Article

The International Journal of Robotics Research
2023, Vol. 42(13) 1169–1184
© The Author(s) 2023
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/02783649231188984
journals.sagepub.com/home/ijr
Sage

Adaptive Discretization using Voronoi Trees for Continuous POMDPs

Marcus Hoerger[1], Hanna Kurniawati[2], Dirk Kroese[1] and Nan Ye[1]

Abstract

Solving continuous Partially Observable Markov Decision Processes (POMDPs) is challenging, particularly for high-dimensional continuous action spaces. To alleviate this difficulty, we propose a new sampling-based online POMDP solver, called **A**daptive **D**iscretization using **V**oronoi **T**rees (ADVT)*. It uses Monte Carlo Tree Search in combination with an adaptive discretization of the action space as well as optimistic optimization to efficiently sample high-dimensional continuous action spaces and compute the best action to perform. Specifically, we adaptively discretize the action space for each sampled belief using a hierarchical partition called* Voronoi tree*, which is a Binary Space Partitioning that implicitly maintains the partition of a cell as the Voronoi diagram of two points sampled from the cell. ADVT uses the estimated diameters of the cells to form an upper-confidence bound on the action value function within the cell, guiding the Monte Carlo Tree Search expansion and further discretization of the action space. This enables ADVT to better exploit local information with respect to the action value function, allowing faster identification of the most promising regions in the action space, compared to existing solvers. Voronoi trees keep the cost of partitioning and estimating the diameter of each cell low, even in high-dimensional spaces where many sampled points are required to cover the space well. ADVT additionally handles continuous observation spaces, by adopting an observation progressive widening strategy, along with a weighted particle representation of beliefs. Experimental results indicate that ADVT scales substantially better to high-dimensional continuous action spaces, compared to state-of-the-art methods.*

Keywords

Planning under uncertainty, motion and path planning, autonomy for mobility and manipulation

Received 23 January 2023; Accepted 19 June 2023

1. Introduction

Planning in scenarios with non-deterministic action effects and partial observability is an essential, yet challenging problem for autonomous robots. The Partially Observable Markov Decision Process (POMDP) (Kaelbling et al. 1998; Sondik 1971) is a general principled framework for such planning problems. POMDPs lift the planning problem from the state space to the *belief space*—that is, the set of all probability distributions over the state space. By doing so, POMDPs enable robots to systematically account for uncertainty caused by stochastic actions and incomplete or noisy observations in computing the optimal strategy. Although computing the optimal strategy exactly is intractable in general (Papadimitriou and Tsitsiklis 1987), the past two decades have seen a surge of sampling-based POMDP solvers (reviewed in Kurniawati (2022)) that trade optimality with approximate optimality for computational tractability, enabling POMDPs to become practical for a variety of realistic robotics problems.

Despite these advances, POMDPs with continuous state, action, and observation spaces remain a challenge, particularly for high-dimensional continuous action spaces. Recent solvers for continuous action POMDPs (Fischer and Tas 2020; Mern et al. 2021; Seiler et al. 2015; Sunberg and Kochenderfer 2018) are generally online—that is, planning and execution are interleaved—and exploit Monte Carlo Tree Search (MCTS) to find the best action among a finite representative subset of the action space. MCTS interleaves guided belief space sampling, value estimation, and action subset refinement to incrementally improve the possibility that the selected subset of actions contains the best action. They generally use UCB1 (Auer et al. 2002) to guide belief

[1]School of Mathematics & Physics, The University of Queensland, Queensland, ACT, Australia
[2]School of Computing, Australian National University, Canberra, ACT, Australia

Corresponding author:
Marcus Hoerger, School of Mathematics & Physics, The University of Queensland, Queensland, ACT 4072, Australia.
Email: m.hoerger@uq.edu.au

space sampling and Monte Carlo backup for value estimation, but differ in the action subset refinement.

Several approaches use the Progressive Widening strategy (Couëtoux et al. 2011) to continuously add new randomly sampled actions once current actions have been sufficiently explored. Examples include POMCPOW (Sunberg and Kochenderfer 2018) and IPFT (Fischer and Tas 2020). More recent algorithms combine Progressive Widening with more informed methods for adding new actions: VOMCPOW (Lim et al. 2021) uses Voronoi Optimistic Optimization (Kim et al. 2020) and BOMCP (Mern et al. 2021) uses Bayesian optimization. All of these solvers use UCT-style simulations and Monte Carlo backups. An early line of work, GPS-ABT (Seiler et al. 2015) takes a different approach: It uses Generalized Pattern Search to iteratively select an action subset that is more likely to contain the best action and add it to the set of candidate actions. GPS-ABT uses UCT-style simulations and Bellman backup (following the implementation of ABT (Hoerger et al. (2018); Klimenko et al. (2014)), though the distinction between Monte Carlo and Bellman backup was not clarified nor explored. All of these solvers have been successful in finding good solutions to POMDPs with continuous action spaces, though for a relatively low (≤ 4) dimension.

To compute good strategies for POMDPs with high-dimensional action spaces, we propose a new MCTS-based online POMDP solver, called **A**daptive **D**iscretization using **V**oronoi **T**rees (ADVT). ADVT is designed for problems with continuous action spaces, while the state and observation spaces can either be discrete or continuous. ADVT contains three key ideas, as briefly described below.

The first key idea of ADVT is a new data structure for action space discretization, called *Voronoi tree*. A Voronoi tree represents a hierarchical partition of the action space for a single sampled belief. It follows the structure of a Binary Space Partitioning (BSP) tree, but each partitioning hyper-plane is only implicitly maintained and computed based on the Voronoi diagram of a pair of sampled actions, which results in a partitioning that is much more adaptive to the spatial locations of the sampled actions, compared to state-of-the-art methods (Bubeck et al. 2011; Lim et al. 2021; Mansley et al. 2011; Sunberg and Kochenderfer 2018; Valko et al. 2013). We additionally maintain estimates of the diameters of the cells in the Voronoi tree. These diameters are used in the other two key ideas, namely, action selection and Voronoi tree refinement, as described in the next two paragraphs. The hierarchical structure of the Voronoi tree allows us to estimate the diameters with an efficient sampling-based algorithm that scales well to high-dimensional action spaces. Section 5 provides a more detailed description on the computational and representational advantages of the Voronoi tree.

The second key idea of ADVT is the use of a cell-diameter-aware upper-confidence bound on the values of actions to guide action selection during planning. This bound represents an upper-confidence bound on the values of *all* actions within a cell, based on the estimated value of the corresponding sampled action and the diameter of the cell. Our bound is a generalization of a bound that was developed for Lipschitz continuum-arm bandit problems (Wang et al. 2020). It is motivated by the observation that in many continuous action POMDPs for robotics problems, the distance between two actions can often be used as an indication of how similar their values are. Using this observation, ADVT assumes that the action value for a belief is Lipschitz continuous in the action space, which in turn allows us to derive the upper bound. The upper bound helps ADVT to exploit local information (i.e., the estimated value of the representative action of a cell and the cell diameter) with respect to the action value function and bias its search toward the most promising regions of the action space.

The third key idea is a diameter-aware cell refinement rule. We use the estimated diameters of the cells to help ADVT decide if a cell needs further refinement: a larger cell will be split into smaller ones after a small number of simulations, while a smaller will require more simulations before it is split. This helps ADVT to avoid an unnecessarily small partitioning of non-promising regions in the action space.

To further support ADVT in efficiently finding approximately optimal actions, we use a stochastic version of the Bellman backup (Klimenko et al. 2014) rather than the typical Monte Carlo backup to estimate the values of sampled actions. Stochastic Bellman backups help ADVT to backpropagate the value of good actions deep in the search tree, instead of averaging them out. This strategy of estimating the action values is particularly helpful for problems with sparse rewards.

Aside from continuous action spaces, continuous observation spaces pose an additional challenge for MCTS-based online POMDP solvers. Recent approaches such as POMCPOW and VOMCPOW apply Progressive Widening in the observation space in conjunction with an explicit representation of the sampled beliefs via a set of weighted particles. Due to its simplicity, we adopt POMCPOW's strategy to handle continuous observation spaces. This enables ADVT to scale to problems with continuous state, action and observation spaces.

Experimental results on a variety of benchmark problems with increasing dimension (up to 12-D) of the action space and problems with continuous observation spaces indicate that ADVT substantially outperforms state-of-the-art methods (Lim et al. 2021; Sunberg and Kochenderfer 2018). Our C++ implementation of ADVT is available at https://github.com/hoergems/ADVT.

This paper extends our previous work (Hoerger et al. 2022) in three ways: First is the extension of ADVT to handle continuous observation spaces. Experimental results on an additional POMDP benchmark problem demonstrate the effectiveness of ADVT in handling purely continuous POMDP problems. Second is a substantially expanded discussion on the technical concepts

of ADVT. And third is an extended ablation study in which we investigate the effectiveness of stochastic Bellman backups when applied to two baseline solvers POMCPOW and VOMCPOW.

2. Background and related work

A POMDP provides a general mathematical framework for sequential decision making under uncertainty. Formally, it is an 8-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, b_0, \gamma \rangle$. The robot is initially in a hidden state $s_0 \in \mathcal{S}$. This uncertainty is represented by an initial belief $b_0 \in \mathcal{B}$, which is a probability distribution on the state space $\mathcal{S}$, where $\mathcal{B}$ is the set of all possible beliefs. At each step $t \geq 0$, the robot executes an action $a_t \in \mathcal{A}$ according to some policy π. It transitions to a next state $s_{t+1} \in \mathcal{S}$ according to the transition model $T(s_t, a_t, s_{t+1}) = p(s_{t+1}|s_t, a_t)$. For discrete state spaces, $T(s_t, a_t, s_{t+1})$ represents a probability mass function, whereas for continuous state spaces, it represents a probability density function.

The robot does not know the state s_{t+1} exactly, but perceives an observation $o_t \in \mathcal{O}$ according to the observation model $Z(s_{t+1}, a_t, o_t) = p(o_t|s_{t+1}, a_t)$. $Z(s_{t+1}, a_t, o_t)$ represents a probability mass function for discrete observation spaces, or a probability density function for continuous observation spaces, respectively. In addition, it receives an immediate reward $r_t = R(s_t, a_t) \in \mathbb{R}$. The robot's goal is to find a policy π that maximizes the expected total discounted reward or the policy value

$$V_\pi(b_0) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| b_0, \pi\right], \tag{1}$$

where the discount factor $0 < \gamma < 1$ ensures that $V_\pi(b)$ is finite and well-defined.

The robot's decision space is the set Π of policies defined as mappings from beliefs to actions. The POMDP solution is then the optimal policy, denoted as π^* and defined as

$$\pi^* = \arg\max_{\pi \in \Pi} V_\pi(b). \tag{2}$$

In designing solvers, it is often convenient to work with the action value or Q-value

$$Q(b, a) = R(b, a) + \gamma \mathbb{E}_{o \in \mathcal{O}}\left[V_{\pi^*}\left(b_a^o\right) \middle| b\right], \tag{3}$$

where $R(b, a) = \int_{s \in \mathcal{S}} b(s) R(s, a) \mathrm{d}s$ is the expected reward of executing action a at belief b, while $b_a^o = \tau(b, a, o)$ is the updated robot's belief estimate after it performs action $a \in \mathcal{A}$ while at belief b, and subsequently perceives observation $o \in \mathcal{O}$. The optimal value function is then

$$V^*(b) = \max_{a \in \mathcal{A}} Q(b, a). \tag{4}$$

A more elaborate explanation is available in Kaelbling et al. (1998).

Belief trees are convenient data structures to find good approximations to the optimal solutions via sampling-based approaches, which have been shown to significantly improve the scalability of POMDP solving (Kurniawati 2022). Each node in a belief tree represents a sampled belief. It has outgoing edges labeled by actions, and each action edge is followed by outgoing edges labeled by observations which lead to updated belief nodes. Naïvely, bottom-up dynamic programming can be applied to a truncated belief tree to obtain a near-optimal policy, but many scalable POMDP solvers use more sophisticated sampling strategies to construct a compact belief tree, from which a close-to-optimal policy can be computed efficiently. ADVT uses such a sampling-based approach and belief tree representation too.

Various efficient sampling-based offline and online POMDP solvers have been developed for increasingly complex discrete and continuous POMDPs in the last two decades. Offline solvers (e.g., Bai et al. 2014; Kurniawati et al. 2011, 2008; Pineau et al. 2003; Smith and Simmons 2005) compute an approximately optimal policy for all beliefs first before deploying it for execution. In contrast, online solvers (e.g., Kurniawati and Yadav 2013; Silver and Veness 2010; Ye et al. 2017) aim to further scale to larger and more complex problems by interleaving planning and execution, and by focusing on computing an optimal action for only the current belief during planning. For scalability purposes, ADVT follows the online solving approach.

Some online solvers have been designed for continuous POMDPs. In addition to the general solvers discussed in Section 1, some solvers (Agha-Mohammadi et al. 2011; Sun et al. 2015; Van Den Berg et al. 2011, 2012) restrict beliefs to be Gaussian and use Linear–Quadratic–Gaussian (LQG) control (Lindquist 1973) to compute the best action. This strategy generally performs well in high-dimensional action spaces. However, they tend to perform poorly in problems with large uncertainties (Hoerger et al. 2020).

Last but not least, hierarchical rectangular partitions have been commonly used to discretize action spaces when solving continuous action bandits and MDPs (the fully observed version of POMDPs), such as HOO (Bubeck et al. 2011) and HOOT (Mansley et al. 2011). However, the partitions used in these algorithms are typically predefined, which are less adaptive than Voronoi-based partitions constructed dynamically during the search. On the other hand, Voronoi partitions have been proposed in VOOT (Kim et al. 2020) and VOMCPOW (Lim et al. 2021). However, their partitions are based on the Voronoi diagram of all sampled actions, which makes the computation of cell diameters and sampling relatively complex in high-dimensional action spaces. ADVT is computationally efficient, just like hierarchical rectangular partitions, and yet adaptive, just like the Voronoi partitions, getting the best of both worlds.

Algorithm 1. ADVT (Initial belief b_0)

1: $b = b_0$
2: $\mathcal{T}$ = initializeBeliefTree(b)
3: $\mathcal{H}(b)$ = Initialize Voronoi tree for belief b
4: isTerminal = False
5: **while** isTerminal is False **do**
6: **while** planning budget not exceeded **do**
7: (e, b_c) = SAMPLEEPISODE($\mathcal{T}$, b) ▷ Algorithm 2
8: **for** $i = |e| - 1$ to 1 **do**
9: (s, a, o, r) = e_i
10: BACKUP($\mathcal{T}$, b_c, a, r) ▷ Algorithm 3
11: b_c = Parent node of b_c in $\mathcal{T}$
12: REFINEVORONOITREE($\mathcal{H}(b_c)$, a) ▷ Algorithm 5
13: **end for**
14: **end while**
15: $a^* = \arg\max_{a \in \mathcal{A}(b)} \widehat{Q}(b, a)$
16: (o, isTerminal) = Execute $a*$
17: $b' = \tau(b, a*, o)$
18: $b = b'$
19: **end while**

3. ADVT: Overview

In this paper, we consider a POMDP $\mathcal{P} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, b_0, \gamma \rangle$, where the action space $\mathcal{A}$ is continuous and embedded in a bounded metric space with distance function d. Typically, we define the metric space to be a D-dimensional bounded Euclidean space, though ADVT can also be used with other types of bounded metric spaces. We further consider the state space $\mathcal{S}$ and observation space $\mathcal{O}$ to be either discrete or continuous.

ADVT assumes that the Q-value function is Lipschitz continuous in the action space; that is, for any belief $b \in \mathcal{B}$, there exists a Lipschitz constant L_b such that for any actions $a, a' \in \mathcal{A}$, we have $|Q(b,a) - Q(b,a')| \leq L_b \ d(a, a')$. Since generally we do not know a tight Lipschitz constant, in the implementation, ADVT uses the same Lipschitz constant L for all beliefs in $\mathcal{B}$, as discussed in Section 4.1.

ADVT is an anytime online solver for POMDPs. It interleaves belief space sampling and action space sampling to find the best action to perform from the current belief $b \in \mathcal{B}$. The sampled beliefs are maintained in a belief tree, denoted as $\mathcal{T}$, while the sampled actions $\mathcal{A}(b)$ for a belief b are maintained in a Voronoi tree, denoted as $\mathcal{H}(b)$, which is adaptively refined. The Voronoi trees form part of the belief tree in ADVT: they determine the sampled action branches for the belief nodes.

Algorithm 1 presents the overall algorithm of ADVT, with details in the sections below. At each planning step, ADVT follows the MCTS approach of constructing a belief tree by sampling a set of *episodes* (line 7 in Algorithm 1), starting from the current belief. Details on the belief tree construction are provided in Section 4. After sampling an episode, the estimated action values $\widehat{Q}(b,a)$ along the sequence of actions selected by the episode are updated using a backup operation (line 10 in Algorithm 1). In addition, ADVT refines the Voronoi tree $\mathcal{H}(b)$ as needed for each belief visited by the episode (line 12 in Algorithm 1), as discussed in Section 5. Once a planning budget is exceeded, ADVT selects an action a^* from the current belief according to

$$a^* = \arg\max_{a \in \mathcal{A}(b)} \widehat{Q}(b, a), \tag{5}$$

executes a^* in the environment to obtain an observation $o \in \mathcal{O}$, updates the current belief to $b' = \tau(b, a^*, o)$ (line 15-17 in Algorithm 1), and proceeds planning from the updated belief. This process repeats until the robot enters a terminal state or a maximum number of planning steps has been exceeded.

4. ADVT: Construction of the Belief Tree

The belief tree $\mathcal{T}$ is a tree whose nodes represent beliefs and the edges are associated with action–observation pairs (a, o), where $a \in \mathcal{A}$ and $o \in \mathcal{O}$. A node b' is a child of node b via edge (a, o) if and only if $b' = \tau(b, a, o)$.

Algorithm 2. SAMPLEEPISODE(Belief tree $\mathcal{T}$, Belief node b_c)

Notations: $\mathcal{H}(b)$ = Voronoi tree associated with belief b; $A(b)$ = Set of candidate actions associated to the leaf nodes of $\mathcal{H}(b)$
e = Empty sequence of state-action-observation-reward quadruples; $b = b_c$; s = A random state sampled from b; newBelief = False
while newBelief is False and s not terminal **do**
  $a = \arg\max_{a_k \in \mathcal{A}(b)} U(b, a_k)$ ▷ equation (6)
  (b', s', o, r) = STEP(b, s, a) ▷ Algorithm 4
  Append (s, a, o, r) to e
  $N(b, a) = N(b, a) + 1$; $N(b) = N(b) + 1$
  if $\mathcal{A}(b') = \varnothing$ **then**
    a = Sample uniformly from $\mathcal{A}$
    $\mathcal{H}(b')$ = Initialize Voronoi tree for belief b'
    Associate $(a, \mathcal{A})$ with the root node of $\mathcal{H}(b')$
    $N(b') = 0$; $N(b', a) = 0$
    newBelief = True
  end if
  $s = s'$
  $b = b'$
end while
$r = 0$
if newBelief is True **then**
  h = calculateRolloutHeuristic(s, b)
  Initialize $\widehat{V}^*(b)$ with h
end if
insert (s, − , − ,r) to e
return (e, b)

To construct the belief tree $\mathcal{T}$, ADVT interleaves the iterative select-expand-simulate-backup operations used in many MCTS algorithms with adaptive discretization. We assume that we have access to a *generative model* $G: \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S} \times \mathcal{O} \times \mathbb{R}$ that simulates the dynamics,

observation, and reward models. In particular, for a given state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, we have that $(s', o, r) = G(s, a)$, where (s', o) is distributed according to $p(s', o|s, a) = T(s, a, s')$ $Z(s', a, o)$, and $r = R(s, a)$. At each iteration, we first *select* a path starting from the root by sampling an episode $s_0, a_0, o_0, r_0, s_1, a_1, o_1, r_1, \ldots$ as follows: We first set the current node b as the root node, and sample s_0 from b. At each step $i \geq 0$, we choose an action $a_i \in \mathcal{A}(b)$ at b using an action selection strategy (discussed in Section 4.1), execute a_i from state s_i via the generative model G to obtain a next state s', an observation o and the immediate reward r_i. For problems with discrete observation spaces, we set the episode's next state s_{i+1} and observation o_i to $s_{i+1} = s'$ and $o_i = o$, respectively. For problems with continuous observation spaces, we select s_{i+1} and o_i according to an observation sampling strategy, as discussed in Section 4.3. Finally, we update b to b's child node via (a_i, o_i). The process terminates when encountering a terminal state or when the child node does not exist; in the latter case, the tree is *expanded* by adding a new node, and a rollout policy is *simulated* to provide an estimated value for the new node. In either case, *backup* operations are performed to update the estimated values for all actions selected by the episode. In addition, new actions are added to $\mathcal{A}(b)$ by refining the associated Voronoi tree $\mathcal{H}(b)$ as needed for each encountered belief. Algorithm 2 presents the pseudo-code for constructing $\mathcal{T}$, while the backup operation and refinement of $\mathcal{H}(b)$ are discussed in Section 4.2 and Section 5, respectively.

4.1. Action Selection Strategy

In contrast to many existing online solvers, which use UCB1 to select the action to expand a node b of $\mathcal{T}$, ADVT treats the action selection problem as a continuum-arm bandit problem. Specifically, it selects an action from the set of candidate actions $\mathcal{A}(b)$ according to Wang et al. 2020

$$a^* = \arg\max_{a \in \mathcal{A}(b)} U(b, a), \text{with} \tag{6}$$

$$U(b, a) = \widehat{Q}(b, a) + C\sqrt{\frac{\log N(b)}{N(b, a)}} + L \ \text{diam}(P), \tag{7}$$

where N(b) is the number of times node b has been visited so far, $N(b, a)$ is the number of times a has been selected atb, $P \subseteq \mathcal{A}$ is the unique leaf cell containing a in $\mathcal{H}(b)$ (see Section 5 for details on the Voronoi tree), and diam(P) = $\sup_{a,a' \in P} d(a, a')$ is the diameter of P with respect to the distance metric d. The constant C is an exploration constant, where larger values of C encourage exploration. In case $N(b, a) = 0$, we set $U(b, a) = \infty$. With the Lipschitz continuity assumption, the value $U(b, a)$ can be seen as an upper-confidence bound for the maximum possible Q-value $\max_{a' \in P} Q(b, a')$ within P, as follows: $\widehat{Q}(b, a) + C\sqrt{\log N(b)/N(b, a)}$ is the standard UCB1 bound for the Q-value $Q(b, a)$, and whenever this upper bounds $Q(b, a)$, we have $U(b, a) \geq Q(b, a')$ for any $a' \in P$, because $U(b, a) \geq Q(b, a) + L$ diam(P) $\geq Q(b, a')$, where the last inequality holds due to the Lipschitz assumption. Since L is unknown, we try different values of L in our experiments and choose the best.

4.2. Backup

Algorithm 3. Backup(Belief tree $\mathcal{T}$, Belief node b_0, Action a, Reward r)

1: b = Parent node of b' in $\mathcal{T}$
2: $\widehat{Q}(b, a) \leftarrow \widehat{Q}(b, a) + \frac{r + \gamma \widehat{V}^*(b') - \widehat{Q}(b, a)}{N(b, a)}$
3: $\widehat{V}^*(b) = \max_{a \in \mathcal{A}(b)} \widehat{Q}(b, a)$

After sampling an episode e, ADVT updates the estimates $\widehat{Q}(b, a)$ as well as the statistics $N(b)$ and $N(b, a)$ along the sequence of beliefs visited by the episode. To update $\widehat{Q}(b, a)$, we use a stochastic version of the Bellman backup (Algorithm 3): Suppose r is the immediate reward sampled by the episode after selecting a from b. We then update $\widehat{Q}(b, a)$ according to

$$\widehat{Q}(b, a) \leftarrow \widehat{Q}(b, a) + \frac{r + \gamma \widehat{V}^*(b') - \widehat{Q}_i(b, a)}{N(b, a)}, \tag{8}$$

where b' is the child of b in the belief tree $\mathcal{T}$ via edge (a, o); that is, the belief we arrived at after performing action $a \in \mathcal{A}$ and perceiving observation $o \in \mathcal{O}$ from b, and $\widehat{Q}_i(b, a)$ is the previous estimate of $Q(b, a)$. This rule is in contrast to POMCP, POMCPOW and VOMCPOW, where the Q-value estimates are updated via Monte Carlo backup, that is

$$\widehat{Q}(b, a) \leftarrow \widehat{Q}(b, a) + \frac{r + \gamma \widehat{V}_e(b') - \widehat{Q}_i(b, a)}{N(b, a)}, \tag{9}$$

where $\widehat{V}_e(b')$ is the the total discounted reward of episode e, starting from belief b'.

Our update rule in equation (8) is akin to the rule used in Q-Learning (Watkins and Dayan 1992) and was implemented in the ABT software (Hoerger et al. 2018; Klimenko et al. 2014), though never explicitly compared with Monte Carlo backup.

The update rule in equation (8) helps ADVT to focus its search on promising parts of the belief tree, particularly for problems where good rewards are sparse. For sparse-reward problems, the values of good actions deep in the search tree tend to get averaged out near the root when using Monte Carlo backups, thus their influence on the action values at the current belief diminishes. In contrast, since stochastic Bellman backups always backpropagates the current largest action value for a visited belief, large action values deep in the search tree have a larger influence on the action values near the root. As we will demonstrate in the experiments, using stochastic Bellman backups instead of Monte Carlo backups can lead to a significant performance benefit for many POMDP problems.

4.3. Observation sampling strategy

Algorithm 4. STEP(Belief node b, State s, Action a)

1: $(s', o, r) = G(s, a)$ ▷ Generative model
2: $b' = null$
3: **if** $\mathcal{O}$ is discrete **then**
4: b' = Child node of b via edge (a, o). (If no such child exists, create one)
5: **else**
6: **if** $|\mathcal{O}(b,a)| > k_o N(b,a)^{\alpha_o}$ **then**
7: o = Sample o uniformly at random from $\mathcal{O}(b,a)$
8: **end if**
9: b' = Child node of b via edge (a, o). (If no such child exists, create one)
10: $w = Z(s', a, o)$
11: $b' \cup \{(s', w)\}$
12: $s' \sim b'$
13: $r = R(s, a, s')$
14: **end if**
15: **return** (b', s', o, r)

During the episode-sampling process, when ADVT selects an action a at a belief b according to equation (6), we must sample an observation to advance the search to the next belief. ADVT uses different observation sampling strategies, depending on whether the observation space $\mathcal{O}$ is discrete or continuous. Algorithm 4 summarizes ADVT ′s observation sampling strategy for both discrete and continuous observation spaces.

For discrete observation spaces, ADVT follows the common strategy of sampling an observation from the generative model G, given the current state of the episode and the selected action, and representing each sampled observation via an observation edge in the search tree. Since the number of possible observations is finite in the discrete setting, this strategy typically works well for moderately sized observation spaces.

For continuous observation spaces, however, the above strategy is unsuitable. In this setting, each sampled observation is generally unique, which leads to a possibly infinite number of observations that need to be represented in the search tree. As a consequence, we cannot expand the search beyond the first step, resulting in policies that are too myopic. Thus, to handle continuous observation spaces, we adopt a strategy similar to the one used by POMCPOW (Sunberg and Kochenderfer 2018) and VOMCPOW (Lim et al. 2021). This strategy consists of two components, as described below.

The first component is to apply Progressive Widening (Couëtoux et al. 2011) to limit the number of sampled observation edges per action edge as a function of $N(b, a)$, that is, the number of times we have selected a from b. In particular, let $\mathcal{O}(b,a)$ be the set of observation children of action aat belief b. We sample a new observation as a child a whenever $|\mathcal{O}(b,a)|$ satisfies $|\mathcal{O}(b,a)| \leq k_o N(b,a)^{\alpha_o}$, where $k_o \geq 0$ and $\alpha_o \geq 0$ are user defined parameters that control the rate at which new observations are added to the tree. If this condition is not satisfied, we uniformly sample an observation from $\mathcal{O}(b,a)$. This enables ADVT to visit sampled observation edges multiple times and expand the search tree beyond one step.

The second component is to use an explicit representation of each sampled belief in the search tree via a set of weighted particles $\{(s_i, w_i)_{i=1}^n\}$, which allows us to obtain state samples that are correctly distributed according to the beliefs. When sampling an episode, suppose ADVT selects action a from belief b, samples a next state s' from the generative model and selects an observation o using the strategy above. The weight corresponding to s' is then computed according to $w = Z(s', a, o)$ and (s', w) is added to the particle set representing belief b' whose parent is b via the edge (a, o). To obtain a next state s' that is distributed according to b', we resample s' from b' with a probability proportional to the particle weights and continue from b'.

Note that in contrast to online POMDP solvers for discrete observation spaces that only require a black-box model to sample observations, we additionally require access to the observation function $Z(s, a, o)$. However, this is a common assumption for solvers that are designed for continuous observation spaces (Sunberg and Kochenderfer 2018; Hoerger and Kurniawati 2021).

Additionally, note that it is possible to apply the observation sampling strategy for continuous observation spaces to discrete observation spaces as well. However, for discrete observation spaces, this introduces unnecessary computational overhead. As discussed above, the purpose of explicit belief representations is to obtain state samples that are correctly distributed according to the beliefs. For discrete observation spaces, this is achieved by sampling a next state s' according to the transition model $T(s, a, s')$ and an observation o according to the observation model $Z(s', a, o)$, and then use s' as a sample from the belief $b' = \tau(b, a, o)$. Since the sampled observations are always distributed according to $Z(s', a, o)$, the state s' is distributed according to b'. This is in contrast to continuous observation spaces, where ADVT samples observations from a distribution that is potentially different to $Z(s', a, o)$ (line 7 in Algorithm 4). Therefore, ADVT handles discrete and continuous observation spaces separately and avoids the weighting a resampling step in the discrete case, thus saving computation time.

5. ADVT: Construction and Refinement of Voronoi Trees

Algorithm 5. REFINEVORONOITREE(Voronoi tree $\mathcal{H}(b)$, Action a)

1: (a, P) = leaf node of $\mathcal{H}(b)$ with its action component being a
2: **if** $C_r N(b, a) \geq 1/\text{diam}(P)^2$ **then**
3: a' = sample from P
4: (P_1, P_2) = Child cells of P induced by a and a'
5: Compute diameters of P_1 and P_2
6: Add (a, P_1) and (a', P') as (a, P)'s children
7: **end if**

For each belief node b in the belief tree, its Voronoi tree $\mathcal{H}(b)$ is a BSP tree for $\mathcal{A}$. Each node in $\mathcal{H}(b)$ consists of a pair (a, P) with $P \subseteq \mathcal{A}$ and $a \in P$ the representative action of P, and each non-leaf node is partitioned into two child nodes. The partition of each cell in a Voronoi tree is a Voronoi diagram for two actions sampled from the cell.

To construct $\mathcal{H}(b)$, ADVT first samples an action a_0 uniformly at random from the action space $\mathcal{A}$, and sets the pair $(a_0, \mathcal{A})$ as the root of $\mathcal{H}(b)$. When ADVT decides to expand a leaf node (a, P), it first samples an action a' uniformly at random from $P \subseteq \mathcal{A}$. ADVT then implicitly constructs the Voronoi diagram between a and a' within the cell P, splitting it into two regions: One is P_1, representing the set of all actions $a'' \in P$ for which the distance $d(a'', a)$ satisfies $d(a'', a) \leq d(a'', a')$, and the other is $P_2 = P \backslash P_1$. The nodes (a, P_1) and (a', P_2) are then inserted as children of (a, P) in $\mathcal{H}(b)$. The leaf nodes of $\mathcal{H}(b)$ form the partition of the action space $\mathcal{A}$ used by belief b, while the finite action subset $\mathcal{A}(b) \subset \mathcal{A}$ used to find the best action from b is the set of actions associated with the leaves of $\mathcal{H}(b)$. Figure 1 illustrates the relationship between a belief, the Voronoi tree $\mathcal{H}(b)$ and the partition of $\mathcal{A}$.

Voronoi trees have a number of representational and computational advantages compared to existing partitioning methods, such as hierarchical rectangular partitions (Bubeck et al. 2011; Mansley et al. 2011) or Voronoi diagrams (Kim et al. 2020; Lim et al. 2021). In contrast to hierarchical rectangular partitions, Voronoi trees are much more adaptive to the spatial locations of sampled actions, since the geometries of the cells are induced by the sampled actions. Furthermore, in contrast to Voronoi diagrams, the hierarchical structure of Voronoi trees allows us to derive efficient algorithms for estimating the diameters of the cells (discussed in Section 5.2) and sampling new actions from a cell (detailed in Section 5.3). For Voronoi diagrams, the diameter computation can be prohibitively expensive in the context of online POMDP planning, since each sampled action results in a re-partitioning of the action space; thus, the diameters of the cells have to be re-computed from scratch. Voronoi trees combine a hierarchical representation of the partition with an implicit construction of the cells via sampled actions, thereby achieving both adaptivity and computational efficiency.

The next section describes how ADVT decides which node of $\mathcal{H}(b)$ to expand.

5.1. Refining the Partition

ADVT decides how to refine the partitioning $\mathcal{H}(b)$ in two steps. First, it selects a leaf node of $\mathcal{H}(b)$ to be refined next. This step relies on the action selection strategy used for expanding the belief tree $\mathcal{T}$ (Section 4.1). The selected leaf node (a, P) of $\mathcal{H}(b)$ is the unique leaf node with a chosen according to equation (6)

In the second step, ADVT decides if the cell P should indeed be refined. This decision is based on the quality of the estimate $\widehat{Q}(b, a)$, as reflected by the number of samples used to estimate $\widehat{Q}(b, a)$, and the variation of the Q-values for the actions contained in P, as reflected by the diameter of P. Specifically, ADVT refines the cell P only when the following criteria is satisfied

$$C_r N(b, a) \geq \frac{1}{\mathrm{diam}(P)^2}, \tag{10}$$

where C_r is an exploration constant. $N(b, a)$, that is, the number of times that a has been selected at b, provides a rough estimate on the quality of the $\widehat{Q}(b, a)$ estimate, whereas diam(P) serves as a measure of variation of the Q-value function within P due to the Lipschitz assumption. This criterion, which is inspired by Touati et al. (2020), limits the growth of the finite set of candidate actions $\mathcal{A}(b)$ and ensures that a cell is only refined when its corresponding action has been played sufficiently often. Larger C_r cause cells to be refined earlier, thereby encouraging exploration.

Our refinement strategy is highly adaptive, in the sense that we use local information (i.e., the diameters of the cells, induced by the representative actions and the quality of the Q-value estimates of the representative actions), to influence the choice of the cell to be partitioned and when the chosen cell is partitioned, and the geometries of our cells are dependent on the sampled actions. This strategy is in contrast to other hierarchical decompositions, such as those used in HOO and HOOT, where the cell that corresponds to an action is refined immediately after the action is selected for the first time, which generally means the Q-value of an action is estimated based only on a single play of the action, which is grossly insufficient for our problem. In addition, our strategy is more adaptive than VOMCPOW. For VOMCPOW, the decision on when to refine the partition is solely based on the number of times a belief has been visited and neither takes the quality of the Q-value estimates, nor the diameters of the Voronoi cells into account. Furthermore, VOMCPOW's refinement strategy is more global in a sense that each sampled action results in a different Voronoi diagram of the action space. On the other hand, ADVT ′s strategy is much more local, since each refinement only affects a single cell.

5.2. Estimating the Voronoi cell diameters

ADVT uses the diameters of the cells in the action selection strategy and the cell refinement rule, but efficiently computing the diameters of the cells is computationally challenging in high-dimensional spaces. We give an efficient approximation algorithm for computing the Voronoi cell diameters below.

Since the cells in $\mathcal{H}(b)$ are only implicitly defined, we use a sampling-based approach to approximate a cell's diameter. Suppose we want to estimate the diameter of the cell P corresponding to the node (a, P) of the Voronoi tree $\mathcal{H}(b)$. To do this, we first sample a set of k boundary points $\mathcal{A}_P(b)$ of P, where k is a user defined parameter. In our

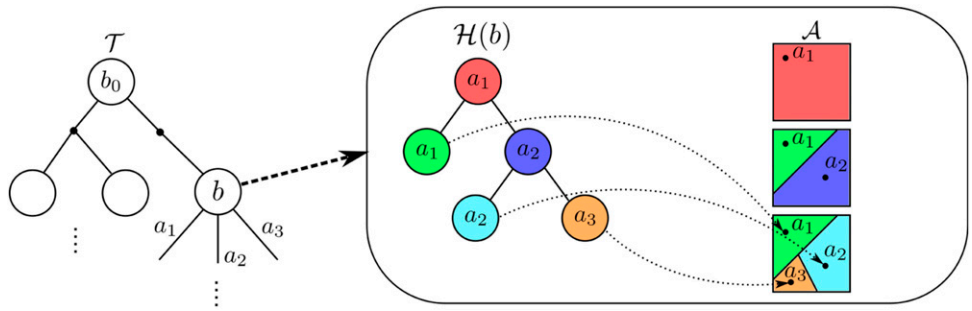

Figure 1. Illustration of the relation between a belief tree $\mathcal{T}$ (left), the Voronoi tree $\mathcal{H}(b)$ associated to belief b (middle) and the partition of the action space induced by the Voronoi tree (right).

experiments, we typically set k to be between 20 and 50. To sample a boundary point $a_P \in \mathcal{A}_P(b)$, we first sample a point α that lies on the sphere centered at a with diameter diam($\mathcal{A}$)—which can be easily computed for our benchmark problems—uniformly at random. The point α lies either on the boundary or outside of P. We then use the Bisection method (Burden et al. 2016) with a and α as the initial end-points, until the two end-points are less than a small threshold ϵ away from each other, but one still lies inside P and the other outside P. The point that lies inside P is then chosen as a boundary point a_P. The diameter of the minimal bounding sphere that encloses all the sampled boundary points in $\mathcal{A}_P(b)$ (Welzl 1991) is then an approximation of the diameter of P.

The above strategy to estimate the diameter of a cell requires us to determine whether a sampled action a lies inside or outside a cell P. Fortunately, a Voronoi tree allows us to determine if $a \in P$ easily. Observe that for each cell P, we have that $P \subseteq$PARENT(P), where PARENT(P) is the cell associated to the parent node of Pin the Voronoi tree. As a result, $a \in P$ implies that $a \in$ PARENT(P). Therefore, to check whether $a \in P$, it is sufficient to check if a is contained in all cells associated to the path in the Voronoi tree from the root to P. Suppose $\zeta = ((a_0, P_0), (a_1, P_1), \ldots, (a_L, P_L))$ is the sequence of nodes corresponding to a path in $\mathcal{H}(b)$ and we want to check whether $a \in P_L$. The point a is inside P_L if, for each $(a_i, P_i) \in \zeta$, we have that $d(a, a_i) \leq d(a, a_i^{'})$, where $a_i^{'}$ is the inducing point of the cell corresponding to the sibling node of (a_i, P_i), and $d(\cdot, \cdot)$ is the distance function on the action space. If this condition is not satisfied, a is outside P_L, that is, $a \notin P_L$.

To further increase the computational efficiency of our diameter estimator, we re-use the sampled boundary points $\mathcal{A}_P(b)$ when ADVT decides to split Pinto two child cells P' and P''. Suppose the diameter of P was estimated using k boundary points. Since every point in $\mathcal{A}_P(b)$ lies either on the boundary of P' or P'', we divide $\mathcal{A}_P(b)$ into $\mathcal{A}_{P'}(b)$ and $\mathcal{A}_{P''}(b)$, such that $\mathcal{A}_{P'}(b) = \{a \in \mathcal{A}_P(b) \mid d(a, a^{'}) \leq d(a, a^{''})\}$ and $\mathcal{A}_{P''}(b) = \mathcal{A}_P(b) \backslash \mathcal{A}_{P'}(b)$, where a' and a'' are the inducing points of P' and P'', respectively. This will leave us with $|\mathcal{A}_{P'}(b)| \leq k$ and $|\mathcal{A}_{P''}(b)| \leq k$ boundary points for cell P' and P'', respectively. For both $\mathcal{A}_{P'}(b)$ and $\mathcal{A}_{P''}(b)$, we then sample $k - |\mathcal{A}_{P'}(b)|$ and $k - |\mathcal{A}_{P''}(b)|$ additional boundary points using the method above, such that $|\mathcal{A}_{P'}(b)| = k$ and $|\mathcal{A}_{P''}(b)| = k$. Using this method, we only need to sample k new boundary points instead of $2k$ when ADVT decides to split the cell P, leading to increased computational efficiency.

5.3. Sampling from the Voronoi Cells

To sample an action that is approximately uniformly distributed in a cell P, we use a simple Hit & Run approach (Smith 1984) that performs a random walk within P. Suppose P is the cell corresponding to the node (a, P) of the Voronoi tree $\mathcal{H}(b)$. We first sample an action a_P on the boundary of P using the method described in Section 5.2. Subsequently, we take a random step from a in the direction towards a_P, resulting in a new action $a' \in P$. We then use a' as the starting point, and iteratively perform this process for m steps, which gives us a point that is approximately uniformly distributed in P.

6. Experiments and Results

We evaluated ADVT on 5 robotics tasks, formulated as continuous action POMDPs. The following section provides details regarding the tasks, while Table 1 summarizes the state, action and observation spaces for each problem scenario.

6.1. Problem Scenarios

6.1.1. Pushbox. Pushbox is a scalable motion planning problem proposed in Seiler et al. (2015) which is motivated by air hockey. A disk-shaped robot (blue disk in Figure 2(a)) has to push a disk-shape puck (red disk in Figure 2(a)) into a goal region (green circle in Figure 2(a)) by bumping into it, while avoiding any collision of itself and the puck with a boundary region (black region if Figure 2(a)). The robot receives a reward of 1000 when the puck is pushed into the goal region, while it receives a penalty of $-1,000$ if the robot or the puck collides with the boundary region. Additionally, the robot receives a penalty of -10 for every

Table 1. Summary of the State, Action, and Observation Spaces for Each Problem Scenario.

	$\mathcal{S}$	$\mathcal{A}$	$\mathcal{O}$
Pushbox2D	$\mathbb{R}^4$	$[-1,1]^2$	24 observations
Pushbox3D	$\mathbb{R}^6$	$[-1,1]^3$	288 observations
Parking2D	$\mathbb{R}^4$	$[-\pi/2, \pi/2] \times [-8, 8]$	4 observations
Parking3D	$\mathbb{R}^5$	$[-\pi/2, \pi/2] \times [-8, 8] \times [-7/2, 7/2]$	4 observations
SensorPlacement-D	$\mathbb{R}^D$	$\mathbb{R}^D$	4 observations
VDP-Tag	$\mathbb{R}^4$	$[0, 2\pi) \times \{0, 1\}$	$\mathbb{R}^8$
LunarLander	$\mathbb{R}^6$	$\mathbb{R}^+ \times \mathbb{R}$	$\mathbb{R}^3$

step. The robot can move freely in the environment by choosing a displacement vector. Upon bumping into the puck, the puck is pushed away and the motion of the puck is affected by noise. We consider two variants of the problem, **Pushbox2D** and **Pushbox3D** that differ in the dimensionality of the state and action spaces. For the **Pushbox2D** problem (illustrated in Figure 2(a)), the robot and the puck operate on a 2D-plane, whereas for **Pushbox3D**, both the robot and the puck operate inside a 3D environment. Let us first describe the **Pushbox2D** variant. The state space consists of the *xy*-locations of both the robot and the puck, that is, $\mathcal{S} = \mathbb{R}^4$, while the action space is defined by $\mathcal{A} = [-1, 1] \times [-1, 1]$. If the robot is not in contact with the puck during a move, the state evolves deterministically according to $f(s, a) = (x_r + a_x, y_r + a_y, x_p, y_p)^T$, where (x_r, y_r) and (x_p, y_p) are the *xy*-coordinates the robot and the puck, respectively, corresponding to state s, and (a_x, a_y) is the displacement vector corresponding to action a. In particular, if the robot bumps into the puck, the next position (x_p', y_p') of the puck is computed as

$$\begin{pmatrix} x_p' \\ y_p' \end{pmatrix} = \begin{pmatrix} x_p \\ y_p \end{pmatrix} + 5r_s \left(\begin{pmatrix} a_x \\ a_x \end{pmatrix} \cdot \vec{n} \right) \left(\vec{n} + \begin{pmatrix} r_x \\ r_y \end{pmatrix} \right), \tag{11}$$

where the " $\cdot$ " operator denotes the dot product, $\vec{n}$ is the unit directional vector from the center of the robot to the center of the puck at the time of contact, and r_s is a random variable drawn from a truncated Gaussian distribution $N(\mu, \sigma^2, l, u)$, which is the Gaussian distribution $N(\mu, \sigma^2)$ truncated to the interval $[l, u]$. For our experiments, we used $\mu = 1.0$, $\sigma = 0.1$, $l = \mu - \sigma = 0.9$ and $u = \mu + \sigma = 1.1$. The variables r_x and r_y are random variables drawn from a truncated Gaussian distribution $N(0.0, 0.1^2, -0.1, 0.1)$.

The initial position of the robot is known and is set to $x_r = 5.5$ and $y_r = 9.5$, respectively. The initial puck position, however, is uncertain. Its initial x_p and y_p coordinates are drawn from a truncated Gaussian distribution $N(5.5, 2.0^2, 3.5, 7.5)$, but the robot has access to a noisy bearing sensor to localize the puck and a noise-free collision sensor which detects contacts between the robot and the puck. In particular, given a state $s \in \mathcal{S}$, an observation (o_c, o_b) consists of a binary component o_c which indicates whether or not a contact between the robot and the puck occurred, and a discretized bearing component o_b calculated as

$$o_b = \text{floor}\left(\frac{\text{atan2}(y_o - y_r, x_o - x_r) + r_o}{\pi/6} \right), \tag{12}$$

where x_r, y_r and x_o, y_o are the *xy*-coordinates of the robot and the puck corresponding to the state s, and r_o is a random angle (expressed in radians) drawn from a truncated Gaussian distribution $N(0.0, (\pi/18)^2, -\pi/18, \pi/18)$. Due to the floor operator in equation (12), the number of discretized bearing observation is 12, thus the observation space consists of 24 unique observations.

Pushbox3D is a straightforward extension of the **Pushbox2D** problem: The state space is defined as $\mathcal{S} = \mathbb{R}^6$, consisting of the *xyz*-locations of the robot and the puck. The action space is $\mathcal{A} = [-1, 1] \times [-1, 1] \times [-1, 1]$, where each $a \in \mathcal{A}$ describes a 3D-displacement vector of the robot. The transition dynamics are defined similarly to **Pushbox2D**, except that all quantities are computed in 3D. The observation space is extended with an additional bearing observation, computed according to equation (12), but with the term $\text{atan2}(y_o - y_r, x_o - x_r)$ being replaced by $\text{atan2}(z_o - z_r, y_o - y_r)$, where z_r and z_o are the *z*-coordinates of the robot and the puck, respectively. The reward function is the same as in the **Pushbox2D** problem.

For both variants of the problem the discount factor is $\gamma = 0.95$ and a run is considered successful if the robot manages to push the puck into a goal region within 50 steps, while avoiding collisions of itself and the puck with the boundary region.

6.1.2. Parking. An autonomous vehicle with deterministic dynamics operates in a 3D environment populated by obstacles, shown in Figure 2(b). The goal of the vehicle is to safely navigate to a goal area located between the obstacles while avoiding any collision with the obstacles (black regions in Figure 2(b)). The vehicle receives a reward of 100 when reaching the goal area, while a collisions with the obstaces are penalized by -100. Additionally, the vehicle receives a penalty of -1 for every step. We consider two variants of the problem, **Parking2D** and **Parking3D**. For **Parking2D**, the vehicle navigates on a 2D-plane, whereas for **Parking3D**, the vehicle operates in 3D space. We first describe the **Parking2D** variant: The state space is $\mathcal{S} = \mathbb{R}^4$ and consists of the *xy*-position of the vehicle on the plane, its orientation θ and its velocity v. The vehicle is controlled via a steering wheel angle a_θ and acceleration a_v, that is, the

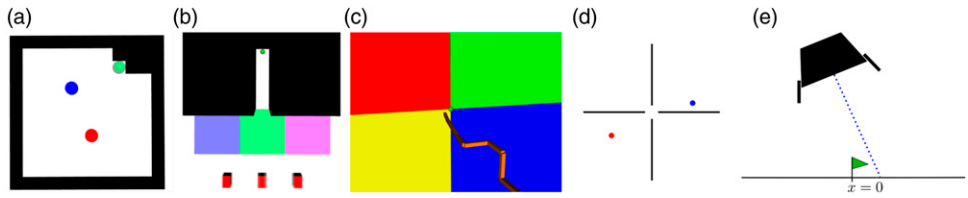

Figure 2. Illustrations of (a) the Pushbox2D, (b) the Parking2D, (c) the SensorPlacement-8, (d) the VDP-Tag, and (e) the LunarLander problems. Goal regions in are marked as green circles or a green flag.

action space is $\mathcal{A} = \Omega \times \Phi$, where $\Omega = [-\pi/2, \pi/2]$ is the continuous set of steering wheel angles and $\Phi = [-8, 8]$ is the continuous set of accelerations. We assume that for a given state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, the state of the vehicle evolves acoording to the following deterministic second-order discrete-time dynamics model

$$f(x, y, \theta, v, a_\theta, a_v) = \begin{bmatrix} x + v\cos(\theta)\Delta \\ y + v\sin(\theta)\Delta \\ \theta + a_\theta\Delta \\ v + a_v\Delta \end{bmatrix}, \tag{13}$$

where x, y, θ and v are the 2D-position, orientation, and velocity corresponding to state s, $\Delta = 1/3s$ is a fixed control duration, and a_θ, a_v are the steering wheel angle and acceleration components of the action. To compute the next state s', given s and a, we numerically integrate equation (13) for 3 steps.

There are three distinct areas in the environment, each consisting of a different type of terrain (colored areas in Figure 2(b)). Upon traversal, the vehicle receives an observation regarding the terrain type, which is only correct 70% of the time due to sensor noise. If the vehicle is outside the terrains, we assume that it deterministically receives a NULL observation. Initially the vehicle starts near one of three possible starting locations (red areas in Figure 2(b)) with equal probability. The exact initial position of the vehicle along the horizontal y-axis is then drawn uniformly from $U[-0.175, 0.175]$ around the starting location. For Parking3D, the vehicle operates in the full 3D space, and we have additional continuous state and action components that model the vehicles elevation and change in elevation, respectively. We assume that the elevation of the vehicle changes according to $z + \Delta a_h$, where $z \in \mathbb{R}$ is the elevation component of the state, and $a_h \in [-7/2, 7/2]$ is the elevation-change component of the action. The discount factor for both variants is $\gamma = 0.95$ and a run is considered successful if the vehicle enters the goal area within 50 steps while avoiding collisions with the obstacles.

Two properties make this problem challenging. First is the multi-modal beliefs which require the vehicle to traverse the different terrains for a sufficient amount of time to localize itself before attempting to reach the goal. Second, due to the narrow passage that leads to the goal, small perturbations from the optimal action can quickly result in collisions with the obstacle. As a consequence, good rewards are sparse and a POMDP solver must discover them quickly in order to compute a near-optimal strategy.

6.1.3. SensorPlacement. We propose a scalable motion planning under uncertainty problem in which a D-DOF manipulator with D revolute joints operates in muddy water inside a 3D environment. The robot is located in front of a marine structure, represented as four distinct walls, and its task is to mount a sensor at a particular goal area between the walls (rewarded with 1000) while having imperfect information regarding its exact joint configuration. To localize itself, the robot's end-effector is equipped with a touch sensor. Upon touching a wall, it provides noise-free information regarding which wall is being touched, while we assume that the sensor deterministically outputs a NULL observation in a contact-free state. However, in order to avoid damage, the robot must avoid collisions (penalized by -500) between any of its other links and the walls. The state space of the robot consists of the set of joint angles for each joint. The action space is $\mathcal{A} \subset \mathbb{R}^D$, where $a \in \mathcal{A}$ is a vector of joint velocities. Due to underwater currents, the robot is subject to random control errors and the joint angles corresponding to a state evolve according to

$$\theta' = \theta + a + r, \tag{14}$$

where θ is the set of joint angles corresponding to the current state, and r is a random vector sampled from a multivariate Gaussian distribution $N(0, \sigma^2 I)$, where I is the identity matrix of size D, and $\sigma^2 = 10^{-3}$. Initially the robot is uncertain regarding its exact joint angle configuration. We assume that the initial joint angles are distributed uniformly according to $U[\theta_l, \theta_u]$, where $\theta_l = \theta_0 - h$ and $\theta_u = \theta_0 + h$, with θ_0 corresponding to the configuration where all joint angles are zero, except for the second and third joint whose joint angles are -1.57 and 1.57, respectively, and $h = (0.1, \ldots, 0.1) \in \mathbb{R}^D$ (units are in radians). We consider four variants of the problem, denoted as SensorPlacement-D, with $D \in \{6, 8, 10, 12\}$, that differ in the degrees-of-freedom (number of revolute joints and thus the dimensionality of the action space) of the manipulator. Figure 2(c) illustrates the SensorPlacement-8 problem, where the colored areas represent the walls and the green sphere represents the goal area. The discount factor is $\gamma = 0.95$ and a run is considered successful if the manipulator mounts the sensor within 50 steps while avoiding collisions with the walls. To successfully mount the sensor at the target location, the robot must use its touch sensor to carefully reduce uncertainty regarding its configuration. This is challenging, since a slight variation in the performed actions can quickly lead to collisions with the walls.

6.1.4. Van Der Pol Tag. Van Der Pol Tag (VDP-Tag) is a benchmark problem introduced in Sunberg and Kochenderfer (2018) in which an agent (blue particle in Figure 2(d)) operates in a 2D-environment. The goal is to tag a moving target (red particle in Figure 2(d)) whose motion is described by the Van Der Pol differential equation and disturbed by Gaussian noise with standard deviation $\sigma = 0.05$. Initially, the position of the target is unknown. The agent travels at a constant speed but can pick its direction of travel at each step and whether to activate a costly range sensor, that is, the action space is $\mathcal{A} = [0, 2\pi) \times \{0, 1\}$, where the first component is the direction of travel and the second component is the activation/deactivation of the range sensor. The robot receives observations from its range sensor via 8 beams (i.e., $\mathcal{O} = \mathbb{R}^8$) that measure the agent's distance to the target if the target is within the beam's range. These measurements are more accurate when the range sensor is active. While the target moves freely in the environment, the agent's movements are restricted by four obstacles in the environment, shown in Figure 2(d). Catching the target is reward by 100, while activating the range sensor is penalized by -5. Additionally, each step incurs a penalty of -1. The discount factor is $\gamma = 0.95$ and a run is considered successful of the agent catches the target within 50 steps. More details of the VDP-Tag problem can be found in Sunberg and Kochenderfer (2018).

Note that in this problem, the action space consists of two disconnected components, namely, $[0, 2\pi) \times \{0\}$ and $[0, 2\pi) \times \{1\}$. To allow a Voronoi tree $\mathcal{H}(b)$ in ADVT to cover the entire action space, we ensure that once the root node of $\mathcal{H}(b)$ is split, we set the range sensor component of the representative actions of the two resulting child cells to 0 and 1, respectively, such that one child cell covers the component $[0, 2\pi) \times \{0\}$, and the other child cell covers the component $[0, 2\pi) \times \{1\}$.

6.1.5. LunarLander. This problem is a partially observable adaptation of Atari's Lunar Lander game. Here, the state, action and observation spaces are all continuous. The objective is to control a lander vehicle to safely land at a target zone located on the moon's surface. The lander operates on a xy-plane and its state is a 6D-vector $(x, y, \theta, \dot{x}, \dot{y}, \dot{\theta})$, where $x \in \mathbb{R}$ and $y \in \mathbb{R}$ are the horizontal and vertical positions of the lander and $\theta \in \mathbb{R}$ its orientation. $\dot{x} \in \mathbb{R}$, $\dot{y} \in \mathbb{R}$ and $\dot{\theta} \in \mathbb{R}$ represent the lander's horizontal, vertical, and angular velocities, respectively. The action space is $\mathcal{A} = \Lambda \times \Psi$, where $\Lambda = \mathbb{R}^+$ is the set of linear accelerations along the lander's vertical axis, and $\Psi = \mathbb{R}$ is the set of angular accelerations about the lander's geometric center. The initial belief of the lander is a multivariate Gaussian distribution with mean $\mu = (0,10,0,0,-10,0)^T$ and covariance matrix $\Sigma = \text{diag}(1.5^2, 1.0^2, 0.1^2, 0^2, 0.5^2, 0.1^2)$.

We assume that the state of the lander evolves according to the following second-order discrete-time stochastic model

$$f(x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}, \tilde{\lambda}, \tilde{\psi}) = \begin{bmatrix} x + \dot{x}\Delta \\ y + \dot{y}\Delta \\ \theta + \dot{\theta}\Delta \\ \dot{x} + (-\tilde{\lambda}\sin(\theta)M)\Delta \\ \dot{y} + (\tilde{\lambda}\cos(\theta)M - G)\Delta \\ \dot{\theta} + H\tilde{\psi}\Delta \end{bmatrix}, \quad (15)$$

where $\tilde{\lambda} = \lambda + e_\lambda$ and $\tilde{\psi} = \psi + e_\psi$, with λ and ψ being the vertical and angular accelerations corresponding to the action, and e_λ and e_ψ are random control errors drawn from zero-mean Gaussian distributions with standard deviations $\sigma_\lambda = 1 \times 10^{-4}$ and $\sigma_\psi = 5 \times 10^{-2}$, respectively. The variable $\Delta = 0.2s$ is a constant step size, whereas $F = 40$ and $H = 2$ are motor constants. The variable $G = -9.81 m/s^2$ is a constant describing the gravitational acceleration along the y-axis. To obtain the next state, given the current state and an action, we numerically integrate the system in equation (15) for 5 steps.

The lander perceives information regarding its state via three noisy sensors: The first two sensors measure the lander's horizontal and angular velocities, whereas the third sensor measures the distance to the ground along the lander's vertical axis (dashed line in Figure 2(e)). The readings of all three sensors are disturbed by standard-Gaussian noise.

The reward function is defined by

$$r_t = \begin{cases} -1000, & \text{if } \theta \geq 0.5 \text{ or } y_t < 0 \\ 100 - |x_t| - |\theta_t| - \dot{y}_t^2, & \text{if } y_t \leq 0.3 \\ -1, & \text{otherwise.} \end{cases} \quad (16)$$

The first term in equation (16) encourages the lander to prevent dangerous angles and crashing into the ground. The second term encourages the lander to safely land at $x = 0$ with an upright angle and a small vertical velocity. The third term encourages the lander to land as quickly as possible. The discount factor is $\gamma = 0.95$ and a run is considered successful if the lander's vertical position reaches $y \leq 0.3$ within 50 steps, without crashing into the ground ($y < 0$).

6.2. Experimental setup

The purpose of our experiments is three-fold: First is to evaluate ADVT and compare it with two state-of-the-art online POMDP solvers for continuous actions spaces, POMCPOW (Sunberg and Kochenderfer 2018) and VOMCPOW (Lim et al. 2021). The results of those experiments are discussed in Section 6.3.1.

Second is to investigate the importance of the different components of ADVT, specifically the Voronoi tree–based partitioning, the cell-diameter-aware exploration term in equation (7), and the stochastic Bellman backups. For this purpose, we implemented the original ADVT and three modifications. First is **ADVT-R**, which replaces the Voronoi decomposition of ADVT with a simple rectangular-based method: Each cell in the partition is a hyper-rectangle that is subdivided by cutting it in the middle along the longest side (with ties broken arbitrarily). The second variant is **ADVT (L=0)**, which is ADVT where equation (7) reduces to the

standard UCB1 bound. For the third variant, **ADVT-MC**, we replace the stochastic Bellman backup in equation (8) with the Monte Carlo backup in equation (9), as used by POMCPOW and VOMCPOW. The results for this study are discussed in Section 6.3.2

Third is to test the effects of two algorithmic components of ADVT when applied to the baselines VOMCPOW and POMCPOW: In the original implementation of VOMCPOW and POMCPOW provided by the authors, the policy is recomputed after every planning step using a new search tree. In contrast, for discrete observation spaces, ADVT applies ABT's (Kurniawati and Yadav 2013) strategy that re-uses the partial search tree (starting from the updated belief) constructed in the previous planning steps and improves the policy within the partial search tree. Therefore, for problems with discrete observation spaces, we also tested modified versions of VOMCPOW and POMCPOW, where we follow ADVT ′s strategy of re-using the partial search trees. Note that for the VDP-Tag and LunarLander problems, we did not test the variants of VOMCPOW and POMCPOW that re-use partial search trees, since each observation that the agent perceives from the environment leads to a new search tree due to the continuous observation spaces. Moreover, to test the effects of ADVT ′s stochastic Bellman backup strategy further, we implemented variants of VOMCPOW and POMCPOW to use stochastic Bellman backups instead of Monte Carlo backups. The results are discussed in Section 6.3.3.

To approximately determine the best parameters for each solver and problem, we ran a set of systematic preliminary trials by performing a grid-search over the parameter space. For each solver and problem, we used the best parameters and ran 1000 simulation runs, with a fixed planning time of 1s CPU time for each solver and scenario. Each tested solver and the scenarios were implemented in C++ using the OPPT-framework (Hoerger et al. 2018). All simulations were run single-threaded on an Intel Xeon Platinum 8274 CPU with 3.2GHz and 4GB of memory.

6.3. Results

6.3.1. Comparison with State-of-the-Art Methods. Table 2 shows the average total discounted rewards of all tested solvers on the Pushbox, Parking, VDP-Tag and LunarLander problems, while Figure 3 shows the results for the SensorPlacement problems. Detailed results for the SensorPlacement problems are presented in Table 3 while results on the success rates of the tested solvers are shown in Table 4 and Table 5.

ADVT generally outperforms all other methods, except for VDP-Tag, where VOMCPOW performs better. Interestingly, as we will see in Section 6.3.2, the variant of ADVT that uses Monte Carlo backups instead of stochastic Bellman backups (ADVT-MC) outperforms all other methods in this problem, which supports ADVT ′s effectiveness in handling continuous actions. The results for the SensorPlacement problems indicate that ADVT scales well to higher-dimensional action spaces. Additionally, the results on the VDP-Tag and LunarLander problems indicate that ADVT is capable of handling continuous observation spaces well.

ADVT performs well in terms of the success rate, too (Table 4 and Table 5). ADVT maintains more than 90% success rate in the Pushbox, Parking, VDP-Tag and LunarLander problems. In the Parking3D problem, VOMCPOW's and POMCPOW's success rate can be as low as $\sim 30\%$ and 12.5%. Similarly, in the SensorPlacement problems ADVT achieves a higher success rate compared to VOMCPOW and POMCPOW. While in the SensorPlacement-6 problem, the gap between ADVT and VOMCPOW is relatively small ($\sim 98\%$ for ADVT and $\sim 92\%$ for VOMCPOW), the gap increases as the dimensionality of the action space increases. In the SensorPlacement-12 problem, ADVT achieves a success rate of $> 72\%$, while VOMCPOW's success rate drops to $< 60\%$. POMCPOW achieves a success rate of $< 74\%$ in the SensorPlacement-6 problem, while only achieving a success rate of $< 32\%$ in the SensorPlacement-12 problem.

6.3.2. Understanding the Effects of Different Components of ADVT. In this section we present results demonstrating the importance of three key algorithmic components of ADVT, namely the Voronoi-based partitions, the cell-size-aware optimistic upper-confidence bound and the Stochastic Bellman backups.

Effects of Voronoi-based partitioning for ADVT. To understand the benefit of our Voronoi-based partitioning method, we compare the results of ADVT with those of ADVT-R. Table 6 shows that ADVT-R slightly outperforms ADVT in the Pushbox2D, Parking2D, VDP-Tag and LunarLander problems, indicating that a rectangular-based partitioning works well for low-dimensional action spaces. However, Table 3 shows that ADVT-R is uncompetitive in the SensorPlacement problems as the dimensionality of the action space increases. For rectangular-based partitionings, the diameters of the cells can shrink very slowly in higher-dimensional action spaces. Additionally, the cell refinement method is

Table 2. Average Total Discounted Rewards and 95% Confidence Intervals of ADVT, VOMCPOW, and POMCPOW on the Pushbox, Parking, VDP-Tag, and LunarLander Problems. The Average is Taken Over 1000 Simulation Runs per Solver and Problem, With a Planning Time of 1s per Step.

	Pushbox2D	Pushbox3D	Parking2D	Parking3D	VDP-Tag	LunarLander
ADVT	**351.6 ± 10.0**	**322.1 ± 14.9**	**35.2 ± 1.9**	**32.6 ± 3.5**	30.5 ± 1.0	**26.01 ± 1.2**
VOMCPOW	129.8 ± 13.3	73.5 ± 13.8	−0.78 ± 2.8	−18.4 ± 1.4	**32.9 ± 0.9**	13.3 ± 2.2
POMCPOW	82.1 ± 14.2	3.6 ± 12.9	−5.1 ± 3.0	−25.7 ± 1.4	28.2 ± 1.1	13.5 ± 2.1

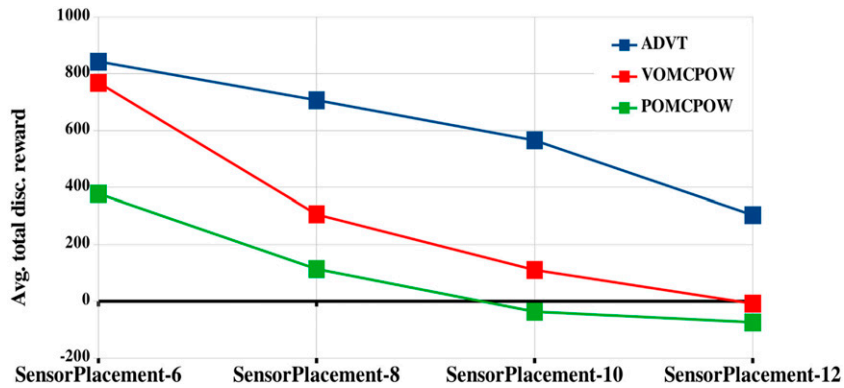

Figure 3. Average total discounted rewards of all tested solvers on the SensorPlacement problems. The average is taken over 1000 simulation runs per solver and problem.

Table 3. Average Total Discounted Rewards and 95% Confidence Intervals of all Tested Solvers on the SensorPlacement Problems. The average is Taken Over 1000 Simulation Runs per Solver and Problem, With a Planning Time of 1s per Step.

	SensorPlacement-6	SensorPlacement-8	SensorPlacement-10	SensorPlacement-12
ADVT	**842.8 ± 9.5**	**706.8 ± 17.5**	**565.1 ± 21.7**	**303.0 ± 19.8**
ADVT-R	676.3 ± 19.7	238.1 ± 33.5	28.7 ± 18.4	−17.3 ± 7.4
ADVT (L = 0)	780.4 ± 12.6	448.8 ± 15.9	325.3 ± 16.2	102.5 ± 7.4
ADVT-MC	812.6 ± 11.4	692.7 ± 17.7	551.2 ± 18.3	293.5 ± 19.6
VOMCPOW	768.5 ± 16.4	305.6 ± 25.8	110.1 ± 24.5	−8.2 ± 13.2
VOMCPOW+our RT+our BB	823.4 ± 15.1	679.1 ± 17.9	481.6 ± 22.2	191.3 ± 17.6
VOMCPOW+our BB	779.2 ± 16.1	654.9 ± 16.3	453.6 ± 22.8	182.4 ± 17.7
VOMCPOW+our RT	817.2 ± 15.8	663.4 ± 18.6	476.0 ± 22.7	189.9 ± 18.0
POMCPOW	377.6 ± 23.5	113.4 ± 24.2	−36.8 ± 11.3	−74.3 ± 12.9
POMCPOW+our RT+our BB	659.3 ± 17.2	428.7 ± 21.5	114.6 ± 16.6	−1.9 ± 6.6
POMCPOW+our BB	562.5 ± 18.3	408.7 ± 19.1	102.6 ± 14.4	−6.5 ± 7.1
POMCPOW+our RT	653.2 ± 17.3	425.2 ± 21.8	111.3 ± 16.8	−2.1 ± 6.8

Table 4. Success Rates of all Tested Solvers on the Pushbox, Parking, VDP-Tag and LunarLander Problems. The Success Rate Is with Respect to 1000 Simulation per Solver and Problem, With a Planning Time of 1s per Step.

	Pushbox2D	Pushbox3D	Parking2D	Parking3D	VDP-Tag	LunarLander
ADVT	0.985	0.969	0.912	0.916	0.985	0.969
ADVT-R	0.987	0.968	0.943	0.906	0.984	0.970
ADVT (L=0)	0.966	0.965	0.857	0.898	0.980	0.931
ADVT-MC	0.989	0.972	0.417	0.337	0.991	0.957
VOMCPOW	0.754	0.815	0.512	0.297	0.987	0.859
VOMCPOW+our RT+our BB	0.985	0.970	0.885	0.886	—	
VOMCPOW+our BB	0.976	0.893	0.882	0.885	0.986	0.958
VOMCPOW+our RT	0.975	0.939	0.597	0.314	—	
POMCPOW	0.712	0.692	0.401	0.125	0.979	0.876
POMCPOW+our RT+our BB	0.974	0.953	0.853	0.534	—	
POMCPOW+our BB	0.970	0.891	0.793	0.246	0.976	0.943
POMCPOW+our RT	0.963	0.969	0.409	0.122	—	

independent of the spatial locations of the sampled actions. Both properties result in loose optimistic upper-confidence bounds of the Q-values, leading to excessive exploration of suboptimal areas of the action space. For Voronoi trees, the geometries (and therefore the diameters) of the cells are much more adaptive to the spatial locations of the sampled actions, leading to more accurate optimistic upper-confidence bounds of the associated Q-values which avoids over-exploration of areas in the action space that already contain sufficiently many sampled actions.

Effects of cell-size-aware optimistic upper-confidence bound. To investigate the importance of the component L diam(P) in the optimistic upper-confidence bound in equation (7), we compare ADVT and ADVT (L=0). The results in Table 6 and Table 3 indicate that the cell-diameter-aware component in the upper-confidence bound in equation (7) is important for ADVT to perform well, particularly in the SensorPlacement problems. The reason is that in the early stages of planning, the partitions associated to the beliefs are still coarse, i.e., only a few candidate actions are considered per belief. If some of those candidate actions have small estimated Q-values, ADVT (L=0) may discard large portions of the action space for a very long time, even if they potentially contain useful actions. The cell-diameter-aware bias term in equation (7) alleviates this issue by encouraging ADVT to explore cells with large diameters. This is particularly important for problems with large action spaces such as the SensorPlacement problems.

Effects of Stochastic Bellman backups. To investigate the effects of this component, let us compare ADVT with ADVT-MC. Table 6 and Table 3 reveal that ADVT which uses stochastic Bellman backups often performs significantly better, particularly in the Parking2D and Parking3D problems. The reason is that in both problems good rewards are sparse, particularly for beliefs where the vehicle is located between the walls and slight deviations from the optimal actions can lead to collisions. The stochastic Bellman backups help to focus the search on more promising regions of the action space. On the other hand, in the VDP-Tag problem, ADVT-MC performs better than ADVT. In this problem, it is important to reduce the uncertainty with respect to the target location during the first few steps by activating the range sensor. However, due to the cost of activating the range sensor, this strategy often seems suboptimal during the early stages of planning, when only a few episodes have been sampled. At the same time, in this problem, the stochastic Bellman backups in ADVT tend to overestimate the Q-values for actions that do not activate the range sensor (this effect is known as *maximization bias* in Q-learning (Sutton and Barto 2018)). As a result, ADVT tends to discard strategies that reduce the uncertainty with respect to the target location during the first few steps. ADVT-MC which uses Monte Carlo backups does not suffer from maximization bias, which helps it to perform better than ADVT in this problem.

Table 5. Success Rates of all Tested Solvers on the SensorPlacement Problems. The Success Rate is with Respect to 1000 Simulation per Solver and Problem, With a Planning Time of 1s per Step.

	SensorPlacement-6	SensorPlacement-8	SensorPlacement-10	SensorPlacement-12
ADVT	0.981	0.962	0.834	0.724
ADVT-R	0.832	0.692	0.756	0.557
ADVT (L = 0)	0.937	0.726	0.791	0.601
ADVT-MC	0.964	0.959	0.828	0.719
VOMCPOW	0.923	0.721	0.645	0.583
VOMCPOW+our RT+our BB	0.979	0.951	0.807	0.703
VOMCPOW+our BB	0.924	0.883	0.798	0.702
VOMCPOW+our RT	0.967	0.891	0.803	0.698
POMCPOW	0.738	0.636	0.519	0.321
POMCPOW+our RT+our BB	0.829	0.794	0.646	0.575
POMCPOW+our BB	0.794	0.779	0.641	0.563
POMCPOW+our RT	0.826	0.781	0.657	0.578

Table 6. Average Total Discounted Rewards and 95% Confidence Intervals of all Tested Solvers on the Pushbox, Parking, VDP-Tag and LunarLander Problems. The average is Taken Over 1000 Simulation Runs per Solver and Problem, With a Planning Time of 1s per Step.

	Pushbox2D	Pushbox3D	Parking2D	Parking3D	VDP-Tag	LunarLander
ADVT-R	371.4 ± 9.8	321.2 ± 15.1	38.9 ± 1.8	24.3 ± 3.4	30.2 ± 1.0	29.6 ± 1.1
ADVT (L=0)	340.8 ± 14.7	294.6 ± 13.3	29.2 ± 3.5	18.6 ± 1.7	28.7 ± 1.1	24.7 ± 1.2
ADVT-MC	319.6 ± 13.7	311.0 ± 16.2	−3.2 ± 1.8	−14.7 ± 0.5	33.5 ± 0.8	21.9 ± 1.7
VOMCPOW+our RT+our BB	322.9 ± 12.1	274.9 ± 14.2	28.2 ± 1.8	24.4 ± 2.4	-	-
VOMCPOW+our BB	316.3 ± 13.6	134.2 ± 17.4	27.5 ± 1.9	23.7 ± 2.5	29.9 ± 1.0	19.9 ± 1.6
VOMCPOW+our RT	316.0 ± 12.3	268.9 ± 14.2	−0.42 ± 2.8	−15.7 ± 1.5	-	-
POMCPOW+our RT+our BB	314.2 ± 13.0	245.7 ± 14.1	27.7 ± 1.8	8.8 ± 2.6	-	-
POMCPOW+our BB	300.6 ± 12.6	128.8 ± 17.5	24.2 ± 1.9	−10.4 ± 2.1	27.5 ± 1.2	18.4 ± 2.2
POMCPOW+our RT	270.6 ± 18.9	203.7 ± 14.3	−5.2 ± 2.9	−22.8 ± 1.3	-	-

6.3.3. Ablation study. Here we investigate how two components of ADVT, namely, re-using partial search trees and using stochastic Bellman backups instead of Monte Carlo backups, affect the baselines VOMCPOW and POMCPOW when we apply these ideas to the baselines. The variants of VOMCPOW and POMCPOW that re-use partial search trees are denoted by VOMCPOW+RT and POMCPOW+RT, respectively. The variants of the baselines that use stochastic Bellman backups are denoted by VOMCPOW+BB and POMCPOW+BB. The variants VOMCPOW+RT+BB and POMCPOW+RT+BB both re-use partial search trees and perform stochastic Bellman backups.

Generally, the results in Table 6 and Table 3 indicate that the baselines that re-use partial search trees perform much better than the baselines VOMCPOW and POMCPOW, respectively, particularly in the Pushbox problems. These results (as well as those of ADVT and all its variants) indicate the benefit of re-using partial search trees that were generated in previous planning steps instead of re-computing the policy at every planning step. Similarly, the baselines that use stochastic Bellman backups tend to outperform the ones that use Monte Carlo backups, except for the VDP-Tag problem. This is consistent with our results for ADVT and ADVT-MC and indicates that stochastic Bellman backup is a simple, yet viable tool to improve the performance of MCTS-based solvers.

7. Conclusion

We propose a new sampling-based online POMDP solver, called ADVT, that scales well to POMDPs with high-dimensional continuous action and observation spaces. Our solver builds on a number of works that uses adaptive discretization of the action space, and introduces a more effective adaptive discretization method that uses novel ideas: A Voronoi tree–based adaptive hierarchical discretization of the action space, a novel cell-size-aware refinement rule, and a cell-size-aware upper-confidence bound. For continuous observation spaces, our solver adopts the Progressive Widening and explicit belief representation strategy, enabling ADVT to scale to higher-dimensional observation spaces. ADVT shows strong empirical results against state-of-the-art algorithms on several challenging benchmarks. We hope this work further expands the applicability of general-purpose POMDP solvers.

Acknowledgements

We thank Jerzy Filar for many helpful discussions.

Declaration of conflicting interests

The author(s) declared the following potential conflicts of interest with respect to the research, authorship, and/or publication of this article: The co-authors Hanna Kurniawati and Nan Ye have a potential CoI with IJRR editor David Hsu, since both co-authors worked with David previously in various capacities.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work is partially supported by the Australian Research Council (ARC) Discovery Project 200101049.

ORCID iD

Marcus Hoerger https://orcid.org/0000-0002-4698-5875

References

Agha-Mohammadi AA, Chakravorty S and Amato NM (2011) *Firm: Feedback Controller-Based Information-State Roadmap. A Framework for Motion Planning under Uncertainty.* IROS, pp. 4284–4291.

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, Hsu D and Lee WS (2014) Integrated perception and planning in the continuous space: a POMDP approach. *IJRR* 33(9): 1288–1302.

Bubeck S, Munos R, Stoltz G, et al. (2011) X-armed bandits. *Journal of Machine Learning Research* 12(5): 128.

Burden RL, Faires JD and Burden AM (2016) *Numerical Analysis*. 10th edition. Boston, MA: Cengage Learning.

Couëtoux A, Hoock JB, Sokolovska N, et al. (2011) Continuous upper confidence trees. In *LION*. 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*. ICML. PMLR, pp. 3177–3187.

Hoerger M and Kurniawati H (2021) An on-line POMDP solver for continuous observation spaces. Proc. IEEE/RSJ Int. Conference on Robotics and Automation (ICRA), May, 2021.

Hoerger M, Kurniawati H and Elfes A (2018) *A Software Framework for Planning under Partial Observability.* IROS, pp. 1–9. https://github.com/RDLLab/oppt

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

Hoerger M, Kurniawati H, Kroese D, et al. (2022) Adaptive discretization using voronoi trees for continuous-action pomdps. In: *Proc. Int. Workshop on the Algorithmic Foundations of Robotics*. Cornell University.

Kaelbling LP, Littman ML and Cassandra AR (1998) Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1-2): 99–134.

Kim B, Lee K, Lim S, et al. (2020) Monte Carlo tree search in continuous spaces using Voronoi optimistic optimization with regret bounds. *AAAI* 34(06): 9916–9924.

Klimenko D, Song J and Kurniawati H (2014) *Tapir: A Software Toolkit for Approximating and Adapting POMDP Solutions Online*. ACRA. https://github.com/rdl-algorithm/tapir.

Kurniawati H (2022) Partially observable Markov decision processes and robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 5(1): To appear

Kurniawati H and Yadav V (2013) An online POMDP solver for uncertainty planning in dynamic environment. *Proc. Int. Symp. on Robotics Research*. Online ahead of print.

Kurniawati H, Hsu D and Lee WS (2008) *SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces*. RSS.

Kurniawati H, Du Y, Hsu D, et al. (2011) Motion planning under uncertainty for robotic tasks with long time horizons. *IJRR* 30(3): 308–323.

Lim MH, Tomlin CJ and Sunberg ZN (2021) Voronoi progressive widening: efficient online solvers for continuous state, action, and observation POMDPs. In: 60th IEEE Conference on Decision and Control (CDC), 2021, pp. 4493–4500.

Lindquist A (1973) On feedback control of linear stochastic systems. *SIAM Journal on Control* 11(2): 323–343. DOI: 10.1137/0311025.

Mansley C, Weinstein A and Littman M (2011) *Sample-based Planning for Continuous Action Markov Decision Processes*. ICAPS.

Mern J, Yildiz A, Sunberg Z, et al. (2021) Bayesian optimized Monte Carlo planning. *AAAI* 35: 11880–11887.

Papadimitriou CH and Tsitsiklis JN (1987) The complexity of Markov decision processes. *Mathematics of Operations Research* 12(3): 441–450.

Pineau J, Gordon G and Thrun S (2003) *Point-Based Value Iteration: An Anytime Algorithm for POMDPs*. IJCAI.

Seiler KM, Kurniawati H and Singh SP (2015) *An online and approximate solver for POMDPs with continuous action space*. ICRA, pp. 2290–2297.

Silver D and Veness J (2010) Monte Carlo planning in large POMDPs. *Advances in Neural Information Processing Systems*, pp. 2164–2172.

Smith RL (1984) Efficient Monte Carlo procedures for generating points uniformly distributed over bounded regions. *Operations Research* 32(6): 1296–1308.

Smith T and Simmons R (2005) *Point-Based POMDP Algorithms: Improved Analysis and Implementation*. UAI.

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

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

Sunberg ZN and Kochenderfer MJ (2018) *Online Algorithms for POMDPs with Continuous State, Action, and Observation Spaces*. ICAPS.

Sutton RS and Barto AG (2018) *Reinforcement Learning: An Introduction*. MIT press.

Touati A, Taiga AA and Bellemare MG (2020) *Zooming for Efficient Model-free Reinforcement Learning in Metric Spaces*. *arXiv preprint 2003*.

Valko M, Carpentier A and Munos R (2013) *Stochastic Simultaneous Optimistic Optimization*. ICML, pp. 19–27.

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. *IJRR* 31(11): 1263–1278.

Wang T, Ye W, Geng D, et al. (2020) *Towards practical Lipschitz bandits*. FODS, pp. 129–138.

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

Welzl E (1991) Smallest enclosing disks (balls and ellipsoids). In: H Maurer (ed), *New Results and New Trends in Computer Science*. Berlin: Springer, pp. 359–370.

Ye N, Somani A, Hsu D, et al. (2017) DESPOT: online POMDP planning with regularization. *Journal of Artificial Intelligence Research* 58: 231–266.