POMHDP: Search-Based Belief Space Planning Using Multiple Heuristics

Sung-Kyun Kim, Oren Salzman, Maxim Likhachev
The Robotics Institute, Carnegie Mellon University
kimsk@cs.cmu.edu, osalzman@andrew.cmu.edu, maxim@cs.cmu.edu

Abstract

Robots operating in the real world encounter substantial uncertainty that cannot be modeled deterministically before the actual execution. This gives rise to the necessity of robust motion planning under uncertainty also known as belief space planning. Belief space planning can be formulated as Partially Observable Markov Decision Processes (POMDPs). However, computing optimal policies for non-trivial POMDPs is computationally intractable. Building upon recent progress from the search community, we propose a novel anytime POMDP solver, Partially Observable Multi-Heuristic Dynamic Programming (POMHDP), that leverages multiple heuristics to efficiently compute high-quality solutions while guaranteeing asymptotic convergence to an optimal policy. Through iterative forward search, POMHDP utilizes domain knowledge to solve POMDPs with specific goals and an infinite horizon. We demonstrate the efficacy of our proposed framework on a real-world, highly-complex, truck unloading application.

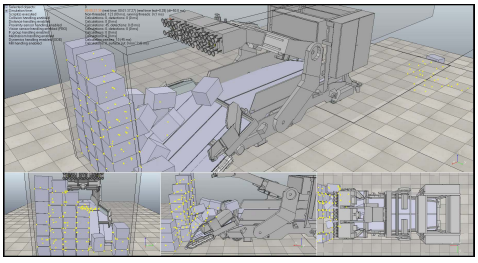
1 Introduction

Motion planning is a well-studied problem in robotics with applications in diverse domains (LaValle 2006; Halperin, Salzman, and Sharir 2017). A key challenge for motion-planning algorithms is accounting for uncertainty, especially in real-world applications.

Consider, for example, the problem of unloading boxes from truck trailers in a warehouse environment (Fig. 1), which motivates this work. Here, we are required to plan robust actions for a custom-build truck unloading robot equipped with two end effectors–a manipulator-like tool with suction-grippers as well as a scooper-like tool. The planning algorithm needs to account for uncertainty as the end effectors interacts with unstructured, unknown and stochastic environments.

Planning such robust motions under uncertainty, also known as *belief space planning*, is a crucial capability for any robot to properly function in the real world. This problem can be formulated in a principled form by a Partially Observable Markov Decision Process (POMDP) (Kaelbling, Littman, and Cassandra 1998; Kochenderfer 2015). However, solving a POMDP is often intractable due to *the curse of dimensionality* (Kaelbling, Littman, and Cassandra 1998) and *the curse of history* (Pineau, Gordon, and Thrun 2003) which correspond to an exponential complexity with the number of states and the planning horizon, respectively.

(a) A truck unloader robot is picking up boxes using suction grippers at its arm end-effector.



(b) A simulated truck unloader robot is pulling up boxes using conveyor belts on its scooper end-effector.

Figure 1: Truck unloader robot.

One approach to solve POMDPs is using heuristic-search such as RTDP-Bel (Geffner and Bonet 1998; Bonet and Geffner 2009). Given informative domain knowledge, it exhibits high sample-efficiency, which is especially beneficial in robotic applications. However, it is often not easy to design a single heuristic from the domain knowledge that captures all the complexities of the problem and effectively guides the search toward the goal. Moreover, a heuristic may not be consistent and admissible, which is required to guar-

antee theoretical bounds on the solution quality.

Another approach that proved to be very effective in solving complex domains is using *point-based* solvers that utilize vector set representation of the value function and restrict its computation to a subset of the belief space. This, in turn, corresponds to performing only local value updates for this subset. However, both these approaches require access to an *explicit* model of the POMDP probability distributions.

Returning to our motivating example of truck unloading this assumption is clearly unrealistic—in our case we only have access to a noisy *generative model* or simulator. Given a state and an action, this simulator provides a sample of a successor state, observation and reward. Monte-Carlo based methods such as POMCP (Silver and Veness 2010) can be applied to such settings. Their favorable traits can be attributed, in part, to representing each belief state using a set of *particles* and performing a Monte-Carlo tree search on these set of particles. Such planners are typically more suited to the *Discounted* POMDP problems where immediate actions (e.g., evasion from adversaries) are more heavily rewarded than future ones (e.g., the agent's location at the end of execution).

However, in many robotic applications, robots are assigned for specific tasks to accomplish (e.g., navigate to a target location, or unload all boxes from a truck). For these problems, future actions are as important as immediate ones, and the objective is to achieve the prescribed goals. It means the planning horizon is unbounded, and effective guidance to the goals is important. Such of POMDPs are called *Goal* POMDPs, and they are what we tackle to solve in this work.

Our key insight is to incorporate recent advances in heuristic search together with a particle representation. Specifically, we make use of *multiple* heuristics to systematically guide our search algorithm in belief space for highly-efficient planning under uncertainty. Utilizing a particle representation of a belief state, our algorithm can be applied to domains where only a generative model is available. We demonstrate the efficacy of our approach on our truck unloading application, computing high-quality plans that removes boxes from a trailer in a time-efficient and risk-averse manner.

2 Related work

In this section, we review related work on belief space planning. Specifically, we detail the three primary approaches in solving POMDPs Point-based methods (Sec. 2.1), Heuristic search-based methods (Sec. 2.2) and Monte Carlo-based methods (Sec. 2.3.

2.1 Point-based methods

Point-based methods have been one of the major approaches to solve relatively large POMDP problems (Pineau, Gordon, and Thrun 2003; 2006; Smith and Simmons 2004; 2005; Kurniawati, Hsu, and Lee 2008; Ross and Chaib-Draa 2007). They represent the value function by a vector set of sampled points and its piece-wise linear combination (also known as α-vectors) (Smallwood and Sondik 1973). Key to their efficiency is that instead of planning on the entire belief space, they concentrate planning resources on beliefs that are reachable from the initial belief.

Point-based methods differ in how they select the core subset of the belief space over which to compute the value function, as well as on the order by which the value at those beliefs are updated. It is important to note that point-based methods maintain the value function using explicit probability-distribution models and update the value function by considering all possible available actions. This requires an explicit transition model and is often unscalable to highly-complex domains. For a recent survey on point-based POMDP methods, see (Shani, Pineau, and Kaplow 2013).

2.2 Heuristic search-based methods

Another major approach to belief-space planning is using heuristic search-based methods (see e.g., (Hansen and Zilberstein 1998; 1999; 2001; Washington 1997; Geffner and Bonet 1998; Bonet and Geffner 2000; Bonet 2002; Geffner and Bonet 1998)). These methods utilize available domain-specific knowledge to guide a search algorithm to efficiently find the optimal solution. Similar to point-based methods and unlike dynamic-programming methods such as value iteration (Poole and Mackworth 2010), heuristic search methods evaluate the value function for a small subset of the belief space. These methods are derived from graph-search algorithms in deterministic environments but operate on graphs where probabilistic transitions along edges should be considered.

A seminal work regarding heuristic search-based methods is the comparison of RTDP-Bel and point-based methods (Bonet and Geffner 2009). Here, the authors present a method to convert a Discounted POMDP to an equivalent Goal POMDP and compared RTDP-Bel with point-based methods. With equivalence-preserving transformation of the problem, many different benchmark tests showed that RTDP-Bel is competitive and often superior to point-based algorithms that utilize Sondik's vector representation of the value function (Smallwood and Sondik 1973).

2.3 Monte Carlo-based methods

Monte Carlo Tree Search (MCTS) is a heuristic search method that uses random sampling to expand promising actions and has been shown to be effective in solving MDPs (Coulom 2006; Gelly and Silver 2007). Recently, Silver et al. suggested Partially Observable Monte Carlo Planning (POMCP), an MCTS-based method for solving POMDPS (Silver and Veness 2010). This, in turn, led to the development of multiple MCTS-based POMDP solvers (see e.g., (Somani et al. 2013; Kurniawati and Yadav 2013; Seiler, Kurniawati, and Singh 2015; Sunberg and Kochenderfer 2017)).

MCTS-based POMDP solvers convert the problem to an MDP by representing a belief state as a set of particles. This implicitly corresponds to sampling the belief state (alleviating the curse of dimensionality) and the belief transitions (alleviating the curse of history). Typically, this approach allows for online computation which interleaves planning and execution so that solution quality is improved while execution a computed policy.

3 Preliminaries: POMDP formulation

In this section we briefly describe our problem formulation assuming the reader is familiar with MDPs and POMDPs. For a general introduction to the subject, see e.g., (Bertsekas 2005; Thrun, Burgard, and Fox 2005; Russell and Norvig 2010)

Let $\mathbb{S}, \mathbb{A}$, and $\mathbb{Z}$ denote the state, action, and observation spaces, respectively. We denote the motion model $T(s, a, s') = P(s' \mid s, a)$, which defines the probability of being at state s' after taking an action a in state s. The observation model $Z(s, a, o) = P(o \mid s, a)$ is the probability of receiving observation o after taking action a in state s.

A belief $b(s)$ is a posterior distribution over all possible states given the past actions and observations, i.e., $b_k(s) = P(s \mid a_{0:k-1}, o_{1:k})$ where the subscript k denotes the time step. Note that a POMDP problem can be formulated as a *Belief MDP* by taking $b(s)$ as an MDP state, also referred to as a *belief state* $b \in \mathbb{B}$, where $\mathbb{B}$ is referred to as *belief space*.

Given a_{k-1}, o_k, and $b_{k-1}(s)$, the updated belief $b_k(s)$ can be computed by Bayesian filtering, which can be divided into two sub-steps as follows.

$$b_{k-1}(s; a_{k-1}) = \sum_{s' \in \mathbb{S}} T(s, a_{k-1}, s') b_{k-1}(s), \tag{1}$$

$$b_k(s; a_{k-1}, o_k) = \eta Z(s, a_{k-1}, o_k) b_{k-1}(s; a_{k-1}), \tag{2}$$

where η is a normalizing constant. For notational convenience, let us denote $b_{k-1}(s; a) = b^a_{k-1}$ and $b_k(s; a, o) = b_k = b^{ao}_{k-1}$ hereafter.

A policy $\pi : \mathbb{B} \to \mathbb{A}$ maps each belief state b to a desirable action a. The expected cost of an action for the true state can be represented as a cost function in a belief space, $c(b, a) \in \mathbb{R}^+$. Given a policy π and a belief $b \in \mathbb{B}$, we can compute the value function,

$$V(b; \pi) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k c(b_k, \pi(b_k))\right], \tag{3}$$

where b_0 is the initial belief state and $\gamma \in (0, 1]$ is a discount factor that reduces the effect of later costs.

We can rewrite Eq. 3 in a recursive form, which is called the Bellman equation.

$$V(b; \pi) = c(b^i, \pi(b)) + \gamma \sum_{b' \in \mathbb{B}} \tau(b, \pi(b), b') V(b'; \pi), \tag{4}$$

where $\tau(b, \pi(b), b') = \sum_{o \in \mathbb{Z}} P(b' | b, \pi(b), o) P(o | b, \pi(b))$ is the transition probability from b to b' under π, which can be derived from Eq. 1 and 2. For further details, see (Ross et al. 2008). It is often convenient to define the so-called Q-value function for an intermediate belief-action pair, (b, a) or simply b^a, as follows.

$$Q(b^a; \pi) = c(b, a) + \gamma \sum_{b' \in \mathbb{B}} \tau(b, a, b') V(b'; \pi). \tag{5}$$

We now restate our POMDP problem as an optimization problem.

$$\begin{aligned} \pi^*(b) &= \underset{\Pi_{0:\infty}}{\operatorname{argmin}} \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k c(b_k, \pi_k(b_k))\right] \\ &= \underset{\Pi_{0:\infty}}{\operatorname{argmin}} V(b_k; \pi_k). \end{aligned} \tag{6}$$

Algorithm 1 RTDP-BEL

Inputs: Initial belief b_0, admissible heuristic h, explicit motion/observation models T, Z

1: **repeat**
2: $b \leftarrow b_0$ ▷ start iteration with initial belief
3: SAMPLE state s with probability $b(s)$
4: **repeat**
5: EVALUATE $q(b^a)$ for each action a using Eq. 5 ▷ initialize $v(b^{ao})$ to $h(b^{ao})$ if not initialized
6: SELECT BEST ACTION a ▷ a minimizes $q(b^a)$
7: UPDATE $v(b)$ to $q(b^a)$
8: SAMPLE next state s' ▷ uses motion model T
9: SAMPLE observation o ▷ uses observation model Z
10: COMPUTE b^{ao} and set $b \leftarrow b^{ao}$, $s \leftarrow s'$
11: **until** b^{ao} is the goal belief
12: **until** CONVERGED()

Note that in this work we consider Goal POMDPs without cost discounting, i.e., $\gamma = 1$, and thus, the value function $V(b; \pi)$ corresponds to the expected *cost-to-goal* from belief state b under policy π.

For notational brevity in the algorithm description, let us denote $V(b; \pi)$ and $Q(b; \pi)$ that are being learned/updated during the belief space planning by $v(b)$ and $q(b)$, respectively, in the following sections.

4 Algorithmic background

4.1 Real-Time Dynamic Programming in Belief Space (RTDP-Bel)

In this section we briefly review Real-Time Dynamic Programming in Belief Space (RTDP-Bel) (Geffner and Bonet 1998) which is an extension of RTDP for MDP (Barto, Bradtke, and Singh 1995) to belief space planning.

RTDP-Bel, described in Alg. 1 performs a series of searches until it converges to the optimal policy (see Eq. 6). Each iteration (Lines 2-12) is a greedy search starting from the initial belief b_0 and terminating at a goal belief[1]. Every iteration of the search consists of sampling a state within our belief (Line 3), evaluating the effect of every action on the sampled state (Line 5), selecting the best action (Line 6) and updating the value function accordingly. We then sample the next state and observation (Lines 8-9) to compute the new belief (line 10).

There are several important details to note here: (i) each iteration is a *path* computed greedily from b_0, (ii) to ensure convergence to the optimal policy we require that the heuristic function $h(\cdot)$ (Line 5) is admissible and that (iii) an explicit transition model is used (Lines 5,8,9).

[1]Here, for ease of description, we assume that there is always a path to the goal state. In practice, this may not be the case and an iteration may be terminated early when the greedy search to the goal has no available action.

4.2 Multi-Heuristic A* (MHA*)

Heuristic functions may have a dramatic effect on the performance of heuristic search-based planners such as A*. However, in many applications, it is difficult to engineer one heuristic function that can guide the search throughout the entire environment. Furthermore, this heuristic function is required to be admissible[2] in order to guarantee bounds on solution quality and completeness.

One approach to address these challenges is by simultaneously using *multiple* heuristic functions. One algorithm that uses this idea is Multi-Heuristic A* (MHA*) (Aine et al. 2016), which takes in a single admissible heuristic called the *anchor* heuristic, as well as multiple (possibly) inadmissible heuristics. For each heuristic, it runs an A*-like search, and the expanded nodes of each search are shared among all searches. This allows to automatically combine the guiding powers of the different heuristics in different stages of the search (for different applications of MHA*, see, e.g., (Kim and Likhachev 2017; Ranganeni, Salzman, and Likhachev 2018; Islam, Salzman, and Likhachev 2018)).

MHA* uses the so-called *anchor test* to evaluate the heuristic value of a state s. Specifically, given some $\varepsilon > 1$, admissible and inadmissible heuristics h_{anchor} and h_{inad}, respectively, the algorithm uses h_{inad} for a state s only if $g(s) + h_{\text{inad}}(s) \leq g(s) + \varepsilon \cdot h_{\text{anchor}}(s)$. Here $g(s)$ represents the cost to reach state s from the initial state (also known as the cost-from-start). It can be shown that using the anchor test, the cost of solutions obtained by MHA* are no more than ε the cost of the optimal solution.

5 Partially Observable Multi-Heuristic Dynamic Programming (POMHDP)

5.1 Key ideas

Our algorithm can be seen as an adaptation of RTDP-Bel. The first set of modification, which is the key contribution of this work, is concerned with how the heuristic function is used (Alg. 1, Line 5). The second set of modification, which we consider a (non-trivial) implementation detail is concerned with relaxing the algorithm's requirement to use explicit transition models (Alg. 1, Lines 5,8,9) and using a particle representation of belief states. We omit further details and refer the reader to the appendix for detailed pseudocode describing the complete algorithm using a generative model.

The first set of modifications makes use of the following key ideas:

I1 Inflation of heuristic value.

I2 Incorporating one inadmissible heuristic with one admissible heuristic.

I3 Incorporating several inadmissible heuristic with one admissible heuristic.

As we will see shortly, ideas **I1** and **I2** are used to heuristically improve how fast the path constructed by the algorithm in each iteration reaches the goal belief. These improvements come at the price of convergence to a policy that is within a given bound of the optimal policy.

[2] A heuristic function is said to be admissible if it never overestimates the optimal cost of reaching the goal.

Algorithm 2 SELECT BEST ACTION

Inputs: current belief b, admissible heuristic h_{anchor}, inadmissible heuristic h_{inad}, approx. factor ε_2

1: $a_{\text{inad}} = \text{argmin}_{a \in \mathbb{A}}\, q_{\text{inad}}(b^a)$ ▷ $b^a = \text{SUCC}(b, a)$
2: $a_{\text{anchor}} = \text{argmin}_{a \in \mathbb{A}}\, q_{\text{anchor}}(b^a)$ ▷ $b^a = \text{SUCC}(b, a)$
3: **if** $q_{\text{inad}}(b^{a_{\text{inad}}}) \leq \varepsilon_2 \cdot q_{\text{inad}}(b^{a_{\text{anchor}}})$ **then**
4: **return** $b^{a_{\text{inad}}}$
5: **return** $b^{a_{\text{anchor}}}$

Idea **I3** is motivated by the insight that a single heuristic may drive the search into a region of the belief space where it may be very hard to reach the goal belief. Roughly speaking, this is because the heuristic value is in *stagnation* (see, e.g., (Dionne, Thayer, and Ruml 2011; Islam, Salzman, and Likhachev 2018))—namely, it is uninformative. In this case we would like to use an alternative heuristic. The best state to expand chosen by the alternative heuristic will generally not be the state considered by the current heuristic. This means that each iteration will now be a *tree* in belief space rooted at the initial belief and the search will end when one state of the tree reaches the goal belief. For visualization of the approach, see Fig. 2.

5.2 Using an inadmissible heuristic (I1 & I2)

Let h_{anchor} and h_{inad} be admissible and inadmissible heuristics, respectively and let $\varepsilon_1, \varepsilon_2 > 1$ be some constants.

Instead of using only the admissible heuristic for value function initialization as in RTDP-Bel (Alg. 1, Line 5), we inflate the admissible heuristic by ε_1 (Idea **I1**). Instead of choosing the best action according to the admissible heuristic as in RTDP-Bel (Alg. 1, Line 6), we perform the anchor test (Idea **I2**).

This requires (i) storing for each state two Q-values (one for the admissible heuristic and one for the inadmissible one) and performing the anchor test. For pseudocode depicting how line 6 in Alg. 1 is now implemented, see Alg. 2.

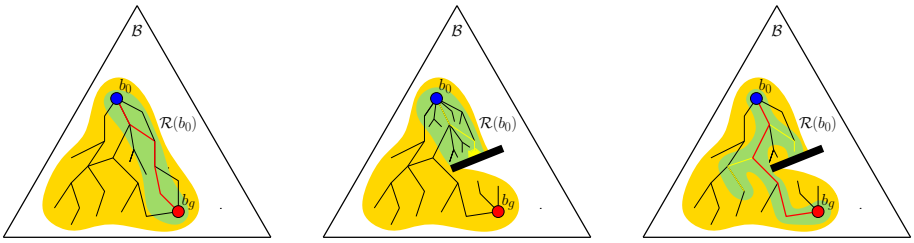
Roughly speaking, Idea **I1** is analogous to the way weighted A* (Pearl 1984) uses an inflated heuristic and Idea **I2** follows the way MHA* ensures bounded suboptimality (see Sec. 4.2).

5.3 Using multiple inadmissible heuristics (I3)

To make use of multiple (inadmissible) heuristics we need (i) a principled way to stop using one (inadmissible) heuristic and start using another (inadmissible) heuristic and (ii) a principled way to choose which state do we want to sample from once we switch to a new (inadmissible) heuristic.

We choose to stop using one (inadmissible) heuristic when we detect that it is no longer informative, or in *stagnation*. There are many ways to define heuristic stagnation (see e.g., (Islam, Salzman, and Likhachev 2018)). Here, we adopted the concept of momentum in stochastic gradient descent (Qian 1999). For pseudo code, see Alg. 3.

To choose which state to sample from once we switch to a new (inadmissible) heuristic, we maintain a priority queue,



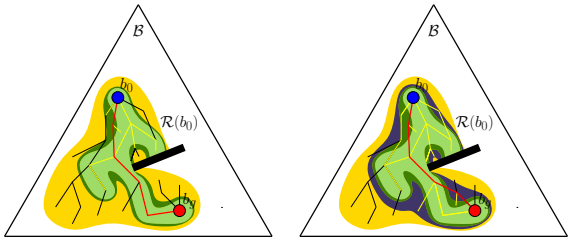
(a) (b) (c)

Figure 2: Illustration of forward search with a single (a, b) and multiple (c) heuristics. (a) Forward search with single heuristic reaches the goal, (b) Forward search with single heuristic gets stuck at local minima. (c) Forward search with multiple heuristics can switch from one heuristic to another until reaching the goal. Yellow and green regions represent the reachable belief space from the start and the explored belief space by the forward search, respectively.

Algorithm 3 IS IN STAGNATION

Inputs: previous state v-value v_{pred},
inadmissible heuristic h_{inad}, previous diff. Δv

1: $q_{\text{succ}} = \min_{a \in \mathbb{A}} q_{i_h}(b^a)$ where $b^a = \text{SUCC}(b, a)$
2: $\Delta v \leftarrow \eta \Delta v + (q_{\text{succ}} - v_{\text{pred}})$
3: **if** $\Delta v \geq 0$ **then**
4: **return** `true`
5: **return** `false`



(a) 2nd iteration (b) 3rd iteration

Figure 3: Illustration of the anytime behavior of POMHDP. After the first iteration of forward search (Fig. 2c), we gradually relax the search space bounds (Alg 4, Line 18) and run another forward search to improve the solution.

OPEN$_i$, of belief-action pairs for each heuristic h_i[3]. Belief-action pairs in OPEN$_i$ are ordered according to the sum of the cost-from-start and cost-to-goal for i-th heuristic, i.e., $\text{KEY}(b^a, i) = g(b^a) + q_i(b^a)$.

When we switch to a new (inadmissible) heuristic h_i, we find the belief-action pair with the minimal key in OPEN$_i$. To ensure that the iterative forward search yields bounded suboptimal solutions, we apply the anchor test between this belief-action pair and the one with the minimal key in OPEN$_0$. Furthermore, to guarantee that the algorithm converges to optimal solution, we decrease the approximation factors, ε_1 and ε_2, between iterations. For visualization, see Fig. 3.

A high-level description of the algorithm is provided in Alg. 4. Lines where the algorithm differs from RTDP-Bel (Alg. 1) are marked in magenta.

5.4 POMHDP—Theoretical properties

In this section we highlight the theoretical properties of POMHDP and provide sketches of proofs for these properties.

Recall that RTDP, which performs asynchronous dynamic programming on an MDP, provides asymptotic convergence to the optimal policy given an admissible heuristic (Barto, Bradtke, and Singh 1995). RTDP-Bel, which approximates a POMDP problem as a belief MDP through belief discretization, similarly provides asymptotic convergence to a resolution-optimal policy (Bonet and Geffner 2009). For the following theoretical analysis, we consider the optimality in belief MDP as an alternative to the resolution-optimality in POMDP (approximated by belief discretization or particle representation). Note that the resolution-optimal policy converges to the optimal POMDP policy as the discretization decreases or the number of particles for belief state respresentation increases (Silver and Veness 2010).

1. **RTDP-Bel using Idea I1 asymptotically converges to ε_1-suboptimal policy.** In belief MDPs, a *cost-to-goal* for each belief state can be decomposed as $v(b) = v_g(b) + \varepsilon_1 \cdot v_h(b)$, where $v_g(b)$ and $v_h(b)$ are initialized to zero and an admissible heuristic value $h(b)$, respectively (Chakrabarti, Ghose, and DeSarkar 1987; Hansen and Zilberstein 1999). Eq. (4) can also decomposed into $v_g(b) = c(b^i, a) + \sum_{b' \in \mathbb{B}} \tau(b, a, b') v_g(b')$ and $v_h(b) = \sum_{b' \in \mathbb{B}} \tau(b, a, b') v_h(b')$. Through iterative forward search, $v_g(b_g) = v_h(b_b) = 0$ for the goal belief is back-propagated to the predecessors, and once converged, $v(b) = v_g(b)$ and $v_h(b) = 0$ are satisfied for each belief. Due to the greedy action selection scheme and inflated heuristic initialization, the forward search is biased to the heuristic guidance, and $v_g(b)$ can be suboptimal but bounded by a factor of ε_1.
2. **RTDP-Bel using Idea I2 asymptotically converges to ε_2-suboptimal policy.** An inadmissible heuristic cannot

[3] In addition to OPEN$_i$, we also maintain two lists, denoted by CLOSED$_{\text{anchor}}$ and CLOSED$_{\text{inad}}$, for the anchor heuristic and the inadmissible heuristics, respectively to detect duplicates. We omit discussing these lists for clarity of exposition and refer the reader to the description of MHA* (Aine et al. 2016) for further details.

Algorithm 4 POMHDP-EXPLICIT

Inputs: Initial belief b_0, admissible anchor heuristic h_0, $n_h - 1$ additional heuristics $h_1, \dots h_{n_h}$ approx. factors $\varepsilon_1, \varepsilon_2$ decaying const. α explicit motion/observation models T, Z

1: **repeat**
2: $b \leftarrow b_0$ ▷ start iteration with initial belief
3: $\forall_i$, OPEN$_i \leftarrow \emptyset$; ▷ start iteration with empty queues
4: curr $\leftarrow 1$ ▷ index of inadmissible heuristic
5: SAMPLE state s with probability $b(s)$
6: **repeat**
7: EVALUATE $q_0(b^a), q_{\text{curr}}(b^a)$ for each action a ▷ initialize $v_i(b^{ao})$ to $\varepsilon_1 \cdot h_i(b^{ao})$ if not initialized ▷ updates OPEN$_0$ and OPEN$_{\text{curr}}$
8: **if** IS IN STAGNATION **then** ▷ Uses Alg. 3
9: SWITCH HEURISTIC ▷ updates curr
10: SELECT BEST ACTION a ▷ runs anchor test OPEN$_{\text{curr}}$'s and OPEN$_0$'s head
11: **else**
12: SELECT BEST ACTION a ▷ Uses Alg. 2
13: UPDATE $v_0(b)$ to $q_0(b^a)$ and , $v_{\text{curr}}(b)$ to $q_{\text{curr}}(b^a)$
14: SAMPLE next state s' ▷ uses motion model T
15: SAMPLE observation o ▷ uses observation model Z
16: COMPUTE b^{ao} and set $b \leftarrow b^{ao}$, $s \leftarrow s'$
17: **until** b^{ao} is the goal belief
18: $\varepsilon_1 \leftarrow \varepsilon_1 e^{-\alpha}$; $\varepsilon_2 \leftarrow \varepsilon_2 e^{-\alpha}$ ▷ decrease approx. factors
19: **until** CONVERGED()

provide any theoretic guarantees by itself, but with the anchor test (Alg. 2, Line 3) it can be incorporated in the forward search guidance without loss of theoretic guarantees. The anchor test allows the inadmissible heuristic to be used for action selection only if its cost-to-goal estimate is not ε_2-times larger than that of the admissible heuristic. This action selection scheme provides suboptimality bounds by a factor of ε_2, *locally*, i.e., from the current belief to the goal belief. By backward induction from the goal belief with the true $v(b_g) = 0$, the iterative forward search can be shown to provide ε_2-suboptimality in the *global* sense.

3. **POMHDP asymptotically converges to the optimal policy.** POMHDP leverages both of ε_1-inflation of heuristics (Idea **I1**) and ε_2-factored anchor test (Idea **I2**). In addition, it makes use of stagnation detection and rebranching over multiple heuristics (Idea **I3**). Note that rebranching helps to avoid stagnation during the forward search and effectively reach the goal, and does not affect the theoretical suboptimality bounds since it has no distinction with the usual evaluation and update step in terms of *asynchronous dynamic programming*. As a result, POMHDP converges to $\varepsilon_1 \cdot \varepsilon_2$-suboptimal solution for constant ε_1 and ε_2. With the step decreasing ε_1 and ε_2 (Alg. 4, Line 18), POMHDP converges to the optimal solution.

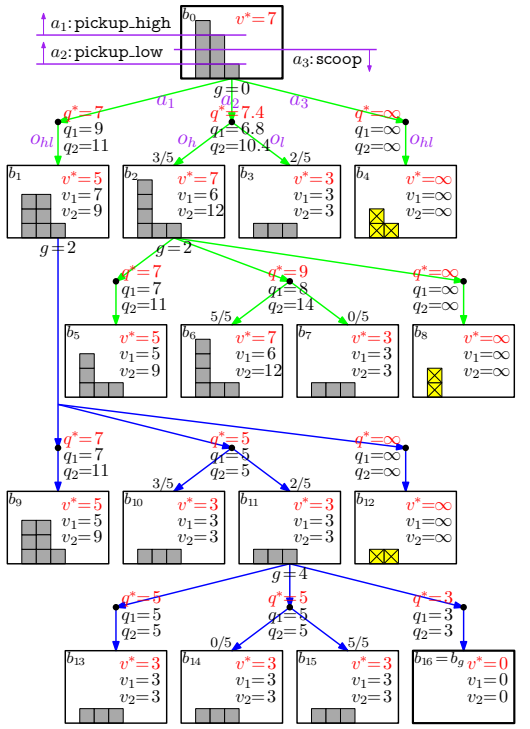


Figure 4: A toy example for truck unloading. POMHDP conducted a forward search using one admissible heuristic and two inadmissible heuristics in the following order: $b_0 \rightarrow b_2 \rightarrow$ (heuristic switch; rebranch) $\rightarrow b_1 \rightarrow b_{11} \rightarrow b_g$. Heuristic switch and rebranching of the forward search helped to get out of the stagnation quickly and reach the goal. (The optimal cost-to-goal values in red color are presented just for reference.)

6 Putting it all together–Illustrative example

Fig. 4 illustrates how POMHDP works on a simplified truck unloading problem. In this toy example, there is a stack of boxes in a truck, and the goal is to remove all the boxes from the truck. Box masses are unknown and unobservable— according to the prior belief, the boxes are `heavy` with a 60 % chance and `light` with a 40 % chance.

There are three available actions for the truck unloader robot; `pickup_high` (a_1), `pickup_low` (a_2), and `scoop` (a_3). The first two actions are to pick up boxes using its arm and release them onto the conveyor belt to remove them from the truck, where `pickup_high` and `pickup_low` try to pick up boxes from the 4$^{\text{th}}$ row and 2$^{\text{nd}}$ row, respectively. The third action is to use its scooper to lift boxes from the floor and pull back using the conveyor belts. The cost of each action is two, two, and three, respectively (here, a scoop action costs more than a pick action as it requires moving the base of the robot which takes more time than only moving the end effector).

Due to the unobservable box masses, transition between belief states is stochastic. If the boxes are `light` (o_l), `pickup_high` or `pickup_low` can lift four boxes in each column of the stack, but if they are `heavy` (o_h), only two boxes

in each column can be lifted. Regardless of the box masses (o_{hl}), scoop is not successful if the number of boxes in each column is larger than two (scooping more than two boxes causes them to fall to the sides of the robot, which in turn, requires manual intervention).

For this truck unloading problem, we have one admissible heuristic, $h_0(b)$, and two inadmissible heuristics, $h_1(b)$ and $h_2(b)$ as follows.

$$h_0(b) = \begin{cases} 3 & (\forall b \in \mathbb{B} \backslash \{b_g\}) \\ 0 & (b = b_g) \end{cases} \quad (7)$$

$$h_1(b) = \mathbb{E}[(\text{\# of all boxes left})] \quad (8)$$

$$h_2(b) = \mathbb{E}[3 \cdot \max(\text{\# of boxes in each column})] \quad (9)$$

where $h_i(b) = \infty$ for $\forall i$ if there is any box fallen out of the robot. $h_0(b)$ refers to a trivially admissible heuristic which assumes a one-time scoop action can unload the all boxes. $h_1(b)$ and $h_2(b)$ are simple heuristic functions that depend on the configuration of remaining boxes and take average of them over the sampled states in the belief.

As shown in Fig. 4, the first forward search starts from the initial belief, b_0, with the heuristic reference index, $i_h = 1$. Let us assume $\varepsilon_1 = 1$, $\varepsilon_2 = 5$, and $\eta = 0$. After evaluating the successor belief-action pairs, $b_0^{a_2}$ with $q_1 = 6.8$ seems to be the best among the successors. Note here that $v_i(b)$ is initialized by $\varepsilon_1 \cdot h_i(b)$. When sampling a successor belief from $b_0^{a_2}$, b_2 is selected for the next evaluation. However, it turns out that all the successor belief-action pairs of b_2 have higher q_1-values than $b_0^{a_2}$, which means the forward search is in stagnation.

Thus, the forward search rebranches from the best belief-action pair in OPEN$_2$, which is $b_0^{a_1}$, after switching i_h to 2. After the evaluation, $b_1^{a_2}$ with $q_2 = 5$ is chosen for the next iteration. When b_{11} is sampled and evaluated, the successor of $b_{11}^{a_3}$, i.e., b_{16}, reaches the goal, and then the first forward search terminates. Note that in Fig. 4 the updated $v_i(b)$ and $q_i(b^a)$ are not visualized due to the space limit, and the optimal $v^*(b)$ and $q^*(b^a)$ are added for the reference.

In this simple example, the solution policy has converged to the optimal after the first forward search (with the expected cost-to-goal of 7), thanks to the heuristic switch and rebranching. Without rebranching, it would need at least two iterations (more specifically, until ε_2 decreases enough to expand $b_0^{a_1}$ based on the admissible heuristic).

7 Simulation results

In this section, we present simulation results in the truck unloader domain to demonstrate the efficacy of the proposed framework. Note that the results in other robotic applications are not included in this paper due to the space limit.

7.1 Problem description

The truck unloader problem—a robot needs to remove boxes from a truck, where there is only limited prior information about the box dimensions and masses—is a highly complex, real-world domain. It is in continuous action/observation space with high stochasticity especially when handling the boxes.

We formulated this problem as a Goal POMDP. A state $s = (T_r,\, q_{1:16},\, \{T_b,\, l_b,\, m_b\}_{1:N_b}) \in \mathbb{S}$ is a tuple of the robot base pose, T_r, and joint configuration, $q_{1:16}$, and the pose, T_b, dimensions, l_b, and mass, m_b, of N_b boxes. Note that the box mass is unknown and unobservable. Then a belief state $b = \{s\}_{1:N_{\text{part}}} \in \mathbb{B}$ is presented by a set of sampled states, i.e., particles. The goal condition is satisfied when all the boxes are out of the truck in all (or almost by a certain threshold) particles.

The action space is discretized into 12 macro actions, such as pickup_high_left, pickup_low_right, or scoop, considering the frequent motions during the truck unloading operations. The observation space is also discretized into 18 cases based on the box poses, such as box_pile_midhigh_left, box_pile_low, or box_pile_none, taking account of the macro action's capability. In the presented simulation results, we assume that we have access to the exact box poses instead of estimating them from simulated visual sensor data and a virtual perception module. Note that even with this assumption, the unobservable box mass and the added noise to the dynamics simulator make the action execution to be still very stochastic.

We defined a cost function and multiple heuristic functions in a similar way with the toy example presented above. The pickup actions have a cost of 2, and the scoop action has a cost of 3. The admissible and inadmissible heuristic functions used in the evaluation are the same as in Eq. 7-9.

7.2 Experiment setup

For evaluation of the proposed work, we first created a simulation model of the truck unloader system in V-REP simulator from Coppelia Robotics GmbH (Fig. 1). Recall that a dynamics simulator serves as the generative model in our framework. To provide feasible solutions for the real-world system, high-fidelity simulation is a strong requirement, while it induces very expensive computational costs. (The real-time factor of this simulation model is approximately 0.3.)

As the baseline algorithms, POMCP and a variant of RTDP-Bel were used. POMCP is considered to be one of the state-of-the-art online POMDP solvers. It intrinsically accepts generative POMDP models and allows us to initialize the value function using any (single) heuristic function. Unlike POMHDP, however, it does not bootstrap with the initialized values during the rollout phase and usually uses a random policy in favor of *unbiased exploration*. RTDP-Bel is a heuristic search-based algorithm with a single heuristic and shown to be competitive to point-based algorithms (Bonet and Geffner 2009). However, it cannot accommodate generative POMDP models, so we used a modified version by our own, namely RTDP-Part, that uses particle representation of a belief state. In addition to the admissible heuristic as in Eq. 7, we also present the results of RTDP-Part with *one* of the inadmissible heuristics in Eq. 8-9, denoted by RTDP-Part(h_i).

7.3 Results

As shown in Table 1, POMCP took significantly longer time than the other algorithms, and thus, sufficient data for further

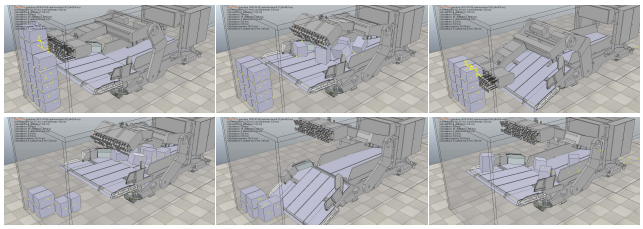


Figure 5: Snapshots of truck unloading task execution based on the solution policy of POMHDP.

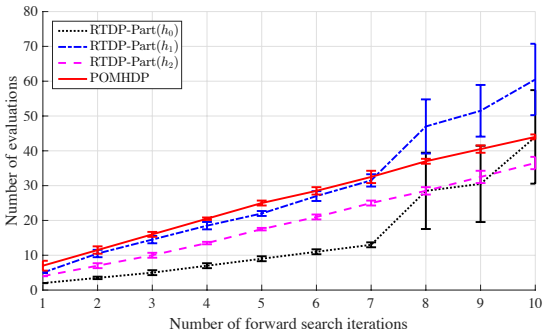


Figure 6: Number of evaluations over the forward search iterations by RTDP-Part with an individual heuristic, and POMHDP with multiple heuristics, respectively.

statistical analysis could not be collected. This is because in the initial forward search iteration only the random rollout policy is being used to select the next action, and thus, it can easily lead to an suboptimal region. Once it enters a *dead-end* state, e.g., when a box fell into the side of the robot, there is no action that the robot can take to recover from it and achieve the goal. Then the forward search indefinitely keeps expanding in dead-end region.

Fig. 6 shows the number of evaluations per each forward simulation iteration, which can be considered as the effectiveness of guiding the search toward the goal. At the earlier iterations, POMHDP needs slightly more evaluations because of rebranching behavior when falling into stagnation. However, considering the small variance of its evaluation numbers over the iterations, the rebranching is shown to provide effectiveness to guide the search in many different scenarios. Note that RTDP-Part with a single heuristic (especially h_0 and h_1) suffers from stagnation and requires many evaluations to reach the goal.

In Fig. 7, the number of boxes in different states after executing a solution policy is presented, where the anytime

Table 1: Average time for the first forward search iteration

Algorithm	POMCP	RTDP-Part			POMHDP
		(h_0)	(h_1)	(h_2)	
Time [min]	≥ 60	1.86	4.83	3.86	6.18

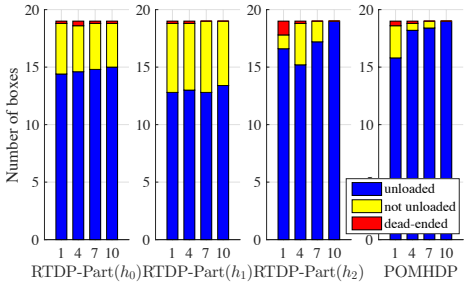


Figure 7: Number of boxes in different states (unloaded, not unloaded, or dead-ended) after executing a solution policy obtained from each number (on the x-axis) of the forward search iterations.

performance of each algorithm can be inferred from. Compared to RTDP-Part(h_0), RTDP-Part(h_1) is relatively conservative (many boxes are not unloaded), so that the solution would not lead to a dead-end. RTDP-Part(h_2) is rather aggressive to achieve the goal (more boxes unloaded), but may fall into a dead-end. POMHDP automatically takes advantage of multiple heuristics and achieves higher number of unloaded boxes, while reducing the chance of failures.

8 Conclusion

In this paper, we proposed a novel belief space planning framework that leverages multiple inadmissible heuristics to solve Goal POMDPs with an infinite horizon. Multiple heuristics can provide effective guidance of the forward search even in the case that the heuristics are possibly uninformative and suffer from search stagnation. From the condition check with an admissible heuristic, it can still guarantee the asymptotic convergence of the suboptimality bounds of the solution. It is an anytime algorithm which can improve the solution quality as time permits by relaxing the search space bounds. The simulation results showed that this approach is also empirically meaningful and can outperform the state-of-the-art methods in robotic problems that can be modeled as Goal POMDPs.

An interesting extension of this work is the online planning version that interleaves replanning and execution. Online POMDP solvers usually have a finite but receding horizon, so they are subject to the local minima problem. Thus, multiple heuristics can be more beneficial to alleviate that problem and achieve better performance.

Another promising revenue of future research is to incorporate the heuristics suggested in the point-based approaches. For example, a heuristic that guides the forward search to the region with a larger gap between the upper and lower bounds can be used as one of inadmissible heuristics in our framework. It would be interesting to investigate the synergetic effects when different approaches in belief space planning are combined together.

Acknowledgments

This research was in part sponsored by ARL, under the Robotics CTA program grant W911NF-10-2-0016.

Appendix: Detailed pseudocode

We present a detailed description of POMHDP, including how we use a particle representation of beleif states. The pseudocode of POMHDP is presented in Alg. 5 and detailed in Alg. 6-12. To incorporate multiple heuristics, we maintain the value function and the Q-value function for each heuristic, denoted by $v_i(b)$ and $q_i(b)$ for $i = 0, ..., n_h$, respectively. Note that we have access to one admissible heuristic (with index $i = 0$) and n_h inadmissible heuristics.

After the initialization, POMHDP iterates the forward search (line 6-23) until the convergence or timeout. Starting from the given initial belief b_0, the forward search 1) evaluates the successor beliefs, 2) backs up (update) the value function, and 3) choose the next belief state to repeat these steps until reaching the goal belief. Once each forward search terminates, i.e., reaches the goal, ε_1 and ε_2 values are reduced exponentially (line 24).

During the evaluation step, successor belief states are generated using a generative model (a black box simulator) if not yet done (line 3-20 in Alg. 8). Then $q_i(b^a)$ for the current belief b and possible action a pair is updated as in (5) (line 26 in Alg. 8). The minimal $q_i(b^a)$ over $\forall a \in \mathbb{A}$ is used to set $v_i(b)$ in the backup step (Alg. 9).

At the end of the evaluation step, the algorithm inserts or updates the intermediate belief-action pair $b^a \in \mathbb{B}^{\mathbb{A}}$ in OPEN_0 and/or OPEN_i for $i = 1, ..., n_h$ if some conditions are met (line 30-35 in Alg. 8). OPEN_i here denotes a priority queue for each heuristic sorted by key values defined as in line 2 in Alg. 8. Notice that POMHDP maintains multiple priority queues for the heuristics, while RTDP-Bel with a single heuristic does not have any.

This is for *rebranching* when the forward search falls into *stagnation*. When choosing the next belief to evaluate and back up, the algorithm first checks for *stagnation* which implies that the cost-to-goal along the forward search is not decreasing (Alg. 6). In such a case, it switches the heuristic reference index i_h to another in a round-robin fashion (Alg. 7), and then invokes OPEN_{i_h} to choose the belief with the minimal key value (Alg. 10). This is referred to as *rebranching* since the forward search stops growing the current belief tree branch and restarts in the middle of the existing branch(es). On the other hand, if the forward search is not in stagnation and has more than one valid successor belief-action pairs, the algorithm selects the best one with the minimal cost-to-goal among the successors (Alg. 11), which results in growing the current branch.

When choosing the belief-action pair with the minimal key value or cost-to-goal value, we have a certain condition called *anchor check* (line 2 in Alg. 10 and line 5 in Alg. 11, respectively). It basically compares the best candidates according to the admissible and inadmissible heuristic and decides which one to use. If the key value or cost-to-goal of the inadmissible's candidate is not larger (i.e., worse) than that of the admissible's one multiplied by ε_2, the inadmissible's candidate is used. Otherwise, admissible's one is used. In fact, Alg. 10 and 11 are mostly equivalent except that the former considers the whole belief tree from the start, while the latter considers only the partial tree rooted at the current belief. The myopic action selection scheme in Alg. 11 may not lead to the optimal solution at first but is still effective for asymptotic convergence to the optimal over the iterative forward search. The action selection in Alg. 10 is non-myopic and may find a better path from the start to a certain belief, but rebranching at every time does not provide a good convergence rate in practice because it takes more time to reach the goal which is the only belief with the true cost-to-goal (trivially, $V^*(b_g) = 0$) at the beginning.

Once a belief-action pair $b^a \in \mathbb{B}^{\mathbb{A}}$ is selected, POMHDP samples an observation and gets a corresponding successor belief $b \in \mathbb{B}$ for the next evaluation/backup iteration (Alg. 12).

Algorithm 5 POMHDP::MAIN

```
1: procedure MAIN()
2:     g(b_0) = 0;  g(b_g) = ∞                          ▷ cost-from-start
3:     for i = 0, ..., n_h do
4:         v_i(b_0) = ε_1 · h_i(b_0);  v_i(b_g) = 0     ▷ cost-to-goal
5:     repeat
6:         for i = 0, ..., n_h do
7:             OPEN_i ← ∅
8:         CLOSED_anchor ← ∅;  CLOSED_inad ← ∅
9:         b ← b_0;  i_h ← 1;  Δv ← Δv_0
10:        while ¬REACHEDGOAL(b, b_g) do
11:            v_pred ← v_{i_h}(b)
12:            EVALUATE(b)
13:            BACKUP(b)
14:            if OPEN_0 = ∅ then
15:                break
16:            if CHECKSTAGNATION(v_pred, i_h) or
17:          GETSUCCS(b) = ∅ then
18:                i_h ← SWITCHHEURISTIC(i_h)
19:                b^{a*} ← NEXTBELACTFROMOPEN(i_h)
20:                Δv ← Δv_0
21:            else
22:                b^{a*} ← NEXTBELACTFROMSUCCS(b,i_h)
23:            b ← SAMPLEBELIEFFROMBELACT(b^{a*})
24:        ε_1 ← ε_1 e^{-α};  ε_2 ← ε_2 e^{-α}
25:    until CONVERGED() or TIMEOUT()
```

Algorithm 6 POMHDP::CHECKSTAGNATION

```
1: procedure CHECKSTAGNATION(v_pred, b, i_h)
2:     q_succ = min_{a∈A} q_{i_h}(b^a) where b^a = SUCC(b, a)
3:     Δv ← ηΔv + (q_succ − v_pred)
4:     if Δv ≥ 0 then
5:         return true
6:     else
7:         return false
```

Algorithm 7 POMHDP::SWITCHHEURISTIC

```
1: procedure SWITCHHEURISTIC(i_h)
2:     i_h ← mod(i_h, n_h) + 1                          ▷ round-robin
3:     return i_h
```

Algorithm 8 POMHDP::EVALUATE

procedure KEY(b^a, i)
  return $g(b^a) + q_i(b^a)$
procedure EXPAND(b)
  for all $a \in \mathbb{A}$ **do**
    for all $o \in \mathbb{O}$ **do**
      $b^{ao} \leftarrow \emptyset$
    for N_{part} times **do**
      $s \sim b$
      $(s', o') \sim \mathcal{G}(s, a)$ ▷ $\mathcal{G}$: generative model
      for all $o \in \mathbb{O}$ **do**
        if $o = o'$ **then**
          $b^{ao} \leftarrow b^{ao} \cup \{s'\}$
    for all $o \in \mathbb{O}$ **do**
      SUCC(b^a, o) $= b^{ao}$
      $\tau(b, a, b^{ao}) = \frac{|b^{ao}|}{|b|}$ ▷ transition probability
      $g(b^{ao}) = \infty$
      for $i = 0, ..., n_h$ **do**
        $v_i(b^{ao}) = \varepsilon_1 \cdot h_i(b^{ao})$
    SUCC(b, a) $= b^a$
    $g(b^a) = g(b)$
procedure EVALUATE(b)
  if b was never expanded **then**
    EXPAND(b)
  for all $a \in \mathbb{A}$ **do**
    for $i = 0, ..., n_h$ **do**
      $q_i(b^a) = c(b, a) + \sum_{o \in \mathbb{O}} \tau(b, a, b^{ao})\, v_i(b^{ao})$
        ▷ cost-to-goal when taking action a where b^a = SUCC(b, a) and b^{ao} = SUCC(b^a, o)
    for all $o \in \mathbb{O}$ **do**
      if $g(b^{ao}) > g(b) + c(b, a)$ **then**
        $g(b^{ao}) = g(b) + c(b, a)$
    if $b^a \notin$ CLOSED$_{\text{anchor}}$ **then**
      Insert/update b^a in OPEN$_0$ with KEY(b^a, 0)
      if $b^a \notin$ CLOSED$_{\text{inad}}$ **then**
        for $i = 1, ..., n_h$ **do**
          if KEY(b^a, i) $\leq \varepsilon_2 \cdot$ KEY(b^a, 0) **then**
            Insert/update b^a in OPEN$_i$ with KEY(b^a, i)

Algorithm 9 POMHDP::BACKUP

procedure BACKUP(b)
  for $i = 0, ..., n_h$ **do**
    $\underline{q}_i = \min_{a \in \mathbb{A}} q_i(b^a)$ where b^a = SUCC(b, a)
    $v_i(b) = \underline{q}_i$

Algorithm 10 POMHDP::NEXTBELACTFROMOPEN

procedure NEXTBELACTFROMOPEN(i_h)
  if OPEN$_{i_h}$.MINKEY() $\leq \varepsilon_2 \cdot$ OPEN$_0$.MINKEY() **then**
    $b^a \leftarrow$ OPEN$_{i_h}$.TOP()
    Insert b^a in CLOSED$_{\text{inad}}$
  else
    $b^a \leftarrow$ OPEN$_0$.TOP()
    Insert b^a in CLOSED$_{\text{anchor}}$
  Remove b^a from OPEN$_i$ for $\forall i = 0, ..., n_h$
  return b^a

Algorithm 11 POMHDP::NEXTBELACTFROMSUCCS

procedure NEXTBELACTFROMSUCCS(b, i_h)
  $a_{\text{inad}} = \text{argmin}_{a \in \mathbb{A}}\, q_{i_h}(b^a)$ where b^a = SUCC(b, a)
  $a_{\text{anchor}} = \text{argmin}_{a \in \mathbb{A}}\, q_0(b^a)$ where b^a = SUCC(b, a)
  $\underline{q}_{\text{inad}} = q_{i_h}(b^{a_{\text{inad}}})$; $\underline{q}_{\text{anchor}} = q_0(b^{a_{\text{anchor}}})$
  if $\underline{q}_{\text{inad}} \leq \varepsilon_2 \cdot \underline{q}_{\text{anchor}}$ **then**
    return $b^{a_{\text{inad}}}$
  else
    return $b^{a_{\text{anchor}}}$

Algorithm 12 POMHDP::SAMPLEBELIEFFROMBELACT

procedure SAMPLEBELIEFFROMBELACT(b^a)
  $o \in \mathbb{O} \sim \tau(b, a, b^{a\boldsymbol{o}})$
  $b^{ao} \leftarrow$ SUCC(b^a, o)
  return b^{ao}

References

Aine, S.; Swaminathan, S.; Narayanan, V.; Hwang, V.; and Likhachev, M. 2016. Multi-Heuristic A*. *International Journal of Robotics Research (IJRR)* 35(1-3):224–243.

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *Artificial intelligence* 72(1-2):81–138.

Bertsekas, D. P. 2005. *Dynamic programming and optimal control, 3rd Edition*. Athena Scientific.

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *International Conference on Artificial Intelligence Planning Systems*, 52–61. AAAI Press.

Bonet, B., and Geffner, H. 2009. Solving POMDPs: RTDP-Bel vs. point-based algorithms. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1641–1646.

Bonet, B. 2002. An e-optimal grid-based algorithm for partially observable Markov decision processes. In *International Conference on Machine Learning (ICML)*.

Chakrabarti, P. P.; Ghose, S.; and DeSarkar, S. 1987. Admissibility of AO* when heuristics overestimate. *Artificial Intelligence* 34(1):97–113.

Coulom, R. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International Conference on Computers and Games*, 72–83. Springer.

Dionne, A.; Thayer, J.; and Ruml, W. 2011. Deadline-aware search using on-line measures of behavior. In *Symposium on Combinatorial Search (SoCS)*, 39–46.

Geffner, H., and Bonet, B. 1998. Solving large POMDPs using Real-Time Dynamic Programming. In *AAAI Fall Symposium on POMDPs*. Citeseer.

Gelly, S., and Silver, D. 2007. Combining online and offline

knowledge in UCT. In *International Conference on Machine Learning (ICML)*, 273–280. ACM.

Halperin, D.; Salzman, O.; and Sharir, M. 2017. Algorithmic motion planning. In *Handbook of Discrete and Computational Geometry, Third Edition*. CRC Press. 1311–1342.

Hansen, E. A., and Zilberstein, S. 1998. Heuristic search in cyclic AND/OR graphs. In *AAAI Innovative Applications of Artificial Intelligence Conference (IAAI)*, 412–418.

Hansen, E. A., and Zilberstein, S. 1999. Solving Markov decision problems using heuristic search. In *AAAI Spring Symposium on Search Techniques from Problem Solving under Uncertainty and Incomplete Information*.

Hansen, E. A., and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129(1-2):35–62.

Islam, F.; Salzman, O.; and Likhachev, M. 2018. Online, interactive user guidance for high-dimensional, constrained motion planning. In *Int. Joint Conference on Artificial Intelligence (IJCAI)*, 4921–4928.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101:99–134.

Kim, S., and Likhachev, M. 2017. Parts assembly planning under uncertainty with simulation-aided physical reasoning. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 4074–4081.

Kochenderfer, M. J. 2015. *Decision making under uncertainty: theory and application*. MIT press.

Kurniawati, H., and Yadav, V. 2013. An online POMDP solver for uncertainty planning in dynamic environment. In *International Symposium on Robotics Research (ISRR)*.

Kurniawati, H.; Hsu, D.; and Lee, W. 2008. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems (RSS)*.

LaValle, S. M. 2006. *Planning Algorithms*. Cambridge University Press.

Pearl, J. 1984. Heuristics: intelligent search strategies for computer problem solving.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1025–1032.

Pineau, J.; Gordon, G.; and Thrun, S. 2006. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research (JAIR)* 27:335–380.

Poole, D. L., and Mackworth, A. K. 2010. *Artificial Intelligence: foundations of computational agents*. Cambridge University Press.

Qian, N. 1999. On the momentum term in gradient descent learning algorithms. *Neural networks* 12(1):145–151.

Ranganeni, V.; Salzman, O.; and Likhachev, M. 2018. Effective footstep planning for humanoids using homotopy-class guidance. In *Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 500–508.

Ross, S., and Chaib-Draa, B. 2007. AEMS: An anytime online search algorithm for approximate policy refinement in large POMDPs. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2592–2598.

Ross, S.; Pineau, J.; Paquet, S.; and Chaib-draa, B. 2008. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research* 32:663–704.

Russell, S. J., and Norvig, P. 2010. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education.

Seiler, K. M.; Kurniawati, H.; and Singh, S. P. 2015. An online and approximate solver for POMDPs with continuous action space. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2290–2297. IEEE.

Shani, G.; Pineau, J.; and Kaplow, R. 2013. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems (AAMAS)* 27(1):1–51.

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

Smallwood, R. D., and Sondik, E. J. 1973. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research* 21(5):1071–1088.

Smith, T., and Simmons, R. 2004. Heuristic search value iteration for POMDPs. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 520–527.

Smith, T., and Simmons, R. 2005. Point-based POMDP algorithms: Improved analysis and implementation. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Somani, A.; Ye, N.; Hsu, D.; and Lee, W. S. 2013. DESPOT: Online POMDP planning with regularization. In *Advances in Neural Information Processing Systems (NIPS)*, 1772–1780.

Sunberg, Z., and Kochenderfer, M. 2017. POMCPOW: An online algorithm for POMDPs with continuous state, action, and observation spaces. *arXiv:1709.06196*.

Thrun, S.; Burgard, W.; and Fox, D. 2005. *Probabilistic robotics*. MIT Press.

Washington, R. 1997. BI-POMDP: Bounded, incremental partially-observable Markov-model planning. In *European Conference on Planning*, 440–451. Springer.