Multilevel Monte Carlo for Solving POMDPs Online

Journal Title
XX(X):1–17

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

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

Abstract

Planning under partial observability 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 remains 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 involving torque control, navigation and grasping indicate that MLPP substantially outperforms state-of-the-art POMDP solvers.

Keywords

Agent-Based Systems, Autonomous Agents, Nonholonomic Motion Planning

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 its state of and/or the state of its operating environment. The Partially Observable Markov Decision Processes (POMDP) Sondik (1971) is a mathematically principled way to solve such planning problems.

Although solving a POMDP exactly is computationally intractable Papadimitriou and Tsitsiklis (1987), the past two decades have seen tremendous progress in developing approximately optimal solvers that trade optimality for computational tractability, enabling POMDPs to start to become practical for various robotic planning problems Bai and Hsu (2012); Horowitz and Burdick (2013); Hsiao et al. (2007); Wandzel et al. (2019); Hoerger et al. (2019b).

Most state-of-the-art on-line solvers, such as POMCP Silver and Veness (2010), DESPOT Somani et al. (2013), and ABT Kurniawati and Yadav (2013) rely on a large number of forward simulations of the system and standard Monte Carlo methods 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 near-optimal policies. MLPP combines the commonly used Monte-Carlo-Tree-Search Kocsis and Szepesvári (2006) with a relatively recent concept in Monte Carlo, called Multilevel Monte Carlo (MLMC) Giles (2015); Heinrich (2001). MLMC is a variance reduction technique that uses cheap and coarse approximations to the system to carry out the majority of the simulations and combines them with a small

[1]School of Computing, College of Engineering & Computer Science, The Australian National University
[2]Rest In Peace. The majority of this work was conducted while the author was with the Robotics and Autonomous Systems Group, Data61, CSIRO
Email: hoergems@gmail.com, hanna.kurniawati@anu.edu.au

number of accurate but expensive simulations to maintain correctness. By constructing a set of correlated samples from a sequence of approximations to 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 torque control, while the other two required planning horizon of more than 10 steps. We also show that under certain conditions, MLPP converges asymptotically to the optimal solution.

This paper extends our previous work Hoerger et al. (2019a) in three ways: First, in Section 3.3 we propose a more efficient strategy to select the levels of approximation to the system's dynamics during planning. This strategy is adaptive to the estimated cost and utility of performing forward simulations on each level of approximation. Second, in Section 4.4.3 we provide new experimental results to understand how the performance of MLPP is affected by dynamics models of varying costs, compared to the two baseline solvers. And third, in Section 5, we present a substantially improved analysis on the asymptotic convergence of MLPP towards 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. At each step, the robot is in some state $s \in \mathcal{S}$, executes an action $a \in \mathcal{A}$, transitions to a next state $s' \in \mathcal{S}$ and perceives an observation $o \in \mathcal{O}$. Additionally, the robot receives a reward according to the reward function $R(s, a)$ when executing action a from state s. However, the effect of executing actions and perceiving observations is uncertain due to errors in control and perception. T and Z model these uncertainties as conditional probability functions $T(s, a, s') = p(s'|s, a)$ and $Z(s', a, o) = p(o|s', a)$.

Due to these uncertainties the current state of the robot and the environment 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, which provides a sufficient statistic that summarizes all previous information necessary to select an optimal action from the current belief.

At each step, the robot selects an action according to a *policy* $\pi(h_t)$, a mapping from histories to actions. The value of a policy π is the expected discounted infinite-horizon future reward the robot receives when following π given history h, i.e. $V_\pi(h) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[r_t|h, \pi]$, where $0 < \gamma < 1$ is a discount factor which ensures that $V_\pi(h)$ is finite and well-defined, while r_t is the immediate reward received at time t. The goal in POMDP-planning is then to compute an optimal policy π^* such that $\pi^* = \arg\max_\pi V_\pi(h)$ with optimal value function $V^*(h) = V_{\pi^*}(h)$. For any POMDP, there is at least one optimal policy with corresponding optimal value function $V^*(h)$.

Note that the above POMDP framework can be extended to consider belief-dependent reward functions Araya et al. (2010); Fischer and Tas (2020), i.e. reward functions of the form $R(b(\cdot, h), a)$. This is helpful for problems where the objective is to explicitly reduce uncertainty regarding the true state, such as pure information gathering tasks Mihaylova et al. (2002). However, in this paper we focus on the "classical" case of state-action dependent reward functions.

2.2 Related POMDP Solvers

While computing optimal policies is computationally intractable for all but the simplest POMDP problems Papadimitriou and Tsitsiklis (1987), in the past two decades we have seen a surge in approximate-optimal POMDP solvers that are tractable for increasingly complex robotic planning problems under partial observability. Key to their success is the use of sampling, which enables them to trade optimality with computational tractability. Existing state-of-the art POMDP solvers can be broadly classified into off-line and on-line solvers.

Off-line solvers such as Kurniawati et al. (2008); Pineau et al. (2003); Smith and Simmons (2005); Kurniawati et al. (2011); Bai et al. (2014) typically construct a sampled representation of the belief space and compute a policy with respect to the sampled beliefs. In contrast, on-line solvers interleave policy computation and policy execution by computing an approximate-optimal policy for the current belief state only. Once a given planning budget is exceeded, the robot executes an action according to the computed policy, receives an observation from the environment, and updates its belief state based on the executed action and perceived observation. This process then repeats from the new belief state.

Many on-line solvers evaluate different sequences of actions by performing forward-search in a lookahead-tree, starting from the current belief, where beliefs are typically represented by sets of particles. For instance POMCP Silver and Veness (2010) and ABT Kurniawati and Yadav (2013) are based on UCT Kocsis and Szepesvári (2006), a Monte-Carlo-Tree-Search (MCTS) method that selects actions during sampling according to Upper Confidence Bounds1 (UCB1) Auer et al. (2002), a popular algorithm for Multi-Arm-Bandit problems Sutton and Barto (2012). DESPOT Somani et al. (2013) and its variants Garg et al. (2019); Luo et al. (2019); Cai et al. (2021) use a combination of Monte-Carlo sampling, heuristic search and branch-and-bound pruning to build a sparse representation of the lookahead-tree. This strategy has enabled the above methods to scale to very large problems, including problems with continuous state spaces.

More recently, MCTS-based methods have been extended to handle problems with large or continuous action and observation spaces. QBASE Wang et al. (2018) can handle problems with up to 10,000 discrete actions by extending the cross-entropy method Rubinstein and Kroese (2013) to MCTS. GPS-ABT Seiler et al. (2015) uses General Pattern Search, a derivative-free optimization method to search for local optima in continuous action spaces. LABECOP Hoerger and Kurniawati (2021) handles

continuous observation spaces by maintaining a set of sampled episodes, i.e. sequences of state-action-observation-reward quadruples to approximate both beliefs and action values. POMCPOW Sunberg and Kochenderfer (2018), BOMCP Mern et al. (2021), VOMCPOW Lim et al. (2020) and IPFT Fischer and Tas (2020) extend POMCP with Double Progressive Widening Couëtoux et al. (2011). These methods incrementally add sampled actions and observations to the lookahead-tree as planning progresses, allowing them to consider both continuous action and observation spaces.

Another line of research aims to alleviate the issue with large action spaces by formulating the belief-space planning problem as a stochastic control problem. For instance, the methods in Van Den Berg et al. (2012, 2011); Sun et al. (2015); Agha-Mohammadi et al. (2011) restrict the transition and observation dynamics to be linear(ized) and the beliefs to be Gaussian to construct Linear-Quadratic-Gaussian (LQG) feedback-controllers Lindquist (1973). While these methods can scale to very large action spaces, the robustness of the resulting policies can suffer for problems with significant non-linearities and large uncertainties, where the Gaussian belief assumption no longer holds Hoerger et al. (2020). A more extensive review of the recent advances in POMDP solving is available in Kurniawati (2022).

This paper focuses on the general POMDP and follows the MCTS-based approach, but we focus on a different issue, compared to the above work. A common bottleneck shared by tree-search-based POMDP solvers is that they typically require large amounts of sampled episodes to accurately estimate the action values. Consequently, for problems with complex non-linear dynamics where sampling even a single trajectory is expensive, their performance quickly deteriorates, particularly when the computational budget is limited. Our method aims to alleviate exactly this issue.

Note that our method is general enough to be applied to many tree-search based POMDP solvers, including those that consider continuous action and observation spaces Sunberg and Kochenderfer (2018); Mern et al. (2021); Lim et al. (2020). However, in this paper we focus on problems with continuous state spaces, but discrete action and observation spaces.

2.3 Multilevel Monte Carlo

Since its introduction in 2001, MLMC has been used to significantly reduce the computational effort in applications that involve computing expectations from expensive simulations Giles (2008); Bierig and Chernov (2016); Anderson and Higham (2012). Here, we provide a brief overview of the underlying concept of MLMC. An extensive overview is available in Giles (2015).

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 approximations increase 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:

$$\mathbb{E}[X] \approx \frac{1}{N_0}\sum_{i=1}^{N_0} X_0^{(0,i)} + \sum_{l=1}^{L}\frac{1}{N_l}\sum_{i=1}^{N_l}(X_l^{(l,i)} - X_{l-1}^{(l,i)}) \tag{2}$$

where the inclusion of the level in the superscript (l, i) indicates that the samples are independent for each of the difference terms on the right-hand side of eq.(2). The key here is that even though the samples at each level are independent, the individual samples $X_l^{(l,i)}$ and $X_{l-1}^{(l,i)}$ for the l-th difference term 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 Giles (2015), 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 represent 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 notation hao is shorthand for appending action a and observation o to the history h). 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 Silver and Veness (2010) and ABT Kurniawati and Yadav (2013): 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: While MLPP uses a single lookahead-tree $\mathcal{T}$ to represent the policy, episodes are sampled using multiple levels of approximations to the transition function. Note therefore, similar to other tree-based POMDP solvers, MLPP uses a single policy and builds a single belief tree, but the computation resources to estimate the Q-value functions is reduced via multilevel approximations of the forward simulations.

Suppose T is the transition function of the POMDP problem. MLPP first defines a sequence of increasingly accurate approximations to 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 a sample of the expected total discounted reward of executing the action a of the k-th quadruple of episode e from a node h of depth k and continuing optimally afterwards. MLPP approximates $Q(h,a)$ according to:

$$\begin{aligned}\widehat{Q}(h,a) &= \widehat{Q}_0^{(0)}(h,a) + \sum_{l=1}^{L}(\widehat{Q}_l^{(l)}(h,a) - \widehat{Q}_{l-1}^{(l)}(h,a)) \\ &= \frac{1}{N_0(h,a)} \sum_{i=1}^{N_0(h,a)} V_k(e_0^{(0,i)}) \\ &\quad + \sum_{l=1}^{L} \frac{1}{N_l(h,a)} \sum_{i=1}^{N_l(h,a)} (V_k(e_l^{(l,i)}) - V_k(e_{l-1}^{(l,i)}))\end{aligned} \tag{3}$$

where an episode e_l on level l is sampled using T_l, and $N_l(h,a)$ is the number of episode samples for the l-th difference term that start from h_0, pass through h and execute a from h. The superscript l in $\widehat{Q}_l^{(l)}(h,a)$ indicates that the estimators for the Q-value difference terms are independent on each level. Details on how the $V_k(h,\cdot)$ values are computed are given in Section 3.1. 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 $V_k(e_l^{(l,i)}) - V_k(e_{l-1}^{(l,i)})$ are *correlated*. The question is, *how do we correlate the sampled episodes?*

We adopt the concepts of *determinization* Somani et al. (2013) and common random numbers Owen (2013), 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 $V_k(e_l^{(l,i)}) - V_k(e_{l-1}^{(l,i)})$.

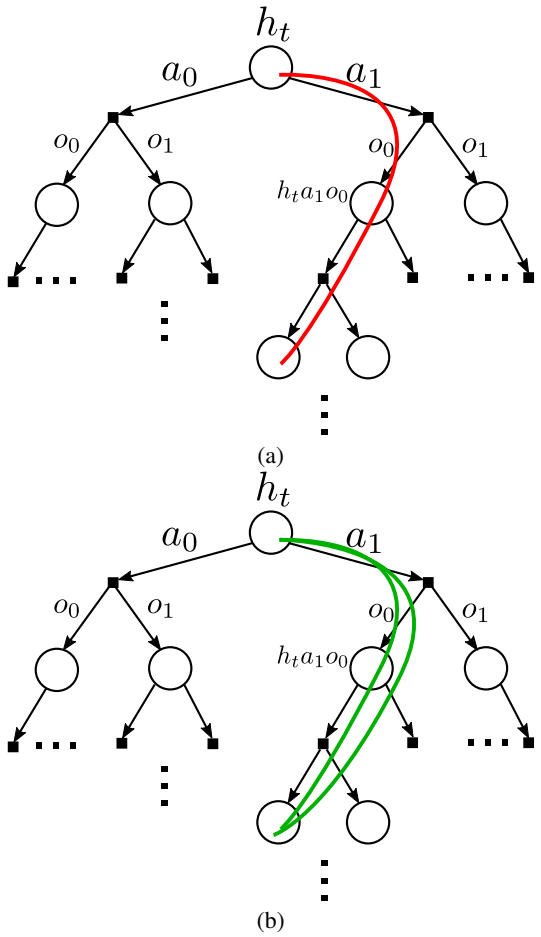


Figure 1. This figure illustrates a partial search tree $\mathcal{T}$ for a POMDP problem with two actions a_1 and a_2 and two observations o_1 and o_2, and the episode sampling process of MLPP. The circles represent histories, whereas the edges represent actions and observations. The root h_t of the tree represents the current history. In the first stage (a), MLPP samples an episode (shown as a red curve) starting from h_t, using the coarsest approximation T_0 to the transition function. In the second stage (b), MLPP samples a pair of correlated episodes (shown as two green curves) on level l and $l-1$. These two stages are repeated until the planning time for the current step is over.

To incorporate the above sampling strategy into 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 to 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 and illustrated in Figure 1. To sample correlated pairs of episodes, MLPP first selects a level l and $l-1$ according to a level selection strategy. In this paper we consider two strategies to select the levels: A default strategy that samples a level l according to

a fixed probability distribution over the levels, and a cost-aware adaptive hybrid sampling (AHS) strategy that adapts the distribution over the levels according to their "utility". Both strategies are detailed in Section 3.3. For the remainder of the paper, we will refer to MLPP that uses the first strategy as **MLPP-Default** and MLPP that uses the second strategy as **MLPP-AHS**.

An overview of MLPP-Default and MLPP-AHS is shown in Algorithm 1 and Algorithm 2 respectively. For both variants of MLPP, 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 1-3 in Algorithm 1). Then, in each planning loop iteration (line 10-14) we first sample an episode using T_0 (line 11), followed by sampling two correlated episodes (line 13). 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 Arulampalam et al. (2002) (line 18) and continue planning from the updated history h_0ao. Note that to get a good approximation of the next belief, the particle filter always uses the original transition function to draw samples from the transition-prior distribution. Since the focus of this paper is to reduce the computational cost of planning, which becomes prohibitively expensive when computing a single forward simulation is costly, we only apply the multi level approximation to planning and not to belief updates during execution.

The process then repeats until a maximum number of steps is reached, or the robot enters a terminal state (we assume that we know when the robot enters a terminal state).

Algorithm 1 MLPP-Default

1: $\mathcal{T}$ = initializeTree()
2: $b = b_0$
3: h = Root of $\mathcal{T}$
4: terminal = False
5: $t = 1$
6: **for** $k = 1$ to L **do** ▷ Initialization for level selection
7: Set probability of level k to $p_k = 2^{-k} / \sum_{j=1}^{L} 2^{-j}$
8: **end for**
9: **while** terminal is False and $t < t_{max}$ **do**
10: **while** planning time not over **do**
11: (e_0, Ψ) = SAMPLEEPISODE($\mathcal{T}$, b, h, $\varnothing$, 0) ▷ Algorithm 3
12: BACKUPEPISODE($\mathcal{T}$, e_0)
13: SAMPLECORRELATEDEPISODES($\mathcal{T}$, b, h, e_0, $[p_1, \cdots, p_L]$) ▷ Algorithm 4
14: **end while**
15: a = get best action in $\mathcal{T}$ from h
16: terminal = Execute a
17: o = get observation
18: $b = \tau(b, a, o)$
19: $h = hao$
20: $t = t + 1$
21: **end while**

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

Algorithm 2 MLPP-AHS

1: $\mathcal{T}$ = initializeTree()
2: $b = b_0$
3: h = Root of $\mathcal{T}$
4: terminal = False
5: $t = 1$
6: **while** terminal is False and $t < t_{max}$ **do**
7: **for** $k = 1$ to L **do** ▷ Initialization for level selection
8: Set weight of level k to $w_k = 1$
9: Set cost of level k to $c_k = 0$
10: Set probability of level k to $p_k = \frac{1}{L}$
11: **end for**
12: **while** planning time not over **do**
13: (e_0, Ψ) = SAMPLEEPISODE($\mathcal{T}$, b, h, $\varnothing$, 0) ▷ Algorithm 3
14: BACKUPEPISODE($\mathcal{T}$, e_0)
15: (l, Δ_v) = SAMPLECORRELATEDEPISODES($\mathcal{T}$, b, h, e_0, $[p_1, \cdots, p_L]$) ▷ Algorithm 4
16: $w_l = w_l \exp(\rho\Delta_v / L\check{p}_l(R_{max} - R_{min}))$ ▷ eq.(9)
17: c_l = updateCostEstimate(l) ▷ eq.(7)
18: **for** $k = 1$ to L **do**
19: $\check{p}_k = (1-\rho)\frac{w_k(t)}{\sum_{j=1}^{L} w_j(t)} + \rho\frac{1}{L}$
20: $p_k = \frac{\check{p}_k / c_k}{\sum_{j=1}^{L} \check{p}_j / c_j}$ ▷ eq.(10)
21: **end for**
22: **end while**
23: a = get best action in $\mathcal{T}$ from h
24: terminal = Execute a
25: o = get observation
26: $b = \tau(b, a, o)$
27: $h = hao$
28: $t = t + 1$
29: **end while**

which will then correspond to the state of the first quadruple of the episode (line 2 in Algorithm 3). To sample a next state and observation, we first need to select an action from h (line 8). 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) Sutton and Barto (2012).

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 are not known in advance. One of the most successful algorithms to solve MAB problems is Upper Confidence Bounds1 (UCB1) Auer et al. (2002). UCB1 selects an action according to $a = \arg\max_{a\in\mathcal{A}} \left(\widehat{Q}(h, a) + C\sqrt{\frac{\log(N_0(h))}{N_0(h,a)}}\right)$, 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 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 16) and, based on ψ and the selected action, generate a next state and observation (line 17) from the model f_0 using

Algorithm 3 SAMPLEEPISODE(Search tree $\mathcal{T}$, Belief b, History node h, Coarse episode e_0, level l)

1: $e =$ init episode
2: $s =$ sample a state from b
3: $\Psi =$ init random number sequence
4: unvisitedAction = False
5: $i = 0$
6: **while** unvisitedAction is False and s not terminal **do**
7: **if** l is 0 **then**
8: (a, unvisitedAction) = UCB1(h)
9: **else**
10: $a = e_0[i].a$
11: **if** a is $\varnothing$ **then**
12: unvisitedAction = True
13: **break**
14: **end if**
15: **end if**
16: $\psi \sim [0, 1]$
17: $(s', o) = f_l(s, a, \psi)$ ▷ Generate (s', o) such that $(s', o) \sim T_l(s, a, s')Z(s', a, o)$
18: $r = R(s, a)$
19: insert (s, a, o, r) to e and ψ to Ψ
20: $s = s'$
21: $h =$ child node of h via edge (a, o). If no such child exists, create one
22: **end while**
23: $r = 0$
24: **if** unvisitedAction is True **then**
25: $r =$ calculateHeuristic(s, h)
26: **end if**
27: insert $(s, -, -, r)$ to e
28: **return** (e, Ψ)

Algorithm 4 SAMPLECORRELATEDEPISODES(Search tree $\mathcal{T}$, Belief b, History node h, Coarse episode e_0, $[p_1, \cdots, p_L]$)

1: $l =$ Sample l proportional to p_l
2: $(e_l, \Psi) =$ SAMPLEEPISODE($\mathcal{T}, b, h, e_0, l$)
3: $e_{l-1} =$ init episode
4: $s = e_l[1].s$ ▷ State of the first quadruple of e_l
5: **for** $i = 1$ to $|h_l|$ **do**
6: $a = e_l[i].a$ ▷ Action of the i-th quadruple of e_l
7: $(s', o) = f_{l-1}(s, a, \Psi[i])$ ▷ s' is generated according to T_{l-1}
8: $r = R(s, a)$
9: insert (s, a, o, r) to e_{l-1}
10: $s = s'$
11: $h =$ child node of h via edge (a, o). If no such child exists, create one
12: **if** s' is terminal **then**
13: break
14: **end if**
15: **end for**
16: $r = 0$
17: **if** i is $|e_l|$ **then**
18: $r =$ calculateHeuristic(s, h)
19: **end if**
20: insert $(s, -, -, r)$ to e_{l-1}
21: $\Delta_v =$ backupRewardDifference($\mathcal{T}$, e_l, e_{l-1}) ▷ Δ_v is the absolute change in the estimated value function of the current history (eq.(8))
22: **return** (l, Δ_v)

T_0, an immediate reward (line 18) 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 21). Note that selecting a previously unselected action always results in a new node. To get a good estimate of $\widehat{Q}_0^{(0)}(h, a)$ for a newly selected action, MLPP computes a problem dependent heuristic estimate (line 25) in its rollout strategy using the last state of the episode. Computing a heuristic estimate for $\widehat{Q}_0^{(0)}(h, a)$ helps MLPP to quickly focus its search on more promising parts of $\mathcal{T}$.

Once we have sampled the episode, we backup the expected discounted reward of the episode all the way back to the current history (line 12 in Algorithm 1) to update the $\widehat{Q}$-values along the selected action sequence. Unlike POMCP Silver and Veness (2010), both versions of MLPP use stochastic Bellman backups, rather than Monte Carlo backups. In particular, suppose h is the history node the i-th episode $e_0^{(0,i)}$ reached after k steps, a_k the action of the k-th quadruple of the episode (i.e. the action that was executed from h) and o_k the observation that was perceived after executing a_k from h. To update the first term on the right-hand side of eq.(3), we compute $V_k(e_0)$ according to

$$V_k(e_0^{(0,i)}) = r_k + \gamma \max_{a' \in \mathcal{A}} \widehat{Q}_0^{(0)}(h', a') \tag{4}$$

where r_k is the immediate reward of the k-th quadruple of the episode, $h' = ha_k o_k$ is the child node of h reached by the episode and $\widehat{Q}_0^{(0)}(h', a')$ is the (updated) estimate of $Q(h', a')$ on the coarsest level.

3.2 Sampling the correlated episodes

Once MLPP has sampled an episode using the coarsest approximation to T, it samples two correlated episodes (line 13 in Algorithm 1), via Algorithm 4. To do this, we first have to select the level l (and corresponding coarse level $l-1$) for which the correlated episodes are sampled on (line 1 in Algorithm 4). The level selection strategies used by MLPP-Default and MLPP-AHS are discussed in detail in the next subsection.

Based on the selected level l, we first sample an episode e_l 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: Since the episode we sampled previously on the coarsest level 0 results in an update of the estimators $\widehat{Q}_0^{(0)}$ along the selected sequence of actions, our aim is to update the estimators $\widehat{Q}$ along this action sequence. Hence, we use the same action sequence for the episode on level l that was used for the episode on level 0 (line 10 in Algorithm 3). Note however that sampling an initial state from the current belief and generating subsequent states and observations from the model f_l is independent of the coarsest episode. Furthermore, note that even though we use the same action sequence that was selected for the episode on the coarsest level, the sequence of histories visited by e_l may be different due to different observation sequences.

As a consequence, we might end up in a history node that hasn't been visited on the coarsest level before. If this is the case, we stop the sampling process for e_l (line 11-14 in Algorithm 3).

After we have sampled an episode e_l on level l, we sample a correlated episode e_{l-1} on level $l-1$ (line 4-20 in Algorithm 4). To do this, we use the model f_{l-1} corresponding to T_{l-1} to generate states and observations, but use 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 e_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 21), to update the estimated Q-value difference between level l and $l-1$, i.e. $\widehat{Q}_l^{(l)}(h,a) - \widehat{Q}_{l-1}^{(l)}(h,a)$ for each action in the sequence. Similarly to eq.(4), we use stochastic Bellman backups to update the Q-value estimates on level l and $l-1$. Note that, similarly as above, the sequence of history nodes visited by e_l and e_{l-1} may be different due to different sequences of observations. Additionally, e_{l-1} might terminate earlier than e_l. If this is the case, we only update the Q-value difference estimates and visitation counts along the sequence of action-edges that is common for both episodes (there is always at least one common action edge, which is the outgoing action of the current history). Furthermore, note that since we only update the Q-value estimates and the statistics along the sequence of common action-edges, we always use the same number of episodes on level l and $l-1$ to update the l-th Q-value difference term on the right-hand side of eq.(3).

The actual Q-value estimates $\widehat{Q}(h,a)$ along the sequence of common action-edges are then updated according to

$$\widehat{Q}(h,a) = \widehat{Q}_0^{(0)}(h,a) + \sum_{l=1}^{L} \omega_l(h,a) \left(\widehat{Q}_l^{(l)}(h,a) - \widehat{Q}_{l-1}^{(l)}(h,a) \right) \quad (5)$$

where $\omega_l(h,a)$ is a weighting function that will be discussed below.

During the early stages of planning, when only a few discounted reward differences have been sampled, the estimators $\widehat{Q}_l^{(l)}(h,a) - \widehat{Q}_{l-1}^{(l)}(h,a)$ might have a large variance, causing it to "overcorrect" the policy. This is particularly relevant for problems with discontinuous reward functions, where small variations between two episodes can lead to large differences regarding their respective total discounted rewards. For instance, in motion planning problems, a small change in a collision-free state trajectory of an episode might result in a trajectory that collides with an obstacle, thereby increasing the variance of the $\widehat{Q}_l^{(l)}(h,a) - \widehat{Q}_{l-1}^{(l)}(h,a)$ estimators. The problem of such discontinuous output functions and their negative effect on the variance of MLMC estimators was already studied in Giles (2015), where the authors proposed several strategies to address this problem, that typically rely on some form of smoothing of the output function.

In this paper, we use a different approach: We use a weighting function ω_l defined as

$$\omega_l(h,a) = \left(1 + \frac{\widehat{\mathbb{V}}\left[Q_l(h,a) - Q_{l-1}(h,a) \right]}{N_l} \right)^{-1} \quad (6)$$

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 episode samples, and N_l is the number of samples used to estimate $Q_l(h,a) - Q_{l-1}(h,a)$. Intuitively, eq.(6) prevents the estimator $\widehat{Q}_l^{(l)}(h,a) - \widehat{Q}_{l-1}^{(l)}(h,a)$ from affecting $\widehat{Q}(h,a)$ too much in case its sample variance is initially large. However, as the number of episode samples on level l and $l-1$ increases, $\omega_l(h,a)$ converges towards 1, hence the limit of eq.(5) is the actual MLMC-estimator of $\widehat{Q}(h,a)$ defined in eq.(3).

3.3 Selecting a Level for the Correlated Episodes

To select a level l for the correlated episodes, we consider two level-selection strategies in this paper. The first strategy used by MLPP-Default samples a level proportional to 2^{-l}, with $l \geq 1$. This strategy 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 Rhee and Glynn (2012). This strategy of selecting the levels is simple and captures the fact that we need fewer and fewer samples as the level increases to get a good estimate of the Q-value differences in eq.(3).

However, the above strategy does not consider the cost of sampling episodes on different levels, nor does it consider the contribution that each Q-value difference term has to the value function $V(h)$ of the current history h. For many problems, the difference $\widehat{Q}_l^{(l)}(h,a) - \widehat{Q}_{l-1}^{(l)}(h,a)$ has a negligible contribution to $V(h)$ as we increase l, but sampling episodes becomes significantly more expensive. Furthermore, the benefit and cost of sampling on different levels depend on the estimated value function so far, which depends on the current history.

Therefore, to balance the benefit and cost, while being adaptive to the current history, we propose a level sampling strategy used by MLPP-AHS that is inspired by the Adaptive Hybrid Sampling (AHS) strategy proposed in Hsu et al. (2005) which is a cost-aware Multi Arm Bandit method for selecting samplers in the context of PRM-planning. This strategy allows our level sampling strategy to be adaptive with respect to both the cost of sampling episodes at level l, as well as to the contribution the Q-value difference terms on level l have to $V(h)$.

Given the current history h, we maintain a weight w_l for each level $l \geq 1$ that reflects the past utility of a level l in terms of its contribution to $V(h)$ and its cost of sampling episodes on level l. At the start of each planning step, we set $w_l = 1$ for each $1 \leq l \leq L$ (line 8 in Algorithm 2).

Suppose c_l is an estimate of the computational cost of sampling two correlated episodes on level l and $l-1$. This cost can vary greatly depending on the current history h from where the episodes are sampled from as well as on the length of the sampled episodes. For instance, in motion

planning problems where a robot has to reach a goal area in the environment, episodes tend to become shorter as the robot gets closer to the goal and are consequently cheaper to sample. Therefore it is, in general, difficult to obtain a good estimate of the computational cost of sampling episodes on level l and $l-1$ a-priori. Thus, we resort to estimating c_l on-line for the current history as the average cost (in terms of CPU time) of sampling two correlated episodes on level l and $l-1$ via

$$c_l = \frac{1}{N_l}\sum_{i=1}^{N_l} c(e_l^{(i)}) + c(e_{l-1}^{(i)}) \tag{7}$$

where $c(\cdot)$ is the cost of a sampled episode, starting from the current history. The cost terms c_l are hereby re-initialized to 0 at every planning step (line 9 in Algorithm 2), i.e. after the robot has executed and action and perceived an observation, which enables us to adapt them to the current history.

Now, suppose at planning loop iteration t level l was chosen and two correlated episodes were sampled on level l and $l-1$. This results in a change of the estimated value function $\widehat{V}(h)$ for the current history h. Let $\widehat{V}_t(h)$ and $\widehat{V}_{t-1}(h)$ be the estimated value functions of h at the current and previous planning loop iterations. The absolute change in the estimated value function, i.e.

$$\Delta_V = \left|\widehat{V}_t(h) - \widehat{V}_{t-1}(h)\right| \tag{8}$$

provides us with a means to estimate the effect the levels l and $l-1$ have on $V(h)$. Using Δ_V we then update the weight of the selected level l according to

$$w_l(t+1) = w_l(t)\exp(\rho\Delta_V/(L\check{p}_l(R_{max}-R_{min}))) \tag{9}$$

where $w_l(t)$ is the current weight of level l, $R_{max} = \max_{s\in\mathcal{S},a\in\mathcal{A}} R(s,a)$ and $R_{min} = \min_{s\in\mathcal{S},a\in\mathcal{A}} R(s,a)$ are the maximum and minimum immediate rewards respectively, $\rho \in (0,1)$ is an exploration constant and $\check{p}_l = (1-\rho)\frac{w_l(t)}{\sum_{j=1}^{L} w_j(t)} + \rho\frac{1}{L}$ is a cost-independent probability of selecting level l. In other words, the current weight $w_l(t)$ of the chosen level l is scaled exponentially with Δ_V. The term $(R_{max}-R_{min})$ in the denominator on the right-hand side of eq.(9) ensures that the weights are upper-bounded by 1. The above method is motivated by Kurniawati and Yadav (2013), where a similar strategy was used to estimate how different rollout heuristics improve the value function at the root node.

To take the cost of sampling two correlated episodes on level l and $l-1$ into account, we update the probability of selecting level l (initially, each level l has equal probability of being selected) according to

$$p_l = \frac{\check{p}_l/c_l}{\sum_{j=1}^{L}\check{p}_j/c_j} \tag{10}$$

This level sampling strategy enables MLPP to quickly focus on levels that have a large effect on the value function $V(h)$ of the current history h, while simultaneously taking the episode sampling cost into account. As we will see in Section 4, this has a sizeable effect on the overall performance of MLPP.

4 Experiments and Results

To evaluate MLPP-Default and MLPP-AHS, we tested them on two motion-planning problems under uncertainty with expensive non-linear transition dynamics and two problems with long-planning horizons. The scenarios are shown in Figure 2 and described below.

4.1 Problem scenarios with expensive transition dynamics

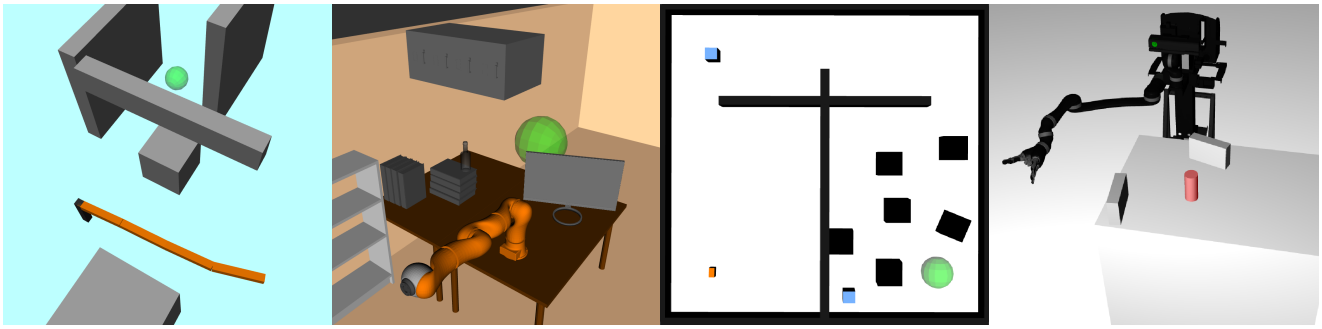
4.1.1 4DOF-Factory A torque-controlled manipulator with 4 revolute joints must move from an initial state to a state where the end-effector lies inside a goal region (colored green in Figure 2(a)), without colliding with any of the obstacles. The state space is the joint product of joint-angles and joint-velocities. 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 Spong et al. (2006). 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.

4.1.2 KukaOffice The scenario is very similar to the `4DOF-Factory` scenario. However, the robot and environment (illustrated in Figure 2(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, for which the initial joint angle is $-1.5rad$.

4.2 Problem scenarios with long planning-horizons

4.2.1 CarNavigation A nonholonomic car-like robot drives on a flat xy-plane inside a 3D environment (shown in Figure 2(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

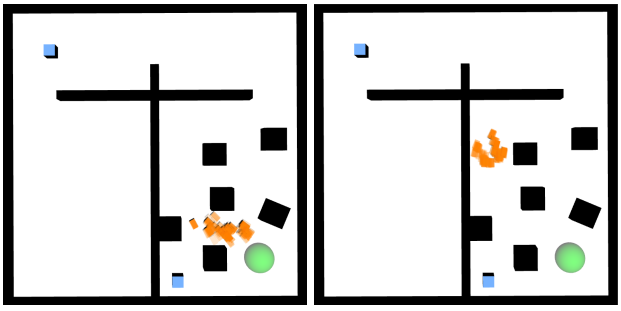


(a) (b) (c) (d)

Figure 2. The test scenarios used throughout the experiments. (a) `4DOF-Factory` (b) `KukaOffice` (c) `CarNavigation` (d) `MovoGrasping`

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 robot's 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 2(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. Additionally, due to the non-linear transition and observation dynamics and the obstacles in the environment, the beliefs are often non-Gaussian or even multimodal, as shown in Figure 3(a) and (b).



(a) (b)

Figure 3. This Figure shows two beliefs (represented as orange particles, where each orange rectangle represents a particle) in the `CarNavigation` scenario. Due to the non-linear transition and observation dynamics and the obstacles in the environment, the beliefs can become non-Gaussian and multimodal.

4.2.2 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 2(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 $closeGripper$ actions 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.

4.3 Experimental setup

For our experiments we compare MLPP-Default and MLPP-AHS with two state-of-the-art on-line POMDP solvers ABT Kurniawati and Yadav (2013) and POMCP Silver and Veness (2010). DESPOT Somani et al. (2013) is not used as a comparator because for the type of problems we try to address, DESPOT's strategy of expanding each belief with every action branch (via forward simulation) is uncompetitive. For example, for the `4DOF-Factory` problem, expanding a single belief takes, on average, $\sim$14.4s using $K = 50$ scenarios (50 is a tenth of what it commonly used Somani et al. (2013)), 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. In our tests we found that, given $K = 50$ scenarios and 1s of planning time/step, DESPOT was unable to expand the belief tree beyond 2 steps in the `MovoGrasp` problem, whereas for the same planning time/step, MLPP typically has a lookahead of around 5-7 steps in this problem. In the `CarNavigation` scenario, for 1s of planning time/step, MLPP achieved a lookahead of around 12-15 steps, whereas the lookahead of DESPOT was limited to a maximum of 4 steps, using $K = 50$ scenarios and 1s of planning time/step. As a result, DESPOT was unable to solve the problem, since its lookahead was insufficient to discover a strategy that drives the car from the lower-left part of the map to the passage near the upper-left part of the map.

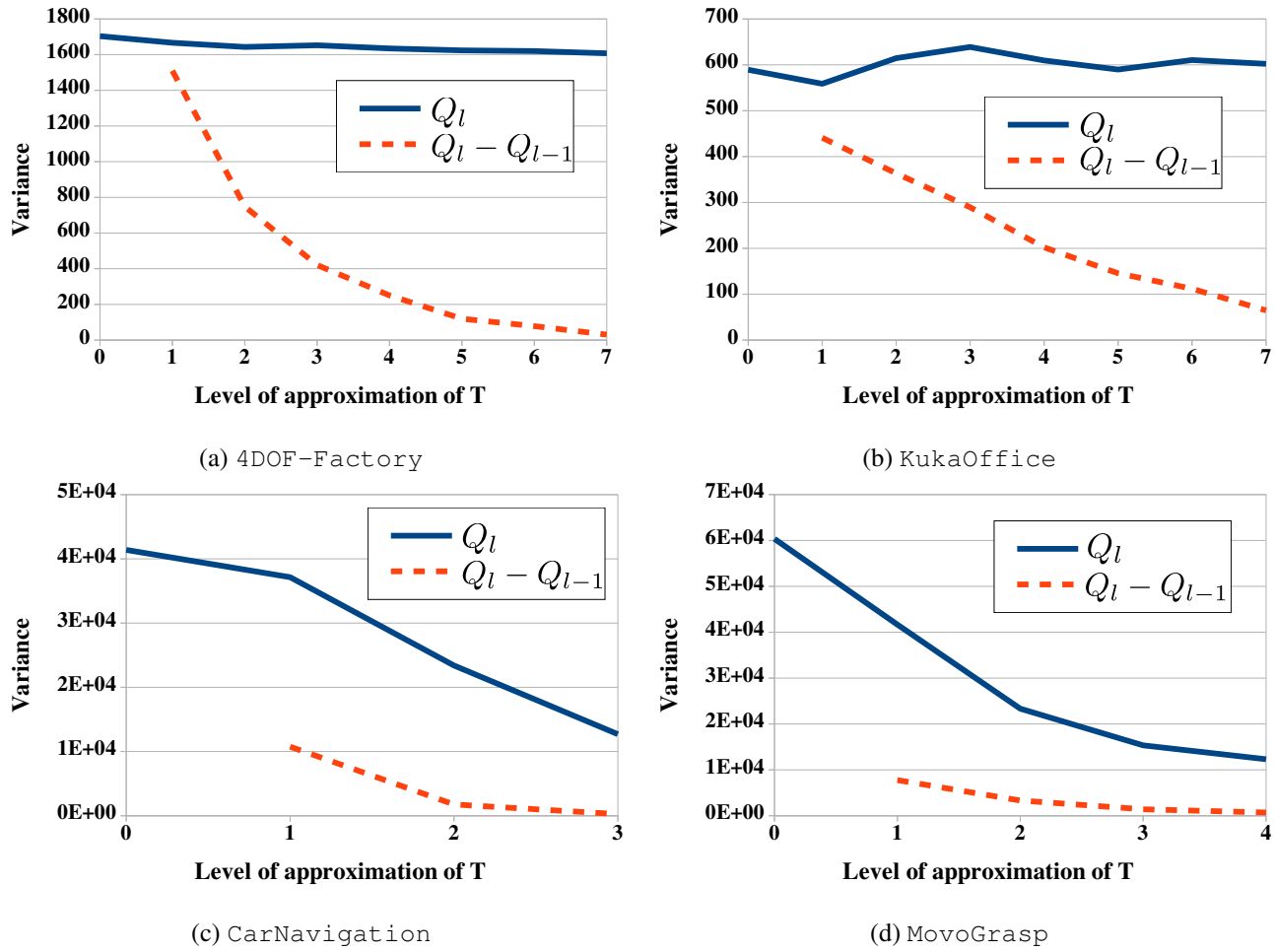
Note that all tested solvers rely on heuristic estimates of the action values in their rollout strategy. For a fair comparison, we use the same heuristic function for all solvers, where we use methods from motion-planning, assuming the problem is deterministic. Additionally, all solvers tested here use a set of particles to represent the current belief that is updated using a SIR particle filter Arulampalam et al. (2002) after the robot has executed an action and perceived an observation. This particle filter always uses the original transition functions to draw samples from the transition-prior distribution for each solver and problem scenario.

For the `4DOF-Factory` and `KukaOffice` problems, we use the ODE physics engine Smith (2001) to simulate the transition dynamics. The levels l used by MLPP-Default and MLPP-AHS in these scenarios are associated with the "discretization" (i.e., δt in seconds) used by the numerical integrator of ODE. In particular, for level l, we set $\delta t(l) = C_1 \cdot 2^{-C_2 l}$. For the `CarNavigation` and `MovoGrasp` problems, since the dynamics of these problems are simple, MLPP associates the levels l to the time-step, i.e., $\Delta t(l) = C_1 \cdot 2^{-C_2 l}$. 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. Note that for each problem, we assume that the transition function T_L corresponding to the most refined level L represents the actual transition function T of the problem. For the exploration factor ρ used by the adaptive level sampling strategy of MLPP-AHS, we set $\rho = 0.1$ throughout the experiments.

The purpose of our experiments is four-fold: First is to test whether our particular choice for the multiple levels of approximation to the transition functions results in a reduction of the variance of the Q-value difference terms in eq.(5). 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-Default 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 the variance $\mathbb{V}[Q_l(h, a) - Q_{l-1}(h, a)]$, 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 on how the variances of the Q-value difference terms in eq.(5) behave as we increase the level of approximation to the transition function.

Second is to compare MLPP-Default and MLPP-AHS with ABT and POMCP. 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` problems, and 5s for the `KukaOffice` problem. For each problem we tested ABT and POMCP using different levels of approximations to T for planning, to see whether using a single approximation to T helps to speed-up computing a good policy, compared to MLPP-Default and MLPP-AHS which use all levels of approximations of T for planning.

Third is to understand how transition dynamics with varying costs affect the performance of MLPP-Default and MLPP-AHS compared to ABT and POMCP. For this experiment we tested all solvers on the `4DOF-Factory` and `KukaOffice` problems where we modify the transition dynamics of the problems to become progressively cheaper. To do this, we set the integration step-size δt of the numerical integrator of ODE to increasingly large values, where we use $\delta t_1 = 0.0004s$, $\delta t_2 = 0.0016s$ and $\delta t_3 = 0.0064s$. In our experiments we found that a step size larger than $\delta t = 0.0128s$ causes the numerical solver of ODE to become unstable, which resulted in chaotic behaviour of the robots in both the `4DOF-Factory` and `KukaOffice` scenarios. For this reason, we cap the largest step size to $\delta t_3 = 0.0064s$



(a) `4DOF-Factory` (b) `KukaOffice`

(c) `CarNavigation` (d) `MovoGrasp`

Figure 4. 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 (larger levels correspond to finer approximations to the transition function) and the y-axis represents the variance.

in this experiment. Similarly to above, we associate the levels used by MLPP with the integration step size of ODE and set, for level l, $\delta t(l) = C_1 \cdot 2^{-C_2 l}$, where we set $C_2 = 5/L$ for δt_1, $C_2 = 3/L$ for δt_2 and $C_2 = 1/L$ for δt_3 with $C_1 = 0.0128$ and $L = 7$ as above. Similarly to the second set of experiments, we use a fixed planning time per step of 1s for the `4DOF-Factory` problem and 5s for the `KukaOffice` problem for all solvers.

Last, we investigated if and how fast MLPP-Default and MLPP-AHS converge 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 in the `4DOF-Factory` problem, starting from 1s to 20s per step.

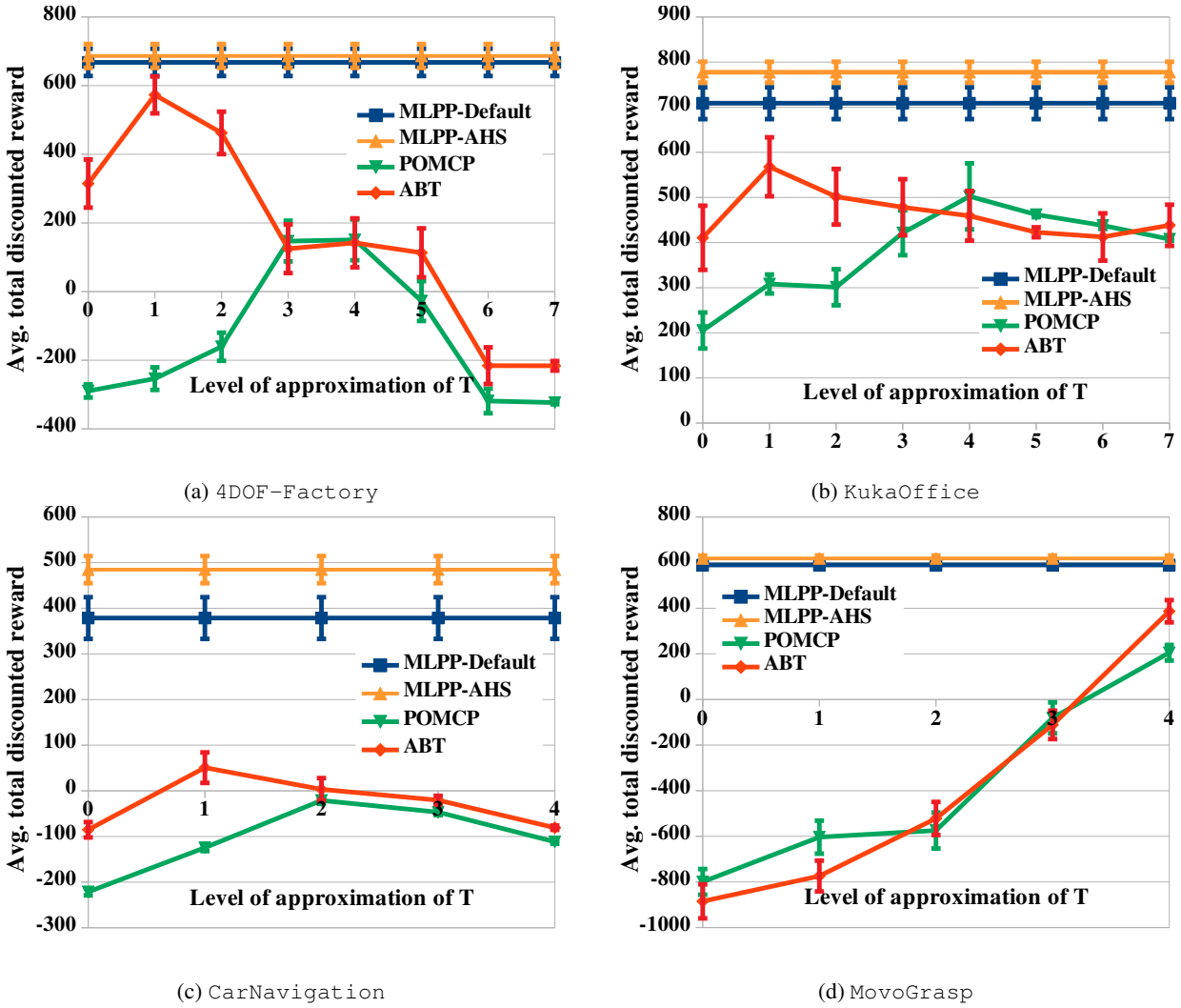
All four test scenarios and the solvers were implemented in C++ within the OPPT framework Hoerger et al. (2018), ensuring that all solvers use the same problem implementation. For ABT we used the implementation provided by the authors Klimenko et al. (2014). For POMCP we used the implementation provided by https://github.com/AdaCompNUS/despot. All simulations were run single-threaded on an Intel Xeon Silver 4110 CPU with 2.1GHz and 128GB of memory.

The results of our experiments are discussed in the next subsection.

4.4 Results

4.4.1 Variances of $Q_l - Q_{l-1}$ Figure 4(a)-(d) shows the average variances of Q_l and $Q_l - Q_{l-1}$ in 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 as we increase 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.

4.4.2 Average total discounted rewards Figure 5(a)-(d) shows the average total discounted rewards achieved by MLPP-Default, MLPP-AHS, ABT and POMCP in all four test scenarios. The average is taken over 500 simulation runs per solver and problem scenario. The results indicate that, for ABT and POMCP, using a single coarse approximation to T for planning can help in computing 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



(a) 4DOF-Factory (b) KukaOffice

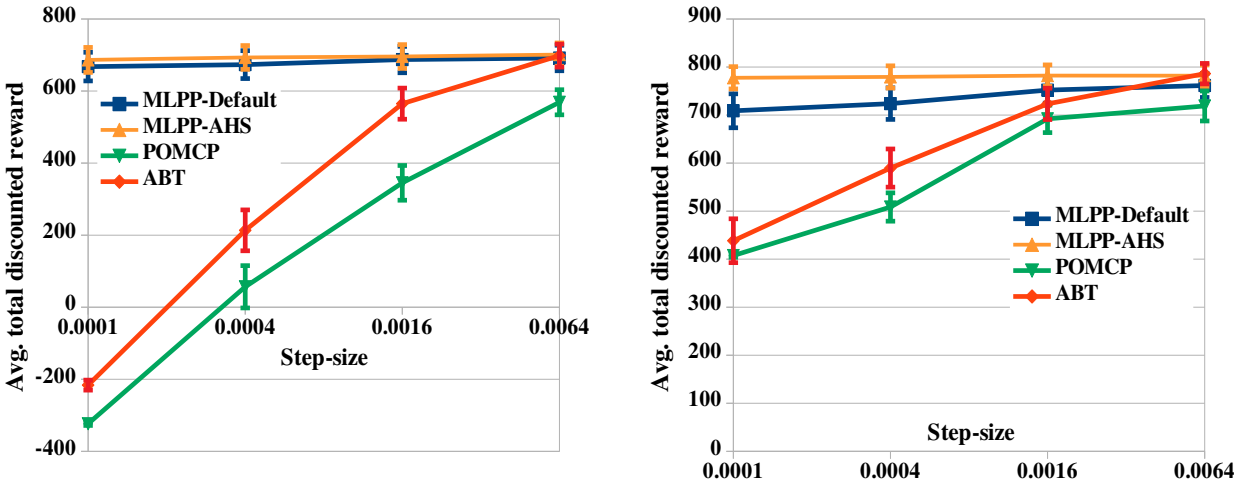
(c) CarNavigation (d) MovoGrasp

Figure 5. Average total discounted reward of MLPP-Default, MLPP-AHS, ABT and POMCP in the (a) 4DOF-Factory, (b) KukaOffice, (c) CarNavigation and (d) MovoGrasp scenarios. The x-axis represents the level of approximation to the transition function used for planning (larger levels correspond to finer approximations). Note that both MLPP-Default and MLPP-AHS use 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 (i.e. the rightmost value on the x-axis) is equal to the original transition function. The y-axis represents the average total discounted reward. The average is taken over 500 simulation runs per solver and problem scenario. Vertical bars are the 95% confidence intervals.

is required. Unlike ABT and POMCP, MLPP-Default and MLPP-AHS cover multiple levels of approximations and are 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 the MovoGrasping as well, where a high accuracy is neccessary for grasping.

Comparing MLPP-Default with MLPP-AHS, we see that MLPP-AHS generally outperforms MLPP-Default, particularly in the KukaOffice and CarNavigation scenarios. This is not surprising. MLPP-Default selects levels of approximations to the transition function according to a fixed distribution that is not necessarily optimal in terms of the cost of sampling episodes on different levels and their effects on the policy. For instance, in the KukaOffice scenario, the cost of sampling correlated episodes increases drastically on the finer levels (in our experiments, the cost of sampling two correlated episodes on level l and $l-1$ approximately doubles as we increase l), whereas the Q-value differences in eq.(3) become significantly smaller (by a factor of up to 10x) as we increase the level. On the other hand, in the CarNavigation scenario, the cost of sampling correlated episodes remains approximately constant across the different levels, while the Q-value differences decrease by a factor of roughly 2x as we increase the level. In both cases MLPP-AHS quickly adapts its level selection strategy to the episode sampling cost on different levels and the importance of the different levels in terms of their contributions to the value function. Subsequently, MLPP-AHS is able to find good policies more efficiently than MLPP-Default.

4.4.3 Transition dynamics with varying costs Figure 6 shows the average total discounted rewards achieved by all solvers in the 4DOF-Factory and KukaOffice scenarios as the transition dynamics of the POMDP problems become progressively cheaper (in terms of sampling cost). Unsurprisingly, the performance of ABT and POMCP



(a) 4DOF-Factory (b) KukaOffice

Figure 6. Average total discounted rewards for MLPP-Default, MLPP-AHS, ABT and POMCP in the (a) 4DOF-Factory and (b) KukaOffice scenarios with transition dynamics of varying cost. The x-axis represents the step-size of the numerical solver used to simulate the transition dynamics, where larger values correspond to cheaper transition dynamics. The y-axis represents the average total discounted reward. The average is taken over 500 simulation runs for each solver and transition function. The vertical bars indicate the 95% confidence intervals.

increases drastically as the cost of the transition dynamics decreases. For the cheapest transition dynamics (i.e. for the transition dynamics that uses an integration step size of $\delta t = 0.0064s$ for the numerical integrator of ODE), ABT matches the performance of both MLPP-Default and MLPP-AHS in the 4DOF-Factory scenario and slightly outperforms both solvers in the KukaOffice scenario. Interestingly, the performance of MLPP-Default and MLPP-AHS in both scenarios is much less affected by the cost of the original transition dynamics. This result indicates that both MLPP-Default and MLPP-AHS are effective in computing an approximate optimal policy, even if the original transition dynamics are expensive.

A slight difference in performance between MLPP-Default and MLPP-AHS occurs in the KukaOffice scenario when the original transition dynamics has a small step-size, which is the most expensive transition dynamics in this set of experiments. This result indicates that an adaptive level selection strategy that balances the cost and benefit will be more advantageous for problems with more expensive transition dynamics.

4.4.4 Increasing planning times Figure 7 shows the average total discounted rewards achieved by each solver in the 4DOF-Factory scenario as the planning time per step increases. The results indicate that both MLPP-Default and MLPP-AHS converge to a good policy significantly faster than ABT and POMCP. ABT requires 15s/step to generate a policy whose quality is similar to the policy generated by MLPP-Default and MLPP-AHS in 2.5s/step, while POMCP is unable to reach a similar level of quality, even with a planning time of 20s/step (in our experiments it took roughly 5 minutes of planning time/step for POMCP to converge to a near-optimal policy).

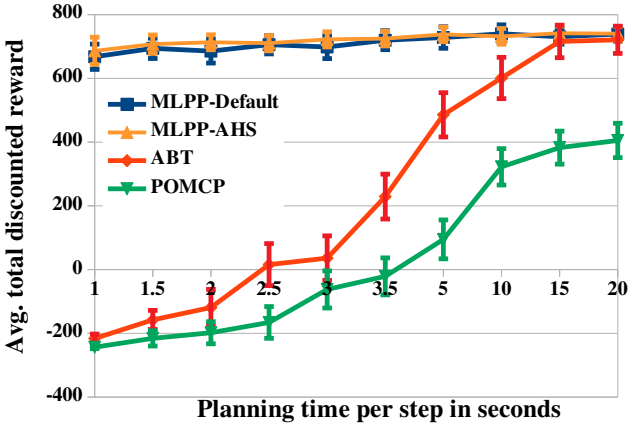


Figure 7. Average total discounted rewards for MLPP-Default, MLPP-AHS, ABT and POMCP in the 4DOF-Factory scenario using increasing planning times per step. The x-axis indicates the planning time per step (in seconds), whereas the y-axis represents the average total discounted reward. The average is taken over 500 simulation runs for each planning time and solver. The vertical bars are the 95% confidence intervals.

5 Discussion on the convergence of MLPP-Default and MLPP-AHS

The above experimental results indicate that both versions of MLPP can converge to a good policy faster than state of the art solvers. Moreover, asymptotic convergence to the optimal solution can be guaranteed for finite horizon POMDPs with horizon d under the assumption below. This result can be expanded to infinite horizons in a similar way many convergence results for finite horizon tree-based POMDP solvers are expanded to infinite horizons (e.g., Silver and Veness (2010)).

Suppose we have an action sequence $(a_1, a_2, a_3, ..., a_K)$ and an initial state s_0, sampled from the current belief. 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 $T_0, T_1, ...T_L$ to the transition function, with $T_L = T$.

Assumption 1. *Given a POMDP P, with transition function T and a sequence of approximations $T_0, T_1, ...T_L$ to the transition function with $T_L = T$, then for any initial state $s_0 \in \mathcal{S}$, any action sequence $(a_1, a_2, a_3, ..., a_K)$ and any observation sequence $(o_1, o_2, o_3, ..., o_K)$, $\prod_{i=1}^{K} T(s_{i-1}, a_i, s_i) Z(s_i, a_i, o_i) > 0$ 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 $\mathcal{T}$ that 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 for $0 \leq l \leq L$. It is true that these levels may not be chosen equally often. In fact, the unequal number of times we use the different levels is indeed the key to the performance of Multi Level Monte Carlo techniques Giles (2008). However, in the limit, each level will be chosen infinitely often because each level will always have a non-zero probability of being selected, regardless of whether we use MLPP-Default or MLPP-AHS.

To show asymptotic convergence of MLPP-Default and MLPP-AHS towards the optimal policy for a finite-horizon POMDP with horizon d, we have to show that under Assumption 1:

(A) At any node $h_{d-1} \in \mathcal{T}$ of depth $d-1$ (we denote a node that is k steps away from the root of $\mathcal{T}$ as h_k), the estimator in eq.(3) converges asymptotically to the true Q-value for any $a \in \mathcal{A}$

(B) The convergence of the Q-values at the nodes of depth $d-1$ implies asymptotic convergence of the Q-values at the root node.

To show that (A) holds, let's suppose h_0 is the current history (i.e. the root node of $\mathcal{T}$), $b(s, h_0)$ is the current belief, and $h_d \in \mathcal{T}$ is a leaf-node with associated action-observation sequence h_d, starting from h_0, that can be reached by episodes sampled using the original transition function. Furthermore, let $h_{d-1} \in \mathcal{T}$ be the parent node of h_d and $b_l(s, h_{d-1})$ be the belief that follows from the current belief and h_{d-1}, under the transition function T_l. Since the Q-values at h_{d-1} only depend on the immediate rewards, for any $a \in \mathcal{A}$, the estimator eq.(3) at h_{d-1} becomes

$$\widehat{Q}(h_{d-1}, a) = \frac{1}{N_0(h_{d-1}, a)} \sum_{i=1}^{N_0(h_{d-1}, a)} R(s_{d-1,0}^{(0,i)}, a) + \sum_{l=1}^{L} \frac{1}{N_l(h_{d-1}, a)} \sum_{i=1}^{N_l(h_{d-1}, a)} R(s_{d-1,l}^{(l,i)}, a) - R(s_{d-1,l-1}^{(l,i)}, a) \tag{11}$$

where $s_{d-1,l}^{(l,i)} \sim b_l(s, h_{d-1})$.

Recall from Section 3 that we use UCB1 as the action selection strategy when sampling episodes using the coarsest approximation to the transition function T_0, whereas for the correlated episodes on level l and $l-1$, we follow the action sequence that was selected at the coarsest level. An important property of UCB1 is the fact that at any node $h \in \mathcal{T}$, each action will be selected infinitely often in the limit. Thus, any action sequence, including the one in h_{d-1}, will be selected infinitely often at the coarsest level and followed on any other level $l > 0$. Since we assume that h_{d-1} can be reached by episodes sampled using the original transition function (i.e. the transition function on level L), it follows by Assumption 1 that h_{d-1} will be reached by episodes sampled using any level of approximation to the transition function infinitely often in the limit. Thus, the estimator eq.(11) converges according to

$$\begin{aligned}
&\lim_{N_0(h_{d-1},a),...,N_L(h_{d-1},a)\to\infty} \widehat{Q}(h_{d-1}, a) \\
&= \int_{s\in\mathcal{S}} b_0(s, h_{d-1}) R(s, a) \mathrm{d}s \\
&\quad + \sum_{l=1}^{L} \int_{s\in\mathcal{S}} b_l(s, h_{d-1}) R(s, a) - b_{l-1}(s, h_{d-1}) R(s, a) \mathrm{d}s \\
&= Q_0(h_{d-1}, a) + \sum_{l=1}^{L} (Q_l(h_{d-1}, a) - Q_{l-1}(h_{d-1}, a)) \\
&= Q_L(h_{d-1}, a) = Q(h_{d-1}, a)
\end{aligned} \tag{12}$$

which guarantees (A).

To show (B), i.e. the asymptotic convergence of the Q-value estimates at the root of $\mathcal{T}$ towards the true values, recall from Section 3 that after sampling an episode using the coarsest approximation to the transition function, we update the estimates $\widehat{Q}_0^{(0)}$ along the selected action sequence using stochastic Bellman backups (see eq.(4)). Similarly, after sampling two correlated episodes on level l and $l-1$, we use stochastic Bellman backups to update $\widehat{Q}_l^{(l)}$ and $\widehat{Q}_{l-1}^{(l)}$ along the common action sequence of both episodes.

For brevity, we only discuss the convergence of $\widehat{Q}_0^{(0)}$, but the result also applies to each $\widehat{Q}_l^{(l)}$ in the difference terms of the right hand side of eq.(3). Subsequently, utilising the results of MLMC Giles (2015), given enough time and the convergence of each term in eq.(3), the estimated Q-value $\widehat{Q}$ on the left-hand side of eq.(3) converges to the correct Q-value. In the following, we discuss the convergence of the mentioned terms.

Recall from Section 3.1 that after sampling an episode on level 0, for each node $h \in \mathcal{T}$ visited by the episode, the estimate $\widehat{Q}_0^{(0)}(h, a)$ (with $a \in \mathcal{A}$ being the action that was executed at h) is updated by computing eq.(4) and updating the first sum on the right-hand side of eq.(3). Thus, after sampling the i-th episode on the coarsest level that executes a at h, this update rule can be written as

$$\begin{aligned}
&\widehat{Q}_{0,i}^{(0)}(h, a) = \widehat{Q}_{0,i-1}^{(0)}(h, a) \\
&+ \alpha_i(h, a) \left(r^{(i)} + \gamma \max_{a' \in \mathcal{A}} \widehat{Q}_{0,i}^{(0)}(hao^{(i)}, a') - \widehat{Q}_{0,i-1}^{(0)}(h, a) \right)
\end{aligned} \tag{13}$$

with $o^{(i)}$ and $r^{(i)}$ being the sampled observation and the immediate reward of the episode after executing a at h and $\alpha_i(h, a) = i^{-1}$. $\widehat{Q}_{0,i-1}^{(0)}(h, a)$ and $\widehat{Q}_{0,i}^{(0)}(h, a)$ refer to the Q-value estimates of a at h on the coarsest level before and after

sampling the episode. Note that eq.(13) is akin to the update-rule used by Q-learning Watkins and Dayan (1992), where in our case, the update is performed in the history space, instead of the state space. It is known that the update rule in eq.(13) is a contraction Bertsekas and Tsitsiklis (1996). The monotonic improvement provided by the contraction property ensures that at each node h, the error $\left|\widehat{Q}^{(0)}_{0,i}(h,a) - Q^{(0)}_0(h,a)\right|$ (with $Q^{(0)}_0(h,a)$ being expected total discounted reward of executing a from h and continuing optimally afterwards, under the transition function T_0) converges to zero for $i \to \infty$ and that this error is upper-bounded by the errors at the child nodes of the action edge ha.

This, together with (A), ensures the asymptotic convergence of MLPP-Default and MLPP-AHS towards the optimal policy.

Assumption 1 is quite strong and might be too restrictive for some problems, since it requires that any observation that has a non-zero probability of being sampled from $Z(s', a, o)$ given a subsequent state $s' \in \mathcal{S}$ sampled from $T(s, a, s')$ also has a non-zero probability of being sampled, given a subsequent sampled from $T_l(s, a, s')$, for $0 \leq l \leq L$. Nevertheless, problems whose transition and observation functions for all state–action pairs are represented as distributions with infinite support (e.g., Gaussian) satisfy the assumption above.

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. They rely on the assumption that each forward simulation can be computed almost instantaneously. This assumption is generally false for problems with complex dynamics.

To alleviate the above shortcomings, we propose MLPP, an on-line POMDP solver that extends Multilevel Monte Carlo to POMDP planning. MLPP samples episodes using multiple levels of approximation to the transition dynamics and computes an approximation to the action-values using a Multilevel Monte Carlo estimator. This strategy enables MLPP to significantly speed-up the planning process while retaining correctness of the action-value estimates. We have presented two variants of MLPP, MLPP-Default and MLPP-AHS and successfully tested them on four robotic planning problems under partial observability that involve expensive transition dynamics and long planning-horizons. In all four problems, MLPP-Default and MLPP-AHS substantially outperform ABT and POMCP, two of the fastest on-line solvers, which shows the effectiveness of the proposed method.

Future work abounds: While this paper focuses on extending the Multilevel Monte Carlo concept to on-line POMDP planning, related Monte Carlo methods that use multiple levels of approximations to models that are expensive to evaluate exist Peherstorfer et al. (2018); Peherstorfer (2019). Some of these methods do not require correlated samples on different levels, which could potentially lead to simpler and more efficient on-line POMDP solvers.

Additionally, while we focus on POMDP problems with expensive transition dynamics, our MLMC-based method could also be applied to complex observation functions by defining a sequence of approximations with increasing accuracy, and hence sampling cost, for the observation function. In many robotics problems, obtaining observations often consists of processing raw sensor data, such as high-resolution images, which can be expensive when many different observation sequences have to be sampled during planning. Applying our method to complex observation functions could be an interesting avenue for future work.

Acknowledgements

This work is partially funded by the ANU Futures Scheme QCE20102

References

Agha-Mohammadi AA, Chakravorty S and Amato NM (2011) 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*. IEEE, pp. 4284–4291.

Anderson DF and Higham DJ (2012) Multilevel monte carlo for continuous time markov chains, with applications in biochemical kinetics. *Multiscale Modeling & Simulation* 10(1): 146–179.

Araya M, Buffet O, Thomas V and Charpillet F (2010) A pomdp extension with belief-dependent rewards. In: Lafferty J, Williams C, Shawe-Taylor J, Zemel R and Culotta A (eds.) *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc. URL https://proceedings.neurips.cc/paper/2010/file/68053af2923e00204c3ca7c6a3150cf7-Paper.pdf.

Arulampalam MS, Maskell S, Gordon N and Clapp T (2002) A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on signal processing* 50(2): 174–188.

Auer P, Cesa-Bianchi N and Fischer P (2002) Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47(2-3): 235–256.

Bai H and Hsu D (2012) Unmanned aircraft collision avoidance using continuous-state pomdps. *Robotics: Science and Systems VII* 1: 1–8.

Bai H, Hsu D and Lee WS (2014) Integrated perception and planning in the continuous space: A pomdp approach. *The International Journal of Robotics Research* 33(9): 1288–1302.

Bertsekas DP and Tsitsiklis JN (1996) *Neuro-dynamic programming*. Belmont, MA: Athena Scientific.

Bierig C and Chernov A (2016) Approximation of probability density functions by the multilevel monte carlo maximum entropy method. *Journal of Computational Physics* 314: 661–681.

Cai P, Luo Y, Hsu D and Lee WS (2021) Hyp-despot: A hybrid parallel algorithm for online planning under uncertainty. *The International Journal of Robotics Research* 40(2-3): 558–573.

Couëtoux A, Hoock JB, Sokolovska N, Teytaud O and Bonnard N (2011) Continuous upper confidence trees.

In: *International Conference on Learning and Intelligent Optimization*. Springer, pp. 433–445.

Fischer J and Tas ÖS (2020) Information particle filter tree: An online algorithm for pomdps with belief-based rewards on continuous domains. In: *International Conference on Machine Learning*. PMLR, pp. 3177–3187.

Garg NP, Hsu D and Lee WS (2019) Despot-alpha: Online pomdp planning with large state and observation spaces. In: *Proc. of Robotics: Science and Systems*.

Giles MB (2008) Multilevel monte carlo path simulation. *Operations Research* 56(3): 607–617.

Giles MB (2015) Multilevel monte carlo methods. *Acta Numerica* 24: 259–328.

Heinrich S (2001) Multilevel monte carlo methods. In: Margenov S, Waśniewski J and Yalamov P (eds.) *Large-Scale Scientific Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-45346-8, pp. 58–67.

Hoerger M and Kurniawati H (2021) An on-line pomdp solver for continuous observation spaces. In: *Proc. IEEE/RSJ Int. Conference on Robotics and Automation (ICRA)*. IEEE.

Hoerger M, Kurniawati H, Bandyopadhyay T and Elfes A (2020) Linearization in motion planning under uncertainty. In: *Algorithmic Foundations of Robotics XII*. Springer, Cham, pp. 272–287.

Hoerger M, Kurniawati H and Elfes A (2018) A software framework for planning under partial observability. In: *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 1–9.

Hoerger M, Kurniawati H and Elfes A (2019a) Multilevel monte-carlo for solving pomdps online. In: *Proc. International Symposium on Robotics Research (ISRR)*.

Hoerger M, Song J, Kurniawati H and Elfes A (2019b) Pomdp-based candy server: Lessons learned from a seven day demo. In: *Proc. AAAI International Conference on Autonomous Planning and Scheduling (ICAPS)*. AAAI, pp. 698–706.

Horowitz M and Burdick J (2013) Interactive non-prehensile manipulation for grasping via pomdps. In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, pp. 3257–3264.

Hsiao K, Kaelbling LP and Lozano-Perez T (2007) Grasping pomdps. In: *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, pp. 4685–4692.

Hsu D, Sanchez-Ante G and Zheng Sun (2005) Hybrid prm sampling with a cost-sensitive adaptive strategy. In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. pp. 3874–3880.

Klimenko D, Song J and Kurniawati H (2014) Tapir: a software toolkit for approximating and adapting pomdp solutions online. In: *Proceedings of the Australasian Conference on Robotics and Automation*.

Kocsis L and Szepesvári C (2006) Bandit based monte-carlo planning. In: *European conference on machine learning*. Springer, pp. 282–293.

Kurniawati H (2022) Partially observable markov decision processes and robotics. *Annual Review of Control, Robotics, and Autonomous Systems* To appear.

Kurniawati H, Du Y, Hsu D and Lee WS (2011) Motion planning under uncertainty for robotic tasks with long time horizons. *The International Journal of Robotics Research* 30(3): 308–323.

Kurniawati H, Hsu D and Lee WS (2008) Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In: *In Proc. Robotics: Science and Systems*.

Kurniawati H and Yadav V (2013) An online pomdp solver for uncertainty planning in dynamic environment. In: *Proc. Int. Symp. on Robotics Research*.

Lim MH, Tomlin CJ and Sunberg ZN (2020) Voronoi progressive widening: Efficient online solvers for continuous state, action, and observation pomdps. *arXiv preprint arXiv:2012.10140* .

Lindquist A (1973) On feedback control of linear stochastic systems. *SIAM Journal on Control* 11(2): 323–343.

Luo Y, Bai H, Hsu D and Lee WS (2019) Importance sampling for online planning under uncertainty. *The International Journal of Robotics Research* 38(2-3): 162–181.

Mern J, Yildiz A, Sunberg Z, Mukerji T and Kochenderfer MJ (2021) Bayesian optimized monte carlo planning. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35. pp. 11880–11887.

Mihaylova L, Lefebvre T, Bruyninckx H, Gadeyne K and De Schutter J (2002) A comparison of decision making criteria and optimization methods for active robotic sensing. In: *International Conference on Numerical Methods and Applications*. Springer, pp. 316–324.

Owen AB (2013) *Monte Carlo theory, methods and examples*.

Papadimitriou CH and Tsitsiklis JN (1987) The complexity of markov decision processes. *Mathematics of operations research* 12(3): 441–450.

Peherstorfer B (2019) Multifidelity monte carlo estimation with adaptive low-fidelity models. *SIAM/ASA Journal on Uncertainty Quantification* 7(2): 579–603.

Peherstorfer B, Willcox K and Gunzburger M (2018) Survey of multifidelity methods in uncertainty propagation, inference, and optimization. *SIAM Review* 60(3): 550–591.

Pineau J, Gordon G and Thrun S (2003) Point-based Value Iteration: An anytime algorithm for POMDPs.

Rhee Ch and Glynn PW (2012) A new approach to unbiased estimation for sde's. In: *Proceedings of the Winter Simulation Conference*. Winter Simulation Conference, p. 17.

Rubinstein RY and Kroese DP (2013) *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media.

Seiler KM, Kurniawati H and Singh SP (2015) An online and approximate solver for pomdps with continuous action space. In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, pp. 2290–2297.

Silver D and Veness J (2010) Monte-carlo planning in large POMDPs. In: *Advances in neural information processing systems*. pp. 2164–2172.

Smith R (2001) Open dynamics engine. http://www.ode.org/.

Smith T and Simmons R (2005) Point-based POMDP algorithms: Improved analysis and implementation.

Somani A, Ye N, Hsu D and Lee WS (2013) Despot: Online pomdp planning with regularization. In: *Advances in neural information processing systems*. pp. 1772–1780.

Sondik EJ (1971) *The Optimal Control of Partially Observable Markov Decision Processes*. PhD Thesis, Stanford, California.

Spong MW, Hutchinson S and Vidyasagar M (2006) *Robot Modeling and Control*, volume 3. Wiley New York.

Sun W, Patil S and Alterovitz R (2015) High-frequency replanning under uncertainty using parallel sampling-based motion planning. *IEEE Transactions on Robotics* 31(1): 104–116.

Sunberg ZN and Kochenderfer MJ (2018) Online algorithms for pomdps with continuous state, action, and observation spaces. In: *Twenty-Eighth International Conference on Automated Planning and Scheduling*.

Sutton R and Barto A (2012) *Reinforcement Learning: An Introduction*. MIT Press.

Van Den Berg J, Abbeel P and Goldberg K (2011) Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information. *The International Journal of Robotics Research* 30(7): 895–913.

Van Den Berg J, Patil S and Alterovitz R (2012) Motion planning under uncertainty using iterative local optimization in belief space. *The International Journal of Robotics Research* 31(11): 1263–1278.

Wandzel A, Oh Y, Fishman M, Kumar N, Wong LL and Tellex S (2019) Multi-object search using object-oriented pomdps. In: *2019 International Conference on Robotics and Automation (ICRA)*. pp. 7194–7200.

Wang E, Kurniawati H and Kroese DP (2018) An on-line planner for pomdps with large discrete action space: A quantile-based approach. In: *ICAPS*. AAAI Press, pp. 273–277.

Watkins CJ and Dayan P (1992) Q-learning. *Machine learning* 8(3-4): 279–292.