An Online POMDP Solver for Uncertainty Planning in Dynamic Environment

Hanna Kurniawati and Vinay Yadav

Abstract Motion planning under uncertainty is important for reliable robot operations in uncertain and dynamic environments. Partially Observable Markov Decision Process (POMDP) is a general and systematic framework for motion planning under uncertainty. To cope with dynamic environment well, we often need to modify the POMDP model during runtime. However, despite recent tremendous advances in POMDP planning, most solvers are not fast enough to generate a good solution when the POMDP model changes during runtime. Recent progress in online POMDP solvers have shown promising results. However, most online solvers are based on replanning, which recompute a solution from scratch at each step, discarding any solution that has been computed so far, and hence wasting valuable computational resources. In this paper, we propose a new online POMDP solver, called Adaptive Belief Tree (ABT), that can reuse and improve existing solution, and update the solution as needed whenever the POMDP model changes. Given enough time, ABT converges to the optimal solution of the current POMDP model in probability. Preliminary results on three distinct robotics tasks in dynamic environments are promising. In all test scenarios, ABT generates similar or better solutions faster than the fastest online POMDP solver today; using an average of less than 50 milliseconds of computation time per step.

1 Introduction

Motion planning under uncertainty is important for reliable robot operation in imperfectly known and dynamic environments. Partially Observable Markov Decision Process (POMDP) is a general and systematic framework for planning under uncertainty. Motion planning under uncertainty problems can be modelled as POMDPs quite naturally. However, solving a POMDP problem exactly is computationally intractable [16]. A lot of effort and tremendous progress have been made in developing efficient approximate POMDP solvers [4, 7, 8, 13, 17, 18, 21, 23], such that today, we have POMDP solvers that can solve simple to moderately difficult motion planning problems within seconds to minutes [9, 10, 12]. However, many of these solvers are not efficient enough to recompute or update its solution when the

Hanna Kurniawati (e-mail: hannakur@uq.edu.au)
Robotics Design Laboratory, School of Information Technology and Electrical Engineering, University of Queensland, Australia.

Vinay Yadav (e-mail: vinayyadav.iitkgp@gmail.com)
Department of Electrical Engineering, Indian Institute of Technology, Kharagpur, India.
All work were done while the author was an internship student at University of Queensland.

POMDP model changes during runtime. Such changes in the POMDP model are often required when a robot operates in dynamic environment. This paper proposes a new approximate POMDP solver that improves and updates its solution online, following changes in the environment.

In imperfectly known and dynamic environment, a robot rarely knows its exact state due to errors in control and sensing. POMDP provides a systematic way to reason about the best action to perform when perfect state information is unavailable. It finds the best action with respect to the set of states that are consistent with the available information so far. The set of states is represented as a probability distribution, called a belief b, and the set of all possible beliefs is called the belief space B. A POMDP solver calculates an optimal policy $\pi^* : B \rightarrow A$ that maps a belief in B to an action in the set A of all possible actions the robot can perform, so as to maximize a given objective function. An offline POMDP solver computes this mapping prior to execution, while an online POMDP solver computes the mapping during runtime.

Methods that use POMDP framework to solve planning in imperfectly known and dynamic environment can be classified into two approaches. The first approach embeds all possible environments and their dynamics as part of the POMDP model. It uses an offline POMDP solver to find a good policy, prior to execution. When the environment and its dynamics are largely unknown, this approach constructs POMDP models too huge to be solved by even the best offline solver today.

The second approach models only the known part of the environment and its dynamics (both stochastic and deterministic), and allows the model to change during execution when more information about the environment becomes available. Key to the success of this approach is an efficient online POMDP solver that can compute a good policy during runtime, following changes in the POMDP model.

Online POMDP solvers have advanced significantly over the past few years [7, 8, 20, 21]. However, most of these solvers [7, 8, 20] are based on replanning, which recompute the best action to perform from scratch at each step, discarding any policy that has been computed so far. As a result, these solvers often waste significant computational resources when changes happen gradually or only to some part of the environment, which are often the case in robotics tasks.

This paper proposes a new online POMDP solver, called Adaptive Belief Tree (ABT), that reuses and improves existing policy at each time step, and update the policy as needed whenever the POMDP model changes. To enable fast policy update, ABT uses the following two observations. First, a change in the POMDP model is directly reflected as a change in the robot's behaviour at a particular set of states. Second, a change in one optimal mapping $\pi^*(b)$ from a belief b, may affect the optimal policy $\pi^*(.)$ at other beliefs that can reach b. Using insight from these observations, ABT represents the policy as pairs of belief and action, and explicitly represents the relation between beliefs, states, and their reachability information, so that it can quickly identify subset of the policy affected by changes in the POMDP model and update the policy fast whenever necessary. To quickly generate a good policy, ABT plans with respect to only a set of representative sampled beliefs. It represents each belief as a set of state particles, and samples a belief b by sampling a set of state trajectories from a particle of the given initial belief b_0. An effective strat-

egy for sampling state trajectories enables ABT to converge to an optimal policy in probability, and quickly generate a good policy. Preliminary results on three distinct robotics tasks in dynamic environment indicate that ABT can generate similar or better motion strategies faster than the best online POMDP solver today [21]. In all three test scenarios, ABT requires an average of less than 400 ms of preprocessing time, and an average of less than 50 ms of online computation time per step.

Furthermore, ABT is designed for POMDP problems with continuous state space and uses a generative model. A generative model is a black box simulator that enables us to generate experiences about the system dynamic and behaviour at various different states. By using a generative model, ABT does not need an explicit model on control error, observation error, and uncertainty about the system dynamics, which are often difficult to obtain in complex robotics tasks.

2 Related work

2.1 POMDP background

A POMDP is defined as a tuple $\langle S, A, O, T, Z, R, b_0, \gamma \rangle$, where S is the set of states, A is the set of actions, and O is the set of observations. At each step, the agent is in a state $s \in S$, takes an action $a \in A$, moves from s to an end state $s' \in S$, and perceives an observation $o \in O$. Due to action uncertainty, the system dynamic from s to s' is represented as a conditional probability function $T(s,a,s') = f(s'|s,a)$. Furthermore, due to sensing uncertainty, after performing action a and ends at state s', the observation that may be perceived by the agent is represented as a conditional probability function $Z(s',a,o) = f(o|s',a)$. At each step, the agent receives a reward $R(s,a)$, if it takes action a from state s. The agent's goal is to choose a suitable sequence of actions that will maximize its expected total reward, while the agent's initial belief is denoted as b_0. When the sequence of actions has infinite length, we specify a discount factor $\gamma \in (0,1)$ so that the total reward is finite and the problem is well defined.

In many problems with large state space, explicit representation of the conditional probability functions T and Z may not be available. However, one can use a *generative model*, which is a black box simulator that outputs an observation perceived, reward received, and next state visited when the agent performs the input action from the input state.

A POMDP planner computes an *optimal policy* that maximizes the agent's expected total reward. A POMDP policy $\pi\colon B \to A$ assigns an action a to each belief $b \in B$. A policy π induces a value function $V(b,\pi)$ which specifies the expected total reward of executing policy π from belief b, and is computed as $V(b,\pi) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t,a_t)|b,\pi]$. A policy can be represented by various representations, e.g., policy-graph [3] and pairs of belief and action [25]. However, most online solvers do not maintain an explicit representation of the policy. Instead, they calculate the mapping from beliefs to actions on the fly. In contrast, our proposed online solver maintains an explicit representation of a subset of the policy, and improves and updates the policy on the fly.

To execute a policy π, an agent executes action selection and belief update repeatedly. For example, if the agent's current belief is b, it selects the action referred to by $a = \pi(b)$. After the agent performs action a and receives an observation o according to the observation function Z, it updates b to a new belief b' given by $b'(s') = \tau(b,a,o) = \eta Z(s',a,o) \int_{s \in S} T(s,a,s') ds$ where η is a normalization constant. We use generative model and represents each belief with a set of particles. The belief update is approximated using particle filter.

2.2 Related POMDP solvers

POMDP is a systematic and general approach for planning under uncertainty. Although solving a POMDP exactly is computationally intractable [16], the past few years have seen tremendous increase in the capability of both offline and online POMDP solvers [13, 21, 23], such that POMDP approach is now practical for solving simple to moderately difficult motion planning problems.

The fastest offline POMDP solvers today are based on point-based approach [13, 17, 22, 23]. This approach reduces the complexity of planning in the belief space B by representing B as a set of sampled beliefs and planning with respect to this set only. To generate a policy, most point-based POMDPs use value iteration, utilizing the fact that the optimal value function satisfies Bellman equation. They start from an initial policy, represented as a value function V. And iteratively perform Bellman backup on V at the sampled beliefs, i.e., $V(b) = \max_{a \in A} \left(R(b,a) + \gamma \sum_{o \in O} \tau(b,a,o) \hat{V}^*(\tau(b,a,o)) \right)$ where $\hat{V}^*(b')$ is the current best value of b'. The iteration is performed until it converges. Over the past few years, impressive progress have been gained by improving the strategy for sampling B [13, 23] and utilizing problem structures [15]. Different approaches have also been proposed for restricted types of uncertainty, e.g., [4, 19] for Gaussian beliefs.

Many online solvers have been proposed too [20]. One of the fastest general online POMDP solvers today is POMCP [21]. Starting from the current belief b, POMCP performs best first search in the belief space. It samples action sequences to quickly focuses on parts of the belief space that is most promising for generating the optimal policy from b. As any sampling based method, the sampling strategy is crucial. POMCP frames the problem of sampling the most promising action sequences as a problem of balancing exploration and exploitation, often called multi-armed bandit problem [24], and uses the Upper Confidence Bounds1 (UCB1) algorithm [1] to select the actions. After POMCP finishes the search, it performs the best action, updates the robot's belief, and repeats the procedure until the goal is reached.

Aside from POMCP, various approaches have been proposed for online POMDP solvers. PUMA [8] and RBSR [7] perform best first search in the belief space, just as POMCP does. However, instead of solving a multi arm bandit problem, PUMA and RBSR sample action sequences using heuristics in the state space, assuming that states are fully observed after an action is performed. The work in [18] plans with respect to only the most likely observation and then replan at each step. Recent work, LQG-Obstacles [5] is very fast, but restricts the belief to be Gaussian and is designed specifically for collision avoidance problems. Our new method is designed as a general online POMDP solver and the beliefs can be any type of distribution.

When the POMDP model changes, in general, the above offline and online solvers recompute the policy from scratch, wasting all computational effort that have been performed so far. In contrast, our new solver ABT can reuse and improve the policy, as well as update it as needed during runtime.

Recent work [14] has proposed a point-based method to modify a pre-computed policy. However, the time it needs to update a policy is too slow to be practical for dynamic environment, and the types of model changes that can be handled are limited to changes in transition, observation, and reward functions. In contrast, our new method can handle any types of changes in the model, and is fast enough to update a policy online, as presented in Section 5.3.

3 Policy Representation

ABT's policy representation must support fast identification on which parts of the policy are affected by changes in the POMDP model, and must support fast update of the policy. To this end, we note two observations. First, a change in any element of the POMDP model can be identified from changes in the robot's behaviour at a particular set of states. By definition, any change in the state space can be identified as addition, reduction, or changes in the types of a set of states. When the transition, observation, or reward function changes, then in the long run, the results of performing an action, the observations perceived, or the reward the robot received at a set of states would change. Therefore, changes in these functions can be identified from changes in the robot's behaviour at a particular set of states, too. Changes in action and observation spaces will affect the transition and observation functions of a set of states, and therefore these changes can be identified from the changes in the robot's behaviour at a particular set of states too. The second observation is a change in an optimal policy $\pi^*(b)$ from a belief $b \in B$ may change the optimal policy $\pi^*(.)$ from other belief(s) that can reach b.

Utilizing the above observations, ABT represents the policy as pairs of belief and action, and explicitly represents the relation between beliefs, states, and their reachability information. This representation helps to quickly identify a subset of the policy that needs to be updated, and to quickly update it. To maintain the relation, ABT represents each belief as a set of state particles, and associates each belief b with state trajectories that reach a particle of b from a particle of the given initial belief b_0. The details of the representation are below.

ABT maintains a set H of sampled episodes. An episode $h \in H$ is a sequence of quadruples (s,a,o,r) of state $s \in S$, action $a \in A$, observation $o \in O$, and immediate reward $r = R(s,a)$. To sample an episode h, ABT samples an initial state $s_0 \in S$ from a given initial belief b_0 and selects an action $a_0 \in A$. The details of action selection are discussed in Section 4.1. After an action a_0 is selected, ABT calls the generative model to sample an observation $o_0 \in O$, an immediate reward r_0, and a next state s_1 when the agent performs a_0 at s_0. ABT inserts the quadruple (s_0, a_0, o_0, r_0) as the first element of h, and iteratively repeats the above steps starting from s_1. The iteration stops after either a terminal state is reached or h has exceeded a certain length. As a last step, ABT inserts $(s,–,–,r)$ as the last element of h, where s is the

next state sampled by the last call to the generative model and $r = R(s)$ is the reward of being at state s.

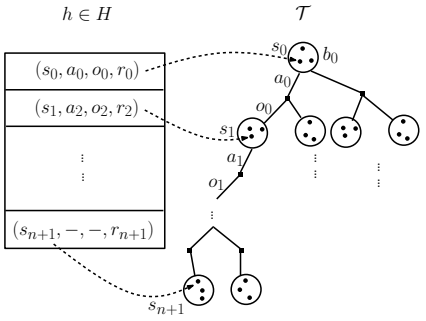

Fig. 1 Illustration of an association between an episode $h \in H$ and path in the belief tree $\mathcal{T}$.

To maintain an explicit relation between beliefs, states, and their reachability information efficiently, ABT maintains a belief tree, denoted as $\mathcal{T}$, and associates it with the set of episodes in H. Each node in $\mathcal{T}$ represents a belief. For writing compactness, we refer to the node and the belief it represents interchangeably. The root of $\mathcal{T}$ represents the initial belief b_0. Each edge $\overline{bb'}$ in $\mathcal{T}$ is labelled by a pair of action and observation *a–o*. An edge $\overline{bb'}$ with label *a–o* means that when a robot at belief b performs action a and perceives observation o, its next belief would be b', i.e., $b' = \tau(b,a,o)$ where $b, b' \in B$, $a \in A$, and $o \in O$.

The paths in the belief tree $\mathcal{T}$ are associated with the episodes in H. Suppose ϕ is a path in $\mathcal{T}$ and $\phi = \langle b_0, a_0, o_0, \ldots, a_n, o_n, b_{n+1} \rangle$, where $b_i, b_{n+1} \in B$, $a_i \in A$, and $o_i \in O$ for $i \in [0,n]$. Then, ϕ is associated with the set of episodes $H_\phi \subseteq H$ which consists of all episodes in H that contains $\langle (s_0, a_0, o_0, *), \ldots, (*, a_n, o_n, *), (*, -, -, *) \rangle$, where s_0 is any state sampled from b_0, a_i and o_i ($i \in [0,n]$) are the corresponding actions and observations in ϕ, and $*$ means any relevant value. Figure 1 illustrates the relation between an episode in H and a path in the belief tree $\mathcal{T}$. *Each episode in H corresponds to exactly one path of $\mathcal{T}$, but a path of $\mathcal{T}$ may be associated with many episodes.*

Each belief in $\mathcal{T}$ is represented by a set of particles, which comprises the states in the corresponding quadruples of the corresponding episodes. Suppose b is a node at level-l of $\mathcal{T}$ (the root has level 0). Suppose $\Phi(b)$ is the set of all paths in $\mathcal{T}$ that starts from the root and contains the node b, and $H_b = \bigcup_{\phi \in \Phi(b)} H_\phi$. Then, b is approximated with the set of particles $\{h_l.s \mid h \in H_b\}$, which comprises the state in the l^{th} quadruple of each episode in H_b (the quadruples are indexed from 0).

The policy π of ABT is embedded in the belief tree $\mathcal{T}$, with

$$\pi(b) = \arg\max_{a \in A(E,b)} \hat{Q}(b,a) \tag{1}$$

$$\text{and value} \quad V(b,\pi) = \max_{a \in A(E,b)} \hat{Q}(b,a) \tag{2}$$

where $b \in B$, E is the set of edges in $\mathcal{T}$, and $A(E,b) \subseteq A$ is the set of actions that have been used to expand b, i.e., the actions that labelled the out-edges of b in $\mathcal{T}$. The value $\hat{Q}(b,a)$ denotes the estimated Q-value. Q-value $Q(b,a)$ is the value of performing action a from belief b and continuing optimally afterwards, i.e., $Q(b,a) = R(b,a) + \gamma \sum_{o \in O} \tau(b,a,o) V^*(\tau(b,a,o))$. ABT estimates $Q(b,a)$ as

$$\hat{Q}(b,a) = \frac{1}{\left|H_{(b,a)}\right|} \sum_{h \in H_{(b,a)}} V(h,l) \tag{3}$$

where $H_{(b,a)} \subseteq H$ is the set of all episodes associated with all paths in $\mathcal{T}$ that start from b_0 and contains the sequence (b, a), l is the depth level of b in $\mathcal{T}$, and $V(h,l)$ is the value of an episode h starting from the l^{th} element. $V(h,l)$ is computed as

$$V(h,l) = \sum_{i=l}^{|h|} \gamma^{i-l} R(h_i.s, h_i.a) \tag{4}$$

where γ is the discount factor and R is the reward function. Note that each state in the l^{th} quadruple of each episode in $H_{(b,a)}$ is a particle of b and the action in that quadruple is the action a. Therefore, it is clear that $\hat{Q}(b,a)$ approximates the first component of $Q(b,a)$ well. However, it may seem odd that eq. (3)-(4) can approximate the second component of $Q(b,a)$, which is $\sum_{o \in O} \tau(b,a,o) V^*(\tau(b,a,o))$, as $V(h,l+1)$ for different h may correspond to different policy. It turns out by using an appropriate action selection strategy when sampling the episodes, one can ensure that as the number of episodes in $H_{(b,a)}$ increases, $V(h,l+1)$ converges to $\sum_{o \in O} \tau(b,a,o) V^*(\tau(b,a,o))$ in probability. This convergence result is based on the convergence result of POMCP [21]. The action selection strategy is discussed in Section 4.1 while the convergence result is discussed in Section 4.3.

The above policy representation and value calculation enable ABT to quickly identify and update the policy following changes in the POMDP model. To identify which parts of the policy need to be updated, ABT only needs to find the episodes in H that contain states that are affected by the changes in the POMDP model. To update the policy, ABT disconnects the association between each affected episode h and its corresponding nodes in $\mathcal{T}$, revises h according to the new POMDP model, and associates it back with the nodes of $\mathcal{T}$ (which may be different than the previously associated path). Then, ABT updates the values and Q-values of beliefs that have new association or disassociation with h. Using eq. (2)-(4), the value and Q-value revisions require only simple arithmetic calculation. With proper data structure, these values can be updated incrementally, and finding the affected episodes and the process of association and disassociation can be done fast. Details on the identification and policy update process are presented in Section 4.2.

4 Offline and online policy computation and update

ABT starts by computing a good approximation to the optimal policy for the a priori POMDP model, offline. During runtime, if the environment and hence the POMDP model changes, ABT identifies subset of the policy that needs to be updated and updates it. Otherwise, ABT improves its current policy. Algorithm 1 presents an overview of ABT.

4.1 Sampling the episodes

The key strategy in generating an initial policy (GENERATE-POLICY function) and improving a policy (IMPROVE-POLICY function) are the same, which is in sampling the episodes. The overall sampling strategy of ABT is in Algorithm 2.

To sample a new episode, ABT starts by sampling a particle state $s \in S$ from a given starting belief $b_{start} \in B$. For the offline policy generation GENERATE-POLICY function, the starting belief is always the given initial belief, while for

Algorithm 1 Adaptive Belief Tree (b_0)

PREPROCESS (OFFLINE)
(H, $\mathcal{T}$) = GENERATE-POLICY(P_0, b_0). {P_i is the POMDP model at time-i.}
Let S' be the set of all sampled states in H, i.e., $S' = \{h_i.s \mid i \in [0, |h|], h \in H\}$
Let $\mathcal{R}$ be a range tree representation of S'.
$b = b_0$.

RUNTIME (ONLINE)
while running **do**
  if $P_t \neq P_{t-1}$ **then**
    H' = IDENTIFY-AFFECTED-EPISODES(P_{t-1}, P_t, H, $\mathcal{R}$, $\mathcal{T}$).
    REVISE-EPISODES(P_t, $\mathcal{T}$, b, H').
    UPDATE-VALUES($\mathcal{T}$, b, H').
  while there is still time **do**
    IMPROVE-POLICY(P_t, H, $\mathcal{R}$, $\mathcal{T}$, b).
  a = Get best action in $\mathcal{T}$ from b.
  Perform action a.
  o = Get observation.
  $b = \tau\,(b, a, o)$.
  $t = t + 1$.

IMPROVE-POLICY, the starting belief is the belief at the current time. After a state is sampled, ABT selects an action and uses the generative model to sample an observation, reward, and next state (line 7–15, 20–30).

To select an action from state $s \in S$ that corresponds to node b in $\mathcal{T}$, ABT uses two strategies. First is the UCB strategy [1, 21], which is used when all actions in A have been used to expand b at least once (line 8–9). UCB strategy frames the action selection problem from each node as a multi-arm bandit problem. Multi-arm bandit problem is a reinforcement learning problem to select a sequence of actions, so as to maximize the total reward when the rewards for selecting the actions are not known in advance. This problem is essentially a problem of balancing exploration and exploitation, i.e., should one uses the action that has shown good performance so far even though it may not be the best action (exploitation) or should one tries other actions that have not shown good performance but may actually be the best action (exploration). To select an action using UCB strategy, ABT uses UCB1 algorithm [1], which selects an action according to

$$a = \arg\max_{a \in A} \left(\hat{Q}(b,a) + c \sqrt{\frac{\log(|H_b|)}{|H_{(b,a)}|}} \right) \tag{5}$$

where H_b is the set of episodes in H that has been associated with b, $H_{(b,a)}$ is the set of episodes in H that corresponds to sequence (b,a), $|.|$ is size of a set, and c is a scalar factor that determines the ratio between exploration and exploitation. UCB1 algorithm is one of the best multi-arm bandit solutions when the reward of performing an action follows a stationary distribution, which may not be known in advance. UCB1 algorithm has also been used for action selection by the fastest online POMDP solver today [21], and has been shown to enable convergence to the optimal policy [11, 21].

Algorithm 2 SAMPLING-AN-EPISODE(P, $\mathcal{T}$, b_{start}, H, ε)

1: $b = b_{start}$
2: Let l be the depth level of node b in $\mathcal{T}$.
3: Let s be a state sampled from b.
   The sampled state s is essentially the state at the l^{th} quadruple of an episode $h' \in H$.
4: Initialize h with the first l elements of h'.
5: Initialize doneMode as UCB.
6: Let A be the action space of POMDP model P.
7: **while** $\gamma^l > \varepsilon$ AND doneMode == UCB **do**
8: Let A' be the set of actions that labelled the edges from b in $\mathcal{T}$.
9: **if** $|A'| == |A|$ **then**
10: a = UCB-ACTION-SELECTION($\mathcal{T}$, b).
11: **else**
12: a = an action sampled from $A \backslash A'$ uniformly at random.
13: doneMode = Rollout.
14: (o, r, s') = GenerativeModel(P, s, a).
15: Insert (s, a, o, r) to h.
16: Add $h_l.s$ to the set of particles that represent belief node b and associate b with h_l.
17: $s = s'$
18: b = child node of b via an edge labelled a-o. If no such child exist, create the child.
19: $l = l + 1$.
20: **if** doneMode == Rollout **then**
21: Let p be a number sampled uniformly at random from $[0, 1]$.
22: **if** $p < p_{policy}$ **then**
23: r = ROLLOUT-POLICY($\mathcal{T}$, s, b).
24: rolloutUsed = policy.
25: **else**
26: r = ROLLOUT-DET(P, s).
27: rolloutUsed = deterministic.
28: **else**
29: r = GenerativeModel(P, s).
30: Insert $(s, -, -, r)$ to h.
31: Add $h_l.s$ to the set of particles that represent belief node b and associate b with h_l.
32: valueImprovement = UPDATE-VALUES($\mathcal{T}$, h)
33: p_{policy} = UPDATE-ROLLOUT-PROB(rolloutUsed, valueImprovement).
34: Insert h to H.

When the condition for using UCB is not satisfied, ABT selects an action towards satisfying the condition of UCB using rollout strategy. To select an action from state $s \in S$ that corresponds to node b of $\mathcal{T}$, rollout strategy samples an action a uniformly at random from the set of actions that has not been used to expand b (line 12).

Furthermore, to generate a good policy fast, ABT also tries to compute a good estimate of the Q-value $Q(b, a)$ in its rollout strategy (line 21–27). A good estimate of the Q-value will help the UCB strategy to converge faster to the optimal action once the condition to run UCB strategy has been satisfied. If the time to generate or improve the policy has run out before the condition to run UCB is satisfied, a good estimate of the Q-value helps ABT choose a good action.

To estimate the Q-value during rollout, ABT uses two heuristics. Suppose rollout strategy selects action $a \in A$ to be performed from state $s \in S$ that corresponds to node b of $\mathcal{T}$. *The first heuristic to estimate $Q(b, a)$ assumes the problem is deterministic* (line 26). ABT uses methods from deterministic motion planning to find a good

solution. It calculates the total reward if the robot starts from state s, performs action a, and continues optimally, assuming the system is deterministic. This total reward is the estimated Q-value of this heuristic and the output of ROLLOUT-DET in line 26. *The second heuristic is based on existing policy* (line 23). For this purpose, ABT first uses the generative model to sample an observation $o \in O$, computes the belief $b' = \tau(b,a,o)$ using particle filter, and finds the node in $\mathcal{T}$ nearest to b'. Any distance metric for distributions can be used. ABT assumes that the state space is a metric space, which is mostly the case for robotics tasks, and defines the distance between two beliefs as the expected state space distance assuming the two beliefs are independent. Suppose the nearest node to b' is $\widehat{b'}$. If the distance between b' and $\widehat{b'}$ is more than a given threshold, ABT uses ROLLOUT-DET to estimate the Q-value. Otherwise, ABT assumes that b' is equal to $\widehat{b'}$. It simulates the robot's movement according to the policy embedded in $\mathcal{T}$ starting from $\widehat{b'}$, until a leaf node of $\mathcal{T}$ is reached. The total discounted reward gathered during this simulation becomes this heuristic's estimate of the Q-value $Q(b,a)$ and the output of ROLLOUT-POLICY in line 23.

Now, the question is which heuristic should be used at a particular rollout operation. This problem is similar to the action selection problem discussed earlier in this section, and similarly ABT frames the problem of choosing which heuristic to use as a multi-arm bandit problem. However for simplicity, this selection is valid globally instead of per belief node as in the case with action selection. Due to this simplification, we cannot use UCB1, as it assumes that the rewards follow a stationary distribution. Instead, we use Exp3 [2], one of the best multi-arm bandit solutions when no statistical assumptions are made about the underlying reward function. Furthermore, Exp3 has been shown to be competitive to the strategy that uses the best action at each step. Using Exp3, ABT assigns a probability to each heuristic strategy, and selects which heuristic to use based on this probability (line 22). The probability is adapted based on how much the heuristic improves the value of the starting belief (line 33), as follows

$$p_i(t+1) = (1-c_r)\frac{w_i(t+1)}{w_1(t+1)+w_2(t+1)} + \frac{c_r}{2}$$
$$\text{where} \quad w_i(t+1) = w_i(t)\exp\left(\frac{c_r\left(\max(0, V_t(b_{start}) - V_{t-1}(b_{start}))\right)}{2p_i(t)}\right) \quad (6)$$

where $i \in [1,2]$ indicates the different heuristics, t is the current time step, and $c_r \in (0,1)$ is the ratio between exploration and exploitation.

4.2 Handling changes in the POMDP model

When the POMDP model changes, ABT identifies a subset of the policy that is affected by the changes (IDENTIFY-AFFECTED-EPISODES function) and updates it (REVISE-EPISODES and UPDATE-VALUES functions).

To identify a subset of the policy that are affected by changes in the POMDP model, ABT needs to identify the set $S_{ch} \subseteq S$ of states affected by the changes, i.e.,

all states $s \in S$ where the robot's behaviour in s changes. In this paper, we do not focus on how to identify changes in the POMDP model. Instead, we assume that either changes in the POMDP model can be identified easily by identifying changes in the environment map, or the user provides information on the set of affected states S_{ch}.

Once the set S_{ch} of affected states is known, ABT finds the set of episodes affected by the changes. An episode $h \in H$ is affected by the changes in the POMDP model if at least one of its state element is affected by the changes, i.e., $\exists i \in [0, |h|] \;\; h_i.s \in S_{ch}$.

To facilitate fast identification of affected episodes, ABT structures the set $S' \subseteq S$ of all sampled states, i.e., the set of all states in each sampled episode $S' = \{h_i.s \mid i \in [0, |h|], h \in H\}$, in a range tree denoted as $\mathcal{R}$. Since multiple episodes may contain the same state, ABT labels each state $s \in S'$ with a set of two-tuple (h, idx) indicating which episodes h of H and which index idx element of h contain s.

To identify episodes in H that are affected by the changes in the POMDP model, ABT finds the intersection between the set S_{ch} of affected states and the set S' of sampled states. For this purpose, ABT constructs a bounding rectangle for each connected component of S_{ch}. Here, rectangle is used in a general sense, referring to hyper-rectangle when the dimension of S is more than two. Then, ABT solves a rectangular query on the range tree $\mathcal{R}$ for each bounding rectangle and checks if the states resulting from the rectangular queries are indeed in S_{ch}. The results of the rectangular queries that lie in S_{ch} are sampled states that are affected by the changes in the POMDP model. The two-tuple labels associated with these sampled states indicate the episodes that are affected by the changes in the POMDP model.

Assuming the range tree has been constructed, the above identification procedure takes $O\left(\log^{dim(S)} |S'| + k + |H'|\right)$, where $dim(S)$ is the dimension of the state space S and k is the total number of states outputted by all rectangular range queries [6]. The first construction of the range tree, which happens offline, takes $O(|S'_0| \log^{dim(S)-1} |S'_0|)$, where S'_0 is the set of all sampled states right after the offline policy generation. During runtime, the time to insert the state of the newly sampled quadruple to the range tree is $O(\log^{dim(S)-1} |S'_t|)$, where S'_t is the set of all sampled states at time t.

Once the set $H' \subseteq H$ of affected episodes are identified, ABT revises affected elements of all episodes in H' according to the new POMDP model. Suppose $h \in H'$ is an affected episode to be revised. First, ABT finds the lowest element index $\underline{idx}$ of h where the state is affected by model changes. Then, ABT revises the episode starting from element index $idx_u = \max(0, \underline{idx} - 1)$ of h until the last element. When $idx_u = 0$, ABT erases the episode h from H, because in this case the entire episode needs to be revised, which means the episode is not reusable and the results would be similar as if ABT samples an entirely new episode. If $idx_u > 0$, ABT uses the generative model to re-sample the sequence of observations perceived, rewards received, and next states visited, when the sequence of actions from element idx_u until the last element of h is performed, starting from the state in element idx_u of h. When this sequence of actions is obviously sub-optimal, e.g., when it causes the robot to collide with a newly added obstacle, ABT uses heuristics to modify the sequence

of actions. The heuristics is the same as that used in ROLLOUT-DET (line 26 of Algorithm 2). It assumes the problem is deterministic and uses methods from deterministic motion planning to find a good sequence of actions. Finally, ABT replaces the content of the quadruples of h with the re-sampled sequence of observations, rewards, and next states, at the respective indices. This revision of $h \in H'$ triggers re-computation of the values and Q-values of all beliefs in $\mathcal{T}$ that correspond to h, and hence update the policy embedded in $\mathcal{T}$.

4.3 Convergence to an optimal policy

ABT converges in probability to the optimal policy from the current belief under the current POMDP model.

First, let us discuss the case when the POMDP model does not change. Key to ABT's convergence is the strategy for sampling the episodes H (Section 4.1), in particular the action selection strategy. ABT frames the problem of selecting an action when sampling an episode, as a multi-arm bandit problem and uses UCB1 algorithm. This action selection strategy has been proven to enable convergence to the optimal policy in probability, regardless of the rollout strategy being used [21]. In fact, [21] has shown that $\hat{Q}(b,a)$ will converge to the optimal Q-value with a rate of $O(\log(|H_{\Gamma(b)}|)/|H_{\Phi(b)}|)$, where $H_{\Phi(b)} \subseteq H$ is the set of episodes that correspond to paths in $\mathcal{T}$ that starts from b_0 and contains node b, and $|.|$ is the size of a set. When $\hat{Q}(b,a)$ converges to the optimal Q-value, the value $\hat{V}(b)$ converges to the optimal value function and the corresponding $\pi(b)$ converges to the optimal policy.

When the POMDP model changes, the existing policy that has been revised can be considered as an initial policy. When enough time is given to improve the policy, the number of sampled episodes keeps increasing, such that the quality of the policy will eventually be dominated by the results of the episode sampling strategy. Therefore, the above results remain valid when the POMDP model changes.

5 Experiments

5.1 Robotics tasks

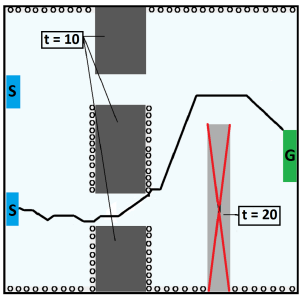

Fig. 2 Underwater Navigation. $|S_{t=0...9}| = 2,652$, $|S_{t\geq 10}| = 2,274$, $|A| = 5$, $|O_{t=0...9}| = 104$, $|O_{t=10...19}| = 142$, $|O_{t\geq 20}| = 138$. Black line is a path generated by ABT.

We have tested ABT on three robotics tasks that require the POMDP model to be modified several times during runtime. To ensure that changes in the environment affect the solution to the problem regardless of how fast or slow a solver is, we define changes in terms of time steps. The three test scenarios are as follows.

Underwater navigation. An Autonomous Underwater Vehicle (AUV) navigates in an environment populated by obstacles and vortices that are not known a priori. In the beginning, we only know the start and goal regions, and the positions of underwater beacons where the AUV can localize perfectly (labelled 'O'). Dur-

ing run time, at time step 10, the obstacles (dark grey) become known. These obstacles possess unique features that can be used by the AUV to localize itself, but at the same time obstruct some of the underwater beacons. To reflect these new information, the POMDP model is modified; the number of states reduces while the number of observation increases. At time step 20, the vortex (light grey with cross mark) becomes known. The POMDP model again changes to reflect the vortex.

The AUV may start from one of the two possible regions labelled 'S' and needs to reach the goal region labelled 'G' while avoiding obstacles and being dragged in a vortex region. The environment is represented as a uniform grid of size 51×52. At each step, the AUV can move one cell in 5 directions, i.e., East, North, South, Northeast, and Southeast. Due to underwater current, the AUV movement is accurate only 80% of the time. The rest of the time, it reaches the left or right of the intended destination with equal probability. The AUV does not have a GPS, but can localize perfectly at cells marked by 'O'. At other cells, no observations are perceived. The AUV receives a high reward for reaching the goal, a small penalty for every action taken, and a high penalty for being in the vortex region.

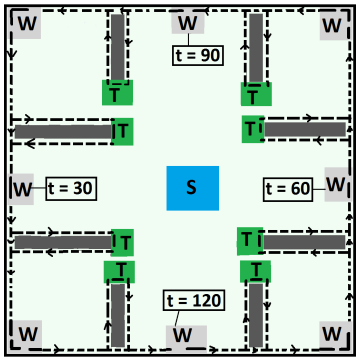

Fig. 3 Homecare. $|S| = 1{,}926{,}912$, $|A| = 9$, $|O| = 200{,}000$.

Homecare. A robot is deployed for caretaking purposes in a home environment. The robot needs to find and attends to an elderly whenever she needs assistance. The elderly moves around in the house, starting from one of the regions labelled 'T'. Her motion is non-deterministic: At each step, she may pause or continue following one of several paths (marked by dashed line). She is likely to stop for longer time in places marked 'W', which represents washroom, regions near dining table, TV, or refrigerator. At time step 30, 60, 90, and 120, new furnitures and appliances where the elderly may pause longer are added (labelled 'W'). To reflect these changes, the transition function of the POMDP model is modified accordingly.

The environment is populated by obstacles (dark grey) and is represented as a uniform grid of size 50×50. The robot starts from a region marked by 'S'. At each step, the robot may stay or move one cell in one of the 8 wind directions. Its motion is accurate only 80% of the time. The rest of the time, it reaches the left or right of the intended destination with equal probability. The elderly calls the robot whenever assistance is needed and turns off the call when assistance is no longer needed. When a call is made, the elderly pause at her current location until the robot comes or until assistance is no longer needed. The robot receives a high reward whenever it reaches the elderly when assistance is still needed. A small penalty is imposed for every move the robot takes to discourage it from wasting energy. The robot has access to four types of observations. First is a GPS. Second is a visibility sensor that enables the robot to localize the elderly exactly if she is within 1 cell away from the robot. Third is from four sensors mounted on the ceiling. These sensors divide the home into four equal regions and can identify which region the elderly is in with

90% accuracy. The rest of the time, the sensor wrongly identifies the region where the elderly is in with equal probability. Last is observation on whether the elderly requires assistance, which is 100% accurate.

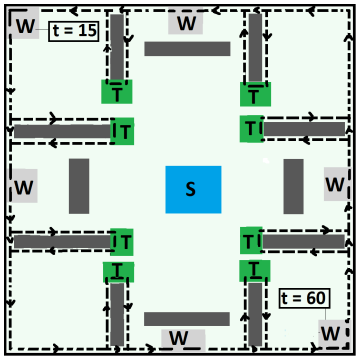

Fig. 4 Target Finding. $|S| = 923,520$, $|A| = 9$, $|O| = 100,000$.

Target finding. This problem is similar to homecare, but here the goal is for the robot to quickly find the elderly, while she is moving in the house. The environment is slightly more complex than homecare, due to additional obstacles. Similar to homecare, the elderly behaviour changes with the addition of new furnitures and appliances. The changes (time step 30 and 60) are reflected in the transition function. The robot's dynamics are the same as in homecare. The robot's observations are also the same as homecare, but without observation on whether the elderly needs assistance.

5.2 Experimental setup

We implement ABT in C++ and test it on the above tasks. To calculate the quality of the motion strategies generated by ABT, we estimate the expected total reward of using ABT to solve each task. To this end, for each task, we first ran a few trial runs to determine the best parameters for ABT to use. Then, we use the best parameters for each task to generate 30 different offline policies. Finally, for each task and each policy, we run 100 simulation runs and computes the total reward of each simulation. The average of these 3,000 simulation runs is the estimated expected total reward.

As a comparator, we also apply POMCP [21], the fastest online POMDP solver today, on the above tasks. For POMCP, we use the software released by the original author, which is written in C++. Similar to ABT, for each task, we first ran a few trial runs to determine the best parameters for POMCP to use. These parameters include the use of additional heuristic to help POMCP's rollout function performs better (knowledge option in POMCP software). We use the best parameters to run 500 simulation runs for each task. The average of these simulation runs is the estimated expected total reward generated by POMCP for solving the task.

All experiments are conducted in a PC with Intel Xeon E5-1620 3.6GHz processor and 16GB RAM.

5.3 Results

The results of ABT and POMCP are in Table 1. All values in Table 1 are in the form of average $\pm$ 0.95 confidence interval. POMCP recomputes the solution at each step using 1,024 particles (note: The POMCP s/w always recomputes from scratch). ABT improves the solution using an additional 1,000 or 2,500 unweighted particles per step. The first column shows the expected total discounted reward. The second column shows the time ABT uses for preprocessing, to generate an initial policy for the a priori POMDP model. POMCP does not perform any preprocessing. The last column shows the average online computation time ABT and POMCP use at

	Total Discounted Reward	Offline Computation Time (ms)	Online Computation Time Average Time/Step (ms)
Underwater navigation			
POMCP*	138.98 ± 47.77	–	754.00 ± 11.23
ABT [1]	185.50 ± 38.23	364.00 ± 45.96	42.90 ± 0.25
Homecare			
POMCP*	2,251.04 ± 275.98	–	1,933.75 ± 13.80
ABT [1]	2,297.34 ± 102.91	63.33 ± 5.82	16.12 ± 0.11
ABT [2]	2,509.38 ± 104.53	63.33 ± 5.82	39.98 ± 0.15
Target finding			
POMCP*	2,237.12 ± 142.32	–	1,221.95 ± 6.59
ABT [1]	2,284.69 ± 60.28	63.00 ± 5.65	15.12 ± 0.16
ABT [2]	2,594.35 ± 60.28	63.00 ± 5.65	44.61 ± 0.33

Table 1 Performance comparison. *: Use 1,024 particles/step. [1]: Improve with 1,000 particles/step. [2]: Improve with 2,500 particles/step.

each step. The average time per step for ABT includes the time to improve existing policy and to update the policy when the POMDP model changes. The average time to perform one policy update for underwater navigation is (87.42 ± 1.13)ms, while the average policy update of all ABT runs for homecare and target finding are less than 1.5 ms, which is below the timer accuracy of our computer system (4 ms).

The results show that in all three scenarios, ABT significantly outperforms POMCP. It can generate similar or better motion strategies up to $120\times$ faster than POMCP. By reusing existing policy, ABT can focus its search faster on parts of the belief space that are most promising for generating the best action strategy from the current belief under the current POMDP model.

In underwater navigation, ABT requires more offline and online computation time, compared to homecare and target finding, even though the size of state, action, and observation spaces are smaller. The reason is underwater navigation requires longer planning horizon and has more complex geometry, compared to the other two problems. As a result, it takes more time to compute the deterministic motion planning heuristic, one of the heuristics used to estimate the Q-value (step 26 of Algorithm 2). This computation is also the reason why policy update time for underwater navigation takes much longer than the other two problems. A more efficient implementation of the deterministic motion planner will reduce the offline and online computation time of underwater navigation.

6 Summary

This paper proposes a new online POMDP solver, called ABT, that reuses and improves existing policy, and updates the policy as needed whenever the POMDP model changes. It is designed for POMDP problems with continuous state space and uses a generative model. We have successfully tested ABT on three different robotics tasks in dynamic environment, where each task requires the POMDP model to change several times during runtime, so as to reflect the environment correctly. Simulation results on these test scenarios show that ABT generates similar or better motion strategies faster than the fastest online POMDP solver today. In all test scenarios, ABT requires an average of less than 400 ms of preprocessing time, and

an average of less than 50 ms of online computation time at each step. These results suggest that ABT brings POMDP a step closer to become practical for non-trivial robotics tasks in uncertain and dynamic environments, even when the environment dynamics are unknown in advance, a class of robotics tasks often deemed too difficult to be solved using POMDP approach.

References

1. P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, May 2002.
2. P. Auer, N. Cesa-Bianchi, Y. Freund, and R.E. Schapire. The non-stochastic multi-armed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2003.
3. H. Bai, D. Hsu, W.S. Lee, and A.V. Ngo. Monte Carlo Value Iteration for Continuous-State POMDPs. In *WAFR*, 2010.
4. J.v.d. Berg, P. Abbeel, and K. Goldberg. LQG-MP: Optimized Path Planning for Robots with Motion Uncertainty and Imperfect State Information. In *RSS*, 2010.
5. J.v.d. Berg, D. Wilkie, S.J. Guy, M. Niethammer, and D. Manocha. LQG-Obstacles: Feedback Control with Collision Avoidance for Mobile Robots with Motion and Sensing Uncertainty. In *ICRA*, 2012.
6. M.d. Berg, O. Cheong, M.v. Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2000.
7. K. Hauser. Randomized Belief-Space Replanning in Partially-Observable Continuous Spaces. In *WAFR*, 2010.
8. R. He, E. Brunskill, and N.Roy. PUMA: planning under uncertainty with macro-actions. In *AAAI*, 2010.
9. M. Horowitz and J. Burdick. Interactive Non-Prehensile Manipulation for Grasping Via POMDPs. In *ICRA*, 2013.
10. K. Hsiao, L.P. Kaelbling, and T. Lozano-Perez. Grasping POMDPs. In *ICRA*, pages 4685–4692, 2007.
11. L. Kocsis and C. Szepesvri. Bandit based monte-carlo planning. In *In: ECML-06. Number 4212 in LNCS*, pages 282–293. Springer, 2006.
12. H. Kurniawati, Y. Du, D. Hsu, and W.S. Lee. Motion planning under uncertainty for robotic tasks with long time horizons. *IJRR*, 30(3):308–323, 2011.
13. H. Kurniawati, D. Hsu, and W.S. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *RSS*, 2008.
14. H. Kurniawati and N.M. Patrikalakis. Point-Based Policy Transformation: Adapting Policy to Changing POMDP Models. In *WAFR*, 2012.
15. S.C.W. Ong, S.W. Png, D. Hsu, and W.S. Lee. Planning under uncertainty for robotic tasks with mixed observability. *IJRR*, 29(8):1053–1068, 2010.
16. C.H. Papadimitriou and J.N. Tsitsiklis. The Complexity of Markov Decision Processes. *Math. of Operation Research*, 12(3):441–450, 1987.
17. J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI*, pages 1025–1032, 2003.
18. R. Platt, R. Tedrake, T. Lozano-Perez, and L.P. Kaelbling. Belief space planning assuming maximum likelihood observations. In *RSS*, 2010.
19. S. Prentice and N. Roy. The Belief Roadmap: Efficient Planning in Linear POMDPs by Factoring the Covariance. In *ISRR*, 2007.
20. S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa. Online planning algorithms for POMDPs. *JAIR*, 32:663–704, 2008.
21. D. Silver and J. Veness. Monte-Carlo Planning in Large POMDPs. In *NIPS*, 2010.
22. T. Smith and R. Simmons. Heuristic search value iteration for POMDPs. In *UAI*, 2004.
23. T. Smith and R. Simmons. Point-based POMDP algorithms: Improved analysis and implementation. In *UAI*, July 2005.
24. R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2012.
25. S. Thrun. Monte carlo POMDPs. In *NIPS*, pages 1064–1070, 2000.