Adaptive Discretization using Voronoi Trees for Continuous-Action POMDPs

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

[1] School of Mathematics & Physics, The University of Queensland, Australia
m.hoerger@uq.edu.au, kroese@maths.uq.edu.au, nan.ye@uq.edu.au
[2] School of Computing, Australian National University, Australia
hanna.kurniawati@anu.edu.au

Abstract. Solving Partially Observable Markov Decision Processes (POMDPs) with continuous actions is challenging, particularly for high-dimensional 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 which we call a *Voronoi tree*. A Voronoi tree is a Binary Space Partitioning (BSP) that implicitly maintains the partition of a cell as the Voronoi diagram of two points sampled from the cell. This partitioning strategy keeps the cost of partitioning and estimating the size of each cell low, even in high-dimensional spaces where many sampled points are required to cover the space well. ADVT uses the estimated sizes of the cells to form an upper-confidence bound of the action values of the cell, and in turn uses the upper-confidence bound to guide the Monte Carlo Tree Search expansion and further discretization of the action space. This strategy enables ADVT to better exploit local information in the action space, leading to an action space discretization that is more adaptive, and hence more efficient in computing good POMDP solutions, compared to existing solvers. Experiments on simulations of four types of benchmark problems indicate that ADVT outperforms and scales substantially better to high-dimensional continuous action spaces, compared to state-of-the-art continuous action POMDP solvers.

Keywords: Planning under Uncertainty · Motion Planning · Partially Observable Markov Decision Process

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) [11, 27] 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 [21], the past two decades have seen a surge of sampling-based POMDP solvers (reviewed in [14]) 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 high-dimensional continuous action spaces remain a challenge. Recent solvers for continuous-action POMDPs [7, 20, 23, 29] 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 [2] to guide belief space sampling and Monte Carlo backup for value estimation, but differ in the action subset refinement.

Several approaches use the Progressive Widening strategy [6] to continuously add new randomly sampled actions once current actions have been sufficiently explored. Examples include POMCPOW [29] and IPFT [7]. More recent algorithms combine Progressive Widening with more informed methods for adding new actions: VOMCPOW [18] uses Voronoi Optimistic Optimization [12] and BOMCP [20] uses Bayesian optimization. All of these solvers use UCT-style simulations and Monte Carlo backups. An early line of work, GPS-ABT [23], 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 [9, 13]), 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 online POMDP solver, called **A**daptive **D**iscretization using **V**oronoi **T**rees (ADVT). ADVT 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 and proposes a new action space discretization mechanism 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. This strategy enables ADVT to keep a low computational cost for partitioning and estimating the diameters of the cells, even in high-dimensional action spaces.

ADVT uses the estimated diameters of the cells in a Voronoi tree to guide belief space sampling and Voronoi tree refinement in two ways. First, they help decide if a cell needs further refinement, which in turn helps ADVT to avoid unnecessarily small partitioning of non-promising action space regions. Our refinement rule, together with the hierarchical partitioning, results in a partitioning that is much more adaptive to the spatial locations of the sampled actions, compared to state-of-the-art methods [4, 18, 19, 29, 31]. The second use of the information on the diameters of the cells is in the action sampling strategy used in MCTS. Instead of using UCB1 [2], ADVT adopts a cell-diameter-aware upper-confidence bound [34], which uses the diameter of a cell to estimate the upper-confidence bound of the value of *all* actions within the cell. The above strategies imply that ADVT uses local information to help construct a highly adaptive discretization of the continuous action space and guide the search for the optimal action. Finally, ADVT applies stochastic Bellman backups, rather than the typical Monte Carlo backup.

Experimental results on a variety of benchmark problems with increasing dimension (up to 12-D) of the action space indicate that ADVT substantially outperforms state-of-the-art methods [18, 29]. Our C++ implementation of ADVT is available at https://github.com/hoergems/ADVT.

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, which is a probability distribution on the state space $\mathcal{S}$. 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 $s_{t+1} \sim T(s_t, a_t, s_{t+1}) = p(s_{t+1}|s_t, a_t)$. In this paper, the function T is a probability density function, as both state and actions spaces are continuous. The robot does not know the state s_{t+1} exactly, but perceives an observation $o_t \in \mathcal{O}$ with probability $Z(s_{t+1}, a_t, o_t) = p(o_t|s_{t+1}, a_t)$. 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; that is, the value $V_\pi(b_0) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[r_t|b_0]$, 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 $\mathit{\Pi}$ of policies defined as mappings from beliefs to actions. The POMDP solution is then the optimal policy, denoted as π^* and defined as $\pi^* = \arg\max_{\pi \in \mathit{\Pi}} V_\pi(b)$. In designing solvers, it is often convenient to work with the action value or Q-value $Q(b, a) = R(b, a) + \gamma \max_{\pi \in \mathit{\Pi}} \mathbb{E}_{o \in \mathcal{O}}[V_\pi(b_a^o)|b]$, 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 $V^*(b) = \max_{a \in \mathcal{A}} Q(b, a)$. A more elaborate explanation is available in [11].

Belief trees are convenient data structures to find good approximations to the optimal solutions via sampling-based approaches, which has been shown to

significantly improve the scalability of POMDP solving [14]. 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 and leading 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 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., [3, 15, 16, 22, 26]) compute an optimal policy for all beliefs first before deploying them for execution. In contrast, online solvers (e.g., [17, 24, 37]) aim to further scale to larger and more complex problems by interleaving planning and execution, and 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 [1, 28, 32, 33] restrict beliefs to be Gaussian and use LQG 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 [8].

Last but not least, hierarchical rectangular partitions have been commonly applied to solve continuous action bandits and MDPs (the fully observed version of POMDPs), such as HOO [4] and HOOT [19]. 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 [12] and VOMCPOW [18]. 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.

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 also consider the state space $\mathcal{S}$ to be continuous or discrete, while we assume the observation space $\mathcal{O}$ to be discrete.

ADVT assumes that the Q-value 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.

Algorithm 1 ADVT(Initial belief b_0)

```
b = b_0;
T = initializeBeliefTree(b)
H(b) = Initialize Voronoi tree for belief b
isTerminal = False
while isTerminal is False do
    while planning budget not exceeded do
        (e, b_c) = SampleEpisode(T, b)                    ▷ Algorithm 2
        for i = |e| − 1 to 1 do
            (s, a, o, r) = e_i
            Backup(T, b_c, a, r)                          ▷ Algorithm 3
            b_c = Parent node of b_c in T
            RefineVoronoiTree(H(b_c), a)                  ▷ Algorithm 4
        end for
    end while
    a = Get best action in T from b
    (o, isTerminal) = Execute a
    b = τ(b, a, o)
end while
```

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 $\mathcal{H}(b)$. 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.

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 $b' = \tau(b, a, o)$.

To construct the belief tree $\mathcal{T}$, ADVT interleaves the iterative select-expand-simulate-backup operations used in many MCTS algorithms with adaptive discretization. Each node b in $\mathcal{T}$ is associated with a finite action set $\mathcal{A}(b) \subset \mathcal{A}$ which is adaptively refined. At each iteration, it first *selects* 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: Set the current node b as the root node, and sample s_0 from b; at each step $i \geq 0$, choose an action $a_i \in \mathcal{A}(b)$ for b using an action selection strategy (discussed in Section 4.1), execute a_i from state s_i to obtain o_i, r_i and s_{i+1} as the observation, the immediate reward and the next state by simulating the dynamics and observation model, and finally 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 encountered actions. In addition, $\mathcal{A}(b)$ is associated with a Voronoi tree $\mathcal{H}(b)$

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

e = init episode; $b = b_c$; s = sample a state from b; newBelief = False
while newBelief is False and s not terminal **do**
  $\mathcal{A}(b)$ = Set of candidate actions associated to the leaf-nodes of $\mathcal{H}(b)$
  $a = \arg\max_{a_k \in \mathcal{A}(b)} U(b, a_k)$ ▷ eq.(2)
  $(s', o, r) = G(s, a)$ ▷ Generative model
  append (s, a, o, r) to e
  $N(b, a) = N(b, a) + 1; N(b) = N(b) + 1$
  $s = s'$
  b = child node of b via edge (a, o) (If no such child exists, $b = null$)
  if $b = null$ **then**
    b = Create a new belief node as a child of b via edge (a, o)
    $\mathcal{H}(b)$ = Initialize Voronoi tree for belief b
    newBelief = True ; $N(b) = 0$
  end if
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)

in ADVT, and it is refined as needed for each encountered belief node b. Algorithm 2 presents the pseudo-code for constructing $\mathcal{T}$, while the backup operation and refinement of $\mathcal{A}(b)$ 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 continuous-arm bandit problem. Specifically, it selects an action from the set of candidate actions $\mathcal{A}(b)$ according to [34]

$$a^* = \arg\max_{a \in \mathcal{A}(b)} U(b, a), \qquad \text{with} \tag{1}$$

$$U(b, a) = \widehat{Q}(b, a) + C\sqrt{\frac{\log N(b)}{N(b, a)}} + L \ \mathrm{diam}(P), \tag{2}$$

where $N(b)$ is the number of times node b has been visited so far, $P \subseteq \mathcal{A}$ is the unique leaf cell containing a in $\mathcal{H}(b)$ (see Section 5 for details on the Voronoi tree), and $\mathrm{diam}(P) = \sup_{a,a' \in P} d(a, a')$ is the diameter of P wrt 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: $\widehat{Q}(b, a) + C\sqrt{\frac{\log N(b)}{N(b,a)}}$ 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 $U(b,a) \geq Q(b,a) + L \operatorname{diam}(P) \geq Q(b,a')$, 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', Action a, Reward r)

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

After sampling an episode, 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 $\widehat{Q}(b,a) = \widehat{Q}(b,a) + N(b,a)^{-1}(r + \gamma \widehat{V}^*(b') - \widehat{Q}(b,a))$, where b' is the child of b in the belief tree $\mathcal{T}$ via edge (a,o); i.e., the belief we arrived at after performing action $a \in \mathcal{A}$ and perceiving observation $o \in \mathcal{O}$ from b. Note that this rule is in contrast to POMCP, POMCPOW and VOMCPOW, where the Q-value estimates are updated via Monte-Carlo backup. This update rule is akin to the rule used in Q-Learning [35] and was implemented in the ABT software [9, 13], though never explicitly compared with Monte Carlo backup. The above update rule helps ADVT to focus its search on promising parts of the belief tree, particularly for problems where good rewards are sparse.

5 ADVT: Construction and Refinement of Voronoi Trees

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 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.

Algorithm 4 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** $C_r N(b,a) \geq 1/\operatorname{diam}(P)^2$ **then**
3: $\quad a'$ = sample from P
4: $\quad (P_1, P_2)$ = Child cells of P induced by a and a'
5: $\quad$ Compute diameters of P_1 and P_2
6: $\quad$ Add (a, P_1) and (a', P') as (a, P)'s children
7: **end if**

To construct $\mathcal{H}(b)$, ADVT first samples an action a uniformly at random from the action space $\mathcal{A}$, and sets the pair $(a, \mathcal{A})$ as the root of $\mathcal{H}(b)$. When ADVT decides to expand the 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_i 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 $d(a'', a) < d(a'', a')$, 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 $\mathcal{A}$.

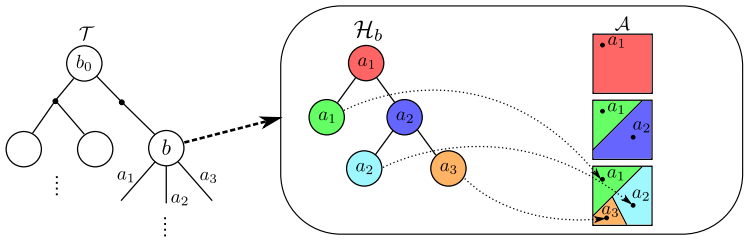


Fig. 1. Illustration of the relation between a belief tree $\mathcal{T}$ (left), the Voronoi tree $\mathcal{H}(b)$ associated to belief b (middle) and the partition of the action space induced by the Voronoi tree (right).

Three key details in constructing the tree $\mathcal{H}(b)$ are elaborated below. Section 5.1 describes how ADVT decides which node of $\mathcal{H}(b)$ to expand. Section 5.2 presents how the diameter of a Voronoi cell is estimated. Finally, Section 5.3 describes how uniform sampling is performed efficiently within a cell.

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 (1)

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

$$C_r N(b, a) \geq \frac{1}{\operatorname{diam}(P)^2}, \tag{3}$$

where C_r is an exploration constant and $N(b, a)$ is the number of times that a has been selected at b, which provides a rough estimate on the quality of the $\widehat{Q}(b, a)$ estimate. This criterion, which is inspired by [30], 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 size of the cells, induced by the sampled 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 in the sense that we use local information to decide when to refine the decomposition.

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 $\mathcal{H}(b)$. Then, we first sample a set of points $\mathcal{A}_P(b)$ that approximately lie on the boundary of P. 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 $\text{diam}(\mathcal{A})$ – 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 [5] 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 a boundary point a_P. The diameter of a bounding sphere that encloses all the sampled boundary points in $\mathcal{A}_P(b)$ [36] is then an approximation of the diameter of P.

5.3 Sampling from the Voronoi Cells

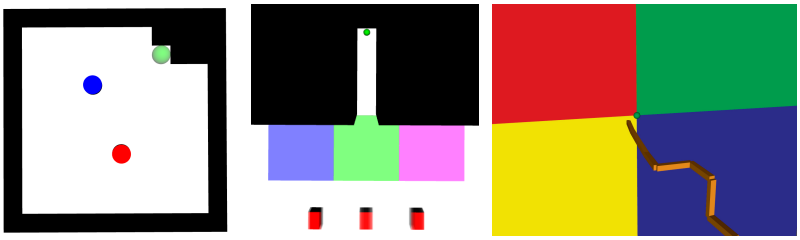
To sample an action that is approximately uniformly distributed in a cell P, we use a simple Hit & Run approach [25] that performs a random walk within P. Suppose P is the cell corresponding to the node (a, P) of the Voronoi tree $\mathcal{H}(b)$. We first sample an action a_P on the boundary of P using 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 4 robotics tasks, formulated as continuous-action POMDPs.

6.1 Problem Scenarios

Pushbox Pushbox is a scalable motion planning problem motivated by air hockey. A disk-shaped robot has to push a disk-shape puck into a goal region (rewarded by $1,000$) by bumping into it, while avoiding collisions of itself and the opponent with a boundary region (penalized by -500). The robot can move



(a) (b) (c)

Fig. 2. Illustrations of (a) the Pushbox2D, (b) the Parking2D and (c) the SensorPlacement-8 problems. Goal regions are marked as green circles.

freely in the environment by choosing a displacement vector. Upon bumping into the puck, it is pushed away and the robot's and puck's motions are affected by noise. The initial puck position is uncertain but the robot has access to a noisy bearing sensor to localize the puck. 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, the robot and the puck operate on a 2D-plane and the state space consists of the 2D-coordinates of the robot and the puck. The action space is $\mathcal{A} \subset \mathbb{R}^2$ such that $a \in \mathcal{A}$ describes a displacement vector of the robot on the 2D-plane. Figure 2(a) illustrates the Pushbox2D problem. For **Pushbox3D**, both the robot and the puck move inside a 3D-environment and the action space is $\mathcal{A} \subset \mathbb{R}^3$ where $a \in \mathcal{A}$ is a 3D-displacement vector. Additional details of the problem can be found in [23].

Parking An autonomous vehicle with deterministic 2nd-order dynamics [32] 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 (rewarded by 100) while avoiding collisions with the obstacles (black areas in Figure 2(b)), which is penalized by -100. We consider two variants of the problem, **Parking2D** and **Parking3D**. For **Parking2D**, the vehicle navigates on a 2D-plane and its state is a 4D-vector consisting of the xy-position of the vehicle on the plane, its orientation and its velocity. The vehicle is controlled via a steering wheel angle and acceleration, i.e., the action space is $\mathcal{A} = \Omega \times \Phi$, where Ω is the continuous set of steering wheel angles and Φ is the continuous set of accelerations. 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. 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. The discount factor is $\gamma = 0.95$.

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 area, the robot requires precise motions in order to avoid collision with the obstacles.

Van Der Pol Tag Van Der Pol Tag (VDP-Tag) is a benchmark problem introduced in [29] in which an agent operates in a 2D-environment. The goal is to tag a moving target (rewarded by 100) 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 (penalized by 5), i.e., 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. More details of the VDP-Tag problem can be found in [29]. For this problem, ADVT needs to discretize the observation space, as detailed in the Appendix.

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 1,000) 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 the touched wall. 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. 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 $h = [0.1]^D$ (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 $\gamma = 0.95$. 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.

6.2 Experimental Setup

The purpose of our experiments is two-fold: First is to evaluate ADVT and compare it with two state-of-the-art online POMDP solvers for continuous actions spaces, POMCPOW [29] and VOMCPOW [18]. Note that 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, whereas ADVT applies ABT's [17] 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, we also tested modified versions of VOMCPOW and POMCPOW, called **VOMCPOW-I** and **POMCPOW-I**, where we follow ADVT's strategy of re-using the partial search trees.

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 eq.(2), 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 eq.(2) reduces to the standard UCB1 bound. For the third variant, **ADVT-MC**, we replace the stochastic Bellman backups described in Section 4.2 with simple Monte-Carlo backups as used by POMCPOW and VOMCPOW. Moreover, to test the effects of stochastic Bellman backups further, we implemented variants of the comparators, **VOMCPOW-B** and **POMCPOW-B**, which modify VOMCPOW-I and POMCPOW-I respectively to use stochastic Bellman backups instead of Monte Carlo backups. Note that for the VDP-Tag problem, 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 space.

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 1,000 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 [10]. All simulations were run single-threaded on an Intel Xeon Platinum 8274 CPU with 3.2GHz and 4GB of memory.

6.3 Results

Comparison with State-of-the-Art Methods Table 1 shows the average total discounted rewards of all tested solvers on the Pushbox, Parking and VDP-Tag problems, while Figure 3 shows the results for the SensorPlacement problems. Detailed results for the SensorPlacement problems and results on the success rates of the tested solvers are in the Appendix.

ADVT generally outperforms all other methods, except for VDP-Tag, where both POMCPOW and VOMCPOW perform better. The reason is that VDP-

Table 1. Average total discounted rewards and 95% confidence intervals of all tested solvers on the Pushbox, Parking and VDP-Tag 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
ADVT	351.6 ± 10.0	$\mathbf{322.1 \pm 14.9}$	35.2 ± 1.9	$\mathbf{32.6} \pm 3.5$	24.3 ± 1.2
ADVT-R	$\mathbf{371.4} \pm 9.8$	321.2 ± 15.1	$\mathbf{38.9} \pm 1.8$	24.3 ± 3.4	24.5 ± 1.2
ADVT (L=0)	340.8 ± 14.7	294.6 ± 13.3	29.2 ± 3.5	18.6 ± 1.7	24.2 ± 1.1
ADVT-MC	319.6 ± 13.7	311.0 ± 16.2	-3.2 ± 1.8	-14.7 ± 0.5	23.9 ± 0.9
VOMCPOW-B	322.9 ± 12.1	274.9 ± 14.2	28.2 ± 1.8	19.1 ± 3.3	-
VOMCPOW-I	316.0 ± 12.3	268.9 ± 14.2	-0.42 ± 2.8	-15.7 ± 1.5	-
VOMCPOW	129.8 ± 13.3	73.5 ± 13.8	-0.78 ± 2.8	-18.4 ± 1.4	$\mathbf{32.9} \pm 0.9$
POMCPOW-B	314.2 ± 13.0	245.7 ± 14.1	27.7 ± 1.8	8.8 ± 2.6	-
POMCPOW-I	270.6 ± 18.9	203.7 ± 14.3	-5.2 ± 2.9	-22.8 ± 1.3	-
POMCPOW	82.1 ± 14.2	3.6 ± 12.9	-5.1 ± 3.0	-25.7 ± 1.4	28.2 ± 1.1

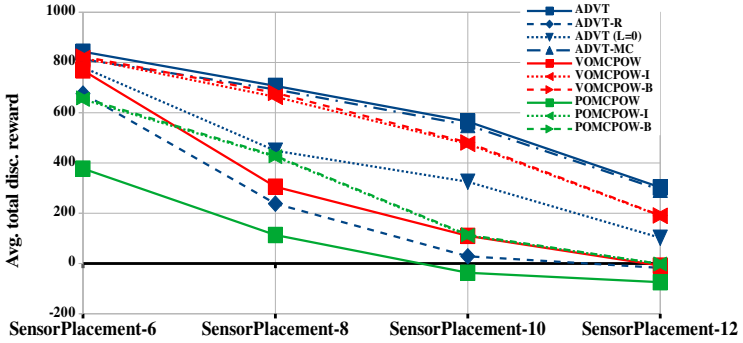


Fig. 3. Average total discounted rewards of all tested solvers on the SensorPlacement problems. The average is taken over 1,000 simulation runs per solver and problem.

Tag's observation space is a continuous space. Both VOMCPOW and POMCPOW are designed for POMDPs with continuous state, action, and observation space, whilst ADVT is **not** designed for continuous observation spaces. In this problem, ADVT relies on discretizing the observation space, which is sub-optimal, considering the observation space is 8-dimensional, whereas VOMCPOW and POMCPOW apply Progressive Widening and weighted-particle representation of the beliefs in the search tree, which scales better to high-dimensional continuous observation spaces. Interestingly, in a VDP-Tag variant with a less noisy dynamics ($\sigma = 0.01$), ADVT is only slightly worse than VOMCPOW and slightly better than POMCPOW, which supports ADVT's effectiveness in handling continuous actions (see the Appendix). The results for the SensorPlacement problems indicate that ADVT scales well to higher-dimensional action spaces.

ADVT performs well in terms of the success rate, too (Appendix 1.B). ADVT maintains more than 90% success rate in the Pushbox, Parking and VDP-Tag problems and $> 72\%$ in the SensorPlacement-12 problem. VOMCPOW's and POMCPOW's success rate can drop as low as $\sim 30\%$ and 12.5% in the Park-

ing3D problem and to $< 60\%$ and $< 40\%$ in the SensorPlacement-12 problem respectively.

Generally, VOMCPOW-I and POMCPOW-I perform much better than 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.

Understanding the Effects of Different Components of ADVT

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 1 shows that ADVT-R slightly outperforms ADVT in the Pushbox2D, Parking2D and VDP-Tag problems, indicating that a rectangular-based partitioning works well for low-dimensional action spaces. However, Figure 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 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 eq.(2), let us compare ADVT and ADVT (L=0). The results in Table 1 and Figure 3 indicate that the cell-diameter-aware component in the upper-confidence bound in eq.(2) 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 eq.(2) 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, as well as VOMCPOW and POMCPOW to VOMCPOW-B and POMCPOW-B, respectively. Table 1 and Figure 3 reveal that the solvers that use stochastic Bellman backups (ADVT, ADVT-R, VOMCPOW-B and POMCPOW-B) perform 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.

7 Conclusion

We propose a new sampling-based online POMDP solver, called ADVT, that scales well to POMDPs with high-dimensional continuous action 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. 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. In future works we are planning to extend ADVT to handle continuous observation spaces as well, which would allow us to tackle even more challenging POMDP problems.

Acknowledgements We thank Jerzy Filar for many helpful discussions. This work is partially supported by the Australian Research Coucil (ARC) Discovery Project 200101049.

References

1. Agha-Mohammadi, A.A., Chakravorty, S., Amato, N.M.: Firm: Feedback controller-based information-state roadmap. A framework for motion planning under uncertainty. In: IROS. pp. 4284–4291. IEEE (2011)
2. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning **47**(2-3), 235–256 (May 2002)
3. Bai, H., Hsu, D., Lee, W.S.: Integrated perception and planning in the continuous space: A POMDP approach. IJRR **33**(9), 1288–1302 (2014)
4. Bubeck, S., Munos, R., Stoltz, G., Szepesvári, C.: X-armed bandits. Journal of Machine Learning Research **12**(5) (2011)
5. Burden, R.L., Faires, J.D., Burden, A.M.: Numerical analysis. Cengage Learning, Boston, MA, tenth edn. (2016)
6. Couëtoux, A., Hoock, J.B., Sokolovska, N., Teytaud, O., Bonnard, N.: Continuous upper confidence trees. In: LION. pp. 433–445. Springer (2011)
7. Fischer, J., Tas, Ö.S.: Information particle filter tree: An online algorithm for POMDPs with belief-based rewards on continuous domains. In: ICML. pp. 3177–3187. PMLR (2020)
8. Hoerger, M., Kurniawati, H., Bandyopadhyay, T., Elfes, A.: Linearization in motion planning under uncertainty. In: Algorithmic Foundations of Robotics XII, pp. 272–287. Springer, Cham (2020)
9. Hoerger, M., Kurniawati, H., Elfes, A.: OPPT. https://github.com/RDLLab/oppt
10. Hoerger, M., Kurniawati, H., Elfes, A.: A software framework for planning under partial observability. In: IROS. pp. 1–9. IEEE (2018)
11. Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. Artificial intelligence **101**(1-2), 99–134 (1998)
12. Kim, B., Lee, K., Lim, S., Kaelbling, L., Lozano-Pérez, T.: Monte carlo tree search in continuous spaces using Voronoi optimistic optimization with regret bounds. In: AAAI. vol. 34, pp. 9916–9924 (2020)

13. Klimenko, D., Song, J., Kurniawati, H.: TAPIR. https://github.com/RDLLab/tapir
14. Kurniawati, H.: Partially observable Markov decision processes and robotics. Annual Review of Control, Robotics, and Autonomous Systems **5**(1) (2022), to appear
15. Kurniawati, H., Du, Y., Hsu, D., Lee, W.S.: Motion planning under uncertainty for robotic tasks with long time horizons. IJRR **30**(3), 308–323 (2011)
16. Kurniawati, H., Hsu, D., Lee, W.S.: SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In: RSS (2008)
17. Kurniawati, H., Yadav, V.: An online POMDP solver for uncertainty planning in dynamic environment. In: Proc. Int. Symp. on Robotics Research (2013)
18. Lim, M.H., Tomlin, C.J., Sunberg, Z.N.: Voronoi progressive widening: Efficient online solvers for continuous state, action, and observation POMDPs. In: 60th IEEE Conference on Decision and Control (CDC). pp. 4493–4500 (2021)
19. Mansley, C., Weinstein, A., Littman, M.: Sample-based planning for continuous action Markov decision processes. In: ICAPS (2011)
20. Mern, J., Yildiz, A., Sunberg, Z., Mukerji, T., Kochenderfer, M.J.: Bayesian optimized monte carlo planning. In: AAAI. vol. 35, pp. 11880–11887 (2021)
21. Papadimitriou, C.H., Tsitsiklis, J.N.: The complexity of Markov decision processes. Mathematics of operations research **12**(3), 441–450 (1987)
22. Pineau, J., Gordon, G., Thrun, S.: Point-based Value Iteration: An anytime algorithm for POMDPs. In: IJCAI (2003)
23. Seiler, K.M., Kurniawati, H., Singh, S.P.: An online and approximate solver for POMDPs with continuous action space. In: ICRA. pp. 2290–2297. IEEE (2015)
24. Silver, D., Veness, J.: Monte-carlo planning in large POMDPs. In: Advances in neural information processing systems. pp. 2164–2172 (2010)
25. Smith, R.L.: Efficient monte carlo procedures for generating points uniformly distributed over bounded regions. Operations Research **32**(6), 1296–1308 (1984)
26. Smith, T., Simmons, R.: Point-based POMDP algorithms: Improved analysis and implementation. In: UAI (2005)
27. Sondik, E.J.: The Optimal Control of Partially Observable Markov Decision Processes. Ph.D. thesis, Stanford, California (1971)
28. Sun, W., Patil, S., Alterovitz, R.: High-frequency replanning under uncertainty using parallel sampling-based motion planning. IEEE TRO **31**(1), 104–116 (2015)
29. Sunberg, Z.N., Kochenderfer, M.J.: Online algorithms for POMDPs with continuous state, action, and observation spaces. In: ICAPS (2018)
30. Touati, A., Taiga, A.A., Bellemare, M.G.: Zooming for efficient model-free reinforcement learning in metric spaces. arXiv preprint 2003.04069 (2020)
31. Valko, M., Carpentier, A., Munos, R.: Stochastic simultaneous optimistic optimization. In: ICML. pp. 19–27. PMLR (2013)
32. Van Den Berg, J., Abbeel, P., Goldberg, K.: LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information. The International Journal of Robotics Research **30**(7), 895–913 (2011)
33. Van Den Berg, J., Patil, S., Alterovitz, R.: Motion planning under uncertainty using iterative local optimization in belief space. IJRR **31**(11), 1263–1278 (2012)
34. Wang, T., Ye, W., Geng, D., Rudin, C.: Towards practical Lipschitz bandits. In: FODS. pp. 129–138 (2020)
35. Watkins, C.J., Dayan, P.: Q-learning. Machine learning **8**(3), 279–292 (1992)
36. Welzl, E.: Smallest enclosing disks (balls and ellipsoids). In: Maurer, H. (ed.) New Results and New Trends in Computer Science. pp. 359–370. Springer, Berlin (1991)
37. Ye, N., Somani, A., Hsu, D., Lee, W.S.: DESPOT: Online POMDP planning with regularization. Journal of Artificial Intelligence Research **58**, 231–266 (2017)

Appendix 1.A Detailed Experimental Results for the SensorPlacement Problems

Table 2. 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-B	823.4 ± 15.1	679.1 ± 17.9	481.6 ± 22.2	191.3 ± 17.6
VOMCPOW-I	817.2 ± 15.8	663.4 ± 18.6	476.0 ± 22.7	189.9 ± 18.0
VOMCPOW	768.5 ± 16.4	305.6 ± 25.8	110.1 ± 24.5	-8.2 ± 13.2
POMCPOW-B	659.3 ± 17.2	428.7 ± 21.5	114.6 ± 16.6	-1.9 ± 6.6
POMCPOW-I	653.2 ± 17.3	425.2 ± 21.8	111.3 ± 16.8	-2.1 ± 6.8
POMCPOW	377.6 ± 23.5	113.4 ± 24.2	-36.8 ± 11.3	-74.3 ± 12.9

Appendix 1.B Success Rates

Table 3. Success rates of all tested solvers on the Pushbox, Parking and VDP-Tag problems. The success rate is with respect to 1,000 simulation per solver and problem, with a planning time of 1s per step.

	Pushbox2D	Pushbox3D	Parking2D	Parking3D	VDP-Tag
ADVT	0.985	0.969	0.912	0.916	0.941
ADVT-R	0.987	0.968	0.943	0.906	0.945
ADVT (L=0)	0.966	0.965	0.857	0.898	0.935
ADVT-MC	0.989	0.972	0.417	0.337	0.892
VOMCPOW-B	0.985	0.970	0.885	0.886	-
VOMCPOW-I	0.975	0.939	0.597	0.314	-
VOMCPOW	0.754	0.815	0.512	0.297	0.987
POMCPOW-B	0.974	0.953	0.853	0.534	-
POMCPOW-I	0.963	0.969	0.409	0.122	-
POMCPOW	0.712	0.692	0.401	0.125	0.979

In addition to the average total discounted rewards in the main document, we also report the success rate of each solver in each problem scenario in Table 3 and Table 4. For the Pushbox problems, a run is considered successful if the robot pushes the opponent into the goal region, while avoiding collisions of itself and the opponent with the boundary region. For the Parking problems, a run is successful if the vehicle reaches the goal area. For the VDP-Tag problem a run is successful if the opponent is being tagged and for the SensorPlacement problems, a run is successful if the end-effector reaches the sensor mounting location. In all problems, the task must be completed within planning steps 50 steps, otherwise problem terminates and the run is considered unsuccessful.

Generally the success rates are closely correlated to the average total discounted rewards achieved by each solver in the problem scenarios. The results in Table 4 further indicate that ADVT scales better to higher-dimensional action spaces compared to the baselines. However, for the SensorPlacement12 problem,

Table 4. Success rates of all tested solvers on the SensorPlacement problems. The success rate is with respect to 1,000 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-B	0.979	0.951	0.807	0.703
VOMCPOW-I	0.967	0.891	0.803	0.698
VOMCPOW	0.923	0.721	0.645	0.583
POMCPOW-B	0.829	0.794	0.646	0.575
POMCPOW-I	0.826	0.781	0.657	0.578
POMCPOW	0.738	0.636	0.519	0.321

the success rates are relatively low, even for ADVT. Thus, for such problems, developing methods that scale even better to high-dimensional action spaces remains a fruitful avenue for future research.

Appendix 1.C Observation Discretization Method for the VDP-Tag Problem

Since the observation space in the VDP-Tag problem is continuous, i.e., $\mathcal{O} = \mathbb{R}^8$, ADVT requires a method to discretize the observations. To this end, we use a simple distance-based discretization: Suppose a sampled episode selects action $a \in \mathcal{A}(b)$ at belief b and perceives an observation $o_i \in \mathcal{O}$. We then compute the Euclidean distance between o_i and each observation corresponding to the outgoing edges $(a, o) \in \mathcal{T}$ that descend b. If there is an observation o_k corresponding to an outgoing edge for which the Euclidean distance to o_i yields a value smaller than a threshold δ (in our experiments we use $\delta = 25$), we continue the search from child node b' of b via edge (a, o_k). Otherwise, we add a new child node to b via edge (a, o_i).

Appendix 1.D Results for the VDP-Tag Problem With Smaller Transition Errors

Table 5. Average total discounted rewards with 95% confidence intervals and success rates of all tested solvers on the VDP-Tag with smaller transition errors ($\sigma = 0.01$). The average is taken over 1000 simulation runs per solver and problem, with a planning time of 1s per step.

	Avg. total discounted reward	Success rate
ADVT	31.5 ± 0.9	0.991
ADVT-R	31.7 ± 0.9	0.986
ADVT L(=0)	30.1 ± 1.0	0.989
ADVT-MC	30.9 ± 0.8	0.984
VOMCPOW	$\mathbf{34.1 \pm 0.8}$	**0.998**
POMCPOW	29.1 ± 0.8	0.990

We additionally tested ADVT as well as VOMCPOW and POMCPOW on a variant of the VDP-Tag problem with smaller transition errors, i.e., the position of the target is disturbed by Gaussian noise with standard deviation $\sigma = 0.01$ instead of $\sigma = 0.05$. The results are shown in Table 5.

It can be seen that for the variant of the problem with $\sigma = 0.01$, ADVT is competitive with POMCPOW. For $\sigma = 0.05$ the uncertainty with respect to the position of the target is large and therefore the agent must carefully decide when to activate its range sensor in order to reduce uncertainty. To achieve this, the solvers require more accurate belief representations in the search tree, which is challenging for ADVT, as it relies on discretizing the continuous observation space. On the other hand, VOMCPOW and POMCPOW use Progressive Widening in the observation space, combined with a weighted-particle representation of the beliefs in the search trees, which helps them to perform well.

For $\sigma = 0.01$, the uncertainty with respect to the position of the target is much smaller, and therefore the solvers are less reliant on accurate belief representations in the search tree, which benefits ADVT. This suggests that our method works well for problems with small belief uncertainties, even if the observation space is continuous. To handle larger uncertainties, we require a better method to handle continuous observation spaces, such as progressive widening and explicit belief representations as used in VOMCPOW and POMCPOW. We are planning to explore this in future works.