Point-Based Policy Transformation: Adapting Policy to Changing POMDP Models

Point-Based Policy Transformation: Adapting Policy to Changing POMDP Models

Hanna Kurniawati and Nicholas M. Patrikalakis

Abstract Motion planning under uncertainty that can efficiently take into account changes in the environment is critical for robots to operate reliably in our living spaces. Partially Observable Markov Decision Process (POMDP) provides a systematic and general framework for motion planning under uncertainty. Point-based POMDP has advanced POMDP planning tremendously over the past few years, enabling POMDP planning to be practical for many simple to moderately difficult robotics problems. However, when environmental changes alter the POMDP model, most existing POMDP planners recompute the solution from scratch, often wasting significant computational resources that have been spent for solving the original problem. In this paper, we propose a novel algorithm, called *Point-Based Policy Transformation (PBPT)*, that solves the altered POMDP problem by *transforming* the solution of the original problem to accommodate changes in the problem. PBPT uses the point-based POMDP approach. It transforms the original solution by modifying the set of sampled beliefs that represents the belief space B, and then uses this new set of sampled beliefs to revise the original solution. Preliminary results indicate that PBPT generates a good policy for the altered POMDP model in a matter of minutes, while recomputing the policy using the fastest offline POMDP planner today fails to find a policy with similar quality after two hours of planning time, even when the policy for the original problem is reused as an initial policy.

1 Introduction

Motion planning under uncertainty that can efficiently take into account changes in the environment is critical for robots to operate reliably in our living spaces. Changes in our living spaces are unavoidable. New furnitures are added, water current changes direction, etc. Despite these changes, robots need to reliably accomplish the given tasks in the midst of various control and sensing errors. For example, imagine an Autonomous Underwater Vehicle (AUV) navigating around highly cluttered offshore oil platforms. Water currents that highly accentuates the AUV's con-

Hanna Kurniawati.
School of Information Technology & Electrical Engineering,
University of Queensland.
e-mail: hannakur@uq.edu.au.
Nicholas M. Patrikalakis.
Department of Mechanical Engineering, Center for Ocean Engineering,
Massachusetts Institute of Technology.
e-mail: nmp@mit.edu.

trol error, changes throughout the day. Many of these changes are very significant, e.g., a 180^0 changes in the currents direction, making an optimal motion strategy at a particular time may actually be a bad strategy when the AUV operates at a different time. Despite the criticality of these changes, most changes are often limited to localized regions. In this preliminary work, we are interested in local changes and assume that the changes are known prior to execution.

To handle uncertainty, PBPT uses the Partially Observable Markov Decision Process (POMDP), which is a systematic and general framework for motion planning under uncertainty. Due to uncertainty, a robot never knows its exact state. Therefore, a POMDP planner computes the best action to perform with respect to a set of states that are consistent with the available information so far. Each set of states is represented as a distribution over the state space, called a belief, and the set of all beliefs is called the belief space B. The action to perform is encoded as a mapping from beliefs to actions, called a policy. Once generated, the optimal policy can be used as a feedback controller for the robot. It is true that solving a POMDP exactly is computationally intractable [15]. However by trading optimality with approximate optimality for speed, point-based POMDP approach [16] has tremendously sped-up POMDP planning, enabling it to be practical for many simple to moderately difficult robotics problems [11, 22].

Now, when the environment changes, the POMDP model changes too. Methods that can be used to handle changes in the POMDP model can be classified into two extreme approaches. First is replanning. Off-line replanning that recomputes the policy from scratch, using existing off-line POMDP planners, may waste significant computation time, especially when the changes are small but significant, and the problem is difficult such that even the fastest offline POMDP solvers today require hours to solve it. Recent replanning methods, such as [7, 8, 17, 21], are fast, as they are designed for on-line planning. But, these methods do not perform global planning in the belief space. Hence, they may cause the robot to fall into a catastrophic state when such state exists. This is undesirable for highly critical tasks. The second approach hedges over all possible changes [20]. It models all possible changes again as a POMDP problem. Each parameter that may change is modeled as a state variable of an enlarged POMDP problem. The enlarged POMDP problem can be solved using existing POMDP planners and the generated policy is optimal over all possible changes in the POMDP model. But, the enlarged problem can quickly become very large, beyond the capability of even the best POMDP solver today.

In this paper, we propose a novel algorithm, called *Point-Based Policy Transformation (PBPT)*, that transform a pre-computed POMDP policy according to changes in the POMDP model. By doing so, PBPT does not enlarge the POMDP problem that needs to be solved, and hence is faster than the hedging approach. Although slower than on-line replanning strategies. PBPT performs global planning in a restricted part of B and finds the best policy in a restricted class of policies with high probability, making it more suitable than on-line replanning for highly critical tasks.

PBPT uses the point-based POMDP approach. Key to point-based approach is that it represents the belief space B using a representative set of sampled beliefs. And then plans, i.e., performs Bellman updates, on only this set of beliefs, instead

of the entire B. Now, when the POMDP model changes, the representative set of beliefs is likely to change too. PBPT uses the difference between the new and the original POMDP models to transform the representative set of beliefs. It identifies beliefs that are affected by the changes and are unlikely to be part of a representative belief set in the new problem. It replaces such beliefs by re-sampling B. Once the set of representative beliefs is fixed, PBPT revises the policy by performing Bellman updates at selected beliefs to avoid unnecessary update operation.

Suppose a policy class Π is the set of all possible policies for the new POMDP model, where the mapping from unaffected beliefs is the same as the mapping from these beliefs in the original policy. Then, PBPT converges to a good approximation of the best policy in Π or better, with high probability. Furthermore, preliminary results indicate that PBPT generates a good policy for the revised POMDP model in a matter of minutes, while recomputing the policy using the fastest offline POMDP planner today fails to find a policy with similar quality after two hours of planning time, even when the original policy is reused as an initial policy.

2 Background and related work

2.1 POMDP background

A POMDP is specified 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. In each step, the agent is in a state $s \in S$, takes an action $a \in A$, and moves from s to an end state s'. Due to the uncertainty in action, the end state s' is modeled as a conditional probability density function $T(s,a,s') = f(s'|s,a)$. The agent may then receive an observation. Due to the uncertainty in observation, the observation result $o \in O$ is again modeled as a conditional probability density function $Z(s',a,o) = f(o|s',a)$. In each step, the agent receives a reward $R(s,a)$, if it takes action a from state s. The agent's goal is to maximize its expected total reward by choosing a suitable sequence of actions. 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.

A POMDP planner computes an *optimal policy* that maximizes the agent's expected total reward. A POMDP policy $\pi\colon B \to A$ prescribes an action a, given the agent's belief b. A policy π induces a value function $V(b,\pi)$ which specifies the expected total reward of executing policy π, and is computed as

$$V(b,\pi) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t,a_t)|b,\pi] \tag{1}$$

A policy can be represented by various representations, e.g., policy-graph [2], α-function [18], or pairs of belief and action [25] for continuous state space.

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

$$b'(s') = \tau(b,a,o) = \eta Z(s',a,o) \int_{s \in S} T(s,s,s')ds \tag{2}$$

where η is a normalization constant.

2.2 Related work

Many motion planners have been proposed to handle changes in the environment, e.g., [4, 9, 14, 23]. However, they do not consider partial observability of the system. The work in [13] considers a limited partial observability property, i.e., partial predictability of the environment, but they do not take into account errors in the robot's control and sensing.

POMDP is a systematic and general approach for planning under uncertainty. Although solving a POMDP exactly is computationally intractable [15], in the past few years, different approaches have been proposed to make POMDP planning more practical. Several work restricts the beliefs to be Gaussian, e.g., [3, 19].

Point-based POMDP does not restrict its distribution and has tremendously advanced POMDP planning. It reduces the complexity of planning in 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 [11, 16, 22]. Although these planners were designed for discrete state space, they can be extended to handle continuous state space, by replacing their policy representation and backup operation with policy representation and backup operation designed for continuous state space, e.g., [2, 18, 25]. However, most work in point-based POMDP does not handle changes in the model. We propose a point-based POMDP planner to handle changes in the POMDP model, where the changes are known prior to execution, by modifying a pre-computed optimal policy of the original problem.

The idea of policy modification resembles path deformation, e.g., [12]. But, most work in path deformation do not consider partially observable systems.

Modifying an optimal policy of one POMDP problem to solve another POMDP problem, can be considered as a transfer learning problem. However, most work in transfer learning focus on finding the mapping from one problem to another or from one solution to another [24]. We propose an algorithm to identify parts of the policy that needs to be modified and how to modify the policy, based on the differences between two POMDP models.

3 Overview

3.1 PBPT's goal

Before defining PBPT's goal more formally, let's first discuss the model changes and their effect on beliefs more formally. Suppose $P_o = \langle S, A, O, T_o, Z_o, R_o, b_0, \gamma \rangle$ is the original POMDP model. PBPT assumes that the state space S is a metric space, and can be continuous or discrete. It assumes that the action space A and observation space O are discrete. Suppose to accommodate environmental changes, the original model P_o is revised to a new POMDP model $P_n = \langle S, A, O, T_n, Z_n, R_n, b_0, \gamma \rangle$. PBPT can handle any changes in the POMDP model, as long as there is a bijection from the state, action, and observation spaces in the original model to their corresponding spaces in the revised model. To simplify notation, here we assume that the bijection is an identity map and use the same notation for these spaces in P_o and P_n. Furthermore, we assume that the initial belief b_0 does not change.

Changes from P_o to P_n affect a subset of S, which we denote as $S_{ch}(P_o, P_n) \subseteq S$ and define as,

$$S_{ch}(P_o, P_n) = \big\{ s \in S \mid \exists_{s' \in S, a \in A} T_o(s,a,s') \neq T_n(s,a,s') \vee \\ \exists_{a \in A, o \in O} Z_o(s,a,o) \neq Z_n(s,a,o) \vee \exists_{a \in A} R_o(s,a) \neq R_n(s,a) \big\}.$$

For writing compactness, we use S_{ch} to refer to $S_{ch}(P_o, P_n)$. PBPT assumes that S_{ch} consists of one or more connected components, where each connected component is a closed set, forming a simple polytope.

Now, we can use the notion of affected states to discuss affected beliefs. Let π_o^* be the optimal policy that has been pre-computed for P_o. Key to PBPT in revising a policy is to first revise the set of representative beliefs. The set of representative beliefs is sampled from the set $\mathscr{R}_o^*(b_0)$ of beliefs that are reachable from the initial belief $b_0 \in B$ under π_o^* in P_o. For writing compactness, we use $\mathscr{R}_o^*$ to refer to $\mathscr{R}_o^*(b_0)$. To identify which beliefs in the original set of representative beliefs need to be replaced, we classify $\mathscr{R}_o^*$ into three classes, i.e., *directly affected, indirectly affected, and unaffected*.

A belief $b \in \mathscr{R}_o^*$ is in the *directly affected* class, denoted as B_{ch}, whenever its support intersects S_{ch}. A belief $b \in \mathscr{R}_o^*$ is in the *indirectly affected* class, denoted as B'_{ch}, whenever its support does not intersect S_{ch} but at least one of the beliefs reachable from b under π_o^* is directly affected. More formally, the set of indirectly affected beliefs is $B'_{ch} = \{ b \in \mathscr{R}_o^* \mid support(b) \cap S_{ch} = \emptyset \wedge support(\mathscr{R}_o^*(b)) \cap S_{ch} \neq \emptyset \}$ where $support(\mathscr{R}_o^*(b)) = \bigcup_{b' \in \mathscr{R}_o^*(b)} support(b')$. Beliefs in $\mathscr{R}_o^*(b_0)$ that are not directly affected nor indirectly affected belong to the *unaffected* class, denoted as B_u.

This classification reflects the effect of model changes on the value function, too. Let $V_o(b, \pi_o^*)$ be the value of executing π_o^* from b in P_o, and $V_n(b, \pi_o^*)$ be the value of executing π_o^* from b in P_n. Then, the relation can be defined more formally as,

Lemma 1. *Suppose $P_o = \langle S, A, O, T_o, Z_o, R_o, b_0, \gamma \rangle$ changes to $P_n = \langle S, A, O, T_n, Z_n, R_n, b_0, \gamma \rangle$. For any $b \in B$, if $support(b) \cap S_{ch} = \emptyset$ and $support(\mathscr{R}_o^*(b)) \cap S_{ch} = \emptyset$, then $V_n(b, \pi_o^*) = V_o(b, \pi_o^*)$.*

Proof. The proof is straightforward by using induction on the planning horizon. For the base case (planning horizon 1), $V_n(b,\pi_o^*) = V_o(b,\pi_o^*)$ because the reward functions of performing any action from any state in $support(b)$ remain the same. Let's assume that $V_n(b,\pi_o^*) = V_o(b,\pi_o^*)$ for planning horizon h. For planning horizon $h+1$, $V_n(b,\pi_o^*) = R(b,\pi_o^*(b)) + \gamma \int_{o\in O} \tau(b,a,o) V(\tau(b,a,o)) do$ where $V(\tau(b,a,o))$ is the value of $\tau(b,a,o)$ computed using h planning horizon. Since b is not affected by any changes, $R(b,\pi_o^*(b))$ and $\tau(b,a,o)$ for any observation $o \in O$ remain the same. Using the assumption on planning horizon h, $V_n(b,\pi_o^*) = V_o(b,\pi_o^*)$. □

Using the above classification, we can define PBPT's goal more formally. Suppose $\Pi(\pi_o^*, P_o, P_n) = \left\{ \pi \in \Pi_n \mid \forall_{b \in B \setminus (B_{ch} \cup B'_{ch})} \pi(b) = \pi_o^*(b) \right\}$, where Π_n is the set of all possible POMDP policies for P_n. Then, PBPT's goal is to find the best policy, i.e., $\pi_n^* = \arg\max_{\pi \in \Pi(\pi_o^*, P_o, P_n)} V_n(b_0, \pi)$, from the policy class $\Pi(\pi_o^*, P_o, P_n)$.

3.2 Overview of the algorithm

Algorithm 1 Point-Based Policy Transformation (π_o^*, P_o, P_n)

```
1: Initialize T with a sampled representation of R*_o.
2: Classify the nodes of T into indirectly affected, directly affected, and unaffected nodes.
3: π_n = Initialize with π*_o.
4: while Termination condition not satisfied do
5:     Sample an indirectly affected node b from T.
6:     Let (b_0, b_1, ..., b_n, b) be the path from b_0 to b in T.
7:     NodeStackToBeBackup.clear();
8:     for i = 0 to n do
9:         NodeStackToBeBackup.push(b_i).
10:    NodeStackToBeBackup.push(b).
11:    while b is not unaffected AND expansion still likely to improve V_n(b_0, π_n) do
12:        Select an action a ∈ A and an observation o ∈ O.
13:        Let b' = τ (b, a, o).
14:        Insert b' as a child of b in T.
15:        NodeStackToBeBackup.push(b').
16:        b = b'.
17:    while NodeStackToBeBackup is not empty do
18:        b = NodeStackToBeBackup.pop()
19:        Backup(b, π_n).
```

Key to PBPT is to transform a policy by revising the set of sampled beliefs that represents B. Given an optimal policy π_o^* for the original problem P_o, PBPT starts by constructing a representative set of sampled beliefs for P_o (line 1 of Algorithm 1). To this end, PBPT samples the set $\mathscr{R}_o^*$ of beliefs reachable under π_o^* in P_o and represents them as a belief tree T. Each node of T represents a sampled belief and the root is b_0. PBPT represents each belief as unweighted particles. In this paper, we refer to the nodes of T and their corresponding beliefs interchangeably, and use the same notation. Each edge $\overrightarrow{bb'}$ represents a pair of action–observation a–o where $a \in A$ and $o \in O$, such that $b' = \tau(b,a,o)$. Details on how PBPT samples $\mathscr{R}_o^*$ are in Section 4.1.

Suppose P_n is the new problem. PBPT uses the difference between P_o and P_n to classify the nodes in T into the directly affected, indirectly affected, and unaffected

nodes (Figure 1), as discussed in Section 3.1 (line 2 of Algorithm 1). Details on how PBPT classifies the nodes are in Section 4.2.

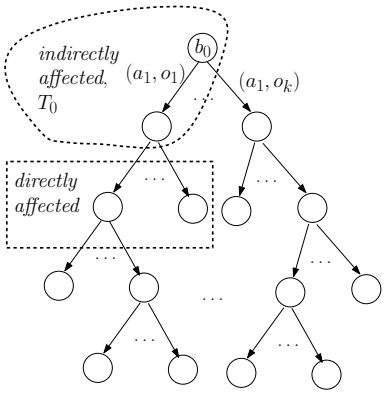


Fig. 1 The initial T. All out-edges of the same node would have the same action label, as the initial T is a sampled representation of $\mathscr{R}^*_o$. The nodes outside the rectangle and bounding curve are unaffected beliefs.

Once the classification is done, PBPT revises T by selecting an indirectly affected node to be expanded (line 5 of Algorithm 1), and performs deep sampling from the selected node (line 11–16 of Algorithm 1). Suppose $\mathscr{R}^*_n$ is the set of beliefs reachable from b_0 under an optimal policy π^*_n for the new problem P_n. Then to revise T, ideally we want to replace nodes of T that are not in $\mathscr{R}^*_n$ with beliefs that lie in $\mathscr{R}^*_n$, and for efficiency, we want to replace the smallest number of nodes possible. Of course, we do not know $\mathscr{R}^*_n$ in advance, as knowing this is the same as knowing π^*_n. But we can use the observation that, even when the changes from P_o to P_n decrease the reward of all states in S_{ch} by much, an indirectly affected belief b may still have a high value, and hence lies in $\mathscr{R}^*_n$, if after reaching b, there is a strategy for the robot to maneuver, to avoid reaching beliefs whose supports intersect S_{ch}. Utilizing this observation, PBPT selects an indirectly affected node to start further expansion of T. Details on how to choose an indirectly affected node for expansion and how a deep sampling is performed are in Section 4.4.

When an unaffected belief is sampled, a deep sampling is stopped (line 11 of Algorithm 1). By doing so, PBPT avoids recomputing the policy at beliefs where the mapping from the original policy does not need to be revised.

To approximate the set B_u of unaffected beliefs, PBPT constructs an end-game region around each unaffected node of T. An end-game region is the set of points that lie within a threshold distance away from an unaffected node of T. When a newly sampled belief falls inside the end-game region, the deep sampling is terminated, as we can and will reuse the mapping from π^*_o. The details on end-game region construction are discussed in Section 4.3.

To revise the policy, PBPT performs repeated backup operations starting from the newly inserted nodes all the way to the root (line 17–19 of Algorithm 1). By doing so, PBPT avoids unnecessary backup operations.

4 Point-Based Policy Transformation

4.1 Initializing the belief tree T

To initialize the belief tree T using a sampled representation of $\mathscr{R}^*_o$, PBPT simulates executing π^*_o from b_0 in P_o multiple times. The required number of simulation runs is discussed in Section 5. Remember that due to uncertainty, the robot does not

know the actual state it visited. Therefore, in each simulation run, PBPT maintains two traces. One is a state trace, representing the sequence of states visited by the robot. Another is a belief trace, representing the sequence of beliefs that estimates the sequence of states visited by the robot.

To start a simulation run, PBPT samples a state $s_0 \in S$ from b_0, and sets s_0 as the actual starting state of the robot. Then, it simulates the robot performing action $\pi^*_o(b_0)$. To simulate the action, PBPT samples the robot's next state $s \in S$ from $T(s_0, \pi^*_o(b_0), s)$, samples an observation $o \in O$ perceived by the robot from $Z(s, \pi^*_o(b_0), o)$, assigns the reward $R(s_0, \pi^*_o(b_0))$ to the robot, and computes the robot's next belief as $b = \tau(b_0, \pi^*_o(b_0), o)$. If the belief b is not yet a child of b_0 in T, PBPT inserts b as a child of b_0. Regardless of whether b is just inserted or is already available, the sampled state s is added to $S_B(b)$, a set of states that represent b's support. Next, PBPT continues the simulation and insertion process iteratively from b and s. This process is iteratively repeated for subsequent beliefs and states until a large number of simulation steps is performed.

4.2 Classifying the nodes of T

To identify nodes of T that are directly affected, PBPT finds the nodes whose support sets intersect S_{ch}. The main consideration here is the ability to handle many different changes to the original POMDP model P_o, efficiently. For this purpose, PBPT takes the union $\bigcup_{b \in T} S_B(b)$ of the support states, and structures them in a range tree data structure. PBPT also computes the bounding hyper-rectangle of each connected subset of the affected state space regions S_{ch}. For each bounding hyper-rectangle, PBPT queries the range tree. Then, it tests each state s in the query result whether s is in the connected subset bounded by the hyper-rectangle. For each s in S_{ch}, PBPT sets each belief b of T that has s in its $S_B(b)$ as a directly affected belief.

Once all directly affected nodes in T have been identified, the indirectly affected nodes are then all ancestors of each directly affected node that are not themselves directly affected. Nodes that are not directly nor indirectly affected are identified as unaffected nodes.

The above strategy requires additional time to construct the range tree, but once the range tree is constructed, finding directly affected nodes of T can be done fast. Suppose T contains N nodes, and the set $S_B(b)$ of each node b in T has size m. To construct the range tree structure, PBPT requires $O(Nm\,log^{d-1}(Nm))$ time [5], where d is the dimension of S. Now, suppose we are given S_{ch} that consists of M connected subset. For each subset, PBPT constructs a bounding hyper-rectangle. The time to query the states that lie inside a hyper-rectangle dominates the time for testing if a state s in a query result lies in S_{ch} and the time for finding the nodes of T that has s in its support. Therefore, the total time to find directly affected nodes is $O(M(\log^d(Nm) + k))$ where k is the largest number of query result.

4.3 Constructing the end-game region

Recall that the end-game region is the set of beliefs within a given threshold distance from at least one of the unaffected nodes of T. Before discussing the distance threshold, let's first discuss the metric we use.

To compute distance in B, PBPT uses the Earth Mover's Distance (EMD). EMD between two beliefs computes the minimum expected state space distance, where the minimum is taken over all possible joint distribution whose marginals are the two beliefs. More precisely,

$$D_B(b,b') = \inf_f \left\{ \int_{s\in S}\int_{s'\in S} D_S(s,s')f(s,s')dsds' \mid b = \int_{s'} f(s,s')ds', b' = \int_s f(s,s')ds \right\}$$

where $D_B(b,b')$ is the EMD between b and b' in B, $D_S(s,s')$ is the distance between s and s' in S, and f is a joint density function.

Most point-based POMDP solvers do not use EMD, instead they use L1 distance as metric in B The main difference between the two metrics is that EMD is dependent on the underlying state space metric, while L1 is not. This difference makes EMD a better indication of the value difference between two beliefs compared to L1, when the state space is a metric space and the behavior of the system at nearby states are similar, which is often the case in robot motion planning. The value function of a belief (eq. (1)) is computed based on two components, the belief and the expected total discounted reward. EMD incorporates both components when the reward function is Lipschitz continuous, while L1 can only incorporate one of them.

Using EMD in end-game region construction requires that nearby beliefs, under EMD, would have similar optimal value, such that with a small error, one belief can be used to represent other nearby beliefs. This requirement is indeed true, as the value function of beliefs in B with EMD satisfies Lipschitz condition, i.e.,

Theorem 1. *Suppose (S,D_S) and (B,D_B) are metric spaces, and for any state $s,s' \in S$ and any action $a \in A$, $|R(s,a)-R(s',a)| \leq C D_S(s,s')$. If $D_B(b,b') \leq \delta$, then $|V^*(b)-V^*(b')| \leq C\,\delta$ for any belief $b,b' \in B$.*

Proof. Let's first define $D'_S(s,s') = C D_S(s,s')$. When the metric in S is D'_S, the EMD of B becomes $D'_B(b,b') = C D_B(b,b')$.

Wlog, suppose the optimal policy is represented by a set Γ of α-functions. Let $Q_b(s,a) = R(s,a) + \gamma \int_{s'\in S} T(s,a,s') \sum_{o\in O} Z(s',a,o)\alpha^*_{b,a,o}(s')ds'$ where $\alpha^*_{b,a,o} = \arg\max_{\alpha\in\Gamma} \int_{s\in S} \alpha(s)\cdot\tau(b,a,o)(s)ds$. Then, the optimal value of $b \in B$ can be written as $V^*(b) = \max_{a\in A}\left(\int_{s\in S} Q_b(s,a)\cdot b(s)ds\right)$ and the value difference, is

$$\begin{aligned}\left|V^*(b)-V^*(b')\right| &= \max_{a\in A}\left(\int_{s\in S} Q_b(s,a)\cdot b(s)ds\right) - \max_{a\in A}\left(\int_{s\in S} Q_{b'}(s,a)\cdot b'(s)ds\right) \\ &\leq \max_{a\in A}\left(\int_{s\in S} Q_b(s,a)\cdot b(s)ds - \int_{s\in S} Q_b(s,a)\cdot b'(s)ds\right) \qquad (3)\end{aligned}$$

We can bound $Q_b(s,a)$ from above, such that $Q_b(s,a) \leq R(s,a) + \gamma\frac{R_{max}}{1-\gamma}$. Using this bound in eq. (3) gives us

$$\left|V^*(b)-V^*(b')\right| \leq \max_{a\in A}\left(\int_{s\in S} R(s,a)\cdot b(s)ds - \int_{s\in S} R(s,a)\cdot b'(s)ds\right) \quad (4)$$

Using the Kantorovich distance [6], i.e., $K(b,b') = \sup_{g\in Lip_1}\left(\int_{s\in S} g(s)b(s)ds - \int_{s\in S} g(s)b'(s)ds\right)$ where Lip_1 is the set of all 1-Lipschitz functions over S, we can bound eq. (4). Using metric D'_S, the reward function R satisfies 1-Lipschitz condition for any action $a \in A$, and so we have $|V^*(b)-V^*(b')| \leq K(b,b')$.

Since the above Kantorovich distance is the dual of EMD D'_B [6], $|V^*(b)-V^*(b')| \leq K(b,b') = D'_B(b,b') = CD_B(b,b')$. □.

In our prior work [10], we have shown Lipschitz condition for value function in B with metric EMD. But, the requirement in the above theorem is much more relaxed than the one in our prior work.

Since EMD is based on the metric in S, we can use heuristics on S to set the threshold distance for constructing end-game region in B. Constructing reasonable heuristics in S is much easier than in B. One heuristics that can be used is based on the intuition that the maximum expected future rewards are similar when the POMDP agent starts from nearby states where the transition and reward functions are the same. Using this heuristics, the threshold distance for a belief b is the maximum between the expected nearest distance to states with different transition and reward function and the expected threshold distance in S.

4.4 Sampling new beliefs

To select which node of T to start deep sampling from, we want to choose a combination of starting nodes that improves the value of b_0 the most in the least time possible. Of course, we do not know which combination of nodes would generate such improvement. Therefore, PBPT uses Exp3.S, an adaptive selection strategy based on sampling history that is guaranteed to be close to the best combination [1].

Suppose $T_0 = \{b_1, b_2, \ldots, b_K\}$. Using Exp3.S, at iteration-t PBPT selects a node $b_i \in T_0$ with probability

$$p_t(b_i) = (1-\beta)\frac{w_t(b_i)}{\sum_{j=1}^{K} w_t(b_j)} + \frac{\beta}{K} \quad (5)$$

where $w_t(b_i)$ is a weight that reflects how much starting deep sampling from b_i has improved the value of b_0 in the past and $\beta \in (0,1)$ is a fixed constant. Once a node b_i is sampled, PBPT starts a deep sampling from b_i. After the deep sampling is terminated, PBPT backups the nodes in the path traversed during deep sampling, starting from the last node inserted to T all the way to the initial belief b_0. Next, PBPT updates the probability in eq. (5) according to how much the value of b_0 improves. It updates the weight as,

$$w_{t+1}(b_i) = w_t(b_i)e^{\beta\frac{x_t(b_i)}{K}} + \frac{e}{MK}\sum_{i=1}^{K} w_t(b_i)$$

where $x_t(b_i) = \frac{V_2^t(b_0,\pi_2) - V_2^{t-1}(b_0,\pi_2)}{|R_{max}|/(1-\gamma)}$ if b_i was selected at iteration-t and 0 otherwise, $V_2^t(b_0,\pi_2)$ is $V_2(b_0,\pi_2)$ after iteration-t. Note that due to backup operation,

$V_2^t(b_0,\pi_2) \geq V_2^{t-1}(b_0,\pi_2)$, and so $x_t(b_i) \geq 0$. The notation M denotes the total number of deep sampling PBPT performs.

PBPT's deep sampling strategy is similar to SARSOP. SARSOP maintains an upper and lower bounds for each node in T. The upper bound is the expected value assuming deterministic action, while the lower bound is the value of the belief in the current policy. Given b, PBPT chooses an action $a \in A$ with the highest upper bound, and an observation $o \in O$ that is expected to reduce the gap between the upper and lower bound the most. A new belief b' is computed as $\tau(b,a,o)$ and then inserted to T. Then, the expansion continues starting from b' until either the gap between the upper and lower bound becomes too small to cause improvement on gap reduction on the root. PBPT stops the expansion that starts from b when the newly sampled belief is unlikely to reduce the gap at the root, or when the newly sampled belief is in the end-game region. When the newly sampled belief is in the end-game region, the mapping from belief to action from the original policy can be used.

5 Convergence

Now, the question is whether PBPT converges to the best policy in $\Pi(\pi_o^*, P_o, P_n)$. To discuss convergence, we need to consider three main components of PBPT. First, the strategy to exclude nodes in the end-game region from expansion and backup operations. Second, the strategy for selecting a starting node for deep sampling. Last, the strategy for deep sampling.

The main concern in excluding beliefs in the end-game region from expansion and backup operations is that, PBPT may misclassify a sampled belief to be in the end-game region, while it is actually not. Suppose B_e is the set of unaffected nodes in the initial T. To identify if a belief b in the initial T is in B_e or not, PBPT samples a set of state traces (Section 4.1), starting from a state sampled from the support of b. If state traces that intersects S_{ch} exist, but PBPT fails to sample even one of them, b will be wrongly identified as unaffected node, and hence wrongly included in the end-game region. Since PBPT will not improve the value of beliefs in the end-game region, this misclassification may cause PBPT to fail to converge to the best policy in $\Pi(\pi_o^*, P_o, P_n)$.

However, it turns out that with a small number of independently sampled state traces, the effect of misclassification in the end-game region can be kept low.

Theorem 2. *Suppose p_i is the probability that a state trace sampled from $\mathscr{R}_o^*(b_i)$ for any b_i in the initial T intersects S_{ch}, and $p_{min} = \min_{b_i \in B_{ch}} p_i$. Assume that each state trace from a node b in the initial T starts from a state in $support(b)$ and that the starting state is sampled independently from the same distribution b. If $\mathscr{R}_o^*(b)$ of each belief b in the initial T is represented by $n \geq \frac{\ln(\varepsilon\delta)}{\ln(1-p_{min})}$ state traces, then $P(\%$ nodes in the initial T that are misclassified as unaffected $< \varepsilon) \geq (1-\delta)$, where $\varepsilon, \delta \in (0,1]$ is a small constant.*

Proof. Let's first compute the smallest number n of sampled state traces needed to ensure that $P(b$ is misclassified$) \leq m$ for any belief $b \in B_e$ and $m \in [0,1]$. Assuming that each state trace starts from a state in $support(b)$ and is sampled independently from the same distribution b, we can compute n using binomial distribution with

success probability p_{min}. The result is that $n \geq \frac{\ln(m)}{\ln(1-p_{min})}$ state traces are needed to guarantee that $P(b \text{ is misclassified}) \leq m$.

Now, let X be the random variable that denotes the number of beliefs in T that are misclassified as unaffected nodes. We can represent X as a binomial distribution with success probability at most m, and computes

$$\begin{aligned} P(X < \varepsilon N) &= 1 - P(X \geq \varepsilon N) \\ &\geq 1 - \frac{Nm}{\varepsilon N} \end{aligned}$$

where N is the total number of nodes in the initial T. The last inequality is computed using Markov inequality. To ensure that $P(X < \varepsilon N) \geq (1-\delta)$, we need $m = \varepsilon\delta$. Inserting this value of m to the lower bound of n that ensures $P(b \text{ is misclassified}) \leq m$ yields the desired result. □.

The above theorem computes the smallest number of traces to guarantee small misclassification in B_e with high probability. But, the end-game region is larger than B_e, as it is the set of all beliefs within a threshold distance from at least one belief in B_e. However, since the value function satisfies Lipschitz condition (Theorem 1), when the threshold distance is small, the difference between $V_2(b, \pi_n^*)$ and $V_2(b', \pi_n^*)$, where b is in the end-game region and $b' \in B_e$ is b's nearest belief in B_e, is small too. Hence, the effect of misclassified beliefs in the end-game region to PBPT convergence depends mostly on the misclassification in B_e.

Opposite to the above discussion, some beliefs that are supposed to be in the end-game region may not be sampled in the initial T. Hence, these beliefs are not identified as part of the end-game region. When they are sampled during expansion of T, PBPT continues expanding and performing backups from them. As a result, PBPT may converge to a policy that is better than the best policy in $\Pi(\pi_o^*, P_o, P_n)$.

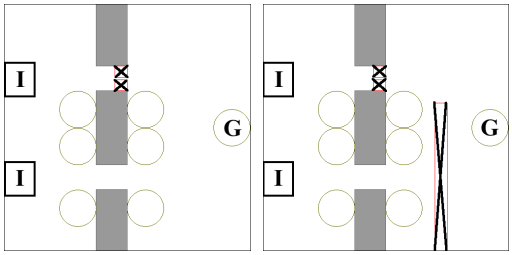
Convergence to better than the best policy in $\Pi(\pi_o^*, P_o, P_n)$ may also happens due to luck. When we use a global policy representation, such as α-functions, the value of beliefs in the end-game region may improve because of the backup operations at some farther beliefs that are not in the end-game region.

The last two components of PBPT do not worsen the convergence results. PBPT deep sampling strategy is the same as SARSOP, which converges to the optimal policy. But unlike SARSOP, PBPT starts sampling from an indirectly affected node in the initial T, instead of from b_0. Starting deep sampling not from b_0 may cause the sampling to be incomplete, in the sense that it may never cover the beliefs that are critical to generate an optimal policy. However, T_0 always contains b_0 and the strategy PBPT uses to sample a starting node from T_0 always assigns non-zero probability for each belief in T_0. Hence, PBPT sampling is complete, which means that PBPT strategies in the first two steps does not change the above convergence results.

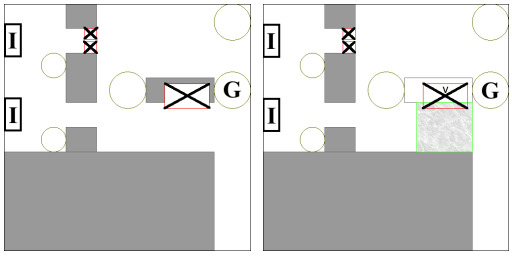
Since PBPT fails to converge only when there are misclassified beliefs in the end-game region, Theorem 2 shows that with a small number of state trace samples, PBPT converges to a good approximation of the best policy in $\Pi(\pi_o^*, P_o, P_n)$ or better, with high probability.

6 Experiments

6.1 Scenarios



(a) Scenario-1.



(b) Scenario-2.

Fig. 2 Scenarios. In each scenario, left: original problem, right: modified problem. The rectangles labeled 'I' is possible initial position of the AUV. The circle labeled 'G' is the goal region. The circle without labels are regions where the AUV can localize well. Rectangles marked with crosses are bounding rectangles of vortex center regions. In the right-most picture, rectangle colored light grey indicates the region significantly affected by a vortex, while v in the large crossed rectangle denotes the vortex center .

We tested PBPT on two scenarios (Figure 2) of an AUV navigating around offshore platforms, where the environment around the platform changes prior to the AUV deployment. In both scenarios, the AUV is set to operate at a particular depth. Hence, these are 2D navigation under uncertainty problems. Although this problem seems simple, the highly accentuated control error due to water current and vortex, unavailability of GPS underwater, and the highly cluttered environments make the problem difficult. Most AUVs handle uncertainty and environmental changes using variants of reactive greedy. But reactive greedy often fails in highly cluttered environments, as the planning horizon is often too short. State of the art POMDP planner requires significant time to generate reasonable policy for these problems.

The first scenario is shown in Figure 2(a). The state space S is a continuous 2D space of size 250m$\times$250m, populated by obstacles (colored dark grey). The initial state of the AUV is not known exactly. It is uniformly distributed inside the rectangles marked with 'I'. The action space is discretized into 5 actions, Move-North, Move-East, Move-South, Move-Northeast, and Move-Southeast. Due to motion error, whenever the AUV performs an action, it never reaches its intended next state, it is always off by within 2.5m radius from its intended destination. For AUV localization, several transponders with various maximum range are placed in the environment. Whenever the AUV is within a transponders' coverage (inside a circle with no label), it can localize itself with 100% accuracy. Otherwise, the AUV does not receive any observation. The AUV receives a reward of 100,000 when it reaches the goal region (circles labeled with 'G'). The environment contains local vortices whose center regions are in the rectangles marked with crosses. These vortex center regions should be avoided by the AUV as it may damage the vehicle. Therefore, if the AUV passes through the crossed rectangle, we assign a penalty of -25,000. Each move

is penalized by -1, as it takes energy. The original problem is in the left picture of Figure 2(a). Due to changes in the water flow, new vortices appear as shown in the right picture of Figure 2(a). Here, the changes are in the reward function of the states inside the new crossed rectangle.

The second scenario is shown in Figure 2(b). The AUV model, including its control and observation model, the reward function, and the legends are the same as in Scenario-1. But here, the changes is in the region affected by the vortex. Due to changes in the speed of the water flow, the vortex on the right becomes much stronger. As a result, when the AUV is at a state s inside the rectangle colored light grey in the right picture of Figure 2(b), the AUV drifts as much as $|\frac{1}{2}(v-s)|$ towards the vortex center, denoted as v. Here, the changes are in the transition function of the states inside the light grey rectangle, but not the reward function.

6.2 Experimental setup

We implemented PBPT in C++, and use MCVI for policy computation from sampled beliefs. We compared PBPT with off-line replanning and policy reuse strategy. Both replanning and policy reuse use SARSOP [11] with MCVI [2] policy computation. In Policy reuse, we use the optimal policy of the original problem as an initial policy. All experiments were conducted in a PC with 2.27GHz Intel processor and 1.5GB RAM. Below is our experimental setup for each problem scenario.

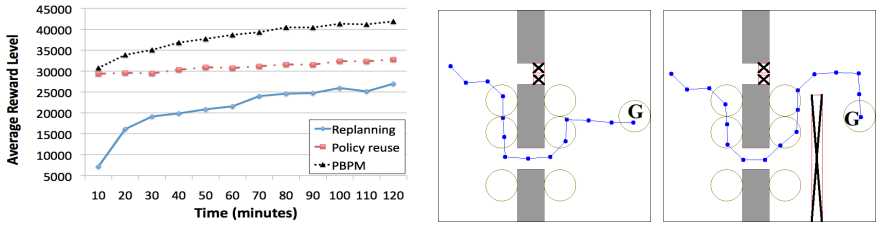
To test PBPT, we first generate 30 different policies for the original problem, using SARSOP+MCVI with 2 hours of planning time. For each policy, we ran PBPT $10\times$ to solve the modified problem. For each original policy, 10 modified policies are generated after every 10 minutes interval, for up to 2 hours. To compute the reward level reached by each modified policy, we ran 500 simulation runs and compute the average total discounted reward of the runs.

To test the performance of policy reuse, we use the same 30 original policies used to test PBPT. For each policy, we ran SARSOP+MCVI with the original policy as the initial policy, for $10\times$ to solve the modified problem. Similar to PBPT, for each original policy, 10 new policies is generated after every 10 minutes interval, for up to 2 hours. And, for each new policy, we ran 500 simulation runs and computed the average total discounted reward. To test the performance of replanning, we generate 30 policies for the modified problems using SARSOP+MCVI, after every 10 minutes interval, for up to 2 hours. To compute the reward level reached by a policy, we ran 500 simulation runs, and computed the average total discounted reward.

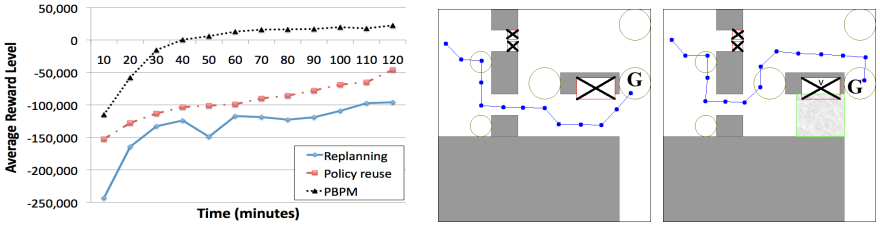
6.3 Results

The results are in Figure 3. They indicate that PBPT can generate a good policy for the modified problems much faster than replanning and policy reuse strategies.

It is interesting to compare PBPT with policy reuse, as policy reuse reuses the policy of the original problem too. The main difference between the two is that in addition to the original policy, PBPT reuses the difference between the original and modified problems to guide belief space sampling. By doing so, PBPT can quickly construct a representative sampled representation of $\mathscr{R}_n^*$ and focus on modifying critical parts of the original policy. Although policy reuse does use the original pol-



(a) Results of Scenario-1.



(b) Results of Scenario-2.

Fig. 3 Simulation results. Left: The average reward level of the policies. Middle: A typical simulation run of the original policy. Right: A typical simulation run of the policy generated by PBPT.

icy as an initial lower bound in SARSOP planning, and hence uses the original policy to indirectly guide its belief space sampling, policy reuse ignores information about the changes in the POMDP model. As a result, it is not as focused as PBPT in its belief space sampling, and hence is much slower in generating a good policy for the modified problems.

The middle and right most pictures in Figure 3 show typical simulation runs of the original policy and the modified policy generated by PBPT. The simulation runs indicate that in both scenarios, a good strategy for passing through the channels in the left should not change, as there are no change in that part of the environment. PBPT utilizes this information and focus more on modifying the strategy for moving after the AUV passes the left channel.

7 Conclusion

We propose a point-based algorithm, called Point-Based Policy Transformation (PBPT), that modifies a pre-computed policy according to changes in the POMDP model. PBPT uses the difference between the original and modified POMDP models to identify subset of $\mathscr{R}_o^*$ that is affected by the changes in the model. It uses this information to guide belief space sampling, which is a critical component for point-based POMDP planners. We show that PBPT converges to a good approximation of the best policy in $\Pi(\pi_o^*, P_o, P_n)$ or better, with high probability. Furthermore, preliminary results indicate that PBPT can generate a good policy for the modified POMDP problems much faster than recomputing the policy using the fastest POMDP planner today, even when the policy for the original problem is reused as an initial policy.

Many avenues are open for future work. We are currently working on speeding up PBPT, so that it can perform the modification on-line.

References

1. 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.
2. H. Bai, D. Hsu, W.S. Lee, and A.V. Ngo. Monte Carlo Value Iteration for Continuous-State POMDPs. In *WAFR*, 2010.
3. 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.
4. J.v.d. Berg and M. Overmars. Roadmap-based motion planning in dynamic environments. *IEEE TRO*, 21(5):885–897, 2005.
5. M.d. Berg, O. Cheong, M.v. Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2000.
6. R.M. Dudley. *Real Analysis and Probability*. Cambridge University Press, 2002.
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. L. Jaillet and T. Siméon. A PRM-based motion planner for dynamically changing environments. In *IROS*, 2004.
10. H. Kurniawati, T. Bandyopadhyay, and N.M. Patrikalakis. Global motion planning under uncertain motion, sensing, and environment map. In *RSS*, 2011.
11. H. Kurniawati, D. Hsu, and W.S. Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *RSS*, 2008.
12. F. Lamiraux, D. Bonnafous, and O. Lefebvre. Reactive path deformation for nonholonomic mobile robots. *IEEE TRO*, 20(6):967–977, 2004.
13. S.M. LaValle and R. Sharma. On motion planning in changing, partially-predictable environments. *IJRR*, 16(6):775–805, 1997.
14. P. Leven and S. Hutchinson. Real-time path planning in changing environments. *IJRR*, 21(12):999–1030, 2001.
15. C.H. Papadimitriou and J.N. Tsitsiklis. The Complexity of Markov Decision Processes. *Math. of Operation Research*, 12(3):441–450, 1987.
16. J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI*, pages 1025–1032, 2003.
17. R. Platt, R. Tedrake, T. Lozano-Perez, and L.P. Kaelbling. Belief space planning assuming maximum likelihood observations. In *RSS*, 2010.
18. J.M. Porta, N. Vlassis, M.T.J. Spaan, and P. Poupart. Point-Based Value Iteration for Continuous POMDPs. *JMLR*, 7(Nov):2329–2367, 2006.
19. S. Prentice and N. Roy. The Belief Roadmap: Efficient Planning in Linear POMDPs by Factoring the Covariance. In *ISRR*, 2007.
20. S. Ross, B. Chaib-draa, and J. Pineau. Bayes-adaptive POMDPs. In *NIPS*, 2007.
21. S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa. Online planning algorithms for POMDPs. *JAIR*, 32:663–704, 2008.
22. T. Smith and R. Simmons. Point-based POMDP algorithms: Improved analysis and implementation. In *UAI*, July 2005.
23. A. Stentz. The Focussed D* Algorithm for Real-Time Replanning. In *IJCAI*, 1995.
24. M.E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *JMLR*, 10(1):1633–1685, 2009.
25. S. Thrun. Monte carlo POMDPs. In *NIPS*, pages 1064–1070, 2000.

Acknowledgements The authors thank Leslie P. Kaelbling and Tomas Lozano-Perez for fruitful discussion, David Hsu for cluster computing usage, and the AdaComp group at SoC, NUS for providing the MCVI code. This work is funded by the Singapore NRF through SMART, CENSAM.