An On-Line POMDP Solver for Continuous Observation Spaces

Marcus Hoerger[1] and Hanna Kurniawati[1]

***Abstract*— Planning under partial obervability is essential for autonomous robots. A principled way to address such planning problems is the Partially Observable Markov Decision Process (POMDP). Although solving POMDPs is computationally intractable, substantial advancements have been achieved in developing approximate POMDP solvers in the past two decades. However, computing robust solutions for problems with continuous observation spaces remains challenging. Most on-line solvers rely on discretising the observation space or artificially limiting the number of observations that are considered during planning to compute tractable policies. In this paper we propose a new on-line POMDP solver, called Lazy Belief Extraction for Continuous Observation POMDPs (LABECOP), that combines methods from Monte-Carlo-Tree-Search and particle filtering to construct a policy reprentation which doesn't require discretised observation spaces and avoids limiting the number of observations considered during planning. Experiments on three different problems involving continuous observation spaces indicate that LABECOP performs similar or better than state-of-the-art POMDP solvers.**

I. INTRODUCTION

Planning under partial observability is both challenging and essential for autonomous robots. The Partially Observable Markov Decision Processes (POMDP) [1] is a mathematically principled way to solve such planning problems. By lifting the planning problem from the robot's state space to its *belief space*, i.e. the set of all probability distributions over the state space, POMDPs enable autonomous robots to systematically reason about different strategies to achieve a given task while having only access to noisy or incomplete observations. Although solving a POMDP exactly is computationally intractable [2], the past two decades have seen tremendous progress in developing approximately optimal solvers that trade optimality for computational tractability, making them applicable to various robotic planning problems [3], [4], [5], [6].

Despite the mentioned advances, solving POMDP problems with continuous observation space remains a challenge. Existing on-line POMDP solvers require some kind of discretisation of the observation space, either explicitly or implicitly. However, discretising the observation space can be difficult, particularly for high-dimensional observation spaces, since it requires a good trade-off between accuracy and planning efficiency. Finer discretisations of the observation space lead to a substantial increase in computation time, whereas coarser discretisations can impair the quality of the resulting policies. More recently, [7] proposed to avoid discretising the observation space by sampling and keeping sampled observations only occasionally. However, this strategy can be sensitive to the rate at which observations are sampled and kept.

In this paper we propose a new on-line POMDP solver, called Lazy Belief Extraction for Continuous Observation POMDPs (LABECOP) to alleviate the above issues. LABECOP avoids any form of observation space discretisation by realising that a set of sampled episodes —that is, sequences of state–action–observation–reward quadruples— is sufficient to represent many different belief sequences and that beliefs, along with their action-values and action-selection strategy, can be estimated using this set via an episode re-weighting method inspired by particle filters. During planning, LABECOP extracts sequences of beliefs and their corresponding action-values and action-selection strategy lazily, in a sense that they are only extracted for the action-observation sequence of the currently sampled episode. This allows LABECOP to consider *every* sampled observation and therefore *every* sequence of beliefs it encounters during planning, without having to discretise the observation space.

Experimental results on three test scenarios indicate that LABECOP is more effective in computing robust policies for problems with continuous observation spaces compared to state-of-the-art methods.

II. BACKGROUND AND RELATED WORK

A. Partially Observable Markov Decision Process (POMDP)

Formally a POMDP is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, \gamma >$, where $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{O}$ are the state, action and observation spaces of the robot. T and Z model the uncertainty in the effect of taking actions and receiving observations as conditional probability functions $T(s, a, s') = p(s'|s, a)$ and $Z(s', a, o) = p(o|s', a)$, where $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$ and $o \in \mathcal{O}$. $R(s, a)$ models the reward the robot receives when performing action a from s and $0 < \gamma < 1$ is a discount factor. Due to uncertainties in the effect of performing actions and receiving observations, the true state of the robot is only partially observable. Hence, instead of planning with respect to states, the robot plans with respect to probability distributions $b \in \mathcal{B}$ over the state space, called beliefs, where $\mathcal{B}$ is the set of all probability distributions over $\mathcal{S}$. The solution of a POMDP is an optimal policy π^*, a mapping from beliefs to actions $\pi^* : b \mapsto a$ such that the robot maximises the expected discounted future reward when following π^*. Once π^* has been computed, it can be used as a feedback-controller: Given the current belief b, the robot performs $\pi^*(b)$, receives an observation $o \in \mathcal{O}$ and updates

*This work is partially funded by the ANU Futures Scheme QCE20102.

[1]College of Engineering & Computer Science, The Australian National University, 7500 Canberra, Australia {marcus.hoerger, hanna.kurniawati}@anu.edu.au

its belief according to $b' = \tau(b, a, o)$, where τ is the Bayesian belief update function. The value achieved by a policy π at a particular belief b can be expressed as

$$V_\pi(b) = R(b, \pi(b)) + \gamma \int_{o \in \mathcal{O}} Z(b, \pi(b), o) V_\pi(\tau(b, \pi(b), o)) \mathrm{d}o \tag{1}$$

where $R(b, a) = \int_{s \in \mathcal{S}} R(s, a) b(s) \mathrm{d}s$ and $Z(b, a, o) = \int_{s' \in \mathcal{S}} Z(s', a, o) \int_{s \in \mathcal{S}} T(s, a, s') b(s) \mathrm{d}s \mathrm{d}s'$. The optimal policy π^* is then the policy that satisfies $\pi^*(b) = \arg\max_\pi V_\pi(b)$.

B. Related On-line POMDP Solvers

POMDP solvers have seen tremendous advances in the past two decades. Key to their success is the use of sampling to trade optimality for computational tractability. These solvers can be broadly classified into off-line and on-line solvers. While off-line solvers [8], [9], [10] compute an approximate-optimal policy for a sampled representation of the belief space, on-line solvers interleave planning and execution by computing a policy for the current belief only. Various on-line solvers have been proposed for continuous state spaces [11], [12], [13], large or continuous actions spaces [14], [15], expensive transition dynamics [16] and long planning horizons [17], [18], [19]. To compute a policy for the current belief, these solvers typically perform forward-search in a lookahead-tree to evaluate sequences of actions starting from the current belief. For instance POMCP [11] and ABT [13] use Monte-Carlo-Tree-Search by sampling many sequences of actions. This allows them to quickly focus the search on the most promising parts of the lookahead-tree. DESPOT [12] uses a combination of Monte-Carlo sampling, heuristic search and branch-and-bound pruning to construct a sparse representation of the lookahead-tree.

For continuous observation spaces most of the aforementioned methods require some form of discretisation of the observation space to trade the branching factor of the lookahead-tree with accuracy. More recently, [20] extended DESPOT to handle large discrete observation spaces. While this approach allows for a finer discretisation of the observation space, it can't handle purely continuous observation spaces. Another approach is POMCPOW [7] and extension of POMCP to handle continuous action and observation spaces. POMCPOW uses Progressive-Widening [21] to slowly add sampled observations to the lookahead-tree as planning progresses. While this approach avoids discretising the observation space, the rate at which sampled observations are added to the tree must be carefully chosen in order to keep the branching factor of the search tree manageable.

III. LAZY BELIEF EXTRACTION FOR CONTINUOUS OBSERVATION POMDPS (LABECOP)

A. Overview

LABECOP is an anytime on-line POMDP solver designed for POMDP problems with continuous state and observation spaces. LABECOP incrementally samples a set of *episodes* $H_{(b)}$, i.e. sequences of state-action-observation-reward quadruples starting from the current belief $b \in \mathcal{B}$ to compute an approximation of the optimal policy for b.

Key to LABECOP is the realisation that the episodes in $H_{(b)}$ provide sufficient information to encode many different sequences of beliefs starting from b and that these beliefs, along with their approximated action values and corresponding action-selection strategy, can be extracted from $H_{(b)}$ on-the-fly during planning, without having to maintain a lookahead-tree. Both components – sequence of visited beliefs and the action-selection strategy – are extracted lazily from $H_{(b)}$ in a sense that we only extract them for the currently sampled episode. This enables LABECOP to consider *every* sampled observation (and therefore *every* encountered belief) during planning which helps LABECOP in discovering good policies that rely on differentiating between many sequences of beliefs. Fig. 1 illustrates the core idea behind LABECOP for a POMDP with two actions.

Whenever we sample a new episode h, we iteratively select and re-weight subsets of $H_{(b)}$, based on the actions and sampled observations associated to h to extract the sequence of beliefs visited by h. Moreover, each quadruple of the episodes in $H_{(b)}$ has an associated value that is computed only once per episode. We use these values to approximate the action values $Q(b, a)$ (i.e. the expected value of executing $a \in \mathcal{A}$ from b and continuing optimally afterwards) at the extracted beliefs via weighted sums of the episode values. The approximated action values are then used to derive an action-selection strategy. The methods that we use to extract sequences of beliefs and derive an action-selection strategy are described in Section III-B and Section III-C respectively.

To sample a new episode, we first sample an initial state $s \in \mathcal{S}$ from the current belief b and select an action $a \in \mathcal{A}$ according to the strategy described in Section III-C. We then sample a next state $s' \in \mathcal{S}$, an observation $o \in \mathcal{O}$ and the immediate reward $r = R(s, a)$ by calling a generative black-box model $(s', o, r) = G(s, a)$ and add the quadruple (s, a, o, r) to the episode. Given the selected action a and perceived observation o, we can now extract an approximation of next belief $b' = \tau(b, a, o)$ from $H_{(b)}$ as described in Section III-B and repeat the above steps from s' and b'. Upon completion, we add the newly sampled episode to $H_{(b)}$. An overview of the episode sampling process is shown in Algorithm 1.

Once the planning time for the current step is over, LABECOP selects an action from the current belief b according to

$$\pi(b) = \arg\max_{a \in \mathcal{A}} Q(b, a) \tag{2}$$

After executing $\pi(b)$ and perceiving an observation, we update the belief using a SIR particle filter [22] and continue planning from the updated belief. This process repeats until a maximum number of planning steps is reached, or the agent enters a terminal state (we assume that we know when the agent enters a terminal state).

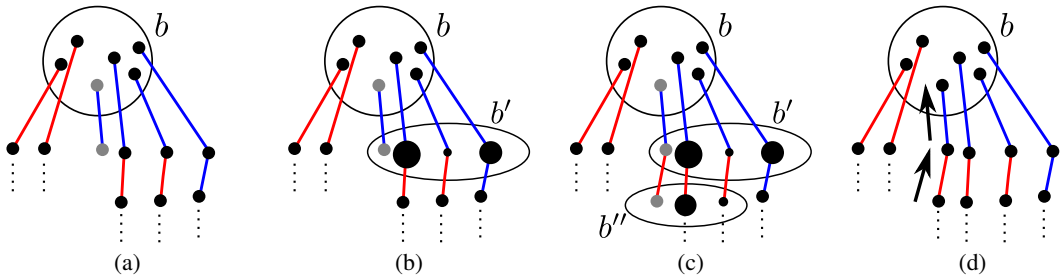


Fig. 1. Illustration of the episode sampling process for a POMDP with two actions a_1 and a_2. (a) Given a set of episodes $H_{(b)}$ (shown as black dots connected by action edges, where the dots represent states and the edges represent actions – blue for a_1 and red for a_2) that start from the current belief b (shown as circle) we first sample a state (shown as grey dot) from the current belief, select an action according to eq.(4) (a_1 in this case), sample a next state from the transition function (second grey dot) and an observation o. (b) We compute a weight (represented by the size of the black dots) for each state associated with the second quadruples of the episodes in $H_{(b,a_1)}$ (black oval), based on the selected action a_1, the sampled observation and the weights of the parent states, giving us a approximation of $b' = \tau(b, a_1, o)$. (c) From b', we select a new action according to eq.(4) sample a next state and observation and continue from the set $H_{(b',a_1)}$ (second black oval). (d) After sampling the episode, we perform Bellman backup to compute its values, add the episode to $H_{(b)}$ and discard the previously computed weights.

B. Extracting a sequence of beliefs for a new episode

After we have sampled an initial state, selected an action a from the current belief b and perceived an observation o, we now have to obtain an approximation of the belief $b' = \tau(b, a, o)$ to select the next action. As mentioned earlier, we can extract an approximation of b' directly from $H_{(b)}$. This is done as follows:

For every episode $h \in H_{(b)}$, the state $h_{(1)}.s$ associated to the first quadruple (we denote the i-th quadruple of h as $h_{(i)}$ and its associated state as $h_{(i)}.s$) has an initial weight $w(h_{(1)}.s)$ corresponding to the weight of the belief particle h starts from. If the current belief is represented by a set of unweighted particles, we set $w(h_{(1)}.s) = 1$. Now, suppose $H_{(b,a)}$ is the set of all episodes in $H_{(b)}$ that contain action a in their first quadruple. For every episode $h \in H_{(b,a)}$ we recursively assign a weight to the state associated to the second quadruple of h according to $w(h_{(2)}.s) = w(h_{(1)}.s)Z(h_{(2)}.s, a, o)/\eta$, where η is a normalisation constant. In other words, we compute the relative likelihood of "seeing" the sampled observation o given that we ended up in $h_{(2)}.s$ after executing a from $h_{(1)}.s$. Thus, the collection of all states associated to the second quadruples of the episodes in $H_{(b,a)}$ along with their computed weights forms a sampled approximation of b'. This is similar to a particle filter.

Once we select the next action a' from b' we set $H_{(b')} = H_{(b,a)}$ and iteratively compute the weights $w(h_{(3)}.s)$ for every $h \in H_{(b',a')} \subseteq H_{(b')}$, given $w(h_{(2)}.s)$, the selected action a' and the next observation. By repeating this process as we continue sampling the episode, we obtain a sequence of sampled approximations of the beliefs visited by the episode. Note that these weights are re-computed every time we sample a new episode since each sampled episode visits a different sequence of beliefs. Algorithm 2 provides an overview of this belief extraction process.

Intuitively, by extracting approximated beliefs using the weighting method above, we obtain richer belief approximations for action sequences that are visited more often, which helps LABECOP to evaluate frequently visited action sequences more accurately. On the other hand, extracting beliefs becomes more expensive as planning progresses, since we have to re-compute the weights for a growing set of episodes. Additionally, compared to many on-line POMDP solvers that only require a generative black-box model for planning, our method explicitly requires us to evaluate the observation function $Z(s, a, o)$ which might not be readily available. The first issue could be mitigated by numerical fitting, i.e. weighting only a small subset of the states and then perform interpolation, similarly to [23]. The second issue could be mitigated by using an approximation for $Z(s, a, o)$ which is often easy to obtain. We are planning to explore both options in future works.

C. Action-selection strategy at the extracted beliefs

To select actions from an approximated belief b of depth d obtained with the method above, LABECOP uses Upper Confidence Bounds1 (UCB1) [24], one of the most widely used strategies in on-line POMDP planning. UCB1 requires two quantities to select an action: Belief-dependent visitation counts $N(b)$ and $N(b, a)$, i.e. the number of times b has been expanded and the number of times each action has been used to expand b, as well as an estimate $\widehat{Q}(b, a)$ of the Q-values. Since we never visit the same belief twice, except for the current belief, we slightly modify UCB1:

Let $W(b) = \sum_{h \in H_{(b)}} w(h_{(d)}.s)$ and $W(b, a) = \sum_{h \in H_{(b,a)}} w(h_{(d)}.s)$. $W(b)$ can be seen as the weighted number of times b has been visited and $W(b, a)$ as the weighted number of times a has been used to expand b. However, we cannot use $W(b)$ and $W(b, a)$ in lieu of $N(b)$ and $N(b, a)$ since $W(b)$ can be smaller than 1, for which UCB1 is undefined. To resolve this we replace $N(b)$ with $N_+(b)$, which is the number of episodes in $H_{(b)}$ for which $w(h_{(d)}.s)$ is greater than 0, and replace $N(b, a)$ with a scaled action weight $\widetilde{W}(b, a) = \frac{W(b,a)}{W(b)} N_+(b)$.

The last ingredient we need for UCB1 are the Q-value estimates. It turns out that, similarly to the beliefs, we can extract those values on-the-fly from $H_{(b)}$ for the currently sampled episode. For a belief of depth d, LABECOP approximates $Q(b, a)$ according to

$$\widehat{Q}(b, a) = \frac{1}{W(b, a)} \sum_{h \in H_{(b,a)}} w(h_{(d)}.s) V(h_{(d)}) \qquad (3)$$

where $V(h_{(d)})$ is the value of the d-th quadruple of episode h, i.e. an approximation of the expected value of

executing $h_{(d)}.a$ from $h_{(d)}.s$ and continuing optimally after seeing $h_{(d)}.o$. These values are computed only once per episode via Bellman backup. Using $N_+(b)$, $\widetilde{W}(b,a)$, and $\widehat{Q}(b,a)$ we can now apply UCB1, i.e. we select an action according to

$$a = \arg\max_{a \in \mathcal{A}} \left(\widehat{Q}(b,a) + c \sqrt{\frac{\log(N_+)}{\widetilde{W}(b,a)}} \right) \tag{4}$$

Note that during the episode sampling process we might end up in an approximated belief for which $\widetilde{W}(b,a)$ is zero for one or more actions. If this is the case we treat these actions as "unvisited" and use a rollout strategy that selects one of these actions uniformly at random. Additionally, we compute a problem-dependent heuristic estimate for $V(h_{(k)})$ given $h_{(k)}.s$ and the randomly selected action, where $h_{(k)}$ is the last quadruple of h. Finally we backup the episode along the sequence of visited beliefs to obtain $V(h_{(i)})$ for each quadruple of the episode and add the episode to $H_{(b)}$. An overview of the action-selection process is shown in Algorithm 3.

Algorithm 1 SampleEpisode(Current belief b, Set of episodes $H_{(b)}$)

h = init episode; $i = 1$; $s \sim b$; unvisitedAction = False
while unvisitedAction is False and s not terminal **do**
  (a, unvisitedAction) = `SelectAction`(b, $H_{(b)}$) // *Algorithm 3*
  $(s', o, r) = G(s, a)$ // *Generative black-box model*
  Insert (s, a, o, r) to h
  $H_{(b,a)} = \{h \in H_{(b)} | h_{(i)}.a = a\}$
  $b' =$ `ExtractBelief`(b, $H_{(b,a)}$, a, o, i) // *Algorithm 2*
  $H_{(b')} = H_{(b,a)}$; $s = s'$; $b = b'$; $i = i + 1$
end while
if unvisitedAction is True **then**
  $V(h_{(i)}) =$ calculateHeuristic(s, h)
end if
insert $(s, -, -, 0)$ to h
backupEpisode(h, $V(h_{(i)})$)
return h

Algorithm 2 ExtractBelief(Belief b, Set of episodes H, Action a, Observation o, Index i)

$b' = \varnothing$;
for $h \in H$ **do**
  $w(h_{(i+1)}.s) = b(h_{(i)}.s) Z(h_{(i+1)}.s, a, o)/\eta$ // $b(h_{(i)}.s) = w(h_{(i)}.s)$
  $b' = b' \cup \{(h_{(i+1)}.s, w(h_{(i+1)}.s))\}$
end for
return b'

Algorithm 3 SelectAction(Belief b, Set of Episodes $H_{(b)}$, Index i

$N_+(b) =$ #episodes in $H_{(b)}$ for which $b(h_{(i)}.s) > 0$ // $b(h_{(i)}.s) = w(h_{(i)}.s)$
Calculate $W(b)$ and $W(b,a)$, $\widetilde{W}(b,a)$, $\widehat{Q}(b,a)$ for each $a \in \mathcal{A}$
$A_{unvisited} = \{a \in \mathcal{A} \mid \widetilde{W}(b,a) = 0\}$
if $|A_{unvisited}| > 0$ **then**
  return (random action from $A_{unvisited}$, True)
end if
$a = \arg\max_{a \in \mathcal{A}} \left(\widehat{Q}(b,a) + c \sqrt{\frac{\log(N_+)}{\widetilde{W}(b,a)}} \right)$
return (a, False)

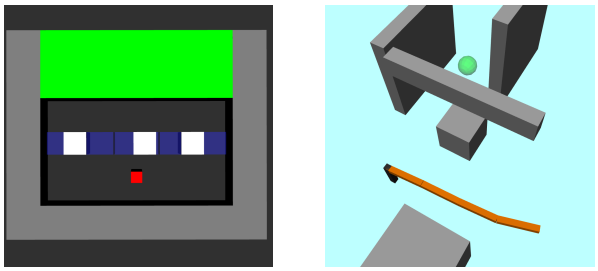
IV. EXPERIMENTS AND RESULTS

To evaluate LABECOP, we tested and compared it with POMCPOW, POMCP, ABT and DESPOT on three POMDP problems with continuous observation spaces, including an agent operating in a light-dark environment, a navigation problem with environment uncertainty and a motion-planning problem under uncertainty problem for a 4DOFs torque-controlled manipulator. The problem scenarios are detailed below.

A. Problem Scenarios

1) LightDark1D: The LightDark1D scenario is a simple benchmark problem introduced in [7] in which an agent navigates inside a one-dimensional discrete domain. The state space of the agent is the set of integers and it has access to four actions $\mathcal{A} = \{-10, -1, 0, 1, 10\}$ to navigate deterministically inside the domain. Executing $a = 0$ results in a terminal state and is rewarded by 100 at $s = 0$, but penalised by -100 at every other state. Additionally, the agent receives a penalty of -1 for every step it takes. Initially the agent is uncertain about its exact location but it receives noisy one-dimensional continuous observations regarding its location that are more accurate in the vincinity of $s = 10$. The discount factor is 0.95. More details and an illustration of the problem can be found in [7].

2) Passage: A robot equipped with a noisy and expensive range sensor operates on a 2D-plane populated by three obstacles and a border region. The robot must navigate from a known initial state to a goal area while avoiding collisions with the obstacles or the border region. The robot has perfect information regarding its own position, the border region as well as the goal area, but the locations of the obstacles are uncertain. The robot has access to 5 actions $\mathcal{A} = \{FORWARD, LEFT, RIGHT, SCAN\}$, where the first three actions move the robot deterministically in the respective direction. The *SCAN* action activates the range sensor and produces a three-dimensional continuous observation, consisting of the euclidean distances of the robot to the three obstacles. These observations become more precises as the robot navigates closer to an obstacle. Note that the robot only receives an observation when it executes the *SCAN* action. The problem terminates upon colliding with the environment, which is penalised by -1,000 or reaching the goal area, which is rewarded by 1,000. Activating the range sensor results in a penalty of -50 and every step incurs a small penalty of -1. Here the discount factor is 0.95. Fig. 2(a)



(a) (b)

Fig. 2. The Passage scenario (a) and the 4DOF-Factory scenario (b).

illustrates the scenario, where the red square represents the robot, the light-grey and green areas represent the border region and the goal area respectively and the white squares are the obstacles. The shaded blue areas represent the initial (uniform) distributions of the obstacles.

3) 4DOF-Factory: A torque-controlled manipulator with 4 revolute joints must move from an initial state to a state where the end-effector lies inside a goal area (colored green in Fig. 2(b)), without colliding with any of the obstacles. The state space is the joint product of joint-angles and joint-velocities. The control inputs are the joint-torques. The action space is set to be the maximum and minimum possible joint torque, resulting in 16 discrete actions affected by control errors. The dynamics of the manipulators are simulated by the ODE physics engine [25] with integration step size $\delta t = 0.004s$ (in [16] we studied a variant of this problem with a significantly more expensive transition function where we used $\delta t = 0.0001s$). The observations are noisy too and consist of the position of the end-effector inside the robot's workspace and joint velocities, resulting in a 7-dimensional continuous observation space. The initial state is known exactly, for which the joint angles and velocities are zero. The robot enters a terminal state and receives a reward of 1,000 upon reaching the goal. To encourage the robot to reach the goal area quickly, it receives a small penalty of -1 for every other action. Collision result in a terminal state too and are penalised by -500. The discount factor is 0.98.

B. Experimental setup

All three problem scenarios and the solvers were implemented in C++ using the OPPT-framework [26]. For ABT we used the implementation provided by OPPT, whereas for DESPOT we used the code released by the original authors (https://github.com/AdaCompNUS/despot). Since the version of POMCPOW released by the authors is written in the Julia programming language, which is incompatible with OPPT, we re-implemented POMCPOW in C++. However, our implementation of POMCPOW achieved better results in the LightDark1D problem than reported in [7]. For POMCP we used our own implementation within OPPT.

All simulations were run single-threaded on an Intel Xeon Silver 4110 CPU with 2.1Ghz and 128GB of memory. We used a fixed planning time of 1s for each solver for the LightDark1D and Passage problems and 2s for the 4DOF-Factory problem. Note that POMCP, ABT and DESPOT require a discretised observation space. For the LightDark1D problem we used a grid-based discretisation whereas for the Passage and 4DOF-Factory problems we used a distance based discretisation, i.e. two observations are considered equal if their euclidean distance is smaller than a given threshold.

For all problems and solvers we first ran a set of systematic preliminary trials to determine the best parameters for each solver. The parameters and their respective values are shown in Table I.

Despite out best efforts, we were unable to obtain results for DESPOT for the 4DOF-Factory problem. The reason is that DESPOTs strategy of expanding each belief using every action via the forward simulation of K scenarios is computationally too demanding for this problem due to its large action space and expensive transition function. In our experiments it took, on average, $\sim$3s to expand a single belief using $K = 50$ scenarios (a tenth of what is commonly used [12]), which is already more than the allowed planning time of 2s.

To study the effect of discretising the observation space on the performance of POMCP, ABT and DESPOT, we first ran a set of experiments in which we varied the discretisation granularity for each problem, i.e. we varied the grid-size in the LightDark1D problem and the observation distance threshold in the Passage and 4DOF-Factory problems. We then ran a set of experiments using the best discretisation granularity for POMCP, ABT and DESPOT and compared the results with POMCPOW and LABECOP using 1,000 simulation runs. The results are shown and discussed in the next section.

	LightDark1D	*Passage*	*4DOF-Factory*
LABECOP			
c	20.0	420.0	10.0
POMCPOW			
c	80.0	500.0	17.0
k_o	3.25	15.25	1.0
α_o	0.01	0.01	0.01
POMCP			
c	40.0	400.0	20.0
ABT			
c	10.0	550.0	20.0
DESPOT			
numScenarios	500	300	–
searchDepth	10	8	–

TABLE I

HYPERPARAMETERS FOR EACH SOLVER USED IN THE EXPERIMENTS. c IS THE UCB1 EXPLORATION CONSTANT. k_o AND α_o ARE THE PROGRESSIVE WIDENING PARAMETERS FOR POMCPOW.

C. Results

Fig. 3 shows the average total discounted reward achieved by each solver in all three test scenarios as we vary the discretisation granularity of the observation space. Since LABECOP and POMCPOW don't discretise the observation space, their results are added as horizontal lines. Table II shows the average total discounted rewards for POMCPOW and LABECOP as well as for POMCP, ABT and DESPOT where we used the best observation space discretisation granularities for the latter three solvers.

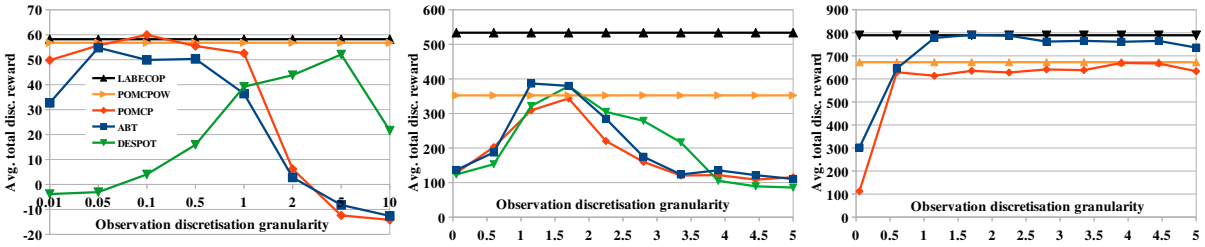


Fig. 3. Average total discounted reward of LABECOP, POMCPOW, POMCP, ABT and DESPOT on the LightDark1D (a), Passage (b) and 4DOF-Factory (c) scenarios. The x-axis represents the discretisation granularity of the observation space (i.e. the size of a grid cell for the LightDark1D problem and the observation distance threshold for the Passage and 4DOF-Factory problems) from fine to coarse, whereas the y-axis represents the average total discounted reward. Note that LABECOP and POMCPOW don't discretise the observation space. Therefore their results are included as horizontal lines.

	LightDark1D	*Passage*	*4DOF-Factory*
LABECOP	58.2 ± 1.1	$\mathbf{533.6 \pm 17.2}$	789.4 ± 17.4
POMCPOW	56.8 ± 1.1	352.3 ± 40.4	672.4 ± 25.2
POMCP	$\mathbf{59.6 \pm 1.1}$	343.6 ± 36.7	669.0 ± 19.7
ABT	54.8 ± 1.4	386.8 ± 39.9	$\mathbf{791.2 \pm 18.4}$
DESPOT	52.1 ± 1.7	379.0 ± 38.4	–

TABLE II

AVERAGE TOTAL DISCOUNTED REWARD AND 95% CONFIDENCE INTERVALS ACHIEVED BY LABECOP, POMCPOW, POMCP, ABT AND DESPOT IN THE THREE PROBLEM SCENARIOS. THE AVERAGE IS TAKEN OVER 1,000 SIMULATION RUNS FOR EACH SOLVER AND SCENARIO. THE BEST RESULT FOR EACH SOLVER AND PROBLEM IS HIGHLIGHTED IN BOLD.

Looking at the results for the LightDark1D and 4DOF-Factory problems, it might seem surprising that solvers (POMCP for LightDark1D and ABT for 4DOF-Factory) that rely on discretising the observations space outperform POMCPOW and slightly outperform LABECOP. LightDark1D has a one-dimensional observation space that is fairly easy to discretise, although the discretisation window in which POMCP outperforms the other methods is quite narrow as seen in Fig. 3(a). For 4DOF-Factory the results in Fig. 3(c) indicate that discretising the observation space causes no issues for POMCP and ABT despite the 7-dimensional continuous observation space, as long as the granularity is not too fine. The reason is that in this problem, it is paramount that the solver has a lookahead of at least 6 steps in order to compute a strategy that avoids collisions with the obstacles. Therefore finer discretisation granularities of the observation space hurt POMCP and ABT due to the increased branching factor of the lookahead trees. Additionally, the transition and observation errors in this problem are small, leading to many sampled observations and their resulting beliefs to be very similar. This enables POMCP and ABT to aggregate observations into 1-3 observation edges at most without a significant performance penalty.

Although in both scenarios the observation space is rather easy to discretise, LABECOP still performs comparable to the best solver in each respective scenario. This indicates that the performance penalty of considering *every* sampled observation for LABECOP is small, even for problems where this is not necessary.

In the Passage problem LABECOP clearly outperforms the other methods as seen in Fig. 3(b). Due to the large initial uncertainty regarding the locations of the obstacles, the robot must plan with respect to many possible observations produced by the `SCAN` action, each of them leading to beliefs with different potential passages between the obstacles. If too many beliefs are aggregated via a coarse observation space discretisation, it is unlikely that the robot discovers a strategy that works well for all the aggregated beliefs. If the discretisation is too fine, the branching factor of the lookahead trees used by POMCPOW, POMCP, ABT and DESPOT increases drastically. Since the robot requires multiple `SCAN` actions to find a safe passage to the goal area, the lookahead becomes too short to discover the long-term benefit of collecting observations. In both cases the robot avoids the `SCAN` action altogether.

LABECOP mitigates both issues. By considering *every* sampled observations during planning, LABECOP can plan with respect to many different sequences of beliefs. At the same time, LABECOP doesn't "branch" over observations, resulting in longer lookaheads. Both properties enable LABECOP to quickly discover the long-term benefit of multiple observation-gathering actions. As a result, LABECOP significantly outperforms the other methods in this scenario.

V. CONCLUSIONS

Despite tremendous progress in on-line POMDP planning, computing robust policies for problems with continuous observation spaces remains challenging. State-of-the-art solvers typically rely on discretising the observation space or limiting the number of observations that are considered during planning, which often impairs robustness. In this paper we present Lazy Belief Extraction for Continuous Observation POMDPs (LABECOP), a new on-line POMDP solver designed to alleviate these shortcomings. LABECOP samples an maintains a set of episodes that is used to lazily extract beliefs and a corresponding action-selection strategy on-the-fly as it samples new episodes, enabling LABECOP to consider *every* sequence of beliefs it encounters during planning, without having to discretise the observation space or maintain a lookahead tree. This results in significant gains in robustness for problems that require reasoning about many different sequences of beliefs. Experiments on three test scenarios indicate that LABECOP performs similar or better than state-of-the-art on-line solvers for problems with continuous observation spaces.

REFERENCES

[1] E. J. Sondik, "The optimal control of partially observable markov decision processes," Ph.D. dissertation, Stanford, California, 1971.

[2] C. H. Papadimitriou and J. N. Tsitsiklis, "The complexity of markov decision processes," *Mathematics of operations research*, vol. 12, no. 3, pp. 441–450, 1987.

[3] H. Bai and D. Hsu, "Unmanned aircraft collision avoidance using continuous-state pomdps," *Robotics: Science and Systems VII*, vol. 1, pp. 1–8, 2012.

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

[5] K. Hsiao, L. P. Kaelbling, and T. Lozano-Perez, "Grasping pomdps," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 4685–4692.

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

[7] Z. N. Sunberg and M. J. Kochenderfer, "Online algorithms for pomdps with continuous state, action, and observation spaces," in *Twenty-Eighth International Conference on Automated Planning and Scheduling*, 2018.

[8] H. Kurniawati, D. Hsu, and W. S. Lee, "Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces," in *In Proc. Robotics: Science and Systems*, 2008.

[9] J. Pineau, G. Gordon, and S. Thrun, "Point-based Value Iteration: An anytime algorithm for POMDPs," 2003.

[10] T. Smith and R. Simmons, "Point-based POMDP algorithms: Improved analysis and implementation," 2005.

[11] D. Silver and J. Veness, "Monte-carlo planning in large POMDPs," in *Advances in neural information processing systems*, 2010, pp. 2164–2172.

[12] A. Somani, N. Ye, D. Hsu, and W. S. Lee, "Despot: Online pomdp planning with regularization," in *Advances in neural information processing systems*, 2013, pp. 1772–1780.

[13] H. Kurniawati and V. Yadav, "An online pomdp solver for uncertainty planning in dynamic environment," in *Proc. Int. Symp. on Robotics Research*, 2013.

[14] K. M. Seiler, H. Kurniawati, and S. P. Singh, "An online and approximate solver for pomdps with continuous action space," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 2290–2297.

[15] E. Wang, H. Kurniawati, and D. P. Kroese, "An on-line planner for pomdps with large discrete action space: A quantile-based approach," in *ICAPS*. AAAI Press, 2018, pp. 273–277.

[16] M. Hoerger, H. Kurniawati, and A. Elfes, "Multilevel monte-carlo for solving pomdps online," in *Proc. International Symposium on Robotics Research (ISRR)*, 2019.

[17] A.-A. Agha-Mohammadi, S. Chakravorty, and N. M. Amato, "Firm: Feedback controller-based information-state roadmap-a framework for motion planning under uncertainty," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 4284–4291.

[18] R. He, E. Brunskill, and N. Roy, "Puma: Planning under uncertainty with macro-actions." in *Proceedings of the National Conference on Artificial Intelligence*, vol. 2, 2010.

[19] H. Kurniawati, Y. Du, D. Hsu, and W. S. Lee, "Motion planning under uncertainty for robotic tasks with long time horizons," *The International Journal of Robotics Research*, vol. 30, no. 3, pp. 308–323, 2011.

[20] N. P. Garg, D. Hsu, and W. S. Lee, "Despot-alpha: Online pomdp planning with large state and observation spaces." in *Proc. of Robotics: Science and Systems*, 2019.

[21] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, "Continuous upper confidence trees," in *International Conference on Learning and Intelligent Optimization*. Springer, 2011, pp. 433–445.

[22] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *IEEE Transactions on signal processing*, vol. 50, no. 2, pp. 174–188, 2002.

[23] T. Li, S. Sun, J. M. Corchado, T. P. Sattar, and S. Si, "Numerical fitting-based likelihood calculation to speed up the particle filter," *International Journal of Adaptive Control and Signal Processing*, vol. 30, no. 11, pp. 1583–1602, 2016.

[24] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256, May 2002.

[25] R. Smith, "Open dynamics engine," http://www.ode.org/.

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