action branch (via forward simulation) is uncompetitive. For example, for `4DOF-Factory`, expanding a single belief takes, on average, ∼14.4s using K=50 scenarios (50 is a tenth of what it commonly used[30]), which is already much more than the time for a single planning step in our experiments (1s). Similarly, for the long planning-horizon problem `MovoGrasp`, DESPOT must expand all 66 actions using K scenarios from every belief it encounters, which quickly becomes infeasible for a planning horizon of more than 5 steps.

Last, we investigated if and how fast MLPP converges to a near-optimal policy compared to ABT and POMCP, when the latter two solvers use the original transition function for planning. To do this, we used multiple increasing planning times per step for the `4DOF-Factory` problem, starting from 1s to 20s per step. The results of all three experiments are discussed in the next section.

5.4 Results

Variances of $Q_l - Q_{l-1}$ Figure 2 shows the average variances of Q_l and $Q_l - Q_{l-1}$ for all four problem scenarios. It is clear that in all scenarios the variance of the Q-value differences decreases significantly as we increase the level l, indicating that we indeed require fewer and fewer episode samples for increasing l. Note that the rate of decrease depends on the particular choice of the sequence of approximate transition functions. Multiple sequences can be possible for a particular problem, but preference should be given to the sequence for which the variance of the Q-value difference decreases fastest.



(a) 4DOF-Factory (b) KukaOffice (c) CarNavigation (d) MovoGrasp

Fig. 2. Average variance of Q_l (solid blue line) and $Q_l - Q_{l-1}$ (dashed red line) for the problem scenarios (a) 4DOF-Factory, (b) KukaOffice, (c) CarNavigation and (d) MovoGrasp. The x-axis represents the level l and the y-axis represents the variance.

Average total discounted rewards Figure 3(a)-(d) shows the average total discounted rewards achieved by ABT, POMCP and MLPP in all four test scenarios. The results indicate that, for ABT and POMCP, using a single coarse approximation of T for planning can help compute a better policy, compared to using the original transition function. However, different regions of the belief space are likely to require different level of approximations. For instance in `4DOF-Factory`, when the states in the support of the belief place the robot in the relatively open area, coarse levels of approximation suffice but, when they are in the cluttered area, higher accuracy is required. Unlike ABT and POMCP, MLPP covers multiple levels of approximations and is able to quickly reduce errors in the estimates of the action values caused by coarse approximations. The lack of coverage causes difficulties for ABT and POMCP in `MovoGrasping` as well, where a high accuracy is neccessary for grasping.

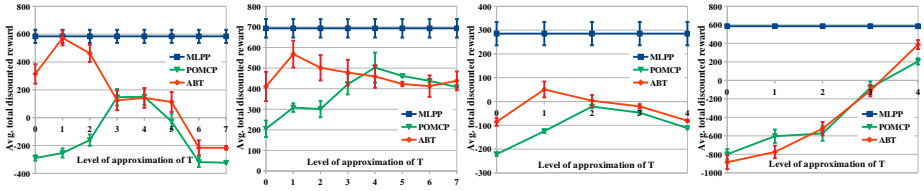


Fig. 3. Average total discounted reward of MLPP, ABT and POMCP on the `4DOF-Factory` (a), `KukaOffice` (b), `CarNavigation` (c) and `MovoGrasp` (d) scenarios. The x-axis represents the level of approximation of the transition function used for planning. Note that MLPP uses all levels for planning (hence the horizontal lines), whereas ABT and POMCP use only a single level as indicated by the x-axis. For each scenario, the largest level of approximation is equal to the original transition function. Vertical bars are the 95% confidence intervals.

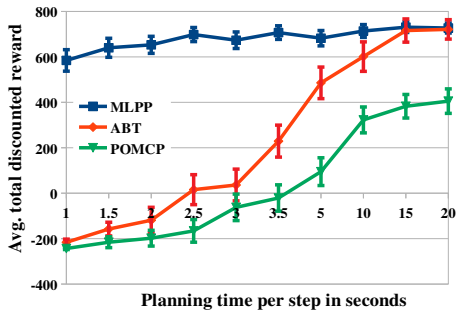


Fig. 4. Average total discounted rewards for ABT, POMCP and MLPP for `4DOF-Factory` using increasing planning times per step. The average is taken over 500 simulation runs for each planning time and algorithm. Vertical bars are the 95% confidence intervals.

Increasing planning times Figure 4 shows the average total discounted rewards achieved by each solver for the `4DOF-Factory` scenario as the planning time per step increases. The results indicate MLPP converges to a good policy much faster than ABT and POMCP: ABT requires 15s/step to generate a policy whose quality is similar to the policy generated by MLPP in 2.5s/step, while POMCP is unable to reach similar level of quality, even with a planning time of 20s/step (in our experiments it takes roughly 5 minutes of planning time/step for POMCP to converge to a near-optimal policy).

6 Conclusion

Despite the rapid progress in on-line POMDP planning, computing robust policies for systems with complex dynamics and long planning-horizons remains challenging. Today's fastest on-line solvers rely on a large number of forward simulations and standard Monte-Carlo methods to estimate the expected outcome of action sequences. While this strategy works well for small to medium-sized problems, their performance quickly deteriorates for problems with transition dynamics that are expensive to evaluate and problems with long planning-horizons.

To alleviate these shortcomings, we propose MLPP, an on-line POMDP solver that extends Multilevel Monte-Carlo to POMDP planning. MLPP samples histories using multiple levels of approximation of the transition function and computes an approximation of the action-values using a Multilevel Monte-Carlo estimator. This enables MLPP to significantly speed-up the planning process

while retaining correctness of the action-value estimates. We have successfully tested MLPP on four robotic tasks that involve expensive transition dynamics and long planning-horizons. In all four tasks, MLPP outperforms ABT and POMCP, two of the fastest on-line solvers, which shows the effectiveness of the proposed method.

References

1. Agha-Mohammadi, A.A., Chakravorty, S., Amato, N.M.: Firm: Feedback controller-based information-state roadmap-a framework for motion planning under uncertainty. In: Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, pp. 4284–4291. IEEE (2011)
2. Anderson, D.F., Higham, D.J.: Multilevel monte carlo for continuous time markov chains, with applications in biochemical kinetics. Multiscale Modeling & Simulation **10**(1), 146–179 (2012)
3. Arulampalam, M.S., Maskell, S., Gordon, N., Clapp, T.: A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. IEEE Transactions on signal processing **50**(2), 174–188 (2002)
4. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning **47**(2-3), 235–256 (2002)
5. Bai, H., Hsu, D.: Unmanned aircraft collision avoidance using continuous-state pomdps. Robotics: Science and Systems VII **1**, 1–8 (2012)
6. Bai, H., Hsu, D., Lee, W.S.: Integrated perception and planning in the continuous space: A pomdp approach. The International Journal of Robotics Research **33**(9), 1288–1302 (2014)
7. Bierig, C., Chernov, A.: Approximation of probability density functions by the multilevel monte carlo maximum entropy method. Journal of Computational Physics **314**, 661–681 (2016)
8. Giles, M.B.: Multilevel monte carlo path simulation. Operations Research **56**(3), 607–617 (2008)
9. Giles, M.B.: Multilevel monte carlo methods. Acta Numerica **24**, 259–328 (2015)
10. He, R., Brunskill, E., Roy, N.: Puma: Planning under uncertainty with macro-actions. In: Proceedings of the National Conference on Artificial Intelligence, vol. 2 (2010)
11. Heinrich, S.: Multilevel monte carlo methods. In: S. Margenov, J. Waśniewski, P. Yalamov (eds.) Large-Scale Scientific Computing, pp. 58–67. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
12. Hoerger, M., Kurniawati, H., Elfes, A.: A software framework for planning under partial observability. In: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1–9. IEEE (2018)
13. Hoey, J., Poupart, P.: Solving pomdps with continuous or large discrete observation spaces. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI'05, pp. 1332–1338. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2005)
14. Horowitz, M., Burdick, J.: Interactive non-prehensile manipulation for grasping via pomdps. In: Robotics and Automation (ICRA), 2013 IEEE International Conference on, pp. 3257–3264. IEEE (2013)
15. Hsiao, K., Kaelbling, L.P., Lozano-Perez, T.: Grasping pomdps. In: Robotics and Automation, 2007 IEEE International Conference on, pp. 4685–4692. IEEE (2007)

16. Klimenko, D., Song, J., Kurniawati, H.: Tapir: a software toolkit for approximating and adapting pomdp solutions online. In: Proceedings of the Australasian Conference on Robotics and Automation (2014)
17. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: European conference on machine learning, pp. 282–293. Springer (2006)
18. Kurniawati, H., Du, Y., Hsu, D., Lee, W.S.: Motion planning under uncertainty for robotic tasks with long time horizons. The International Journal of Robotics Research **30**(3), 308–323 (2011)
19. Kurniawati, H., Hsu, D., Lee, W.S.: Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In: In Proc. Robotics: Science and Systems (2008)
20. Kurniawati, H., Yadav, V.: An online pomdp solver for uncertainty planning in dynamic environment. In: Proc. Int. Symp. on Robotics Research (2013)
21. Luo, Y., Bai, H., Hsu, D., Lee, W.S.: Importance sampling for online planning under uncertainty. The International Journal of Robotics Research p. 0278364918780322
22. Owen, A.B.: Monte Carlo theory, methods and examples (2013)
23. Papadimitriou, C.H., Tsitsiklis, J.N.: The complexity of markov decision processes. Mathematics of operations research **12**(3), 441–450 (1987)
24. Pineau, J., Gordon, G., Thrun, S.: Point-based Value Iteration: An anytime algorithm for POMDPs (2003)
25. Rhee, C.h., Glynn, P.W.: A new approach to unbiased estimation for sde's. In: Proceedings of the Winter Simulation Conference, p. 17. Winter Simulation Conference (2012)
26. Seiler, K.M., Kurniawati, H., Singh, S.P.: An online and approximate solver for pomdps with continuous action space. In: Robotics and Automation (ICRA), 2015 IEEE International Conference on, pp. 2290–2297. IEEE (2015)
27. Silver, D., Veness, J.: Monte-carlo planning in large POMDPs. In: Advances in neural information processing systems, pp. 2164–2172 (2010)
28. Smith, R.: Open dynamics engine. http://www.ode.org/
29. Smith, T., Simmons, R.: Point-based POMDP algorithms: Improved analysis and implementation (2005)
30. Somani, A., Ye, N., Hsu, D., Lee, W.S.: Despot: Online pomdp planning with regularization. In: Advances in neural information processing systems, pp. 1772–1780 (2013)
31. Sondik, E.J.: The optimal control of partially observable markov decision processes. Ph.D. thesis, Stanford, California (1971)
32. Spong, M.W., Hutchinson, S., Vidyasagar, M.: Robot Modeling and Control, vol. 3. Wiley New York (2006)
33. Sunberg, Z.N., Kochenderfer, M.J.: Online algorithms for pomdps with continuous state, action, and observation spaces. In: Twenty-Eighth International Conference on Automated Planning and Scheduling (2018)
34. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press (2012)
35. Wang, E., Kurniawati, H., Kroese, D.P.: An on-line planner for pomdps with large discrete action space: A quantile-based approach. In: ICAPS, pp. 273–277. AAAI Press (2018)