Planning under Uncertainty with Multiple Heuristics

Sung Kyun Kim

May 2019
CMU-RI-TR-19-26

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Maxim Likhachev (Chair)
Matthew T. Mason
Oliver Kroemer
Ali-akbar Agha-mohammadi, NASA/JPL-Caltech

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Abstract

Many robotic tasks, such as mobile manipulation, often require interaction with unstructured environments and are subject to imperfect sensing and actuation. This brings substantial uncertainty into the problems. Reasoning under this uncertainty can provide higher level of robustness but is computationally significantly more challenging. More specifically, sequential decision making under motion and sensing uncertainty can be formulated in a principled form as a Partially Observable Markov Decision Process (POMDP). Solving POMDPs exactly is computationally intractable due to their exponential complexity with the number of states and the depth of the planning horizon, so called, the curse of dimensionality and the curse of history, respectively.

In this work, we propose a novel search-based robust planning framework that sample-efficiently finds a solution with theoretical suboptimality bounds by leveraging multiple heuristics that are designed using domain knowledge. The main contributions of this work can be summarized as follows: 1) it works with generative models, in the absence of mathematical models, similarly to the reinforcement learning paradigm, but 2) it still exhibits high sample-efficiency by bootstrapping with the domain knowledge in the form of heuristics, and 3) it is empowered with effective guidance of the search toward the goal through the systematic employment of multiple heuristics. Its solution can be returned in an anytime fashion, and converges to the bounded suboptimality with theoretical guarantees.

We validate the proposed frameworks through simulation and robot experiments, which includes 2D rover navigation, PR2 parts assembly, and full size mobile manipulator truck unloading tasks. The truck unloading task is especially interesting, where the custom-built robot is to unload up to several thousands of boxes from a truck. It is a challenging real-world manipulation problem in continuous space with excessively high uncertainty, and we demonstrate the efficacy of our approach in such a domain.

Acknowledgments

First of all, I would like to thank my advisor, Maxim Likhachev, for his well-balanced advising which allowed me to explore research topics but also provided acute inspiring directives, technical guidance and supports to develop my skills and expertise while conducting the projects, and personal acknowledgments as a colleague researcher throughout my Ph.D. studies. I would also like to thank my thesis committee members, Matt Mason, Oliver Kroemer, and Ali Agha for the insightful, constructive, and supportive feedback and suggestions for my thesis. I enjoyed the time serving as a teaching assistant for Matt's class and appreciate giving me a chance to work together. The internship at JPL was also a great part in my thesis work, and I thank Ali Agha for being my mentor and providing inspiring and precise guidance.

I also would like to acknowledge my colleagues who had worked with me during my Ph.D. studies. I thank Oren Salzman for leading the project that is being an important part of this thesis, and thank Anirudh Vemula, Andrew Dornbush, Fahad Islam, Adrian Schoisengeier, and Abhijeet Tallavajhula for the great collaboration throughout the project. In addition, I thank Soonkyum Kim, Eui-Jung Jung, Mike Phillips, Venkat Narayanan, Kalin Gochev, and Siddharth Swaminathan, Rohan Thakker for working together during my early days of graduate studies. I also enjoyed the time to discuss the research directions with Wei Du, Tushar Kusnur, Ishani Chatterjee, Kalyan Vasudev, Kyungzun Rim, Shervin Javdani, and Jiaji Zhou.

Finally, and most importantly, I would like to give thanks to my family. This work would not have been done without the love and prayers of my parents and my sister. My wife, Ji Ae, is the one who deserves to be congratulated for the completion of this thesis. Her devotion and supports made me stay strong and move forward. I also thank my lovely children, Olivia Yewon and Aiden Yejun, for their energizing happy smiley faces.

Contents

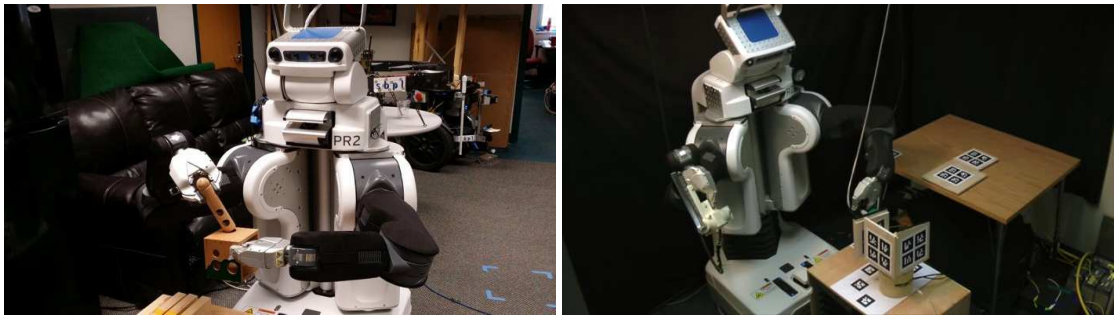
Chapter 1

Introduction

1.1 Motivation and Challenges

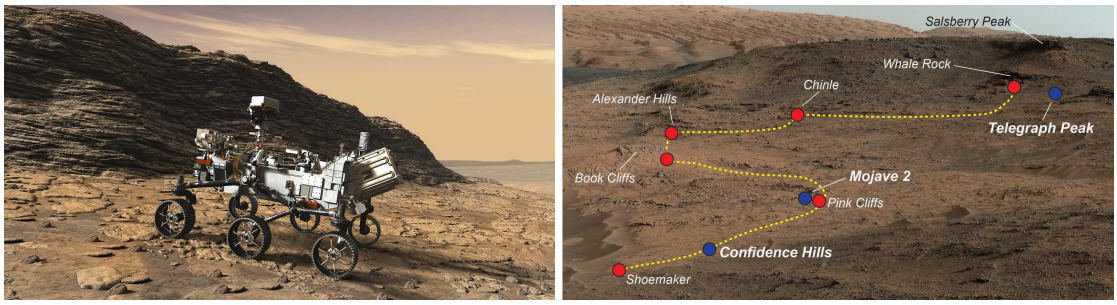
Consider a scenario where a mobile manipulator needs to assemble multiple parts for a birdhouse (Fig. 1.1b). The robot has to depend on its visual sensor data with limited resolution and accuracy for identifying and localizing a specific part. When the robot picks up a part using a vacuum gripper, noise in the real world comes in and makes the pose of the part slightly change. Under these sensing and motion uncertainty, the robot often suffers from putting the pieces in right place before nailing them together. To find a robust motion plan, it needs to take into account the amount of uncertainty at present and the effect of taking a sequence of actions.

We can also think of a Mars rover navigation example under uncertainty (Fig. 1.2). The rover's mission is to navigate through an obstacle-laden environment to scientific target locations. It is desired to complete the mission as soon as possible, while its safety should always be guaranteed. However, the rough terrain on Mars makes the exact motion of the rover unpredictable. When the rover is on feature-poor terrain such as sand, it cannot localize itself on the map precisely. Thus, it is important to find a plan that the rover can reach the target quickly while keeping itself away from hazard with a sufficiently small localization error.



(a) Peg-in-hole task. (b) Birdhouse assembly task.

Figure 1.1: Examples of manipulation tasks under motion and sensing uncertainty using PR2.



(a) NASA's Mars 2020 Rover. (Courtesy of NASA/JPL-Caltech.)

(b) Scientific targets on *Pahrump Hills*. (Courtesy of NASA/JPL-Caltech/MSSS.)

Figure 1.2: Examples of Mars rover navigation under motion and sensing uncertainty.

The above-mentioned problems are instances of a general problem of decision-making under uncertainty in the presence of risk and constraints. This problem in its most general and principled form can be formulated as a Partially Observable Markov Decision Process (POMDP) [1, 2]. It is known that the optimal solution of POMDP is intractable: *PSPACE-complete* for a finite horizon [3] and *NP-hard* for an infinite horizon [4].

The two main reasons of this complexity are called *the curse of dimensionality* [1] and *the curse of history* [1]. In a POMDP problem with n distinct states, a belief state which is a probability distribution over the state space is in $(n-1)$-dimensional continuous space. Even in the case the belief space is discretized, the complexity still grows exponentially with the number of states. When thinking of the value iteration algorithm for POMDPs, it starts from the initial belief and updates the values of the successor nodes in a breadth-first manner. Then the number of distinct action-observation sequence grows exponentially with the depth in the belief tree or the planning horizon at the end (Fig. 1.3a).

In this work, we particularly tackle a challenging class of POMDPs, here referred to as a *Goal POMDP*. (See section 2.2.2 for more information.) A Goal POMDP has an infinite horizon without discounting but has terminal belief states or conditions. Its extensive formulation is able to accommodate a variety of the robot motion planning problems under uncertainty with specific goals or safety-critical constraints, including the two scenarios introduced above.

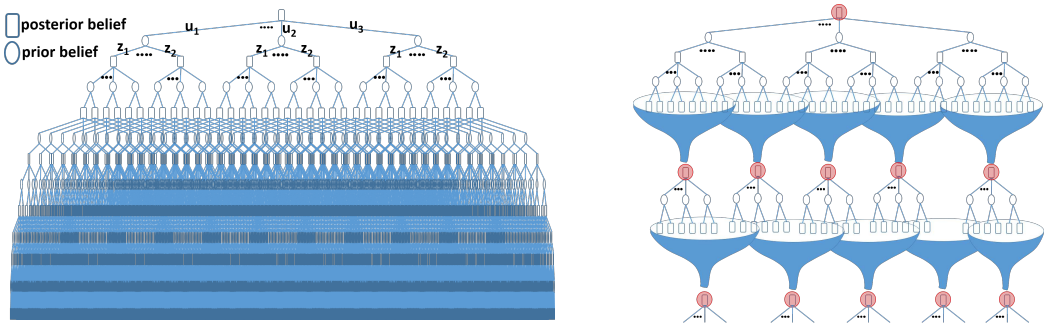
1.2 Proposed Approach

Many of the state-of-the-art POMDP solvers use forward search from the initial belief to the finite horizon or discount horizon to explore the reachable belief space. However, in Goal POMDP problems with infinite horizons in complex environments, it is not trivial to find a valid path that satisfy the goal condition. For example, a stochastic shortest-path finding problem in a 2D maze requires the planner to solve the maze at every episodic forward search. In this case, random rollout policy or simple heuristic such as preferred action set cannot efficiently guide the forward search to reach the goal.

In terms of heuristics, it is often not easy to design a single heuristic that captures all the complexities of the problem. Moreover, it is possible that a heuristic derived from domain-specific knowledge is informative but not consistent and admissible for the theoretic optimality guarantees.

We propose a new search-based belief space planning framework that leverages *multiple heuristics* to remedy these problems. A more complex guidance scheme can be represented by an ensemble of multiple heuristics, so that the forward search can be much more efficient in complex environments [5]. Given at least one admissible heuristic, we can guarantee its *completeness* and theoretic *suboptimality bounds* in conformant planning. In contingent planning, we can guide the search to explore ϵ-optimally reachable region in an anytime fashion, which helps the planner to cover the reachable region more efficiently and then find the solution faster [6].

As formalizing the belief space planning as a graph search problem, we adopt the recent graph construction technique into belief space. State-Lattice with Controllers (SLC)



(a) Belief tree constructed without local controllers.

(b) Belief graph constructed with local controllers.

Figure 1.3: Illustration of belief tree and graph obtained from forward simulation [8].

extends state-lattice based graph construction by incorporating controller-based motion primitives [7]. In other words, it gradually grows the graph by repeatedly adding successor states obtained from forward simulation of the local controls to the graph.

In belief space, this technique allows us to construct a belief graph instead of a belief tree. As depicted in Fig. 1.3, forward simulation under uncertainty in (continuous) belief space results in a belief tree where almost no belief states coincide with each other. However, a local feedback-controller being used in forward simulation can play a role as a belief funnel that merges nearby beliefs into a single belief, which effectively produces a belief graph.

Note that a belief tree directly suffers from the curse of history that causes exponential complexity with the search depth. On the other hand, a belief graph can have at best linear complexity with the search depth by breaking the dependency on the history. Another benefit of using local controllers is that we can pre-process or reuse policies as we can revisit the funnel nodes again.

1.3 Contributions

We summarize the contributions of this work. The proposed belief space planning framework is featured as follows.

- **Guidance by multiple heuristics:** We introduce how to make use of multiple

heuristics to guide the forward search in belief space. Multiple heuristics allow us to hedge our bets; we do not need to depend on a single heuristic nor struggle to design a heuristic that can capture all the complexities of the problem. Note that this framework can accommodate any domain-specific heuristics including ones that are used in point-based POMDP solvers. This is particularly effective when solving Goal POMDPs in complex environments with infinite horizons.

- **Belief graph construction with controllers:** The controller-based graph construction scheme is adopted into belief space. Given a generative simulation model, a belief graph is constructed from repeated forward simulations under the local controllers. There are two main benefits in this approach. First, the constructed graph represents the physically feasible world model. Second, the local controllers behave as funnels in belief space that can alleviate the exponential complexity with the search depth.

- **Scalable online-offline combination:** We provide a scalable online-offline POMDP solver framework. The near-optimal POMDP solvers including the proposed belief planner still have limited scalability due to POMDP's intrinsic complexity. We show the advantages of combining them with long-range approximate POMDP solvers such as Feedback-based Information RoadMap (FIRM) [9]. In this framework, a long-range approximate planner generates a sparse but global policy in offline, and a short-range near-optimal planner interleaves replanning and execution of a local policy that is connected to the offline global policy. The results show that this online-offline combination can provide better scalability while preserving the near-optimal solution quality.

- **Real-world application:** We aim at applying the proposed belief space planner to a real-world mobile manipulation problem with high complexity. More precisely, we are working on a package unloading task. The packages have different dimensions and weights that are unknown to the robot, and of course, the sensing and motion in the real world are under uncertainty. The mobile manipulator robot has more than 20 joints, and the number of packages to be unloaded can be more than several thousands. To solve this large and highly complex problem, we devised

a hierarchical belief space planning framework which consists of a low-level deterministic motion planner, a mid-level macro action instantiator, and a high-level belief space planner. The belief space planner generates plans over the macro action space, and the corresponding macro action is adaptively instantiated by the mid-level action instantiator based on the current observation.

Chapter 2

Background

In this chapter, we provide background information about search-based motion planning and Partially Observable Markov Decision Process (POMDP) and define belief space motion planning problems.

2.1 Search-based Motion Planning

Search-based motion planning is one of the core algorithms for robot motion planning in deterministic environments. Graph search algorithms are for solve graph traversal problems, i.e., the process of visiting nodes along their edges in a graph. They are used in many areas, including motion planning, where a problem can be formulated as a graph.

In the case of motion planning, a configuration state that uniquely specifies the entire system including robot joint positions, object poses, etc. is represented as a node in a graph. Transition relationship from a configuration state to another configuration state is represented by an edge in the graph. An action that invokes transition of a configuration state to another takes an action cost which is encoded in the graph as an edge weight. Then for a given start and goal configuration states and a graph representing the problem, a graph search algorithm can find a solution that minimizes the total cost from the start to the goal.

In the following subsection, we review more detail information about graph search algorithms.

2.1.1 Graph Search Algorithms

A* Search

One of the most popular graph search algorithms is A* search [10]. A* algorithm finds a minimum cost path from a start state $s_{start} \in S$ to a goal state $s_{goal} \in S$ in a directed graph $G(S, E)$. S is the set of nodes in the graph and E is the set of edges that connect the nodes. Note here that A* only works for deterministic graphs where each edge $e \in E$ connects a single node s to another single node s'. An edge cost function $c : S \times S \rightarrow \mathbb{R}^+$ maps a pair of states, i.e., each edge $e(s, s')$, to a positive scalar-valued cost. If s and s' are not connect in the graph, $c(s, s') = \infty$.

In A* algorithm, each state is associated with three values. The g-value $g(s)$ is the minimum path cost accumulated over the edges on the best path from s_{start} to s. The h-value $h(s)$ is a heuristically estimated minimum path cost accumulated over the edges on the best path fro s to s_{goal}. The f-value $f(s) = g(s) + h(s)$ is an estimated minimum path cost from s_{start} to s_{goal} that passes through s and is also referred to as $priority$ of state s.

Algorithm 1 shows the A* algorithm. OPEN represents a priority queue called *OPEN list* that contains all the states that have been discovered but not yet expanded. Starting with OPEN containing s_{start} only, A* repeatedly expands the state in OPEN with the minimum f-value, i.e., with the highest priority. The state expansion process, ExpandState(), consists of two operations. One is to find or generate a successor of the expanded state for each action. Then the g-value of the successor is updated by the best path cost found so far. When s_{goal} is to be expanded, i.e., s_{goal} has the highest priority in OPEN, the search process terminates and return the best path found.

Several theoretic properties of A* search depends on the heuristic function $h(s)$. If the heuristic is *consistent*, i.e., it satisfies the triangle inequality $h(s) \leqslant h(s') + c(s, s'), \forall s, s' \in S$ and $h(s_{goal}) = 0$, then A* is guaranteed not to expand a state more than once. If the

Algorithm 1 A* Search

```
1: procedure ExpandState(s)
2:   remove s from OPEN
3:   for all s' ∈ GetSuccessors(s) do
4:     if s' was not visited before then
5:       g(s') ← ∞
6:     if g(s') > g(s) + c(s, s') then
7:       g(s') ← g(s) + c(s, s')
8:       insert/update s' in OPEN with priority f(s') = g(s') + h(s')
9: procedure A*(s_start, s_goal)
10:  OPEN ← ∅
11:  g(s_start) ← 0
12:  f(s_start) ← h(s_start)
13:  insert s_start in OPEN with priority f(s_start)
14:  while s_goal is not expanded do
15:    if OPEN is empty then return null
16:    remove s with the smallest f-value from OPEN
17:    ExpandState(s)
18:  return ReconstructPath()
```

heuristic is *admissible*, i.e., it never over-estimates the best path cost from s to s_{goal} for $\forall s \in S$, then A* is guaranteed to find the optimal (minimum cost) path. Note that if a heuristic is consistent, it is also admissible, but not vice versa.

Variants of A* Search

There are many variants of A* search. Weighted A* inflates the heuristic function, so that the search is more goal-directed [11]. This inflation leads to suboptimality of the returned solution, but it is proven that the suboptimality is bounded by the factor of inflation. Anytime Repairing A* finds an initial solution quickly with high inflation of the heuristic and then improves the solution quality over time by decreasing the inflation factor and replan [12]. Multi-Heuristic A* leverages multiple and possibly inadmissible heuristics to guide the search in a sophisticated manner [5]. By limiting the use of inadmissible heuristics based on admissible heuristic evaluation, it preserves the theoretic guarantees on the suboptimality bounds.

2.2 Partially Observable Markov Decision Process

2.2.1 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., [13, 14, 15]

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.

$$b_{k-1}(s; a_{k-1}) = \sum_{s' \in \mathbb{S}} T(s, a_{k-1}, s') b_{k-1}(s), \tag{2.1}$$

$$b_k(s; a_{k-1}, o_k) = \eta Z(s, a_{k-1}, o_k) b_{k-1}(s; a_{k-1}), \tag{2.2}$$

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,

$$V(b; \pi) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k c(b_k, \pi(b_k))\right], \tag{2.3}$$

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. 2.3 in a recursive form, which is called the Bellman equation.

$$V(b; \pi) = c(b^i, \pi(b)) + \gamma \sum_{b' \in \mathbb{B}} \tau(b, \pi(b), b') V(b'; \pi), \tag{2.4}$$

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. 2.1 and 2.2. For further details, see [16]. 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.

$$Q(b^a; \pi) = c(b, a) + \gamma \sum_{b' \in \mathbb{B}} \tau(b, a, b') V(b'; \pi). \tag{2.5}$$

Then (2.4) can be written as follows.

$$V(b; \pi) = \min_{a \in \mathbb{A}} Q(b, a; \pi) \tag{2.6}$$

We now restate our POMDP problem as an optimization problem.

$$\begin{aligned} \pi^*(b) &= \operatorname*{argmin}_{\Pi_{0:\infty}} \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k c(b_k, \pi_k(b_k))\right] \\ &= \operatorname*{argmin}_{\Pi_{0:\infty}} V(b_k; \pi_k). \end{aligned} \tag{2.7}$$

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.

2.2.2 Discounted POMDP vs. Goal POMDP

The POMDP problems can be categorized into two classes: *Discounted POMDP* and *Goal POMDP*. We discuss each class of POMDP problems in this subsection.

Discounted POMDP

Some POMDP problems have specific terminal conditions and others do not. In the case there is no terminal condition or the size of the problem is too large, it is often to discount the cost or reward far from the current belief, which effectively limits the (possibly infinite) planning horizon to a finite one [17]. We refer to this class of POMDP problems as *Discounted POMDPs*. Many of the POMDP solvers aim at this class of problems including the point-based methods that will be reviewed in section 3.1.1.

Discounted POMDP formulation is well suited for dynamic environments or adversarial settings. In those cases, the effect of actions that will happen far later is under high uncertainty, and thus, it is not much worth to deliberately compute their effects on the total cost or reward of the current belief.

Goal POMDP

The POMDP problems with terminal conditions and without cost/reward discount is referred to as *Goal POMDPs*. It can be understood as a stochastic shortest path problem (with the minimum cost) for the given start and goal under motion and sensing uncertainty. As it equally takes into account the actions that will happen later as the immediate actions, Goal POMDP formulation is suitable for static environment or safety-critical systems.

It should be noted that, however, any types of Discounted POMDPs can be transformed into Goal POMDPs by adding a terminal node and encoding the discount factor in the transition probabilities [18]. This means that POMDP problems for dynamic environments or adversarial settings can also be properly represented by Goal POMDPs, thanks to the richer expressiveness of Goal POMDPs.

2.2.3 Belief Space Planning

There are two kinds of belief space planning [19]. We briefly review them in this subsection.

Conformant Planning

Conformant planning refers to planning with incomplete information but *with no sensor feedback* [20]. It is also called *conditional planning* in some literature [21]. It considers a problem under motion uncertainty where state transition by an action can be modeled as either a non-deterministic function or a probability distribution.

Conformant planning can be conveniently formulated as deterministic planning in belief space. It means that the non-deterministic or probabilistic transition can be encoded in the belief state and then the transition between belief states by an action becomes deterministic. Although the belief state is still dependent on the history of the applied action sequence, this problem can be solved by deterministic planning algorithms such as A* search.

Contingent Planning

Contingent planning refers to planning with incomplete information *with sensor feedback* [22]. It considers a problem under motion and sensing uncertainty where both action and observation are non-deterministic or probabilistic. Especially when action and observation are modeled as probabilistic distributions, it can be formulated as a POMDP.

Since observations cannot be predicted or selected like actions, it is not possible to transform contingent planning into deterministic planning. When formulating this problem as a graph, it will be an AND-OR graph, not a deterministic graph. Therefore deterministic search algorithms cannot solve this problem but, rather, dynamic programming such as value iteration should be used.

Chapter 3

Related Work

In this chapter, we review previous work in three primary areas: motion planning in deterministic environments, belief space planning algorithms, and applications of motion planning under uncertainty. Works on belief space planning are not limited to motion planning, so we will cover general algorithms in several categories based on their approaches. At last, interesting applications related to motion planning under uncertainty are reviewed.

3.1 Belief Space Planning

In belief space planning, we categorized the previous work into four. We review each of categories and address its key ideas to tackle POMDP problems. Note that assuming the belief space planning problems we consider here are formulated in probabilistic representation, the terms, *a belief space planner* and *a POMDP solver*, will be used interchangeably.

3.1.1 Point-based Methods

Point-based methods have been one of the major approaches to solve larger POMDP problems. They are named as point-based POMDP solvers because they in common

represent the value function by a vector set of sampled points and its piece-wise linear combination [23, 24]. This representation enables them to improve the lower bound (and the upper bound in some algorithms) of the value function (in the reward setting) by sampling a new belief state. The process of updating the value function based on a new sample is called *backup*. By iterative sampling and backup, it is proven to converge to the optimal solution.

Another important idea in this approach is to restrict the search space only that is reachable from the given initial belief state. By starting from the initial belief and following a certain heuristic strategy for the next sampling, the search space grows that is reachable from the initial belief, where the final solution would reside. There are many different heuristic strategies how to guide the sampling, and this has been the main contributions of the variants of this approach. Note that a heuristic in point-based methods does not necessarily mean consistent and admissible heuristic as in search-based methods.

One thing to notice here is that this approach maintains a value function for explicit probability distribution models and updates the value function by considering all possible actions in a full-width planning manner. This helps to generalize the value function for the neighbor belief states of the sampled ones, but induces higher complexity and poor scalability to larger problems.

Pineau *et al.* have first established this point-based method called Point-Based Value Iteration (PBVI) with lower bound of the value function [24, 25]. They introduced the idea of searching for the reachable space only and address the benefit of maintaining a set of vectors as a representation of the lower bound of the value function. This representation is possible because the optimal value function is always piece-wise linear and convex in the belief space [23]. They also pointed out *the curse of history* in POMDP problems.

Smith and Simmons proposed Heuristic Search Value Iteration (HSVI) with a heuristic strategy that guides the search where the gap between the upper and lower bounds of the value function is large [26, 27]. More precisely, they select an action with the greatest upper bound and an observation with the largest contribution of the bound gap at the initial belief. The basic idea behind it is that exploring a region with high uncertainty helps to converge to the optimal faster. Unlike the lower bound, the upper bound of the

value function cannot be represented by a vector set. Thus, they represent the upper bound by a point set and its convex hull that should be solved by linear programming. The initial values of the upper bound is computed by assuming full observability and solving the MDP version of the problem [28]. Other approaches to initialize the upper and lower bounds can be found in [16, 29].

The state-of-the-art point-based POMDP solver (in offline) is considered to be SARSOP (Successive Approximations of the Reachable Space under Optimal Policies) presented by Kurniawati *et al.* [30]. This algorithm follows the same ideas of other point-based approaches. It constructs a belief tree by full-width expansion through forward search while maintaining the upper and lower bounds of the value function. The novelty of this algorithm lies in that it tries to expand *optimally* reachable space unlike PBVI or HSVI. It carefully keeps a balance in the termination condition of forward search, i.e., it tries to make the sampling path as shallow as possible but as deep enough to reduce bound gap at the initial belief and reach a high expected reward. This is accomplished by terminating the forward search if the lower bound of the sample node no longer increases and the upper bound of the sample node provides a sufficiently small gap at the root with discounting. It also utilizes pruning of the optimally reachable belief tree for the sake of backup efficiency.

Point-based methods can be extended to anytime online versions. Anytime Error Minimization Search (AEMS) by Ross *et al.* is one of the early successful online planner [31], which guides the forward search where the expected error between the lower and upper bounds can be minimized. More online point-based methods can be found in [16].

3.1.2 Heuristic Search-based Methods

Another major approach to belief space planning is a search-based method using consistent and admissible heuristics. The key idea of this approach is to utilize available domain-specific knowledge to guide the search and find the optimal solution faster. Unlike dynamic programming such as value iteration, heuristic search methods searches from the start to the goal and do not need to evaluate the entire belief space. Notice

that this is a similar idea in point-based methods to restrict the search space to that is reachable.

As its name suggests, heuristic search-based approach is derived from graph search algorithms in deterministic environments. Under motion and sensing uncertainty in a POMDP setting, the tree or graph is no longer deterministic, in other words, probabilistic transition along an edge should be considered. To cope with this stochastic graph structure, several methods have been developed. Note that in heuristic search-based methods, the value function is usually in the cost setting (the lower the better) unlike in the reward setting (the higher the better) in point-based methods.

LAO*, which represents AO* algorithm for MDPs or belief MDPs with Loops, was proposed by Hansen and Zilberstein [32, 33, 34]. AO* algorithm is an extension of A* search algorithm that can solve problems formalized as an acyclic AND-OR graph (i.e., without loops). In deterministic graphs an edge connect one node to another single node, but in AND-OR graphs an edge, sometimes called *hyperarc* or *k-connector* can connect a node to multiple nodes. The chance of transition from one node to each successor node is represented by a transition probability distribution. AO* algorithm repeats *forward expansion* of the best node in the current belief tree and backward induction from the leaf node back to its predecessors. Forward expansion is guided by an admissible heuristic, which helps to explore a region where the optimal solution is likely to reside. LAO* generalizes the backward induction process to value iteration or policy iteration. Thus, it requires more computation compared to AO* but can handle AND-OR graphs with loops. Once it reaches the goal node for the first time, then it returns the solution, and it is proven to converge to the optimal solution. One thing to note is that the order of value iteration may heavily affect the efficiency of this algorithm.

Washington suggested Bounded, Incremental Search for Partially Observable Markov Decision Processes (BI-POMDP) [35]. It is also an extension of AO* algorithm for MDPs or belief MDPs. By running AO* in an incremental fashion, it could achieve better memory efficiency. Additional feature of this method is that it keeps track of the upper bounds of costs (i.e., lower bounds of rewards) as well as the lower bounds of costs. The initial lower and upper bound values (in the cost setting) are computed by solving

offline MDP optimally and in the worst-case, respectively. Then the forward expansion is biased to nodes with larger gaps between the upper and lower bounds, which is also adopted in the point-based methods such as HSVI [26, 27].

RTDP-Bel presented by Bonet and Geffner is another heuristic search-based algorithm but from a different perspective [19, 36, 37, 38]. While LAO* or BI-POMDP updates the values for the whole belief tree (or within a finite horizon in BI-POMDP) after every single forward expansion, RTDP-Bel runs forward simulation along a single branch from the start to the goal and updates the values only for the nodes on the branch. RTDP-Bel is originated from LRTA* [39] for deterministic environments and RTDP [40] for MDP problems under motion uncertainty. The forward simulation along a branch is called a *trial*, and these algorithms keep updating the value function (heuristic function in LRTA* case) over iterative trials. As heuristic search-based methods, RTDP-Bel initializes the value function by an admissible heuristic and converges to the optimal value function (up to belief discretization resolution) through repeated trials. The admissible heuristics for initialization can be computed by solving MDP with full observability assumption [28]. In each expansion during forward search, it follows the greedy policy (by fully expanding the node for all possible actions and selecting the best action) and gets the probabilistic outcomes (successors). The value update is done immediately after selecting the next action in an asynchronous dynamic programming fashion [40].

There are other several variants of RTDP that can be applied to MDPs or belief MDPs. One is Labeled RTDP by Bonet and Geffner [41] that labels states with converged values as *solved*, so that the value function can converge faster within a bounded number of trials. The other is Bounded RTDP by McMahan *et al.* [42] which maintains the upper bound of the value function (in the cost setting) as well as the lower bound and guide the search toward less-well-understood states with larger gaps between the bounds.

One remarkable paper for heuristic search-based methods is the comparison of RTDP-Bel and point-based methods [18]. In this work, they converted a *Discounted POMDP* problem into an equivalent *Goal POMDP* problem and compared RTDP-Bel and other point-based methods. In general, heuristic search-based methods aim at *Goal POMDPs* in the cost setting that have some specific terminal states or conditions without cost

discounting, while many point-based methods are for *Discounted POMDPs* in the reward setting where there is discounting but not necessarily terminal conditions. 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 utilizes Sondik's vector representation of the value function [23].

3.1.3 Monte Carlo-based Methods

Monte Carlo-based methods for POMDPs are initiated by Silver and Veness who proposed Partially Observable Monte Carlo Planning (POMCP) [43]. Monte Carlo Tree Search (MCTS) has been successful in solving MDPs [44, 45], and they extended this approach to belief space planning. Two important features in this work are the use of UCT (Upper Confidence Bound applied to Trees) algorithm for guiding the tree search and the belief state representation as a set of particles. UCT algorithm helps to balance between the exploration and exploitation, and it is proven that UCT algorithm converges to the optimal in the fully observable MDP setting (as its bias toward exploration disappears as the number of visits to a state goes to infinite). In order to convert a POMDP to a MDP, they represent a belief state as a set of particles, which implicitly means that they sample the belief state (alleviating the curse of dimensionality) and sample the belief transitions (alleviating the curse of history). Additionally, it is an online POMDP solver, which means it interleaves planning and plan execution so that it can improve the solution quality during execution. POMCP, with particle representation instead of explicit probabilistic models and without any upper/lower bound updates, is shown to be working surprisingly well in larger problems, and inspired many other variants.

DESPOT (Deteminized Sparse Partially Observable Tree) suggested by Somani *et al.* is one of the successful variants [46]. POMCP is sometimes too much greedy and thus leads to a bad worst-case performance. DESPOT tries to improve the worst-case performance by considering a small number of challenging scenarios and improving the optimality within the sparse tree.

ABT (Adaptive Belief Tree) by Kurniawati and Yadav is also based on POMCP [47]. It

reuses and improves the existing solution and update the solution as needed whenever the POMDP model changes. It has an extension to continuous action space called GPS-ABT (General Pattern Search - Adaptive Belief Tree) [48].

POMCPOW (Partially Observable Monte Carlo Planning with Observation Widening) is proposed by Sunberg and Kochenderfer to extend POMCP to continuous action and observation space [49]. As POMCP depends on UCT algorithm for tree search, which forces to expand an unvisited node if exists, it is only able to expand a single-depth layer of belief tree in continuous action or observation space. To remedy this limitation, POMCPOW limits the number of child nodes based on Progressive Widening technique both for action and observation space, and then it can construct a deeper belief tree.

3.1.4 Feedback Controller-based Methods

There is another camp in belief space planning that utilizes local feedback controllers. Agha-mohammadi *et al.* proposed Feedback-based Information RoadMap (FIRM) that adopted a probabilistic roadmap scheme in belief space [9, 50]. In continuous space under motion and sensing uncertainty there is a only rare chance for a belief to evolve to the same belief state that has been visited before, which results in exponential complexity with the planning horizon, i.e., *the curse of history*. However, FIRM leverages local feedback controllers to make the current belief to converge to an existing belief, which is called a FIRM node, in the roadmap. Convergence to a FIRM node implies that any histories, i.e., previous action and observation sequences, that have brought a belief to this FIRM node are no longer needed to be considered. The belief states in the roadmap are obtained from random sampling and connected to their neighbor beliefs. The edges between FIRM nodes are denoted by FIRM edges. Then the constructed roadmap represents a belief MDP, which can be solved efficiently. For the given start and goal belief states, FIRM can quickly return a solution that bring the belief to a nearby FIRM node and guide it along the FIRM edges.

FIRM-Rollout is an extension of FIRM with online replanning [8]. On top of offline FIRM policy, it interleaves online replanning and execution in order to improve the solution

quality as well as to adapt to dynamic changes in the environment.

On the other hand, van den Berg *et al.* presented a belief space variant of iterative Linear Quadratic Gaussian control (BS-iLQG) [51]. It assumes a belief state to be a Gaussian distribution and utilizes LQG controllers for iterative local optimization. In more detail, it computes an initial solution, for example, through Rapidly-growing Random Tree (RRT) algorithm [52] in a deterministic setting, then it performs forward simulation and backward recursion to improve the solution quality over iterations. By iterative local optimization it can converge to a local minimum, but its convergence to the global minimum is subject to the initial solution.

3.2 Applications of Motion Planning under Uncertainty

In this section, we review some of interesting applications using motion planning under uncertainty, specifically, for two areas of interest: ground vehicle navigation and mobile manipulation.

3.2.1 Ground Vehicle Navigation

In the real world, ground vehicle navigation undergoes a lot of uncertainty. For example, the controlled motion is not as the same as desired due to many noises, such as uneven terrain or slippery surface, that cannot be modeled precisely. The perception for the vehicle is not perfect as well, which can increase the risk of collision with obstacles.

For robust navigation, there are two conditions that should be pursued. One is collision avoidance. The vehicle should not collide with obstacles for safety reasons, and this is a strict condition that should be satisfied. The other is accurate localization both for obstacles and vehicle itself. While GPS (Global Positioning System) is one of the most source for localization in outdoors, visual sensor data is also important sensor data for localizing the vehicle on the map and static/dynamic obstacles around the vehicle. For better vision-based localization, it is crucial to obtain feature-rich observations.

Gonzalez and Stentz have addressed this ground navigation problems while considering both of collision avoidance and accurate localization [53, 54]. They modeled the belief state as a Gaussian distribution to formulate the problem as a belief MDP and used classical graph search algorithms in belief MDPs. The results showed that this approach could find a path with the minimal cost that ensures the vehicle has a very little chance of collision with obstacles by approaching to feature-rich landmarks as necessary.

Other related work also assumes Gaussian belief space for the sake of its compact belief representation. Prentice and Roy constructed a Gaussian belief roadmap that can approximate the continuous space into a finite graph structure [55]. (Note that this approach does not utilize feedback controllers to stabilize a belief to the graph node beliefs and break the curse of history as in FIRM.) Bry and Roy extended RRT to Gaussian belief space so that they can incrementally construct a belief tree and find a path to the goal with a small chance of collision [56]. Lenz *et al.* used heuristic search algorithms for a Gaussian belief MDP while considering non-holonomic constraints of the ground vehicle [57].

3.2.2 Mobile Manipulation

There are many applications in manipulation that considers motion and sensing uncertainty. Manipulation involves a lot of interaction with the environment which is a main source of uncertainty, and thus, many previous work tried to reduce such uncertainty for more robust manipulation.

Some earlier work in this context targeted robust part feeding by utilizing contact between the parts and the environment to align its pose [58, 59]. Basically, these approaches tried to reduce uncertainty in object orientation on a plane by a sequence of actions. They required rigorous analysis of the configuration space that particularly depends on the shape of the object, which is usually possible in 2D space only.

More recent works inheriting this scheme can be found in [60, 61]. In these works, the robot exploits contact with the environment or other objects for robust grasping or in-hand manipulation. (It is called *extrinsic dexterity* in [60].) They rely on the idea of using

contact to reduce uncertainty and successfully demonstrate its effectiveness. However, there was no general motion planner and the motion sequences are hand-scripted.

Zhou *et al.* proposed a belief space planning algorithm for 2D object pose identification [62]. This work uses particle filter representation for a belief state and open-loop search over a finite number of actions to construct a belief tree in offline. From this process, it generates a policy that can be used to determine a sequence of actions for object pose identification.

Narayanan and Likhachev presented a work for motion planning under environment model uncertainty [63]. Assuming uncertainty in the articulated object models, such as doors or drawers, but not in the robot state and object poses, it constructs a belief MDP with a probability distribution of the articulated object model. They solved this belief MDP using LAO* algorithm and showed that a robot can identify the articulation type of the given object by a sequence of actions from the planner.

Haustein *et al.* and Koval *et al.* proposed belief space planners for 2D object pushing under uncertainty [64, 65]. They utilize a simulator for physical reasoning and RRT or A* variant search algorithms for motion planning. Especially in [65], they decomposed the policy into pre- and post-contact stages to reduce the complexity. The post-contact policy that involves interaction with the planar object under uncertainty is solved offline on a discretized belief space using SARSOP, while the pre-contact policy is modeled as an open-loop trajectory assuming no observations and belief updates until the contact.

There are abundant recent works on manipulation based on reinforcement learning approach [66, 67]. They used a reinforcement learning technique called *Guided Policy Search* for convolutional neural networks for different parts assembly tasks. They trained the deep network in the MDP setting using visual sensor data and joint positions as inputs and joint torques as outputs, and showed that the robot can perform a robust manipulation task using a global policy that is generated from several hours of training for that specific task.

Chapter 4

Planning under Motion Uncertainty with Multiple Heuristics

4.1 Introduction

In this chapter, we consider a conformant planning problem under motion uncertainty. As a concrete example, parts assembly task is being presented here [68].

Parts assembly, in a broad sense, is to make multiple objects to be in specific relative poses in contact with each other. One of the major reasons that make it difficult is uncertainty. Because parts assembly involves physical contact between objects, it requires higher precision than other manipulation tasks like collision avoidance. The key idea of this work is to use simulation-aided physical reasoning while planning with the goal of finding a robust motion plan for parts assembly. Specifically, in the proposed approach, a) uncertainty between object poses is represented as a distribution of particles, b) the motion planner estimates the transition of particles for unit actions (motion primitives) through physics-based simulation, and c) the performance of the planner is sped up using Multi-Heuristic A* (MHA*) search that utilizes multiple inadmissible heuristics that lead to fast uncertainty reduction. To demonstrate the benefits of our framework, motion planning and physical robot experiments for several parts assembly tasks are provided.

Parts assembly is happening not only in factories but also in most of our living spaces. It is an operation of arranging, stacking, or combining objects. Many physical manipulation tasks that are to set relative poses between multiple objects in contact can be considered as parts assembly in a broad sense.

The challenge, however, is that contact involves complex physical phenomena such as friction and force interaction in addition to kinematics and dynamics of objects. These phenomena are hard to estimate or approximate well. Moreover, parts assembly usually requires higher precision in motion because it is a practice of placing objects to fit in, not separating them apart.

Therefore, it is important to reason about the underlying physics and to execute manipulation plans that are robust to uncertainty during assembly. In this work, we propose a framework to utilize a physics-based simulator for physical reasoning and use graph search algorithms to find a robust motion plan in belief space.

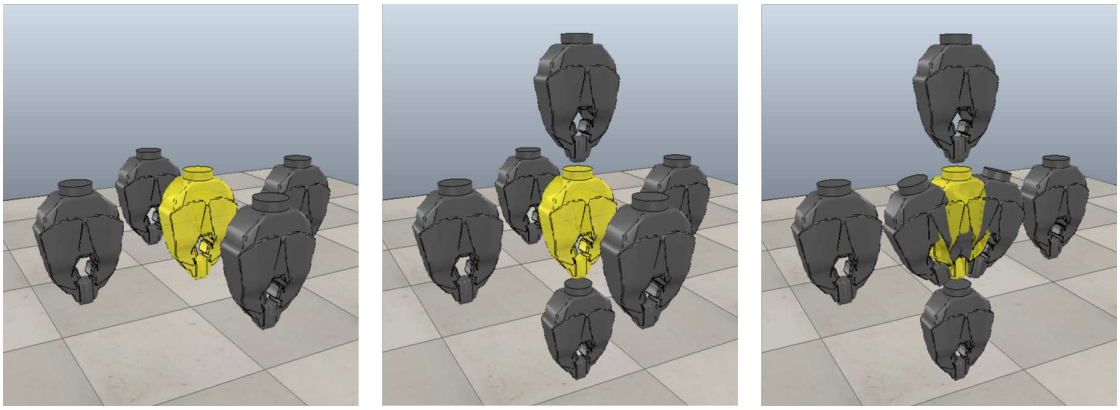
The rationale behind selecting a simulator for physical reasoning is that modeling complex manipulation actions to sufficient degree of fidelity is infeasible. Instead, we should exploit state-of-the-art in physics-based simulation while planning.

By sampling uncertainty distribution and running physics-based simulation of actions, the planner constructs a belief space. The planner then runs a heuristic search to find a plan while reducing the uncertainty for a higher chance of success.

4.2 Belief Space Representation

4.2.1 Belief State

Since we assume there is uncertainty in the state of robot and objects, we need to encode uncertainty into the state of the graph. In this work, we represent a state of a graph s as a belief state which is a set of particle sub-states $\mathbb{P} = \{P_0, P_1, ..., P_m\}$ (see Fig. 4.3). Each particle P_j consists of the pose information of the robot and objects $\{(p_r, R_r), (p_{o1}, R_{o1}), ..., (p_{on}, R_{on})\}$.



(a) 2D (x, y) (b) 3D (x, y, z) (c) 3D (y, z, R)

Figure 4.1: An example of belief state representation: particles of cross-polytopes (hyperoctahedron) in different search spaces.

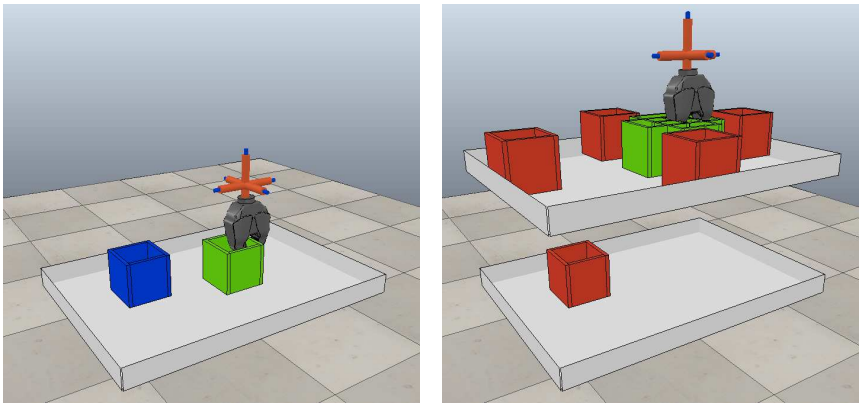
For parts assembly tasks, we accept Gaussian distribution as the initial uncertainty of a given task, but note that any distributions can be used as long as the number of particles are sufficient to represent the distribution.

The typical way to draw particles from a given Gaussian distribution is random sampling. However, we cannot employ plenty of random samples to properly represent the original distribution because the simulation for each particle is computationally expensive. Alternatively, we use vertices of a uniform polytope as the particles since it can be seen as an approximation of c_p-sigma ellipsoid where c_p is a constant which is set to 2 in this work. Amongst uniform polytopes, we used a cross-polytope which is a higher dimensional octahedron for particle generation as shown in Fig. 4.1.

4.2.2 Cost Function in Belief Space

Before getting into the details for belief space search, we describe the special structure of the configuration space under uncertainty.

First, let us take an example. As shown in Fig. 4.2(a), the robot gripper is holding a box and wants to place it at the position of the blue box accurately. (This task will be called *box-on-table* hereafter.) However, there is uncertainty in the initial pose of the gripper and the box (depicted in a green color). How can we represent the amount of uncertainty of



(a) A given planning problem. (b) (x, y, u)-configuration space.

Figure 4.2: Illustration of 2D box-on-table task. (a) The green box is the start pose, and the blue box is the goal pose. (b) The red boxes are the contact and goal attractors for inadmissible heuristics. Uncertainty u in the configuration space is visualized in z-axis. The green box on the upper level has high uncertainty and needs to get contact with the walls to reduce pose uncertainty and get down to the lower level of uncertainty where the goal state exists.

each belief state?

Let us define a particle-aggregated belief state vector X which describes the distribution of particles.

$$X = \begin{bmatrix} x \\ \omega \\ u \end{bmatrix} = \begin{bmatrix} \mathbb{E}_j[p_j^{tool}] \\ \mathbb{E}_j[\omega_j^{tool}] \\ u \end{bmatrix} \tag{4.1}$$

where j is the index for particles. p is a position vector, and $\omega = \theta\hat{\omega}$ is an orientation vector that can be obtained from angle-axis representation where θ and $\hat{\omega}$ are the rotation angle and rotation axis, respectively. Basically it is composed of mean vectors of position and orientation of the tool frame in addition to a scalar-valued uncertainty measure. (The tool frame is a local frame attached the object held by the gripper and, in this example, is at the center of the bottom of the box.) Note that position and orientation are with respect to the local frame of the table which is the target object.

In this work, uncertainty is defined as a weighted sum of traces of sample covariance

matrices for position and orientation as follows:

$$
\begin{aligned}
u = w_p \mathrm{Tr}\left(\mathbb{E}_j\left[\left(p_j^{tool} - \mathbb{E}_j\left[p_j^{tool}\right]\right)\left(p_j^{tool} - \mathbb{E}_j\left[p_j^{tool}\right]\right)^T\right]\right) \\
+ w_o \mathrm{Tr}\left(\mathbb{E}_j\left[\left(\omega_j^{tool} - \mathbb{E}_j\left[\omega_j^{tool}\right]\right)\left(\omega_j^{tool} - \mathbb{E}_j\left[\omega_j^{tool}\right]\right)^T\right]\right)
\end{aligned} \tag{4.2}
$$

where w_p and w_o are weights for position and orientation, respectively. Note that the trace of covariance matrix is used instead of the determinant due to its numerical unstability.

Now notice that we have a uncertainty measure coordinate in addition to pose coordinates in the belief state vector. It means that we have (d+1)-dimensional configuration space for d-dimensional planning problem under uncertainty. Figure 4.2(b) illustrates the (x, y, u)-configuration space for a planning problem in (x, y)-space.

One important physical fact to note is that the uncertainty of the relative pose between objects cannot be reduced without getting in contact with each other, provided that all the executable actions are to move around in the space. In the box-on-table example, a belief state on a high uncertainty level (the green box in Fig. 4.2(b)) cannot get down to the lower level of uncertainty without contact with the walls. In a word, this phenomenon leads to *foliation* of the configuration space.

4.3 Belief Graph Construction by State-Lattice with Controllers

In order to formulate motion planning as a graph search problem, we need to represent the search space as a graph. One can discretize the world into n-dimensional grid and take each cell as a node on a graph, but in this work, we adopt a graph construct called *State Lattice with Controllers (SLC)* [7].

SLC is composed of a set of states $\mathbb{S}$ and their connecting edges $\mathbb{E}$. It adds more states and edges to the graph by computing the successor function $\textsc{Succ}(s)$ for $s \in \mathbb{S}$ that is already in the graph. In $\textsc{Succ}(s)$, the original State Lattice construct uses pre-computed

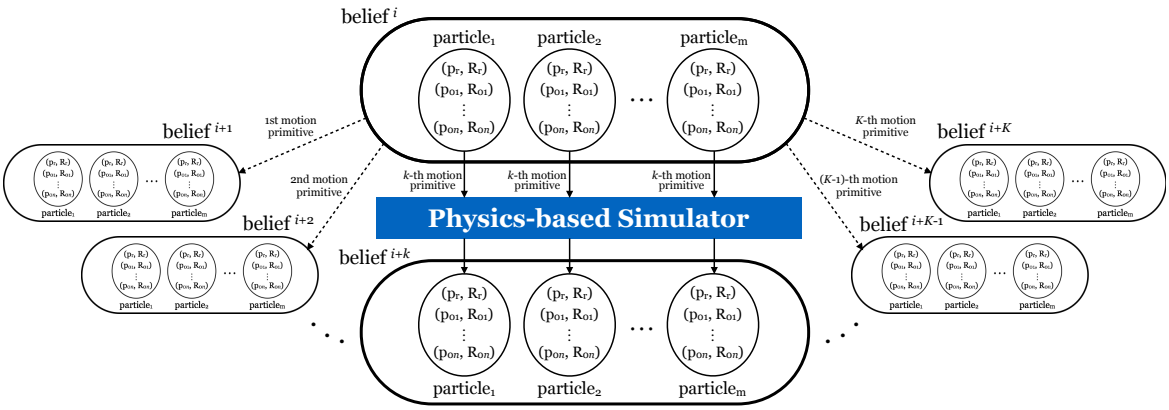


Figure 4.3: Belief space representation in a graph and its expansion scheme. belief ^{i} denotes a belief state $s^i = \mathbb{P}^i$, and particle$_j$ denotes a particle sub-state P_j. Each particle in a predecessor belief state is simulated for each motion primitive to get a successor belief state.

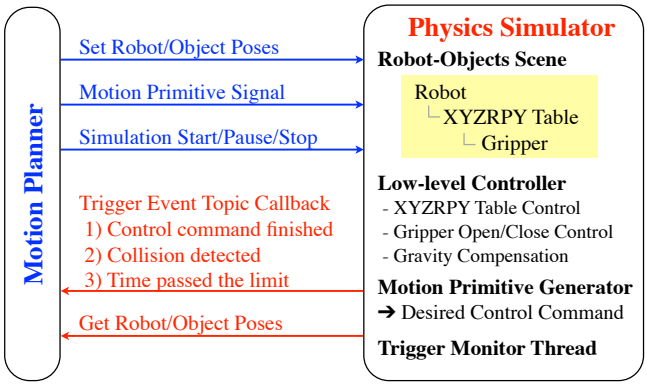


Figure 4.4: Integration of a motion planner and a physics-based simulator. The simulator has all the information about the robot and objects, including geometric shapes, inertial properties, and controller characteristics. Simulation of a motion primitive is terminated if one of the trigger events is detected.

metric motion primitives only, but SLC allows us to use controller-based motion primitives as well. This means that motion primitives can be simulated in a realistic controller and trigger setting to generate more plausible successor states. (A trigger is an event, such as detecting a marker or reaching the target position, that is used to terminate the simulation and return the result.) Based on this scheme, a more physically reasonable graph can be constructed for the given robot and the environment.

4.3.1 Physically Feasible Graph from Physics-based Simulation

In order to get physically reasonable $\textsc{Succ}(s)$ for a predecessor belief state, we exploit a physics-based simulator. More specifically, we use V-REP (Virtual Robot Experimentation Platform) simulator that supports Vortex Dynamics Engine [69], and integrated it with the motion planner as shown in Fig. 4.4.

As shown in Fig. 4.3, every particle sub-state in the predecessor belief state is simulated for each motion primitive, and the results constitute the successor belief states. Note that all the motion primitives being used are controller-based ones in the context of SLC, which means that a motion primitive is terminated by a trigger event in the simulation. Another note is that each motion primitive is to assign incremental displacement to the gripper, not the absolute pose in the global frame.

A motion primitive is triggered when 1) the control command is finished, 2) any change of the collision state is detected, or 3) the simulation time exceeded the limit. The first trigger is usually detected when the robot gripper remains in a free space without any collision until reaching the target pose. The second trigger is detected when the collision (or contact) states between any objects is switched from `NonContact` to `Contact`, or `Contact` to `NonContact`. The third trigger is for exception handling for the case that the simulator gets stuck.

4.4 MHA* for Deterministic Belief MDP

4.4.1 Algorithm

For the motion planning in the above-mentioned foliated belief space, Multi-Heuristic A* (MHA*) search algorithm [5, 70] is used. Refer to the pseudo code of the algorithm in Alg. 2.

MHA* has one consistent heuristic, which is called an anchor heuristic, and N arbitrary inadmissible heuristics of any necessity. This algorithm cycles through each inadmissi-

ble heuristics in a round-robin fashion, and determines whether to use the inadmissible heuristic or the anchor heuristic based on the condition in line 37 in Alg. 2. In such a way, it can control the suboptimality of the solution by a factor of $w_1 * w_2$ where w_1 and w_2 are the weights for weighted A* search (in line 22) and anchor suboptimality (in line 37).

To adopt MHA* for our problem, two major revisions are made: particle generation/transition and attractor-based heurisitics. As a belief space search problem, we need to generate a set of particles and obtain reasonable transition of them, which are explained in section 4.3. For the foliated belief space, we need to find good (inadmissible) heuristics that can help the search to go through the narrow passages fast, which can be in the form of (4.3) in section 4.4.2.

The heuristic function in (4.3) needs two input parameters, s_{cont} and s_{goal}, and they are being searched in AttractorSearch(s_{seed}) as shown in line 8 in Alg. 2. It is a quite simple operation that checks the amount of uncertainty reduction after applying motion primitives. It can apply a single long motion primitive or a sequence of them, and it can start from s_{start} or s_{goal}. As of now it is a naive process, but can possibly be developed as a sophisticated one.

4.4.2 Foliated Heuristic Function

As discussed in the previous section, the search configuration space is foliated, which means that transition between states with different uncertainty measures is mostly blocked by (virtual) obstacles except a few narrow passages, such as the walls in the box-on-table example.

In order to tackle this narrow-passage problem, we introduce a foliated inadmissible heuristic function as follows:

$$\begin{aligned} h(s) &= h\left(d_c(s), d_g(s), u(s)\right) \\ &= \begin{cases} w_d d_c(s) + w_u u(s) & (u(s) > u_{tol}) \\ w_d d_g(s) + w_u u(s) & (u(s) \leq u_{tol}) \end{cases} \end{aligned} \tag{4.3}$$

Algorithm 2 Multi-Heuristic A* Search in a Foliated Belief Space

input: The start state s_{start} and the goal state s_{goal}, suboptimality bound factor w_1, w_2 (both ≥ 1), and one consistent heuristic h_0.
output: A path from s_{start} to s_{goal} whose cost is within $w_1 * w_2 * g^*(s_{goal})$.

procedure CrossPolytopeParticles(μ, Σ)
  create a nominal particle P_0 from μ
  $\mathbb{P} \leftarrow \emptyset$
  for $d \in$ SearchSpaceCoordinates **do**
    $\mathbb{P} \leftarrow \mathbb{P} \cup \{P_0 + c_p * \Sigma_{(d,d)}\hat{\mathbf{e}}_d\}$ ▷ $\{\hat{\mathbf{e}}_d\}$: orthonormal basis of SearchSpace
    $\mathbb{P} \leftarrow \mathbb{P} \cup \{P_0 - c_p * \Sigma_{(d,d)}\hat{\mathbf{e}}_d\}$
  return $\mathbb{P}$
procedure AttractorSearch(s_{seed})
  $\mathbb{S}_{attractor} \leftarrow \emptyset$
  for $m_k \in$ LongMotionPrimitives **do**
    $s' \leftarrow$ Succ(s_{seed}, m_k)
    if s'.Uncert() $< c_u * s_{seed}$.Uncert() **then**
      $\mathbb{S}_{attractor} \leftarrow \mathbb{S}_{attractor} \cup \{s'\}$
  return $\mathbb{S}_{attractor}$
procedure NewInadmissHeuristic($s_{attractor}$, s_{goal})
  create a new instance of a heuristic class, h'
  h'.attractor $\leftarrow s_{attractor}$
  h'.goal $\leftarrow s_{goal}$
  return h'
procedure Key(s, v)
  $h_v \leftarrow$ H.Get(v)
  return $g(s) + w_1 * h_v(s)$
procedure Main()
  $s_{start} \leftarrow$ CrossPolytopeParticles(μ_{start}, Σ_{start})
  H $\leftarrow \emptyset$, $N \leftarrow 0$
  H.Add(h_0)
  for $s_{attractor}$ in AttractorSearch(s_{start}) **do**
    H.Add(NewInadmissHeuristic($s_{attractor}$, s_{goal}))
    $N \leftarrow N + 1$
  $g(s_{start}) \leftarrow 0$, $g(s_{goal}) \leftarrow \infty$
  for $v = 0, 1, ..., N$ **do**
    OPEN$_v \leftarrow \emptyset$
    insert s_{start} in OPEN$_v$ with Key(s_{start}, v)
  CLOSED$_{anchor} \leftarrow \emptyset$, CLOSED$_{inad} \leftarrow \emptyset$
  while OPEN$_0$.Minkey() $< \infty$ **do**
    for $v = 1, 2, ..., N$ **do**
      if OPEN$_v$.Minkey() $\leq w_2$ * OPEN$_0$.Minkey() **then**
        if $g(s_{goal}) \leq$ OPEN$_v$.Minkey() **then**
          if $g(s_{goal}) < \infty$ **then**
            terminate and return a solution path
        else
          $s \leftarrow$ OPEN$_v$.Top()
          ExpandState(s)
          insert s in CLOSED$_{inad}$
      else
        if $g(s_{goal}) \leq$ OPEN$_0$.Minkey() **then**
          if $g(s_{goal}) \leq \infty$ **then**
            terminate and return a solution path
        else
          $s \leftarrow$ OPEN$_0$.Top()
          ExpandState(s)
          insert s in CLOSED$_{anchor}$

where w_d and w_u are weights for distance and uncertainty, respectively. d_c and d_g are Euclidean distances to the contact attractor state and the goal attractor state that are given parameters to the heuristic function. u is the uncertainty measure defined in (4.2), and u_{tol} is the goal tolerance for u. d_c and d_g are defined as follows:

$$\begin{aligned} d_c(s) &= d(s, s_{cont}) \\ &= \frac{1}{m}\sum_{j=1}^{m}\left(w_p\sqrt{(x_s - x_c)^T(x_s - x_c)} + w_o\Delta\theta(s, s_{cont})\right) \end{aligned} \tag{4.4}$$

$$\begin{aligned} d_g(s) &= d(s, s_{goal}) \\ &= \frac{1}{m}\sum_{j=1}^{m}\left(w_p\sqrt{(x_s - x_g)^T(x_s - x_g)} + w_o\Delta\theta(s, s_{goal})\right) \end{aligned} \tag{4.5}$$

where s_{cont} is the contact attractor state, and s_{goal} is the goal state. x is the position vector of a belief state defined in (4.1), and subscripts s, c and g stand for the current, contact attractor, and the goal attractor states, respectively. $\Delta\theta(s, x_{goal})$ is the rotation angle from the current state s to the goal state s_{goal}, which can be computed in angle-axis representation. w_p and w_o are weights for position and orientation, respectively.

As a high-level explanation, this foliated heuristic function is to lead the graph to expand toward the contact attractor state in high uncertainty region and toward the goal attractor state in low uncertainty region. For example, if all the states in OPEN of this heuristic are on the same high uncertainty level and have the same g-value, then the state nearest to the contact attractor will have the minimum h-value and f-value, and it will be selected as the predecessor for expansion.

Also note that w_d and w_u on the right hand side of (4.3) should be chosen carefully, so that the second term for the uncertainty is signficantly larger than the first term for the distance. It is because the transition in uncertainty coordinate is more difficult due to the foliation of the search space.

We define a consistent *anchor* heuristic function (to adopt MHA* framework which will

be introduced in section 4.4.1) and an edge cost function as follows:

$$h_0(s^i) = h_0(s^i, s_{goal}) = w_d d(s^i, s_{goal}) + w_u u(s^i) \tag{4.6}$$

$$g(s^{i-1}, s^i) = w_d d(s^{i-1}, s^i) + w_u u(s^{i-1}) \tag{4.7}$$

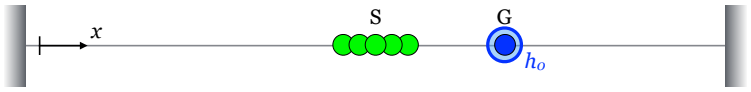
where superscript i is the index to represent predecessor-successor relationship in the graph. $d(s^{i-1}, s^i)$ means the Euclidean distance between s^{i-1} and s. Then, we can define $f_0(s^i)$ for the anchor heuristic as follows:

$$\begin{aligned} f_0(s^i) =& \sum_{t=1}^{i} g(s^{t-1}, s^t) + h_0(s^i, s_{goal}) \\ =& \sum_{t=1}^{i} \left(w_d d(s^{t-1}, s^t) + w_u u(s^{t-1}) \right) \\ & + w_d d(s^i, s_{goal}) + w_u u(s^i) \\ =& w_d \left(\sum_{t=1}^{i} d(s^{t-1}, s^t) + d(s^i, s_{goal}) \right) \\ & + w_u \left(\sum_{t=1}^{i} u(s^{t-1}) + u(s^i) \right) \end{aligned} \tag{4.8}$$

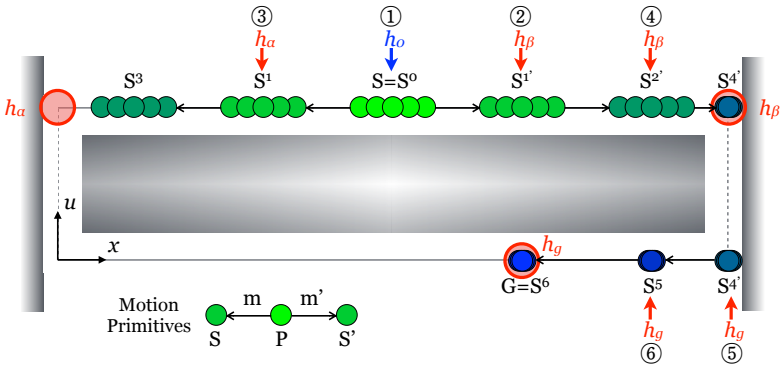
where s^0 is equivalent to s_{start}. Note that the term for uncertainty u in (4.7) is not the difference between predecessor and the successor states, but the remaining uncertainty of the parent state. This is because, in the former case, the second term in (4.8) will be the same for all paths that connect s_{start} and s_{goal}. On the other hand, the latter case can effectively penalize a path that remains in high uncertainty region for a long time.

4.4.3 Toy Example

Let us take a look at a 1-dimensional toy example to see how MHA* works in a foliated belief space. As shown in Fig. 4.5(a), the initial belief state at S has high uncertainty in x-position. From the belief state vector representation, we can construct a 2-dimensional configuration space with additional u-coordinate as shown in Fig. 4.5(b). There are two inadmissible heuristics, h_α and h_β, that have their contact attractors on the left and right



(a) A given problem: move balls at S to G.
Consistent anchor heuristic (h_0).



(b) Inadmissible contact heuristics (h_α and h_β) and goal heuristic (h_g).

Figure 4.5: Illustration of a 1-dimensional toy example.

walls, respectively, and the goal attractors at the goal state position.

As presented in Table 4.1, MHA* algorithm starts to cycle each inadmissible heuristic for graph expansion. For the first iteration, h_α takes the turn but couldn't satisfy the condition of line 37 in Alg. 2. So, OPEN_0 of h_0 is used to select the state for expansion, and then S^0 is expanded. For the second run, h_β takes the turn and satisfied the condition, and $\text{S}^{1'}$ is expanded. In the fourth run, $\text{S}^{2'}$ is expanded by applying motion primitives, m and m′, and the successor $\text{S}^{4'}$ gets contact with the wall. Its particles are gathered at one place, so its uncertainty measure reduces to near zero. For the states with low uncertainty, the goal attractor is being used, and after two more expansions, the algorithm

Table 4.1: MHA* Search Process for a Toy Example

Index	Turn for Round-Robin	Satisfied Line 37?	Selector Heuristic	Selected Parent	Child States
1	α	No	h_0	$S^0 = S$	S^1, $S^{1'}$
2	β	Yes	h_β	$S^{1'}$	$S^{2'}$
3	α	Yes	h_α	S^1	S^3
4	β	Yes	h_β	$S^{2'}$	$S^{4'}$
5	α	Yes	h_g	$S^{4'}$	S^5
6	β	Yes	h_g	S^5	$S^6 = G$

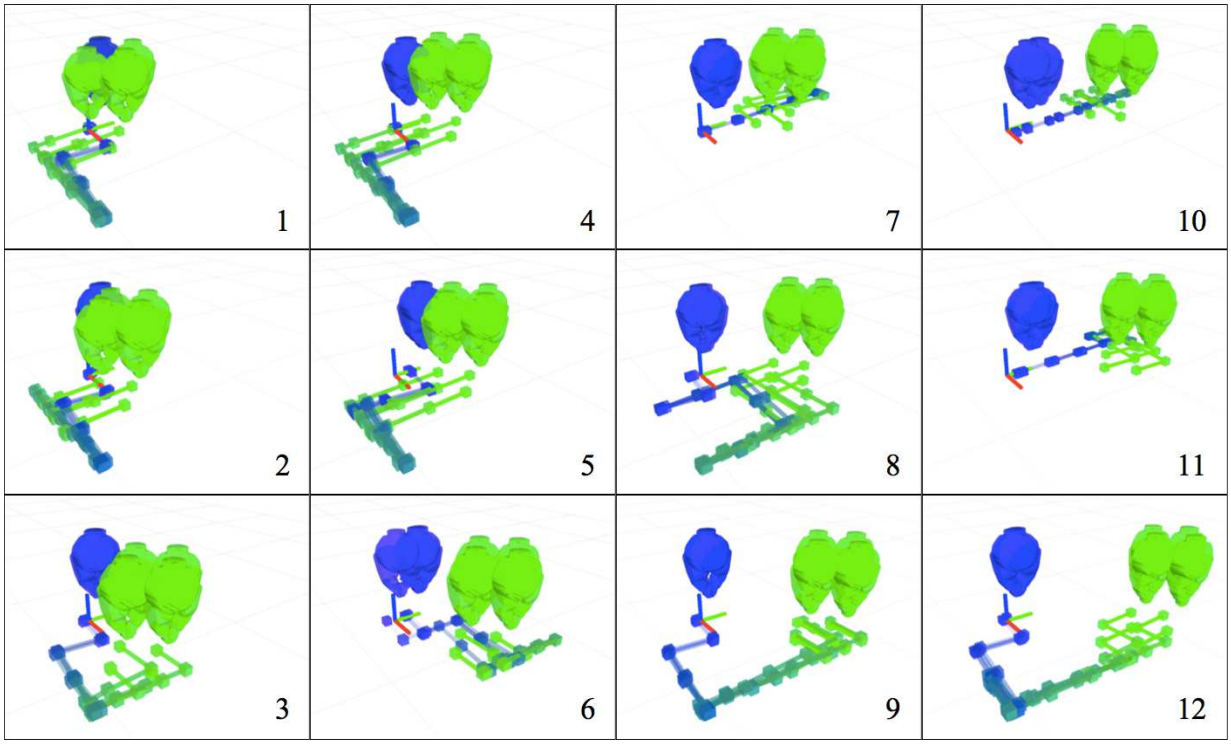


Figure 4.6: Planning results for 2D box-on-table tasks with 12 different start poses. The green and blue markers are the start and final poses of robot gripper particles, respectively, and an RGB-colored frame marker is the goal pose for the object held by the robot.

successfully finds a path to the goal with reduced uncertainty.

4.5 Experimental Results

4.5.1 Parts Assembly

Motion Planning

Box-on-Table (2D):

This task is to place a box held by the robot gripper to a specified pose on the table (Fig. 4.2). For analysis purposes, 12 different start poses are used for the motion planning, and the results are shown in Fig. 6 and Table 2. It can be seen that the planned motion utilizes contact with the walls close to the start and goal poses to reduce uncer-

Table 4.2: Planning Results for 2D Box-on-Table Tasks
(The step size for translational motion primitives is 0.10 m, and the initial offset from the nominal start pose of each particle is 0.06 m.)

Task ID	Time [s]	Cost	Node # in Path / Expansion #	Pos. Error Mean [m]	Pos. Error Std Dev [m]
1	367	42032	9/10	0.0165	0.0085
2	482	46544	12/12	0.0143	0.0072
3	364	35812	8/10	0.0163	0.0067
4	484	45172	10/11	0.0158	0.0055
5	502	47644	13/13	0.0439	0.0066
6	396	47847	11/11	0.0273	0.0290
7	488	42900	13/14	0.0146	0.0059
8	685	48839	14/16	0.0296	0.0059
9	638	45951	12/16	0.0171	0.0063
10	463	41891	9/13	0.0382	0.0203
11	711	42198	16/19	0.0370	0.0099
12	637	47356	13/16	0.0162	0.0089
Mean	518.1	44515.5	11.6/13.4	0.0239	0.0101
Std Dev	121.2	3705.5	2.3/2.8	0.0108	0.0072

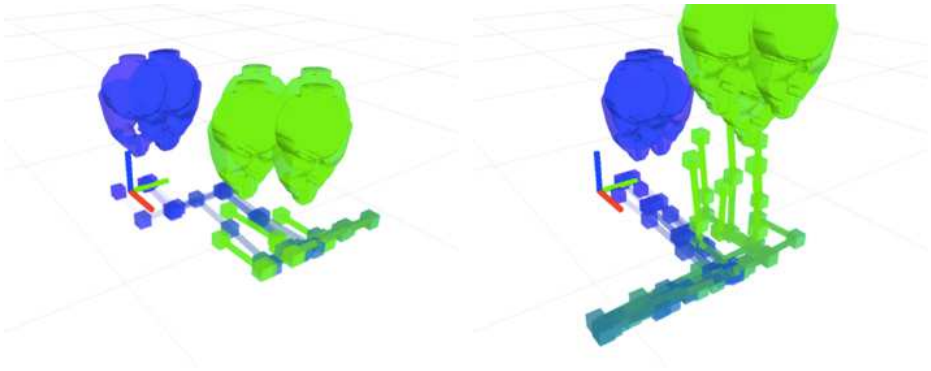


Figure 4.7: Comparison of planning results of 2D (left) and 3D (right) box-on-table tasks.

tainty.

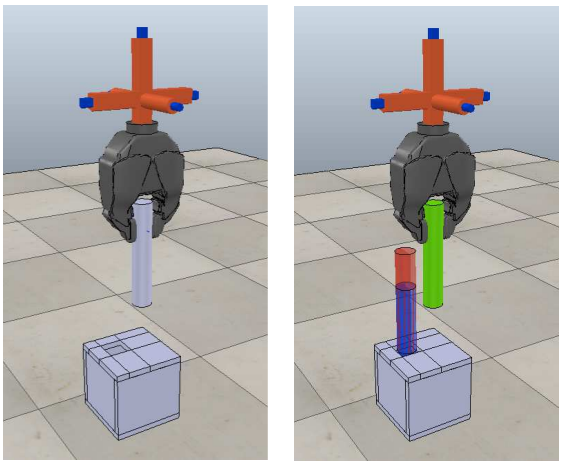
The contact attractors were at the walls, not at the corners (as in Fig. 4.2(b)), but many motion plans got to the corner. It can be interpreted that contact attractors don't introduce significant artifacts to the solution path.

Box-on-Table (3D):

Planning results for box-on-table task in 3D space are presented in Fig. 7 and Table 3. It is compared to the 2D planning case that only differs in the initial z-position. As can be seen in planning time and number of expansions, the 3D planning case obviously suffers

Table 4.3: Comparison of Planning Results of 2D and 3D Box-on-Table Tasks
(The motion step sizes and initial particle offsets are the same, but the numbers of particles are 4 for 2D case and 6 for 3D case, respectively, and the nominal pose in 3D case is 0.13 m higher than the goal pose.)

Task ID	Time [s]	Cost	Node # in Path / Expansion #	Pos. Error Mean [m]	Pos. Error Std Dev [m]
6 (2D)	396	47847	11/11	0.0273	0.0290
6′ (3D)	1322	42958	11/17	0.0611	0.0009



(a) Models of a peg and a hole.

(b) A given problem.

Figure 4.8: Illustration of 2D peg-in-hole task. (b) The green peg is the start pose, and the blue peg is the goal pose. The red peg is a goal attractor for an inadmissible heuristic.

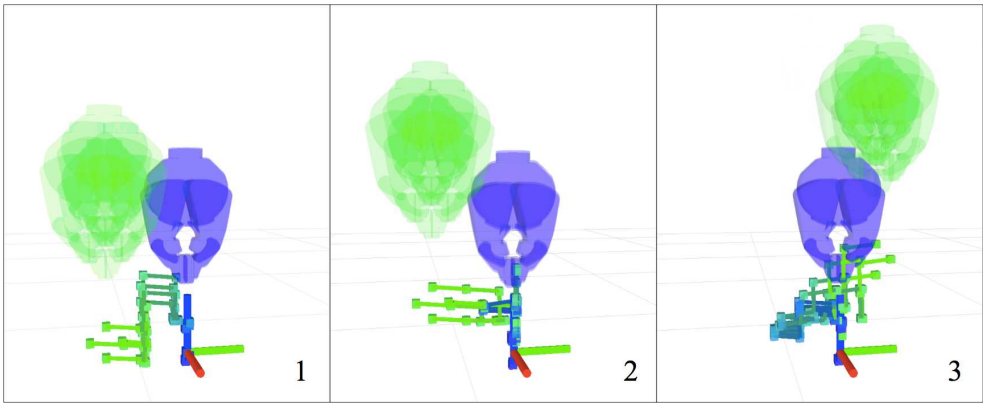


Figure 4.9: Planning results for 2D peg-in-hole tasks with 3 different start poses.

from the curse of dimensionality. This is somewhat expected because the number of simulations per expansion equals to the number of particles times the number of motion primitives, which should be (4×4) and (6×6) for 2D and 3D, respectively.

Peg-in-Hole (2D):

Table 4.4: Planning Results for 2D Peg-in-Hole Tasks
(The step size for translational motion primitives is 0.10 m, and the initial offset from the nominal start pose of each particle is 0.04 m.)

Task ID	Time [s]	Cost	Node # in Path / Expansion #	Pos. Error Mean [m]	Pos. Error Std Dev [m]
1	706	12164	16/21	0.0032	0.0009
2	604	12474	16/19	0.0029	0.0018
3	830	15311	22/25	0.0035	0.0011
Mean	713.1	13316.3	18.0/21.6	0.0032	0.0013
Std Dev	113.2	1734.4	3.1/3.5	0.0003	0.0005

This task is to insert the peg into the hole. One interesting point of planning task is that, by backward attractor search from the goal, an attractor state was found at the entrance of the hole and set as a goal attractor, not a contact attractor in a high uncertainty region. This is reasonable because the state at the entrance should have low uncertainty as the goal state. The results for three different cases are shown in Fig. 9 and Table 4. The first and third cases reduce uncertainty by contact with the top and the left side of the box, but the second case does that by contact with the top and the inner side of the hole.

Motion Execution

Box-on-Table (2D):

Physical robot experiments were conducted for a 2D box-on-table task (Fig. 10). 10 different runs used the same motion plan searched for the given (nominal) start and goal poses, but the actual start pose for each run was perturbed by Gaussian noise with 0.05 m standard deviation. As shown in Fig. 11, the standard deviation of the final pose reduced to tenth of the initial one. However, there were remaining errors from the goal due to relatively large motion primitive displacements.

Peg-in-Hole (2D):

Robot experiment results for peg-in-hole tasks are presented in Fig. 4.12 and Fig. 4.13. As in the box-on-table case, 10 different runs used the same planned motion computed by the planner for the given start and (nominal) goal poses. The right gripper of the robot held the box with a hole (see Fig. 4.12), but it was just used to provide a perturbed goal

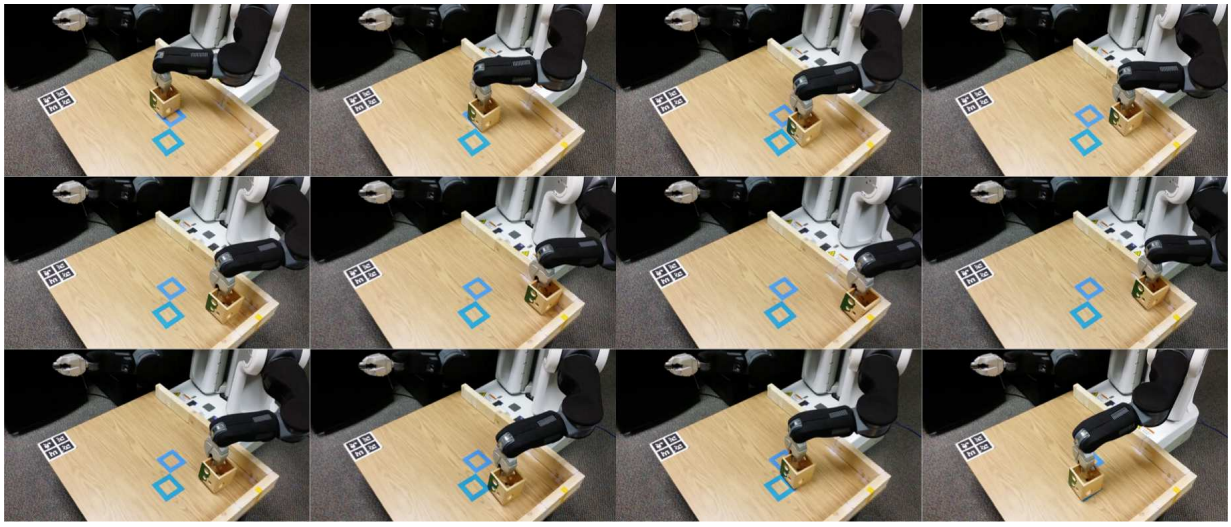


Figure 4.10: Snapshots of a robot experiment for a 2D box-on-table task. The motion was planned for a nominal start position (upper square in the picture) and a goal position (lower square in the picture). The actual start position deviated from the nominal start position by 6 cm, but the goal could be accomplished.

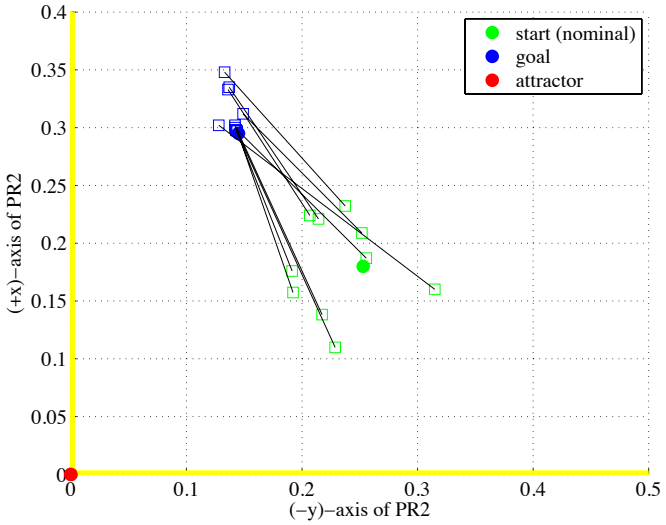


Figure 4.11: Plot of robot experiments for 2D box-on-table tasks. (This plot is rotated by 90 degrees, so that it can be depicted as the PR2 views the table.) The green filled circle is the nominal start position, and the blue filled circle is the goal position. Green and blue squares are the initial and the final positions of 10 individual test runs, and the pairs are connected by black solid lines. The red filled circle is the attractor used for planning, and the yellows bars represent the walls of the table. The standard deviation of start position was set to 0.05 m, and the perturbed start position of each test is randomly drawn from the corresponding Gaussian distribution. The step size of translational motion primitives is 0.10 m. The mean and standard deviation of position errors to the goal are (0.0175, 0.0053) and (0.0189, -0.0061) in meter, respectively.

Figure 4.12: Snapshots of a robot experiment for a 2D peg-in-hole task.

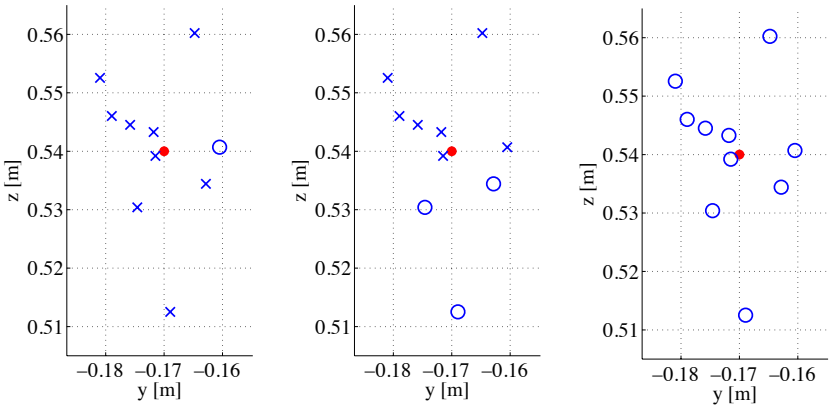
pose for each run. The motion plan from our proposed planner was able to successfully insert the peg into the perturbed hole in all 10 runs.

In order to demonstrate the significance of the proposed approach, two baselines are compared for these tasks. One is a *single-particle planner* which considers only one particle without the notion of uncertainty. The other is a *contact-greedy planner*, an extension of the single-particle planner, which tries to get to the nearest contact point as soon as possible, exploiting the uncertainty reduction effect of contact. In both planners, physics-based simulation was necessary to construct physically feasible graphs.

As shown in Fig. 4.13, the motion plan from the single-particle planner was very fragile under uncertainty so that it succeeded only in one case which has the almost same height with the nominal goal pose. The contact-greedy planner was more robust than the single-particle planner, but as it also considers only one particle, its motion plan failed in more than the half of the runs.

4.6 Summary and Discussion

We proposed a motion planning framework for parts assembly under uncertainty. This problem is formalized as a graph search problem in a foliated belief space where uncer-



(a) Baseline (single-particle)

(b) Baseline (contact-greedy)

(c) Our planner

Figure 4.13: Plot of robot experiments for 2D peg-in-hole tasks. The red filled circle is the nominal goal position (the true mean of the Gaussian distribution of the goal position) that is used for planning, and the blue circles or crosses are the actual goal poisitions for 10 test runs that are drawn from Gaussian distribution with a standard deviation of 0.01 m. The step size of translational motion primitives was 0.04 m. Note that in these experiments the peg held by the left gripper is set to be at the same start position for all the test runs, but the box with a hole held by the right gripper which serves as an environment under uncertainty (without any sensor feedback) is set to be at a perturbed goal position. A circle marker represents that a peg-in-hole task for the corresponding goal position was successfully completed, i.e., the peg was inserted into the hole, by the planned motion of a specific planner, while a cross marker represents that the corresponding peg-in-hole task was not successful.

tainty reduction is only possible in narrow passages of contact. Physics-based simulation is used to construct a physically reasonable graph, and foliated heuristic functions with contact and goal attractors are adopted in Multi-Heuristic A* search framework to accelerate the search. The planning and experimental results for box-on-table and peg-in-hole tasks demonstrated the effectiveness of our approach.

As a future work, it would be interesting to study how sampling-based methods can help finding good attractor states for inadmissible heuristics and be combined with combinatorial search algorithms. Based on the fact that simulation-based reasoning is highly expensive, we would like to incorporate Experience Graph (E-Graph) into our framework, so that we can reuse the previous planning results for other similar tasks [71].

Chapter 5

Planning under Motion and Sensing Uncertainty with Multiple Heuristics

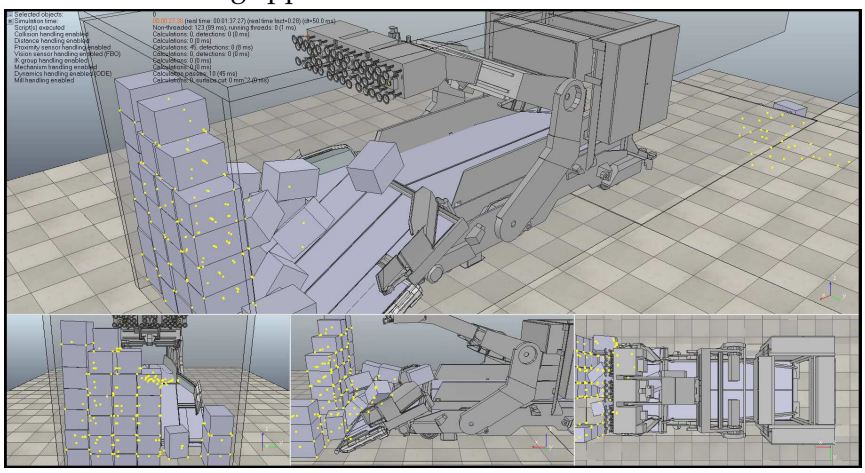
5.1 Introduction

Motion planning is a well-studied problem in robotics with applications in diverse domains [72, 73]. 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. 7.1), 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) [1, 2]. However, solving a POMDP is often intractable due to *the curse of dimensionality* [1] and *the curse of history* [24] which correspond to an exponential complexity

(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 5.1: Truck unloader robot.

with the number of states and the planning horizon, respectively.

One approach to solve POMDPs is using heuristic-search such as RTDP-Bel [18, 38]. 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 guarantee 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 [43] 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.

5.2 Algorithmic Background

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

In this section we briefly review Real-Time Dynamic Programming in Belief Space (RTDP-Bel) [38] which is an extension of RTDP for MDP [40] to belief space planning.

RTDP-Bel, described in Alg. 3 performs a series of searches until it converges to the optimal policy (see Eq. 2.7). 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).

5.2.2 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

[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.

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

Algorithm 3 RTDP-BEL

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

repeat
  $b \leftarrow b_0$ ▷ start iteration with initial belief
  SAMPLE state s with probability $b(s)$
  repeat
    EVALUATE $q(b^a)$ for each action a using Eq. 2.5
      ▷ initialize $v(b^{ao})$ to $h(b^{ao})$ if not initialized
    SELECT BEST ACTION a ▷ a minimizes $q(b^a)$
    UPDATE $v(b)$ to $q(b^a)$
    SAMPLE next state s' ▷ uses motion model T
    SAMPLE observation o
      ▷ uses observation model Z
    COMPUTE b^{ao} and set $b \leftarrow b^{ao}$, $s \leftarrow s'$
  until b^{ao} is the goal belief
until CONVERGED()

functions. One algorithm that uses this idea is Multi-Heuristic A* (MHA*) [5], 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., [74, 75, 76]).

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 $g(s) + h_{\text{inad}}(s) \leqslant g(s) + \varepsilon \cdot h_{\text{anchor}}(s)$. 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.

5.3 Partially Observable Multi-Heuristic Dynamic Programming (POMHDP)

5.3.1 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. 3, 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. 3, 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 pseudo-code 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.

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., [76, 77])—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. 5.2.

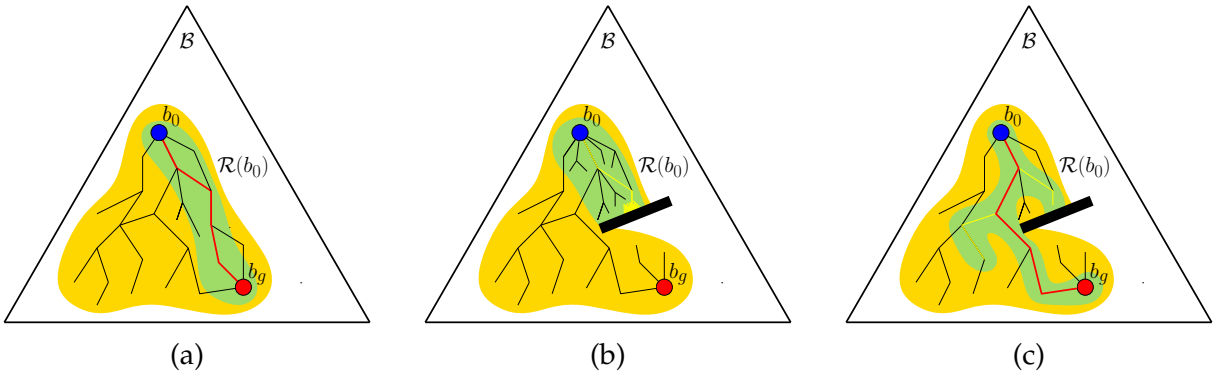


Figure 5.2: Illustration of forward search with a single (a, b) and multiple (c) heuristics. (a) Forward search with single heuristic reaches the goal, (b) Forward search with single heuristic gets stuck at local minima. (c) Forward search with multiple heuristics can switch from one heuristic to another 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.

5.3.2 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. 3, 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. 3, 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. 3 is now implemented, see Alg. 4.

Roughly speaking, Idea **I1** is analogous to the way weighted A* [78] uses an inflated heuristic and Idea **I2** follows the way MHA* ensures bounded sub-optimality (see Sec. 5.2.2).

5.3.3 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

Algorithm 4 SELECT BEST ACTION

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

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

Algorithm 5 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 $b^a = \textsc{Succ}(b, a)$
2: $\Delta v \leftarrow \eta \Delta v + (q_{\text{succ}} - v_{\text{pred}})$
3: **if** $\Delta v \geqslant 0$ **then**
4: **return** `true`
5: **return** `false`

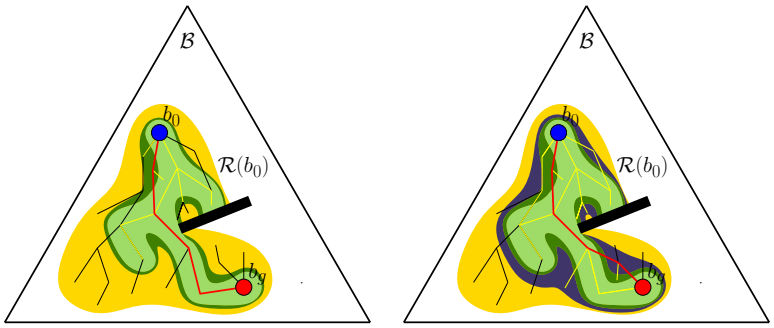
(ii) a principled way to choose which state do we want to sample from once we switch to a new (inadmissible) heuristic.

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., [76]). Here, we adopted the concept of momentum in stochastic gradient descent [79]. For pseudo code, see Alg. 5.

To choose which state to sample from once we switch to a new (inadmissible) heuristic, we maintain a priority queue, 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., $\textsc{Key}(b^a, i) = g(b^a) + q_i(b^a)$.

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 con-

[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* [5] for further details.



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

Figure 5.3: Illustration of the anytime behavior of POMHDP. After the first iteration of forward search (Fig. 5.2c), we gradually relax the search space bounds (Alg 6, Line 18) and run another forward search to improve the solution.

verges to optimal solution, we decrease the approximation factors, ε_1 and ε_2, between iterations. For visualization, see Fig. 5.3.

A high-level description of the algorithm is provided in Alg. 6. Lines where the algorithm differs from RTDP-Bel (Alg. 3) are marked in magenta.

5.3.4 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 [40]. RTDP-Bel, which approximates a POMDP problem as a belief MDP through belief discretization, similarly provides asymptotic convergence to a resolution-optimal policy [18]. 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 [43].

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 $v(b) =$

Algorithm 6 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

repeat
  $b \leftarrow b_0$ ▷ start iteration with initial belief
  $\forall_i, \text{OPEN}_i \leftarrow \emptyset$;
    ▷ start iteration with empty queues
  curr $\leftarrow 1$ ▷ index of inadmissible heuristic
  SAMPLE state s with probability $b(s)$
  repeat
    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 $\text{OPEN}_{\text{curr}}$
    if IS IN STAGNATION **then** ▷ Uses Alg. 5
      SWITCH HEURISTIC ▷ updates curr
      SELECT BEST ACTION a
        ▷ runs anchor test $\text{OPEN}_{\text{curr}}$'s and OPEN_0's head
    else
      SELECT BEST ACTION a ▷ Uses Alg. 4
    UPDATE $v_0(b)$ to $q_0(b^a)$ and , $v_{\text{curr}}(b)$ to $q_{\text{curr}}(b^a)$
    SAMPLE next state s' ▷ uses motion model T
    SAMPLE observation o
      ▷ uses observation model Z
    COMPUTE b^{ao} and set $b \leftarrow b^{ao}$, $s \leftarrow s'$
  until b^{ao} is the goal belief
  $\varepsilon_1 \leftarrow \varepsilon_1 e^{-\alpha}$; $\varepsilon_2 \leftarrow \varepsilon_2 e^{-\alpha}$
    ▷ decrease approx. factors
until CONVERGED()

$v_g(b) + \varepsilon_1 \cdot v_h(b)$, where $v_g(b)$ and $v_h(b)$ are initialized to zero and an admissible heuristic value $h(b)$, respectively [33, 80]. Eq. (2.4) 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 provide any theoretic guarantees by itself, but with the anchor test (Alg. 4, 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. 6, Line 18), POMHDP converges to the optimal solution.

5.4 Illustrative Example

Fig. 5.4 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, respec-

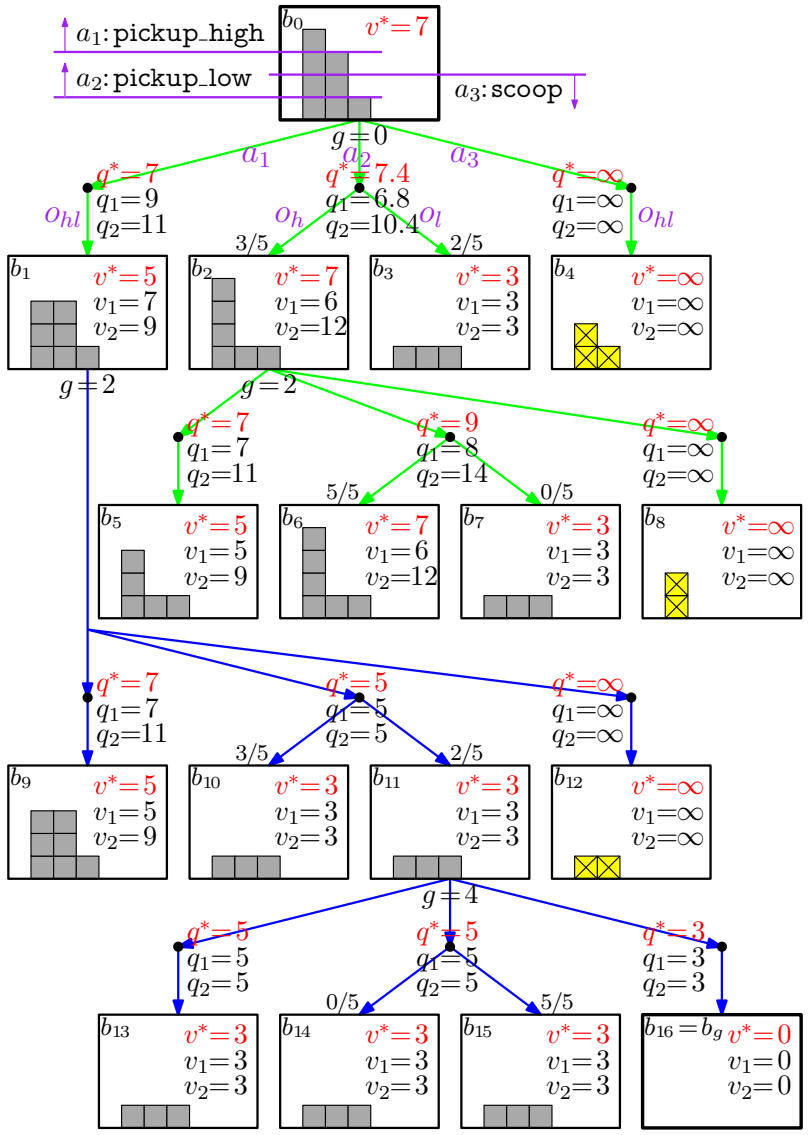


Figure 5.4: A toy example for truck unloading. POMHDP conducted a forward search using one admissible heuristic and two inadmissible heuristics in the following order: $b_0 \rightarrow b_2 \rightarrow$ (heuristic switch; rebranch) $\rightarrow b_1 \rightarrow b_{11} \rightarrow b_g$. Heuristic switch 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.)

tively. 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.

$$h_0(b) = \begin{cases} 3 & (\forall b \in \mathbb{B} \backslash \{b_g\}) \\ 0 & (b = b_g) \end{cases} \tag{5.1}$$

$$h_1(b) = \mathbb{E}[(\text{\# of all boxes left})] \tag{5.2}$$

$$h_2(b) = \mathbb{E}[3 \cdot \max(\text{\# of boxes in each column})] \tag{5.3}$$

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. 5.4, 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. 5.4 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).

5.5 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.

5.5.1 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 $s = (T_r,\, q_r^{1:16},\, \{T_b,\, l_b,\, m_b\}_{1:N_b}) \in \mathbb{S}$ is a tuple of the robot base pose, T_r, and joint configuration, $q_r^{1:16}$, 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. 5.1-5.3.

5.5.2 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. 7.1). 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 [18]. 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. 5.1, we also present the results of RTDP-Part with *one* of the inadmissible heuristics in Eq. 5.2-5.3, denoted by RTDP-Part(h_i).

Table 5.1: Average time for the first forward search iteration

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

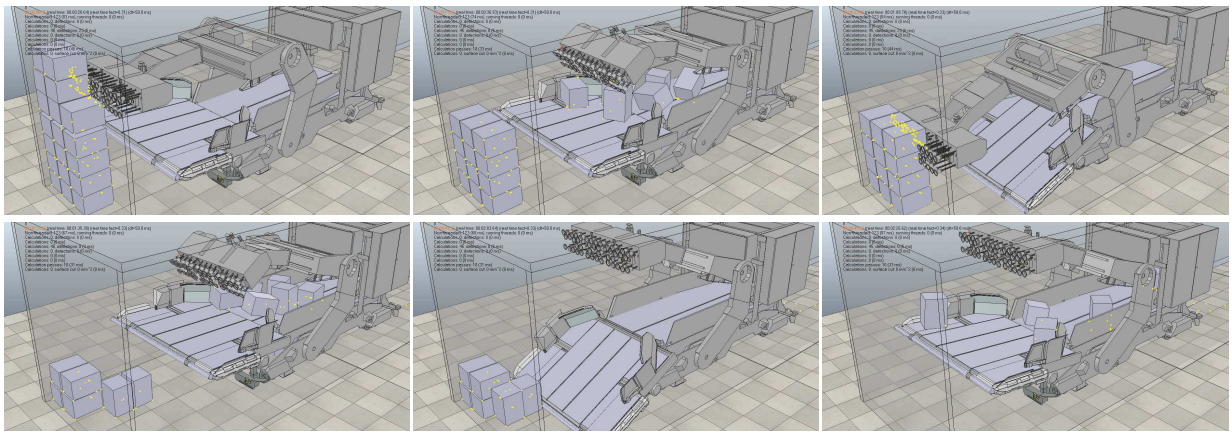


Figure 5.5: Snapshots of truck unloading task execution based on the solution policy of POMHDP.

5.5.3 Results

As shown in Table 5.1, POMCP took significantly longer time than the other algorithms, and thus, sufficient data for further 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. 5.6 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. 5.7, the number of boxes in different states after executing a solution policy is presented, where the anytime 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

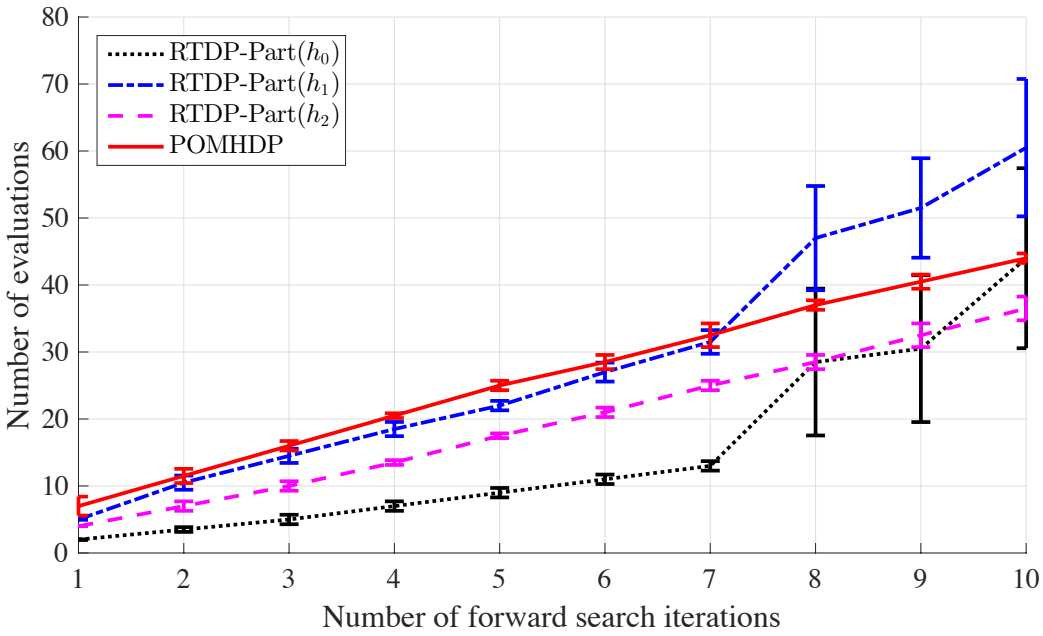


Figure 5.6: Number of evaluations over the forward search iterations by RTDP-Part with an individual heuristic, and POMHDP with multiple heuristics, respectively.

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.

5.6 Summary and Discussion

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.

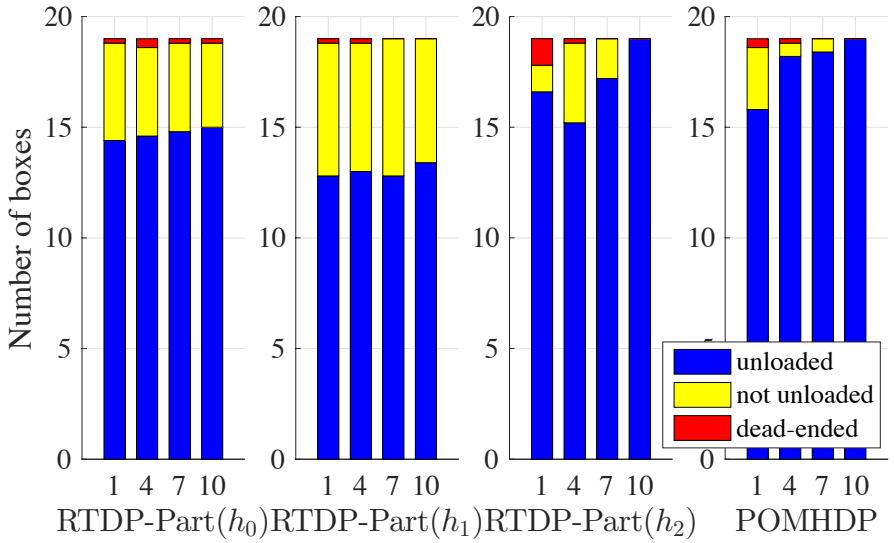


Figure 5.7: Number of boxes in different states (unloaded, not unloaded, or dead-ended) after executing a solution policy obtained from each number (on the x-axis) of the forward search iterations.

An interesting extension of this work is the online planning version that interleaves re-planning 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.

5.A 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. 7 and detailed in Alg. 8-14. 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. 10). Then $q_i(b^a)$ for the current belief b and possible action a pair is updated as in (2.5) (line 26 in Alg. 10). The minimal $q_i(b^a)$ over $\forall a \in \mathbb{A}$ is used to set $v_i(b)$ in the backup step (Alg. 11).

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. 10). OPEN$_i$ here denotes a priority queue for each heuristic sorted by key values defined as in line 2 in Alg. 10. 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. 8). In such a case, it switches the heuristic reference index i_h to another in a round-robin fashion (Alg. 9), and then invokes OPEN$_{i_h}$ to choose the belief with the minimal key value (Alg. 12). This is referred to as *rebranching* since the forward search stops growing the current belief

Algorithm 7 POMHDP::MAIN

procedure MAIN()
  $g(b_0) = 0;\ g(b_g) = \infty$ ▷ cost-from-start
  for $i = 0, ..., n_h$ **do**
    $v_i(b_0) = h_i(b_0);\ v_i(b_g) = 0$ ▷ cost-to-goal
  repeat
    for $i = 0, ..., n_h$ **do**
      $\text{OPEN}_i \leftarrow \emptyset$
    $\text{CLOSED}_{\text{anchor}} \leftarrow \emptyset;\ \text{CLOSED}_{\text{inad}} \leftarrow \emptyset$
    $b \leftarrow b_0;\ i_h \leftarrow 1;\ \Delta v \leftarrow \Delta v_0$
    while $\neg$REACHEDGOAL(b, b_g) **do**
      $v_{\text{pred}} \leftarrow v_{i_h}(b)$
      EVALUATE(b)
      BACKUP(b)
      if $\text{OPEN}_0 = \emptyset$ **then**
        break
      if CHECKSTAGNATION(v_{pred}, i_h) or
       GETSUCCS$(b) = \emptyset$ **then**
        $i_h \leftarrow$ SWITCHHEURISTIC(i_h)
        $b^{a^*} \leftarrow$ NEXTBELACTFROMOPEN(i_h)
        $\Delta v \leftarrow \Delta v_0$
      else
        $b^{a^*} \leftarrow$ NEXTBELACTFROMSUCCS(b,i_h)
      $b \leftarrow$ SAMPLEBELIEFFROMBELACT(b^{a^*})
    $\varepsilon_1 \leftarrow \varepsilon_1 e^{-\alpha};\ \varepsilon_2 \leftarrow \varepsilon_2 e^{-\alpha}$
  until CONVERGED() or TIMEOUT()

Algorithm 8 POMHDP::CHECKSTAGNATION

procedure CHECKSTAGNATION(v_{pred}, b, i_h)
  $q_{\text{succ}} = \min_{a \in \mathbb{A}} q_{i_h}(b^a)$ where $b^a =$ SUCC(b, a)
  $\Delta v \leftarrow \eta \Delta v + (q_{\text{succ}} - v_{\text{pred}})$
  if $\Delta v \geqslant 0$ **then**
    return `true`
  else
    return `false`

Algorithm 9 POMHDP::SWITCHHEURISTIC

procedure SWITCHHEURISTIC(i_h)
  $i_h \leftarrow \text{mod}(i_h, n_h) + 1$ ▷ round-robin
  return i_h

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. 13), 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. 12 and line 5 in Alg. 13, 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. 12 and 13 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. 13 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. 12 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. 14).

Algorithm 10 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) $\leqslant \varepsilon_2 \cdot$ KEY($b^a, 0$) **then**
            Insert/update b^a in OPEN$_i$ with KEY(b^a, i)

Algorithm 11 POMHDP::BACKUP

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

Algorithm 12 POMHDP::NEXTBELACTFROMOPEN

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

Algorithm 13 POMHDP::NEXTBELACTFROMSUCCS

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

Algorithm 14 POMHDP::SAMPLEBELIEFFROMBELACT

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

Chapter 6

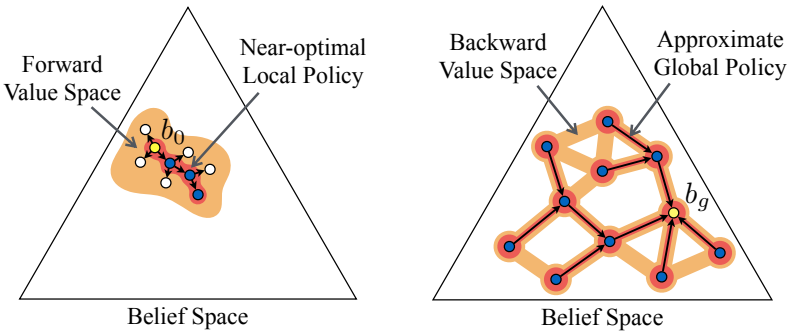
Extension: Online-Offline Combination for Scalability

6.1 Motivation

Consider a scenario where an autonomous mobile robot (e.g., a rover or flying drone) needs to navigate through an obstacle-laden environment under both motion and sensing uncertainty. In spite of these uncertainties, the robot needs to guarantee safety and reduce the risk of collision with obstacles at all times. This, in particular, is a challenge for safety-critical systems and fast moving robots as the vehicle traverses long distances in a short time horizon. Hence, ensuring system's safety requires risk prediction over long horizons.

The above-mentioned problem is an instance of general problem of decision-making under uncertainty in the presence of risk and constraints, which has applications in different mobile robot navigation scenarios. In particular, in this work, we focus on a challenging class of POMDPs, here referred to as RAL-POMDPs (Risk-Averse, Long-range POMDPs). A RAL-POMDP reflects some of challenges encountered in physical robot navigation problems, and is characterized with the following features:

1. Long planning horizons (beyond 10^4 steps) without cost discounting, i.e., safety is



(a) Forward value space (b) Backward value space

Figure 6.1: Illustration of bi-directional value learning. For a given belief space, the first thread expands *forward value space* from the start as its belief tree grows. This provides a near-optimal local policy. The second thread constructs *backward value space* connected to the goal by solving belief MDP in a sampled subspace. This returns an approximate global policy which can be used to guide the forward search.

equally critical throughout the plan. In RAL-POMDP, the termination of the planning problem is dictated by reaching the goal (terminal) state rather than reaching a finite planning horizon.

2. RAL-POMDP is defined via high-fidelity continuous state, action, and observation models.

3. RAL-POMDP incorporates computationally expensive costs and constraints such as collision checking.

4. RAL-POMDP requires quick policy updates to cope with local changes in the risk regions during execution.

In recent years, value learning in partially observable settings has seen impressive advances in terms of the complexity and size of solved problems. There are two major classes of POMDP solvers (see Fig. 6.1). The first class is forward search methods [24, 30, 43, 81]. Methods in this class (offline and online variants) typically rely on forward simulations to search the reachable belief space from a given starting belief and learn the value function. Partially Observable Monte Carlo Planning (POMCP) [43], DESPOT [46], and ABT [82] are a few examples of recent methods in this class that can efficiently learn and update the policy while executing a plan using Monte Carlo simulation. The second class is approximate long-range solvers [9, 83]. These methods typically address continuous POMDPs but under Gaussian assumption. They typically rely

on sampling-based graph construction and feedback controllers to solve larger problems. Through offline planning, they can learn an approximate value function on the representative (sampled) graph.

The features of RAL-POMDP problems make them a challenging class of POMDPs for above-mentioned solvers. Forward search-based methods typically require cost discounting and a limited horizon (shorter than 100 steps) to be able to handle the planning problem. Also, they typically require at least one of the state, action, or observation space to be discrete. Continuous approximate long-range methods suffer from suboptimality since actions are generated based on a finite number of local controllers due to the underlying sparse sampling-based structure.

This work tackles the RAL-POMDP problem induced by fast-moving robot navigation in safety-critical scenarios. Several seconds of operation can translate to thousands of decision making steps in these systems. The main objective of this work is to provide probabilistic safety guarantees for the long-horizon decision making process (beyond thousands of steps). The second objective of this work is to generate solutions for the RAL-POMDP that are closer to the globally optimal solution compared to the state-of-the-art methods. In parallel to these objectives, we intend to satisfy other requirements of the RAL-POMDP such as incorporating high-fidelity continuous dynamics and sensor models into the planning.

In this work, we propose Bi-directional Value Learning (BVL) method, a POMDP solver that searches the belief space and learns the value function in a bi-directional. In the one thread (can be performed offline) we learn a risk-aware approximate value function backwards from the goal state toward the starting point. In the second thread (performed online), we expand a forward search tree from the start toward the goal. BVL significantly improves the performance (optimality) of the backward search methods by locally updating the policy through rapid online forward search during the actual execution. BVL also enhances the probabilistic guarantees on system's safety by performing computationally intensive processes, such as collision checking, over long planning horizons in the offline phase.

6.2 Risk-Averse, Long-Range POMDPs

In this work, we focus on a Risk-Averse, Long-Range POMDP (RAL-POMDP) as a special case of the above-mentioned POMDP problem. Formally, in a RAL-POMDP, $\mathbb{X}$, $\mathbb{U}$, and $\mathbb{Z}$ are continuous spaces, and f and h represent locally differentiable nonlinear mappings. There exists a goal termination set $B^{goal} \subset \mathbb{B}$ such that $J(b_g) = 0$ for $\forall b_g \in B^{goal}$. There also exists a failure termination set $F \subset \mathbb{B}$ which represents the risk region (e.g., obstacles in robot motion planning) such that $J(b_f) \to \infty$ for $\forall b_f \in F$. As the risk is critical throughout the plan, a RAL-POMDP does not allow cost discounting, i.e., $\gamma = 1$.

In our risk metric discussion, we follow definitions in [84, 85]. Accordingly, our risk metric falls in the category of risk for sequential decision making with deterministic policies, satisfying *time-consistency* (see [84, 85] for details). Specifically, we formalize the risk by compounding the failure probability, $p(F|b, \pi(b)) = \sum_{b_f \in F} p(b_f|b, \pi(b))$, of each action along the sequence. Accordingly, the risk metric of a policy given a belief b_0 is measured as follows.

$$\rho(b_0; \pi) = 1 - \mathbb{E}\left[\prod_{k=0}^{\infty}(1 - p(F|b_k, \pi(b_k))\right] \tag{6.1}$$

The second term on the right-hand side is the expected probability to reach the goal without hitting the risk region. Note that $\rho(b_g; \pi) = 0$ for $\forall b_g \in B^{goal}$ and $\rho(b_f; \pi) = 1$ for $\forall b_f \in F$ for $\forall \pi \in \Pi$. It can be rewritten in a recursive form:

$$\begin{aligned}\rho(b; \pi) &= 1 - \sum_{b' \in \mathbb{B}} p(b'|b, \pi(b))(1 - \rho(b'; \pi)) \\ &= \sum_{b' \in \mathbb{B}} p(b'|b, \pi(b))\rho(b'; \pi)\end{aligned} \tag{6.2}$$

Now we show that in RAL-POMDPs where $J(b_f) = J^F \to \infty$ for $\forall b_f \in F$, the optimal policy π^* in Eq. (2.7) also minimizes $\rho(b; \pi^*)$ in Eq. (6.2) for $\forall b \in \mathbb{B}$.

Lemma 6.1. *In RAL-POMDPs where $J^F \to \infty$ and $\gamma = 1$, the following is satisfied for $\forall b \in \mathbb{B}$.*

$$\rho(b;\pi) = \lim_{J^F \to \infty} \frac{J(b;\pi)}{J^F} \tag{6.3}$$

Proof. We prove this by *backward induction*.

Consider the terminal beliefs first. Trivially, from Eq. (2.3) and Eq. (6.1), $\rho(b_g;\pi) = \lim_{J^F\to\infty} \frac{J(b_g;\pi)}{J^F} = 0$ for $\forall b_g \in B^{goal}$, and $\rho(b_f;\pi) = \lim_{J^F\to\infty} \frac{J(b_f;\pi)}{J^F} = 1$ for $\forall b_f \in F$. Thus, Eq. (6.3) is satisfied for terminal beliefs.

Next, consider a belief such that its every successor is either $b_g \in B^{goal}$ or $b_f \in F$, i.e., $\sum_{b_g \in B^{goal}} p(b_g|b,\pi(b)) + \sum_{b_f \in F} p(b_f|b,\pi(b)) = 1$. Then from Eq. (6.2),

$$\rho(b;\pi) = \sum_{b_f \in F} p(b_f|b,\pi(b)) \cdot 1 \tag{6.4}$$

and from Eq. (2.4) with $\gamma = 1$ we have:

$$\lim_{J^F\to\infty} \frac{J(b_f;\pi)}{J^F} = \lim_{J^F\to\infty} \frac{\sum_{b_f \in F} p(b_f|b,\pi(b)) \cdot J^F}{J^F} \tag{6.5}$$

Thus, all such belief b satisfies Eq. (6.3).

Now we consider a belief b such that its all successors $\{b'|b,\pi(b)\}$ satisfy Eq. (6.3). By injecting Eq. (6.3) for the successors into Eq. (6.2),

$$\rho(b;\pi) = \sum_{b' \in \mathbb{B}} \lim_{J^F\to\infty} p(b'|b,\pi(b)) \frac{J(b';\pi)}{J^F} \tag{6.6}$$

By dividing Eq. (2.4) by J^F,

$$\lim_{J^F\to\infty} \frac{J(b;\pi)}{J^F} = \lim_{J^F\to\infty} \frac{1}{J^F} \sum_{b' \in \mathbb{B}} p(b'|b,\pi(b)) J(b';\pi) \tag{6.7}$$

Thus, it satisfies Eq. (6.3).

Finally, by backward induction, Eq. (6.3) is satisfied for $\forall b \in \mathbb{B}$ in RAL-POMDPs. □

Theorem 6.1. *In RAL-POMDPs where $J^F \to \infty$ and $\gamma = 1$, the optimal policy π^* that minimizes $J(b;\pi^*)$ also minimizes $\rho(b;\pi^*)$ for $\forall b \in \mathbb{B}$.*

Proof. First, we can rewrite Eq. (2.7) as follows for RAL-POMDPs where $J^F \to \infty$.

$$\pi^*(b) = \underset{\pi\in\Pi}{\operatorname{argmin}} \lim_{J^F\to\infty} J(b;\pi),\ \forall b \in \mathbb{B} \tag{6.8}$$

By dividing the objective function in Eq. (2.7) by a constant J^F, we have:

$$\pi^*(b) = \underset{\pi\in\Pi}{\operatorname{argmin}} \lim_{J^F\to\infty} \frac{J(b;\pi)}{J^F},\ \forall b \in \mathbb{B} \tag{6.9}$$

Then by Lemma 6.1, we prove the theorem.

$$\pi^*(b) = \underset{\pi\in\Pi}{\operatorname{argmin}} \rho(b;\pi),\ \forall b \in \mathbb{B} \tag{6.10}$$

□

6.3 Bi-directional Value Learning (BVL)

6.3.1 Algorithm

6.3.2 Structure of Bi-directional Value Learning

Figure 6.2 conceptually shows how the combination of forward short-range and backward long-range planners works. When merging these two classes of methods the overall cost-to-go (or value) consists of three terms: 1) cost learned by the short-range planner C^{sr}, 2) cost learned by the long-range planner C^{lr}, and 3) cost learned by the bridge planner that connects the short-range policy to the long-range policy C^{br}.

$$\begin{aligned}\pi(\cdot) = \underset{\Pi}{\operatorname{argmin}}\, \mathbb{E}\,[&C^{sr}(b^{sr};\pi(\cdot)) \\ &+ C^{br}(b^{br},\pi(\cdot)) + C^{lr}(b^{lr},\pi(\cdot))]\end{aligned} \tag{6.11}$$

In the following section, we formally develop the above combined policy with a concrete instance of *bi-directional long-short-range* POMDP solvers.

6.3.3 FIRM: Approximate Long-Range Planner

FIRM is an approximate long-range planner which utilizes sampling-based roadmap in belief space and feedback controllers as funnels in belief space [9]. It has the following characteristics.

1. It assumes a Gaussian belief model to alleviate the curse of dimensionality by its parametric representation.
2. It creates a graph in belief space where a belief state can stabilize to its nodes by local feedback controllers, so that it can break the curse of history.
3. A global policy to the goal is obtained by concatenating the local policies between the nodes on the graph.

Now we formally describe FIRM's components and its global policy. We define the i-th FIRM node B^i as a set of belief states near a center belief $b_c^i \equiv (v^i, P_c^i)$, where v^i is a sampled point in state space and P_c^i is the node covariance. Nearby belief states can be steered to this node by a local controller.

$$B^i = \{b : ||b - b_c^i|| \leqslant \epsilon\} \tag{6.12}$$

where ϵ decides the node size. $\mathbb{V}^g = \{B^i\}$ is the set of all FIRM nodes.

For each FIRM node B^j, a node controller (stabilizer) μ_s^j can be designed by a Stationary LQR controller. For each FIRM edge between two FIRM nodes B^i and B^j, an edge controller $\bar{\mu}^{ij}$ can be designed by time-varying LQR controllers. Then, by concatenating $\bar{\mu}^{ij}$ and μ_s^j, we can construct a local controller μ^{ij} that can steer the belief from B^i to B^j. Formally, controller μ^{ij} is a mapping from the belief space to the action space $\mu^{ij} : \mathbb{B} \to \mathbb{U}$. We denote the set of all local controllers as $\mathbb{M}^g = \{\mu^{ij}\}$ and the set of all local controllers originated from B^i as $\mathbb{M}(i) \subset \mathbb{M}$, respectively.

A FIRM edge cost is computed by running Monte Carlo simulations from the center belief state of a FIRM node, b_c^i, to the vicinity of another FIRM node, B^j, using the local controller, μ^{ij}.

$$\tilde{C}^g(B^i, \mu^{ij}) = \sum_{k=0}^{\mathcal{K}^{ij}} c(b_k, \mu^{ij}(b_k)) \tag{6.13}$$

where $b_0 = b_c^i$. $\mathcal{K}^{ij}$ is the number of time steps controlled by μ^{ij} until b_k reaches B^j by satisfying (6.12).

A FIRM graph policy is a mapping from FIRM nodes to FIRM edges, $\tilde{\pi}^g : \mathbb{V}^g \to \mathbb{M}^g$, while a general POMDP policy is a mapping from a belief state to a control input, $\pi : \mathbb{B} \to \mathbb{U}$.

For a FIRM graph policy, an approximate cost-to-go can be computed as follows.

$$\tilde{J}^g(B^i; \tilde{\pi}^g) = \mathbb{E}\left[\sum_{k=0}^{\infty} \tilde{C}^g(B_k, \tilde{\pi}^g(B_k))\right] \tag{6.14}$$

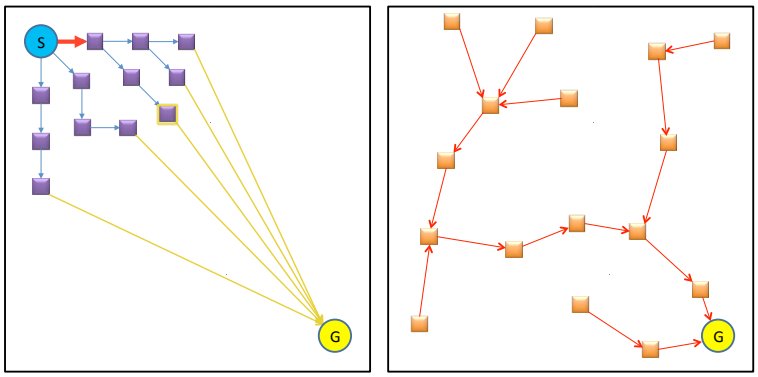
where $B_0 = B^i$. We denote by $\mathbb{N}(B^i)$ the set of neighbor FIRM nodes of B^i. Note that this approximate cost-to-go are computed by concatenating FIRM edge costs that are separately computed without considering the actual history from the start. Equation (6.14) can also be rewritten in a recursive form as follows.

$$\begin{aligned}\tilde{J}^g(B^i; \tilde{\pi}^g) = \; & \tilde{C}^g(B^i, \tilde{\pi}^g(B^i)) \\ & + \sum_{B^j \in \mathbb{N}(B^i)} \mathbf{P}^g(B^j | B^i, \tilde{\pi}^g(B^i)) \tilde{J}^g(B^j; \tilde{\pi}^g)\end{aligned} \tag{6.15}$$

where $\mathbf{P}^g(B^j | B^i, \tilde{\pi}^g(B^i))$ is the transition probability from B^i to B^j under $\mu^{ij} = \tilde{\pi}^g(B^i)$.

Then FIRM solves the following optimization problem by value iteration.

$$\tilde{\pi}^{g^+}(\cdot) = \underset{\tilde{\Pi}^g}{\operatorname{argmin}} \, \tilde{J}^g(B_k; \tilde{\pi}^g) \tag{6.16}$$



(a) Local policy by a short-range planner

(b) Global policy by a long-range planner

Figure 6.2: Illustration of policies (thin arrows) and action selections (thick red arrows) by a forward short-range planner, a backward long-range planner, and a combined bi-directional long-short-range planner.

6.3.4 POMCP: Near-optimal Short-Range Planner

POMCP is an online POMDP solver that uses Monte Carlo Tree Search (MCTS) in belief space and particle representation for belief states [43]. The followings are the POMCP's characteristics.

1. It represents a belief state by a set of particles. This state sampling helps to ameliorate the curse of dimensionality.

2. It uses MCTS with UCT (Upper Confidence Bound applied to Trees) algorithm to control the exploration-exploitation trade-off. This history sampling tempers the curse of history.

3. It can efficiently explore the forward value space considering the actual history due to its forward simulation strategy and balancing between exploration-exploitation.

POMCP's action selection during forward simulation is governed by two policies: a tree policy within the constructed belief tree, and a rollout policy out of the tree but up to the finite discount horizon.

Algorithm 15 GlobalSetup()

parameters:
N_{mc}: the number of Monte Carlo simulations
K^{sr}: short-range planner's fixed horizon
K^{br}_{max}: bridge planner's maximum horizon limit
η_q: constant for exploration bonus in a tree policy
η_w: constant for exploration bonus in a rollout policy
global variables:
$\mathbb{V}^g = \{B^j\}$: a set of FIRM nodes
$\tilde{J}^g(B^j)$: cost-to-go of $B^j \in \mathbb{V}^g$
$\mathbb{T} = \{b\}$: a set of belief states on POMCP tree
$J(b)$: cost-to-go of $b \in \mathbb{T}$
$N(b)$: visitation count of $b \in \mathbb{T}$
$Q(b, u)$: Q-value for $b \in \mathbb{T}$ and $u \in \mathbb{U}$
$N(b, u)$: visitation count for $b \in \mathbb{T}$ and $u \in \mathbb{U}$
$\mathbb{M}(b) = \{\mu^{\cdot j}\}$: a set of local controllers for $b \in \mathbb{T}$

The tree policy selects an action based on PO-UCT algorithm as follows.

$$Q^{\ominus}(b, u) = Q(b, u) - \eta_q \sqrt{\frac{\log(N(b))}{N(b, u)}} \tag{6.17}$$

$$u^* = \underset{u \in \mathbb{U}}{\operatorname{argmin}}\, Q^{\ominus}(b, u) \tag{6.18}$$

where $Q(b, u)$ is defined in (2.5), and $N(b)$ and $N(b, u)$ are the visitation counts for a belief node and an intermediate belief state-action node, respectively. η_q is a constant for exploration bonus in the tree policy. As η_q gets larger, PO-UCT will be more explorative in action selection.

The rollout policy can be a random policy if there is no domain knowledge available. Otherwise, it can choose an action from a preferred action set which is pre-defined heuristically but without any sense of the global policy. A balanced setup of tree policy and rollout policy helps POMCP to efficiently explore the belief space and learn the values.

After each Monte Carlo simulation, $Q(b, u)$ is updated by the following rule.

$$Q'(b, u) = Q(b, u) + \frac{R - Q(b, u)}{N(b, u)} \tag{6.19}$$

where $Q'(b, u)$ denotes the updated $Q(b, u)$ value.

Algorithm 16 OfflinePlanning()

input:
$\mathbb{X}_{free}$: free space map
b^{goal}: a goal belief state
output:
$\mathbb{V}^g$: a set of FIRM nodes
$\tilde{J}^g(B^j)$: cost-to-go of a FIRM node, $B^j \in \mathbb{V}^g$
B^{goal}: a goal FIRM node
procedure OFFLINEPLANNING()
  Sample PRM nodes $\mathcal{V} = \{v^j\}$ s.t. $v^j \in \mathbb{X}_{free}$
  $v^{goal} \leftarrow \hat{m}^{goal}$, where $b^{goal} = (\hat{m}^{goal}, \hat{P}^{goal})$
  $\mathcal{V} \leftarrow \mathcal{V} \cup \{v^{goal}\}$
  Construct PRM edges $\mathcal{E} = \{e^{ij}\}$, where e^{ij} is an edge from v^i to $v^j \in \mathbb{N}(v^i)$
  for all $v^i \in \mathcal{V}$ **do**
    Design a node controller μ_s^i for v^i
    Construct a FIRM node B^i and its node center b_c^i
    if v^i is v^{goal} **then**
      $B^{goal} \leftarrow B^i$
  for all $e^{ij} \in \mathcal{E}$ **do**
    Design an edge controller $\bar{\mu}^{ij}$ along e^{ij}
    Sample belief paths from b_c^i to B^j using a local controller μ^{ij}, i.e., $\bar{\mu}^{ij}$ followed by μ_s^j
    Compute transition cost $\tilde{C}^g(B^i, \mu^{ij})$ and transition probability $\mathbf{P}^g(B^j|B^i, \mu^{ij})$
  $\mathbb{V}^g \leftarrow \{B^j\}$
  Compute cost-to-go $\tilde{J}^g(B^j)$ and FIRM feedback policy $\tilde{\pi}^g(B^j)$
    for $B^j \in \mathbb{V}^g$ by value iteration
  return B^{goal}

6.3.5 BVL: POMCP on top of FIRM

BVL is a locally-optimal long-range POMDP solver that leverages backward search of FIRM as well as the forward search of POMCP. Formally, as a concrete instantiation of (6.11), we can write down the optimization problem for BVL as follows.

$$\pi(\cdot) = \operatorname{argmin} \mathbb{E}\left[\sum_{k=0}^{K^{sr}-1} c(b_k, \pi_k(b_k)) + \sum_{k=K^{sr}}^{K^{sr}+K^{br}-1} c(b_k, \pi_{(K^{sr}-1)}(b_k)) + \tilde{J}^g(B^{j^+}; \tilde{\pi}^{g^+})\right] \tag{6.20}$$

where K^{sr} is a fixed horizon for the short-range planner and K^{br} is a variable horizon of the bridge planner such that $b_{(K^{sr}+K^{br})} \in B^{j^+}$ for $B^{j^+} \in \mathbb{N}(b_{(K^{sr}-1)})$. The first two terms are handled by POMCP modified for the long-range planner framework, and the third term is computed by FIRM during the offline phase.

Algorithm 17 OnlinePlanningAndExecution()

input:
b_0: an initial belief state
B^{goal}: a goal FIRM node
output:
R: total execution cost
procedure OnlinePlanningAndExecution(b_0, B^{goal})
  $b \leftarrow b_0$
  $R \leftarrow 0$
  while $b \notin B^{goal}$ **do**
    $u^* \leftarrow$ Search(b)
    $z' \leftarrow$ ExecuteAndObserve(u^*)
    $b' \leftarrow$ EvolvedBeliefState(b, u^*, z')
    $c' \leftarrow$ ComputeExpectedCost(b, b')
    $R \leftarrow R + c'$
    $b \leftarrow b'$
  return R

Algorithm 18 Search()

procedure Search(b)
  for $i = 1, 2, ..., N_{mc}$ **do**
    $x \sim b$ ▷ draw a sample from belief b
    Simulate($x, b, 0,$ `nil`)
  $b \leftarrow$ GetMatchingBeliefNode($\mathbb{T}, b$)
  $\mu^* \leftarrow \text{argmin}_{\mu^{\cdot j} \in \mathbb{M}(b)} Q(b, \mu^{\cdot j}(b))$
  $u^* \leftarrow \mu^*(b)$
  return u^*

In the rest of this section, we discuss several aspects of the proposed framework and contrast them with the original POMCP framework. For the complete details, see Algorithm 15–20.

Bridging POMCP to FIRM: The shortcomings of POMCP in RAL-POMDP problems mainly come from lack of proper guidance beyond its horizon. In contrast, BVL bootstraps FIRM's global policy to guide the forward search and improve the safety guarantees and optimality. There are two places where FIRM's cost-to-go information is being used.

1) When we add a new belief node b to the BVL forward search tree, it is initialized using the cost-to-go from the underlying FIRM graph.

$$Q_{init}(b, u^j) = \mathscr{C}(b, B^j) + \tilde{J}^g(B^j) \tag{6.21}$$

$$J_{init}(b) = \min_{u^j \in \mathbb{U}(b)} Q_{init}(b, u^j) \tag{6.22}$$

Algorithm 19 Simulate()

procedure SIMULATE($x, \mathcal{b}, k, \mu^-$)
  if $k > K^{sr}$ **then**
    return ROLLOUT($x, \mathcal{b}, k, \mu^-$)
  $b \leftarrow$ GETMATCHINGBELIEFNODE($\mathbb{T}, \mathcal{b}$)
  if b is `nil` **then**
    $\mathbb{T} \leftarrow$ ADDNEWBELIEFNODETOTREE($\mathbb{T}, \mathcal{b}$)
    $b \leftarrow \mathcal{b}$
    $\mathbb{M}(b) \leftarrow \{\}$
    $\mathbb{N}(b) \leftarrow$ GETNEARESTNEIGHBORS(b)
    for all j s.t. $B^j \in \mathbb{N}(b)$ **do**
      $\mu^{\cdot j} \leftarrow$ GETLOCALCONTROLLER(b, B^{j^*})
      $\mathbb{M}(b) \leftarrow \mathbb{M}(b) \cup \{\mu^{\cdot j}\}$
      $\mathscr{C}^j \leftarrow$ HEURISTICEDGECOST(b, B^j)
      $Q(b, \mu^{\cdot j}(b)) \leftarrow \mathscr{C}^j + \tilde{J}^g(B^j)$
    $J(b) \leftarrow \min_{\mu^{\cdot j} \in \mathbb{M}(b)} Q(b, \mu^{\cdot j}(b))$
    $N(b) \leftarrow 0$
    return ROLLOUT($x, \mathcal{b}, k, \mu^-$)
  else
    $b \leftarrow$ UPDATEBELIEFNODE($b, \mathcal{b}$)
    $\mu^* \leftarrow \operatorname{argmin}_{\mu^{\cdot j} \in \mathbb{M}(b)} Q(b, \mu^{\cdot j}(b)) - \eta_q \sqrt{\frac{\log N(b)}{N(b,\, \mu^{\cdot j}(b))}}$
    $u^* \leftarrow \mu^*(b)$
    $(x', z', c') \leftarrow$ GENERATIVEMODEL(x, u^*)
    $\mathcal{b}' \leftarrow$ EVOLVEDBELIEFSTATE($\mathcal{b}, u^*, z'$)
    $R \leftarrow c' +$ SIMULATE($x', \mathcal{b}', k{+}1, \mu^*$)
  $N(b, u^*) \leftarrow N(b, u^*) + 1$
  $N(b) \leftarrow N(b) + 1$
  $Q(b, u^*) \leftarrow Q(b, u^*) + \frac{R - Q(b,\, u^*)}{N(b,\, u^*)}$
  $J(b) \leftarrow \min_{\mu^{\cdot j} \in \mathbb{M}(b)} Q(b, \mu^{\cdot j}(b))$
  return $J(b)$

where $\mathbb{U}(b) = \{\mu^{\cdot j}(b) \,|\, \mu^{\cdot j} \in \mathbb{M}(b)\}$ and $\mathbb{M}(b)$ is a set of local controllers steers a belief from b to its neighbor FIRM node $B^j \in \mathbb{N}(b)$. $\mathscr{C}(b, B^j)$ is a heuristically estimated edge cost from b to B^j, and $\tilde{J}^g(B^j)$ is the cost-to-go computed by FIRM in the offline phase. The visitation counts are initialized to zeros, i.e., $N_{init}(b, u^j) = 0$ and $N_{init}(b) = 0$. Note that the action space $\mathbb{U}(b)$ is only a subset of the continuous action space which is based on local controllers toward neighbor FIRM nodes.

2) The rollout policy selects an action by random sampling from a probability mass function which is based on FIRM's cost-to-go, rather than a uniform distribution. In more detail, for each $B^j \in \mathbb{N}(\mathcal{b})$, the weight w^j is computed as

$$w^j = (\mathscr{C}(\mathcal{b}, B^j) + \tilde{J}^g(B^j))^{-1} + \eta_w \tag{6.23}$$

Algorithm 20 Rollout()

procedure ROLLOUT($x, \mathcal{b}, k, \mu^-$)
  if $k > K^{sr}$ **then**
    $B^{j^-} \leftarrow$ GETTARGETFIRMNODE(μ^-)
    if $\mathcal{b} \in B^{j^-}$ **then**
      return $\tilde{J}^g(B^{j^-})$
    else
      $\mu^* \leftarrow \mu^-$
      $u^* \leftarrow \mu^*(b)$
  else
    $\mathbb{N}(\mathcal{b}) \leftarrow$ GETNEARESTNEIGHBORS($\mathcal{b}$)
    for all j s.t. $B^j \in \mathbb{N}(\mathcal{b})$ **do**
      $\mathscr{C}^j \leftarrow$ HEURISTICEDGECOST($\mathcal{b}, B^j$)
      $w^j \leftarrow \frac{1}{\mathscr{C}^j + \tilde{J}^g(B^j)} + \eta_w$
    $j^* \sim \frac{w^j}{\sum_{j'} w^{j'}}$
    $\mu^* \leftarrow$ GETLOCALCONTROLLER($\mathcal{b}, B^{j^*}$)
    $u^* \leftarrow \mu^*(\mathcal{b})$
  $(x', z', c') \leftarrow$ GENERATIVEMODEL(x, u^*)
  $\mathcal{b}' \leftarrow$ EVOLVEDBELIEFSTATE($\mathcal{b}, u^*, z'$)
  return $c' +$ ROLLOUT($x', \mathcal{b}', k+1, \mu^*$)

where $\mathcal{b}$ denotes the sampled belief state in the current Monte Carlo simulation. η_w is a constant for the exploration bonus in the rollout policy. As η_w gets larger, the rollout policy will be more explorative. $\eta_w = \infty$ will lead to pure exploration, i.e., random sampling from uniform distribution.

Based on the computed weight w^j, the rollout policy selects an action by random sampling $u^* \sim p(u^j; \mathcal{b})$ from the following probability mass function.

$$p(u^j; \mathcal{b}) = \frac{w^j}{\sum_{j'} w^{j'}} \text{ for } \forall u^j \in \mathbb{U}(\mathcal{b}) \tag{6.24}$$

Backup/update of cost-to-go: In the original POMCP, $J(b) = \min_{u \in \mathbb{U}(b)} Q(b, u)$ in (2.6) is not backed up or updated due to its preference to *unbiasedness*. This is reasonable in dynamic or adversarial environments, such as Go, which are the target scenarios of the original POMCP [43, 81]. However, lack of J backup leads to higher variance and slower convergence of $Q(b, u)$.

This problem gets more significant in RAL-POMDP problems with longer horizon without discounting and computationally expensive forward simulation. For example, con-

sider a scenario where there is a safety risk near to the goal but far from the current state and the optimal path is to avoid this risk only after the robot gets closer to it. Without discounting, this delayed risk will percolate to all the values along its forward simulation path, and then POMCP without $J(b)$ backup will spend a lot of time to simulate many other paths branched off near the current state. Further, robot simulation and safety evaluation are orders of magnitude computationally more expensive than the game evaluation. Thus, overall, performing many runs and slow convergence is not an option for applications to complex physical systems.

To remedy this, in BVL, we backup $J(b)$ as described in (2.6) in addition to $N(b)$, $N(b, u)$, and $Q(b, u)$. As can be seen in line 28 of Algorithm 19, $J(b)$ is updated at every iteration when $Q(b, u)$ for $\forall u \in \mathbb{U}$ is updated.

6.4 Experimental Results

In this section, we first define a representative RAL-POMDP problem and a set of metrics to evaluate the performance of different planners. Next, we describe three baseline methods that are compared with the proposed algorithm in terms of safety, scalability, and optimality under different scenarios.

6.4.1 Rover Navigation Problem

As a representative RAL-POMDP problem, we consider the real-world problem of the Mars rover navigation under motion and sensing uncertainty. In the Rover Navigation Problem (RNP) introduced here, the objective is to navigate a Mars rover from a starting point to a goal location while avoiding risk regions such as steep slopes, large rocks, etc. The rover is provided a map of the environment which is created by a Mars orbiter satellite [86] and a Mars helicopter [87] flying ahead of the rover. This global map contains the location of landmarks, which serve as information sources that the rover can use to localize itself on the global map. The map also contains the location of risk regions that the rover needs to avoid and regions of science targets which needs to be visited by the

rover to collect data or samples.

Many robotic navigation problems, in particular Mars rover missions, fall into the category of safety-critical missions which induce RAL-POMDP problems. Hence, the planner should ensure the safety of the rover throughout its execution under motion and sensing uncertainty.

Motion model: The motion of the rover is noisy due to factors like wheel slippage, unknown terrain parameters, etc. In the RNP introduced here, we assume a nonlinear motion model (but still a holonomic one to provide a simple benchmark). Specifically, we use the model in [88], where the state $x = [{}^g x, {}^g y, {}^g \theta]^T \in \mathbb{R}^3$ represents the 2D position and heading angle of the rover in the global world frame. Control input $u \in \mathbb{R}^3$ represents the velocity of each robot wheel. Using [88], we obtain the discrete motion model as follows:

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1}) \tag{6.25}$$

where w is motion noise drawn from a Gaussian distribution with zero mean.

Observation model: In the RNP, we assume the rover can measure the range and bearing to each landmark. Denoting the displacement vector to a landmark L^i by $d^i = [d^i_x, d^i_y]^T \equiv L^i - p$, where $p = [{}^g x, {}^g y]^T$ is the position of the robot, the observation model is given by:

$$z^i = h^i(x, v^i) = \left[||d^i||,\ \tan^{-1}(d^i_y/d^i_x) - {}^g\theta\right]^T + v^i \tag{6.26}$$

$$R^i = \text{diag}\left((\xi_r||d^i|| + \sigma^r_b)^2,\ (\xi_\theta||d^i|| + \sigma^\theta_b)^2\right) \tag{6.27}$$

where $v^i \sim \mathcal{N}(0, R^i)$. The measurement quality decreases as the distance of the robot from the landmark increases, and the weights ξ_r and ξ_θ control this dependency. σ^r_b and σ^θ_b are the bias standard deviations.

Cost and risk metrics: In the RNP, we consider the localization accuracy as well as the mission completion time as the main elements of the cost function. Localization accuracy depends on the distribution of landmarks in the global map and their locations relative to the rover. Risk in RNP denotes the probability of the collision with obstacles,

which is modeled as a soft constraint. We use the following cost function the reflects the localization accuracy of the rover:

$$c(b_k, u_k) = \text{tr}(P_k),\ J = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k c(b_k, \pi_k(b_k))\right] \tag{6.28}$$

where $\xi_p = 100$, $\xi_u = 0$, $\xi_T = 1$, and $\xi_c = 10^9$ for the experiments. Note that the total execution cost of a mission is $J(b_0, \pi(\cdot)) = \sum_{k=0}^{\infty} c(b_k, \pi_k(b_k))$.

RNP scalability: To test different algorithms under different RNP complexities, we parameterize the Rover Navigation Problem as $RNP_s(e, o)$, where e represents the size of the environment, o represents the size/density of obstacles, and s represents different environment types. We compare three key attributes (safety, scalability, and optimality) of each algorithm in three different environments ($RNP_{InformationTrap}$, $RNP_{ObstaclesWall}$, and $RNP_{RockForest}$). A detailed description of these scenarios is given in the following subsections.

6.4.2 Baseline Methods

As baseline methods, we consider three algorithms. From the class of forward search methods, we consider the POMCP method [43]. From the approximate long-range methods, we consider the FIRM [9] and its variant [89] which is referred to as online graph-based rollout (OGR) here.

FIRM: FIRM is an execution of closed-loop controls returned by FIRM offline planning algorithm. FIRM relies on belief-stabilizing local controllers at each FIRM node to ameliorate the curse of history. Hence, it can solve larger problems, but it is usually suboptimal compared to optimal online planners.

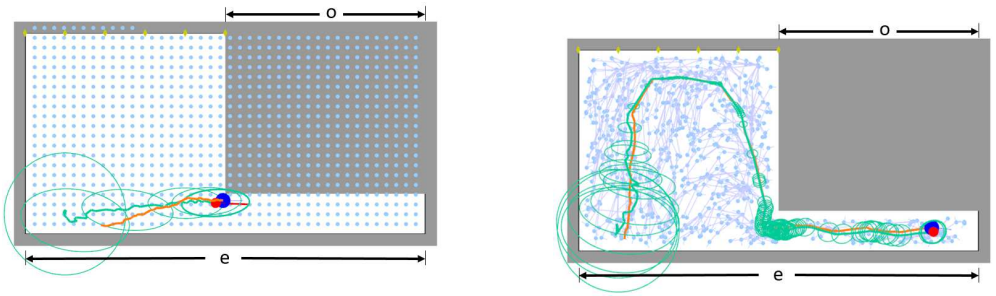
OGR: OGR is an online POMDP solver that improves the optimality of a base graph-based method (particularly, FIRM in our implementation). At every iteration, an OGR planner selects the next action by simulating all different possible actions and picking the best one. Compared to BVL, OGR expands the belief tree in a full-width but only for one-step lookahead. While it can improve the performance of its base graph-based

planner, it is prone to local minima due to the suboptimality in base planner's cost-to-go and OGR's myopic greedy policy. Additionally, OGR discards the performed forward simulation results in the next iteration, while BVL leverages them at each step to enrich the underlying tree structure.

URM-POMCP: POMCP lacks in the capability of working in a larger and continuous space because it has a finite discount horizon and requires a finite action set. To assist POMCP to solve large and continuous space problems like RAL-POMDP, we extend it to be able to estimate the cost-to-go beyond the horizon and to discretize the action space into a finite action set. This extended version is referred to as URM-POMCP (Uniform RoadMap POMCP).

Firstly, URM-POMCP uses a heuristic cost-to-go function to cope with POMCP's finite horizon limitation. Each Monte Carlo simulation in POMCP should be episodic, which means that each forward simulation should reach either the discounted horizon or terminal states. In RAL-POMDP problems with infinite horizon and a goal state, it is almost impossible to satisfy this condition unless it is close to the terminal states. Instead of computing a cost-to-go by actual forward simulation, URM-POMCP uses a heuristic function to estimate a cost-to-go from the end of the tree to the goal state. In this work, the heuristic cost-to-go function $\tilde{J}(b, x)$ is implemented as $\tilde{J}(b, x) \approx \frac{d(b,b_g)}{\Delta \dot{x}_{max}}(\xi_p \text{tr}(P_c) + \xi_T \Delta t)$, where $d(b, b_g)$ is the distance from the current belief state to the goal state, $\Delta \dot{x}_{max}$ is the (estimated) maximum velocity of the rover, and P_c is the stationary covariance for $m \in \mathbb{X}$ of the current belief $b = (m, P)$. This heuristic optimistically assumes that the belief can reach the goal by following the direct path at the maximum velocity without collision with the obstacles. It needs to be optimistic, otherwise it will make the planner to prefer conservative actions such as staying around the landmarks forever rather than moving toward the goal.

Additionally, URM-POMCP utilizes a uniformly distributed roadmap in belief space to construct a finite set of actions for POMCP. Each points in the uniformly distributed roadmap serves as the target point of a time-varying LQG controller, so that the controller can generate control inputs for a belief to move toward the point. This enables POMCP to utilize the Gaussian belief model in generating a control input and updating



(a) $RNP_{InformationTrap}(20, 10)$ for URM-POMCP

(b) $RNP_{InformationTrap}(20, 10)$ for BVL

Figure 6.3: Execution trajectories of URM-POMCP (left) and BVL (right). The orange line is the trajectory of the true state, and the green line and ellipses are the trajectory and covariances of the belief state, respectively.

the current belief from an observation. To prevent the robot from getting stuck in a state due to local minima, we also penalize the actions for staying at the same state.

6.4.3 Safety

Reducing risk and ensuring system's safety is the most important goal of the proposed framework. We compare the risk aversion capability of our planner with the baseline methods using $RNP_{InformationTrap}(e, o)$ problem as shown in Fig. 6.3, where e is the length of the environment and o is the length of the obstacle.

In the $RNP_{InformationTrap}$ problem the rover needs to reach the goal by passing through the narrow passage without colliding with any obstacles. As shown in Fig. 6.3, BVL reduces risk of collision by executing a longer trajectory that goes close to the landmarks (yellow diamonds) and reduces the localization uncertainty before entering the narrow passage. Since the URM-POMCP algorithm plans in a shorter horizon and depends on a heuristic cost-to-go estimation beyond the horizon, it takes a greedy approach to go towards the goal thus taking a higher risk of colliding with the obstacles. This can also be seen in Fig. 6.4 that shows the probability of collision of the rover as the length of the obstacle increases. The probability of collision here was estimated by running 20 Monte Carlo simulations of rover executing policies by different planners.

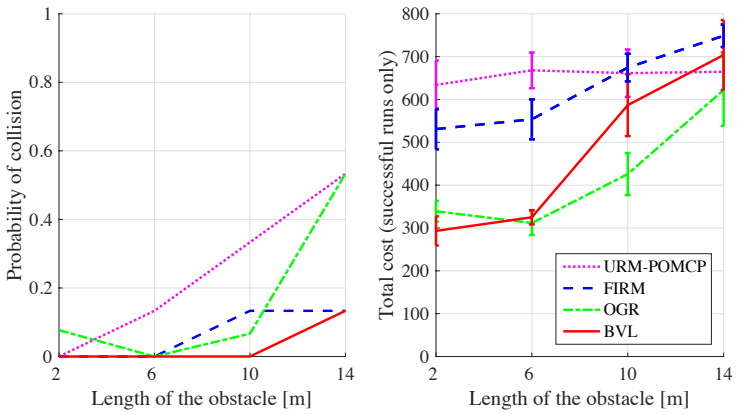


Figure 6.4: Plots of the probability of collision and the total cost over different obstacle lengths. They are evaluated by running 20 Monte Carlo simulations of rover executing policies generated by the planners. Note that total cost mean and standard deviation are computed using the successful (collision-free) executions only, which means that algorithms with higher collision probability will have much higher expected total cost when considering the collision penalty.

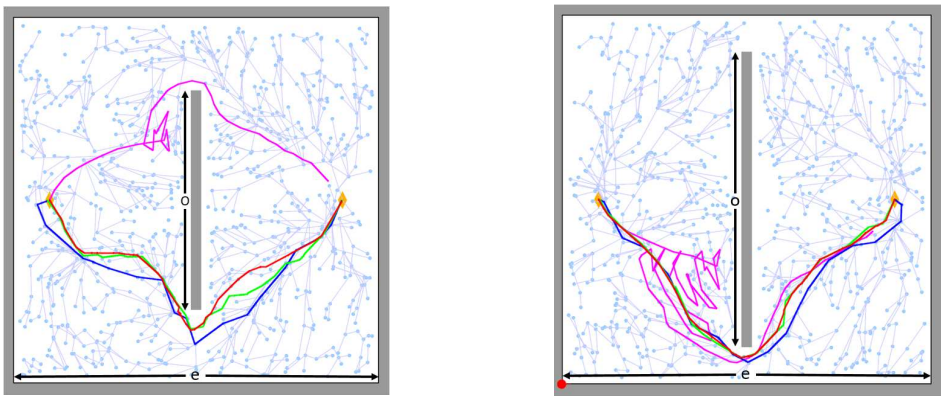
6.4.4 Scalability in Planning Horizon

Bi-directional learning of the value function enables the proposed planner to scale to infinite planning horizons. To compare this scalability with the baseline methods, we consider the $RNP_{ObstacleWall}(e, o)$ problem shown in Fig. 6.5, where e is the length of the environment and o is the length of the obstacle shown in the figure.

Notice that in Fig. 6.6, as the obstacle gets larger, the local minimum gets deeper and the performance of URM-POMCP becomes worse. The number of time steps to get to the goal grows exponentially for URM-POMCP, while it grows linearly for BVL and others. This shows the effectiveness of guidance by long-range solver's global policy in larger problems as opposed to a naive heuristic guidance in URM-POMCP.

6.4.5 Optimality

The fundamental contribution of this method is to achieve policies that are closer to the globally optimal policies while reducing the risk of collisions over long horizons. To compare the optimality of the planners, we consider the $RNP_{RockForest}(e, o)$ problem shown in Fig. 6.7, where e represents the length of the environment and o represents the



(a) $RNP_{ObstacleWall}(20, 10)$ (b) $RNP_{ObstacleWall}(20, 16)$

Figure 6.5: Execution trajectories of URM-POMCP (pink), FIRM (blue), OGR (green), and BVL (red). The start and the goal states are on the left and the right of the wall, respectively.

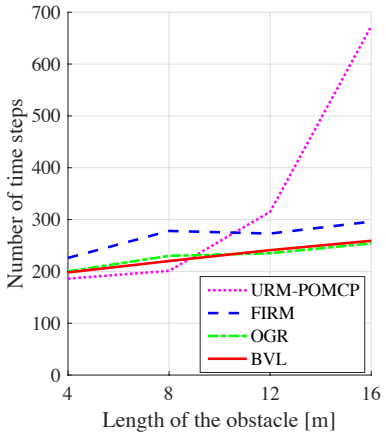
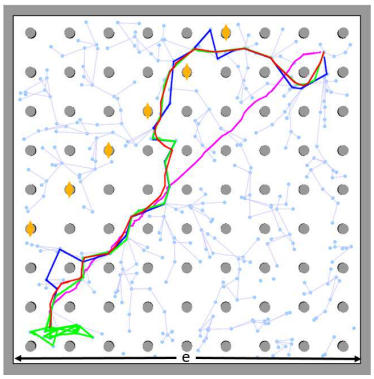


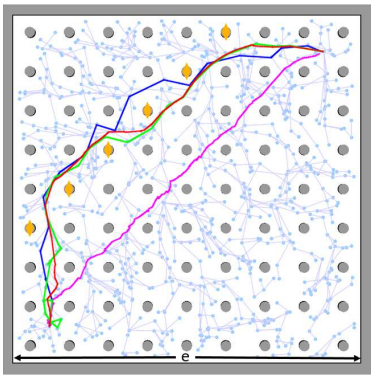
Figure 6.6: Plot for scalability tests. A longer wall and more landmarks induce deeper local minima for URM-POMCP due to its heuristic cost-to-go estimation. URM-POMCP performs worse as having more local minima, but other methods with FIRM's approximate cost-to-go are less affected.

number of obstacles. We vary the density of the underlying belief graph to demonstrate its effect on the proposed method.

As can be seen in Fig. 6.7 and Fig. 6.8, as the density of the underlying graph gets higher, the performance of the FIRM solution increases. However, it will reach a maximum sub-optimal bound due to its sampling-based nature (i.e., it requires stabilization of the belief to the stationary covariance of the graph nodes before leaving them). In this complex environment, OGR with myopic online replanning frequently gets stuck at local minima, while it sometimes outperforms FIRM. Its performance is brittle and subject to the coverage of the underlying belief graph. In contrast, BVL performs well even with a smaller



(a) $RNP_{RockForest}(20, 81)$ with 350 FIRM nodes



(b) $RNP_{RockForest}(20, 81)$ with 800 FIRM nodes

Figure 6.7: Execution trajectories for URM-POMCP (pink), FIRM (blue), OGR (green), and BVL (red). The start and the goal states are on the left bottom and the right top of the map, respectively. While both OGR and BVL take shortcuts instead of following FIRM's offline policy, OGR suffers from local minima near the start state.

number of nodes in the underlying graph.

While actions of the BVL are selected from local controllers connecting to the nodes of the underlying belief graph, online belief tree search process fundamentally improves its behavior such that it is much less dependent on the density and coverage of the underlying graph. BVL not only generates trajectories that are much closer to global optimum but also reduces the risk of collision over an infinite horizon.

6.5 Summary and Discussion

In this work, we proposed BVL, a novel bi-directional value learning algorithm that incorporates locally near-optimal forward search methods and globally safety-guaranteeing approximate long-range methods to solve challenging RAL-POMDP problems. As shown in Fig. 6.9, BVL provides better probabilistic safety guarantees than forward search methods (URM-POMCP) and is closer to the optimal performance than approximate long-range methods (FIRM). It also shows more consistency in different environments compared to online graph-based rollout methods (OGR).

In future work, we will study the theoretical properties of this algorithm more rigorously and extend this work to more general and challenging robotic applications, such as ma-

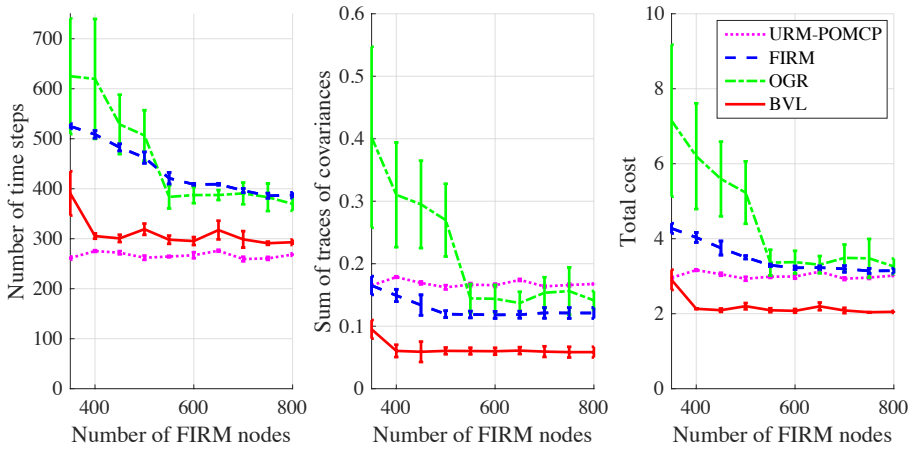


Figure 6.8: Plots for number of time steps, sum of traces of covariances, and total costs over different number of FIRM nodes. BVL performs the best and is least affected by the FIRM node density.

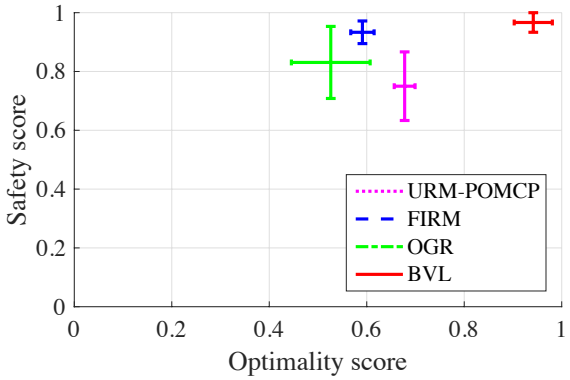


Figure 6.9: Plot for overall optimality and safety evaluation. Optimality score is computed by dividing the minimal total cost over all runs by the individual total cost for $RNP_{RockForest}(e, o)$. Safety score is obtained by subtracting the probability of collision from 1 for $RNP_{InformationTrap}(e, o)$.

nipulation. It is also interesting to investigate bi-directionaly value learning methods for dynamic environments or in adversarial setting.

Chapter 7

Extension: Application to Real-world Mobile Manipulation

7.1 Introduction

We consider the problem of planning and execution of robot motions to complete complex tasks in real-world application. Specifically, we consider the problem of unloading boxes from truck trailers in a warehouse environment onto a conveyor belt (Fig. 7.1). Here, we are required to plan robust actions for a custom-built truck-unloading robot equipped with two end-effectors—a manipulator-like tool with suction grippers as well as a scooper-like tool.

The planning module needs to continuously decide to choose which end-effector to use (manipulator or scooper) and how to use the chosen end-effector (where to pick, how fast to scoop). This needs to be done with partial information (imperfect perception module, unknown box masses, etc.) and for a wide range of environments. While there is a wide range of environments that need to be handled, the same high-level choices can be used in many similar environments. For example, for neatly-stacked walls of boxes, the robot should use the manipulator to unload several rows of boxes (from top to bottom) until the remaining boxes can be scooped. On the other hand, unordered piles of boxes should be scooped slowly to avoid jamming the system. However, even for similar scenarios

Figure 7.1: Truck unloader robot.

that share the same sequence of high-level choices or actions, the system is required to plan the robot's motion for each action in each individual scenario—environments may slightly differ and a collision-free path for one environment may be in-collision with the obstacles in a similar environment.

Thus, we suggest to pre-compute high-level plans using a simulator in an offline stage for recurring settings. These pre-computed high-level plans which we term *strategies* are stored in a library of learned strategies. During execution, the system needs to pick the most appropriate precomputed strategy to execute, given the current state of the system. Once such a strategy is picked, motion plans for individual actions need to be planned and executed.

While the approach described is general and we anticipate that it can be applied to many systems, in this work we focus on the success of the architecture for our specific truck-unloading application (Sec. 7.2). We detail the different modules described (Sec. 7.3) and demonstrate its efficacy both in simulation and in unloading real trucks (Sec. 7.5). We conclude with a discussion highlighting open research questions that emerge from our proposed planning, learning and reasoning architecture (Sec. 7.6).

(a) (b)

(c) (d)

Figure 7.2: Examples of different trailers that need to be unloaded by our system.

7.2 Problem Description

In many warehouse applications, boxes need to be unloaded from incoming trailers and on to conveyor belts where they will be singulated, scanned and sorted. Currently, these trailers are both loaded and unloaded by human workers—a labor-intensive and time-consuming step. The system we consider in this work is aimed at automating the unloading step by a custom-designed mobile robot that can autonomously enter truck trailers, pick boxes using its end-effectors (manipulator-like and scooper-like tools) and place them onto conveyor belts that are located on the robot. These conveyor belts are attached to an extendable conveyor belt which, in turn, is attached to a static conveyor belt where boxes need to be placed (these are the belts where human workers currently have

to unload the boxes to).

Truck trailers come from different vendors and contain boxes in a variety of shapes, sizes, and weights. Furthermore, they are loaded manually by different workers and tend to shift and move when transported. Thus, there are a wide range of environments that need to be considered and a large uncertainty regarding the specific environment encountered when a new truck arrives. For a depiction of different trailers encountered by the system, see Fig. 7.2.

The objective of the system is to unload boxes as quickly and efficiently as possible. However, boxes should not be damaged in this process (e.g., by causing them to fall from large heights). Furthermore, the scooper-like end-effector is wider than the rear conveyor in order to maximize the range of boxes that can be unloaded. Thus, unloading boxes too aggressively may cause system jams where the conveyors become narrower. This, in turn, may require human intervention—a costly operation in terms of time as the system needs to be shut down.

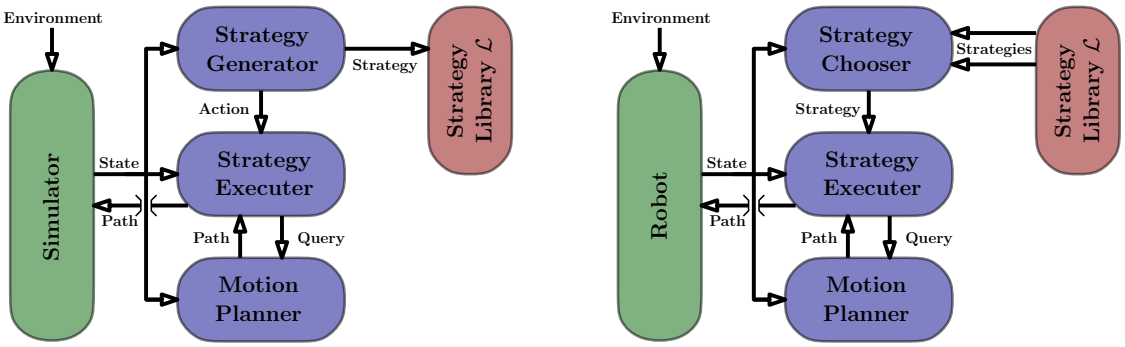
Finally, the system is equipped with a wide suite of sensors that allow it to avoid collision with the trailer, estimate box poses, and detect jams.

We assume that the system has access to a simulator Sim that *estimates* the outcome of an action.

7.3 System Architecture

The planning, learning and reasoning (PLR) component of our system, which is the focus of this work, is tasked with planning the collision-free motions of the robot (end-effectors as well as base) that will maximize the system's throughput (rate of boxes unloaded) while minimizing damage to boxes.

It receives as an input, an estimated system state $\tilde{s}$ provided by the perception module and outputs a trajectory to be executed. Executing actions such as moving the robot's base or picking objects with the end-effectors typically take time that is in the order of several seconds. We assume that the perception module updates $\tilde{s}$ at a high frequency



(a) Offline phase: Generation of strategy library

(b) Online phase: Truck unloading using the strategy library and the strategy chooser

Figure 7.3: System architecture.

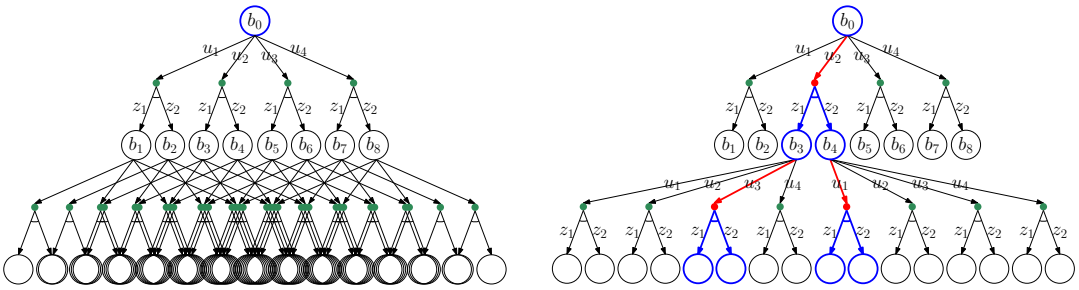
allowing PLR to constantly reason what is the next action to be taken.

Before we detail the system's modules, we introduce a key notion that drives the suggested architecture—*strategies*.

7.3.1 Representation of Robust Motion Plans

We represent a robust motion plan as a decision tree of sequential actions and observations which is referred to as a *strategy* hereinafter. More technically, a strategy is a partial mapping that describes what action to take given a system's state (see Fig. 7.4). Note that the system does not have access to the true state s but to the estimated state $\tilde{s}$, which can be inferred from the initial *belief* (probability distribution over states) and the *history* (the sequence of past actions and observations. Instead of explicitly representing the estimated states, a strategy encodes them in a decision tree where the root is the initial belief and the branches represent the possible histories.

For example, strategies can be simplistic such as "move forward at a constant speed while scooping; if no boxes are left, terminate; otherwise try another scooping" or more complex such as "try to pick up boxes from low-left section of the truck by selecting a target pick point and finding a collision-free motion to it; if the boxes were successfully picked up, try to pick up boxes from low-right section; else if the suction grippers lost contact with the boxes while pulling them back, try to pick up boxes from high-left section; else



(a) Belief tree over all possible actions/observations.

(b) Decision tree with best actions specified for each reachable node.

Figure 7.4: Graphical illustration of a belief tree that can be constructed over all possible actions (u_1, u_2, u_3, and u_4) and observations (z_1 and z_2) along the sequence given the initial belief b_0, and a decision tree, i.e., a *strategy*, that specifies the best action (red arrows) for each belief state (node) reachable from the initial belief.

if no valid pick point was found, move the robot base back and take another observation; ...”

It is possible that a human expert manually designs a decision tree that provides decent performance in general. We refer such a decision tree as a *hardcoded strategy*. Due to the high stochasticity in the domain of interest, a hardcoded strategy should be quite complex, which makes it highly non-trivial to be developed, maintained, or updated. As unforeseen scenarios are encountered or unintended results of certain actions are discovered, the designer of the hardcoded strategy is required to assess how new changes affect the system. If a new vendor is known to typically ship small but fragile products and asks for more careful handling of the packages, then a significant amount of the hardcoded strategy may need to be amended and its performance should be re-evaluated.

To this end, we suggest to *automatically* generate a set of strategies in an offline phase using the simulator Sim. When the system is deployed, at any given time, it *chooses* the most appropriate strategy and executes the actions defined by this strategy. As we will see, this framework allows the system to generate near-optimal strategies for sampled environments and also effectively generalize them to adapt to the stochastic environments at run time.

7.3.2 System Overview

As mentioned previously, PLR uses an offline phase to design strategies which are then used in the online, truck-unloading phase. Several of the system modules are used only in one of the two phases while others are used by both phases. For a system diagram, see Fig. 7.3.

- *Motion Planner*, used in both phases, receives as an input an estimated state $\tilde{s}$ as well as a goal and plans a collision-free trajectory to reach the goal. The goal can be either a specific configuration, a set of configurations, or a specific position for one or more of the robot's end-effector's positions.

- *Strategy Executor*, used in both phases, receives as an input a specific strategy $\mathcal{S}$ and an estimated state $\tilde{s}$. It is tasked with executing it by instantiating the high-level strategy into motion-planning queries and assessing which queries are collision-free (using the motion planner). Once a collision-free trajectory is computed, the strategy executor sends it to the robot's controllers or to the simulator Sim.

- *Strategy Generator*, used in the offline phase, generates a *library* $\mathcal{L}$ of strategies, each tailored for a different given environment. It uses the strategy executor as well as the motion planner and the simulator to simulate the outcome of different actions in order to plan the optimal strategy for the given environment.

- *Strategy Chooser*, used in the online phase, is to decide which strategy to execute given the current state. Effectively, it is a mapping from a specific state to a strategy. This module is required as the strategies planned by the strategy generator were done for specific environments. It is unclear upfront, which strategy to use given a new, unforeseen environment. To learn a generalized mapping from a finite number of (*environment*, *strategy*) pairs in the library, supervised learning techniques, such as k-Nearest Neighbor algorithm, can be employed.

7.4 Robust Motion Planning in Belief Space

In this section, we detail how the strategies are generated by belief space planning. First, we describe the components that are involved in the offline strategy generation process, i.e., the motion planner, the strategy executor, and the strategy generator. Next, we present the problem formulation for the belief space planning algorithm to generate the strategy library.

7.4.1 Detailed System Description

Motion Planner The motion planner module is tasked with planning collision-free motions for the robot. The motion planner must be able to handle a large variety of task-specific path and goal constraints. For example, a given planning query might be for the manipulator-like end-effector to reach a particular pose by moving either the base, the arm or both, without move the scooping tool. For another query, we may want to relax checking for collisions between the manipulator-like end-effector and the boxes, if our intention is to pick them. Yet another query might be to find a goal pose from which another pose is reachable, in the event that we will subsequently attempt to pick at that pose.

In our system, minimizing execution time is of utmost important to unload a large volume of boxes efficiently. Additionally, short planning times are desired to enable the strategy executor to rapidly evaluate many potential motions, when determining the best way to accomplish a high-level action.

Finally, planning times should be *consistent*. Namely, similar queries should roughly take the same amount of planning time. Unfortunately, sampling-based planners such as RRT [90] and its many variants often have a large variance in their planning times due to the stochastic nature of the algorithm.

To this end, we choose to use ARA* [91] as our planner. ARA* is an anytime heuristic search-based planner which tunes its performance bound based on available search time. Specifically, it computes an initial plan quickly and refines its quality as time permits.

Our search space consists of a uniformly discretized state lattice with motion primitives [92], where each motion primitive is a short, atomic motion executable by the robot. To produce efficient-to-execute paths, we provide a cost function that approximates the time to perform each motion primitive using a fixed value, related to the length of the motion in the configuration space.

Strategy Executor The strategy executor module is tasked with efficiently executing a high-level strategy. This is done by continually evaluating the current state of the world as received from the robot's sensors and translating high-level actions into specific motion-planning queries which are then executed by the robot. This is done until the strategy is completed or an alternative one is provided.

Consider the following example strategy: "If there are any boxes that can be picked by the manipulator-like end-effector, attempt to pick them, otherwise scoop the first layer of boxes. Repeat until the truck is unloaded".

To execute this simple strategy, there are several sub-tasks to determine what motions the robot should make to accomplish its goal. How do we evaluate whether any boxes are able to be picked? If there are boxes that can be picked, where should the tool be placed and how should the whole robot move to reach the boxes? If there are no boxes that can be picked, how aggressively should we scoop the boxes?

Each one of these sub-tasks can be accomplished in a number of ways, with previous decisions affecting future decisions. For example, the decision of where to pick may affect the number of picks that need to be made in the future before the remaining boxes can be scooped. The strategy executor can leverage having the full definition of the current strategy to intelligently make these decisions.

Strategy Generator The strategy generator module is tasked with precomputing a set of robust and efficient strategies $\mathcal{L}$ for a small set of given environments (e.g., environments depicted in Fig. 7.2) that contains uncertainty. In generating strategies, we assume that the planner only has access to a perceived environment. A common example of data that is unknown to the planner is box masses which affect the system's dynamics.

This problem can be modeled as the problem of planning robust motions under uncer-

tainty, also known as belief space planning, which can be formulated in a principled form as a Partially Observable Markov Decision Process (POMDP) [93, 94]. In contrast to the assumption used in many POMDP solvers, we do not have access to an explicit model of the POMDP probability distributions but to a generative model, the simulator Sim. Given a state and an action, Sim provides a sample of a successor state, observation, and cost. Monte-Carlo based methods such as POMCP [95] 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. However, such planners are not well suited to a *Goal* POMDP where the objective is to achieve a specific goal (e.g., unloading the boxes from the truck in our setting). They typically lack in effective guidance toward the goal, and thus, require a large number of simulations to converge.

To this end, we incorporate recent advances in heuristic search together with a particle representation. Specifically, we employ POMHDP algorithm that uses multiple heuristics to effectively guide the search toward the goal, and improve the solution quality through repeated searches over time. The heuristics are used to estimate the cost needed to reach the goal from the current state, and in our application, it can be a function of the number of the remaining boxes or the maximum height of the stacks of boxes. For additional details, see [96].

7.4.2 Belief Space Planning Formulation

Consider POMHDP algorithm for the belief space planning to generate a strategy for a given environment. We assume that we have access to a simulation model Sim and have a prior probability distribution of the initial states of the environments, such as box masses, dimensions, and poses.

From the aspect of the belief space planning, a strategy can be formally defined as a mapping from a history to an action for the given initial belief state, i.e., $\pi : \mathbb{B} \times \mathbb{H} \rightarrow \mathbb{A}$ or $\pi(b_0, h) = a$. A belief state $b \in \mathbb{B}$ (to be detailed below) encodes the probability distribution over states and can be approximately represented by a set of sampled states, i.e.,

a set of particles. A history $h \in \mathbb{H}$ implies a sequence of past actions and observations, $\{a_0, o_1, a_1, o_2, ..., a_{t-1}, o_t\}$. Given an initial belief state $b_0 \in \mathbb{B}$, h effectively represents $b \in \mathbb{B}$ at time step t which can be inferred from Bayesian filtering. At a high level, a strategy is in the form of a decision tree that specifies which action to take at a decision node after following a sequence of decision nodes and chance nodes from the root of the tree.

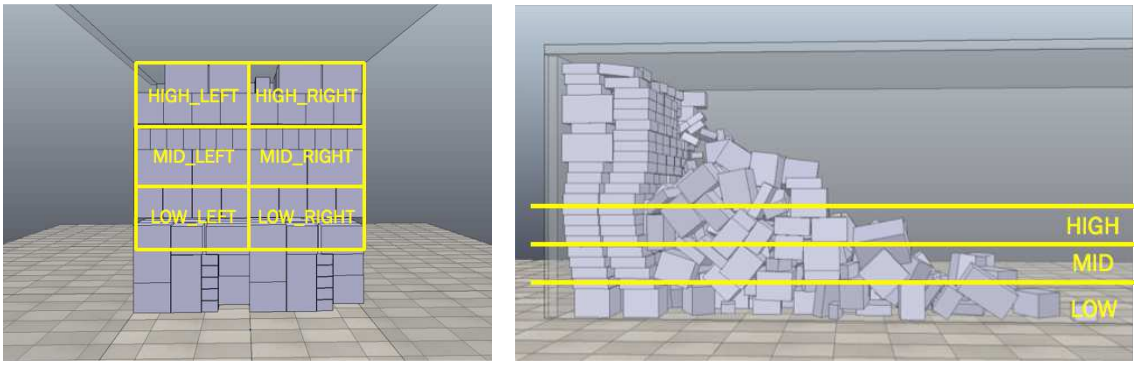
Then we define the components of this belief space planning problem as follows:

- State space: A state describes the configuration of the robot base and joints, and the masses, dimensions, and poses of all N_b boxes in the environment, i.e., $s \in \mathbb{S} = (T_r, q_o^{1:16}, \{m_o^i, l_o^i, T_o^i\})$. T, q, m stand for the pose in 3D space, robot joint positions, and object mass, respectively. The subscript r and o denote the robot and the object, respectively, and the superscript $i = 1, ..., N_b$ represents the index of each box in the environment.

- Belief space: A belief state is a probability distribution over states that essentially encodes the estimated true state. Note that the box masses are unknown and unobservable, so it is not possible to access the true state. Here, a belief state is approximated by a set of sampled states, i.e., a set of particles $b = \{s_1, s_2, ..., s_{N_{part}}\} \in \mathbb{B}$ where N_{part} is the total number of particles.

- Action space: Within the hierarchical system architecture described in Section 7.3, the action space for the belief space planner is a finite set of macro actions that are supported by the strategy executor. For example, `pickup_high_left`, `pickup_low_right`, or `scoop_mid` (see Fig. 7.5). In the current system, there are 10 macro actions available for the belief space planner. Each macro action is instantiated by the strategy executor either in the simulation or in the real robot by taking account of the latest observation.

- Observation space: The observation space is also discretized into 18 cases based on the poses of the boxes that are visible from the robot. For example, `box_pile_high_left`, `box_pile_low`, or `box_pile_none`. In the offline phase we can extract the visible box poses from the simulator's ground truth, while in the online phase we will get such information from the perception module using the real



(a) Pick actions: (`height`, `side`). (b) Scoop actions: (`height`).

Figure 7.5: Macro action parameters for pick and scoop actions. Each macro action is instantiated by the strategy executor based on the corresponding parameters and the latest observation.

sensors on the robot.

- Cost function: We defined a cost function for an individual action as the sum of the time to execute that action and the estimated time for human intervention as needed. The human operator intervention is needed when any boxes make the robot gets stuck, for example, when some boxes fall into the side of the robot so that they can be damaged by any further robot operation.

Note that there persists motion and sensing uncertainty in the described problem. The motion uncertainty comes from the stochastic motion of each box when interacting with the robot or with other boxes. The sensing is also imperfect in terms of the box pose estimation as well as in the fact that the box mass and some of the box dimensions are not observable.

In general, this problem is very challenging and cannot be solved by a single belief space planning query. Unloading a full truck with hundreds or thousands of boxes requires tens or hundreds of macro actions, and to evaluate a single macro action takes about 5 minutes due to the slow speed of the high-fidelity robot simulation.

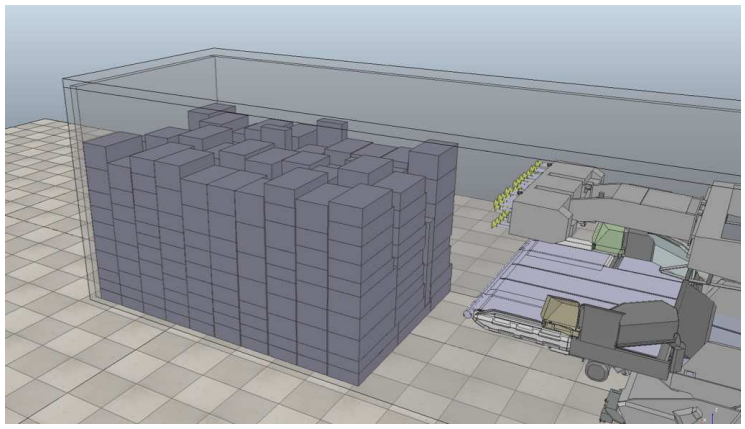
Considering the exponential complexity with the depth of planning horizon in POMDPs, it is practically intractable to solve this as a single infinite-horizon planning problem. Even with POMHDP that alleviates this complexity, the time complexity of the strategy generation is $O(T_{sim}N_{part}N_{action}N_{depth}N_{iter}/N_{sim})$, where T_{sim} is the time to execute an action in simulation, N_{part} is the number of particles, N_{action} is the number of

possible actions, N_{depth} is the depth of planning horizon, N_{iter} is the number of iterations until convergence or timeout, and N_{sim} is the number of simulators running in parallel. In the current setting of $T_{sim} = 5$ min, $N_{part} = 5$, $N_{action} = 10$, and $N_{sim} = 25$, it roughly becomes $(10 \cdot N_{depth} \cdot N_{iter})$ min. Note that N_{iter} may still grow exponentially with N_{depth} in the worst case (according to the quality of the heuristics and the complexity of the problem). If the number of actions required to unload the full truck is 100, one would need $N_{depth} = 100$ and $N_{iter} = 100$, which results in more than 1600 hours for planning.

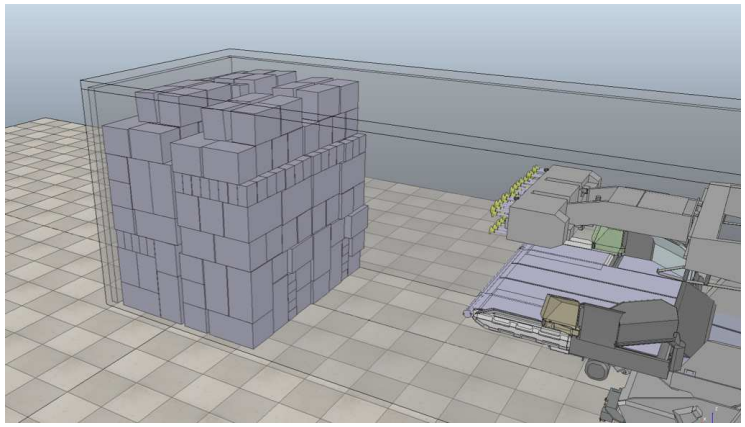
Instead, we incorporate receding horizon control scheme into our framework. Given the initial state, we plan for a finite-horizon strategy (with $N_{depth} = 6$ and $N_{iter} = 5$). After following the most-likely path when executing the finite-horizon strategy, we repeat this planning process again. All the finite-horizon strategies obtained from the beginning until the end of sequential planning episodes are saved in the strategy library. We repeat this process for other initial environments sampled from the prior, and construct a large set of (*environment*, *strategy*) pairs in the library, so that the strategy chooser can find the best strategy for the current environment at run time.

This still may cause suboptimality in the global sense, but greatly reduces the problem complexity, making this approach tractable. In the presented PLR framework, this approach particularly makes more sense. In PLR, there is the online strategy chooser who constantly monitors the strategy execution and re-assigns another strategy to execute whenever necessary. Thanks to its adaptive behavior, the short-horizon plans can still be effective strategies at run time. Moreover, the truck unloading task is locally repetitive, i.e., the front box configurations at the early stage and the late stage look similar. So, for example of a simple case, a strategy with six sequential actions can unload boxes of the first layer, and then the same strategy can be re-used to unload the second layer.

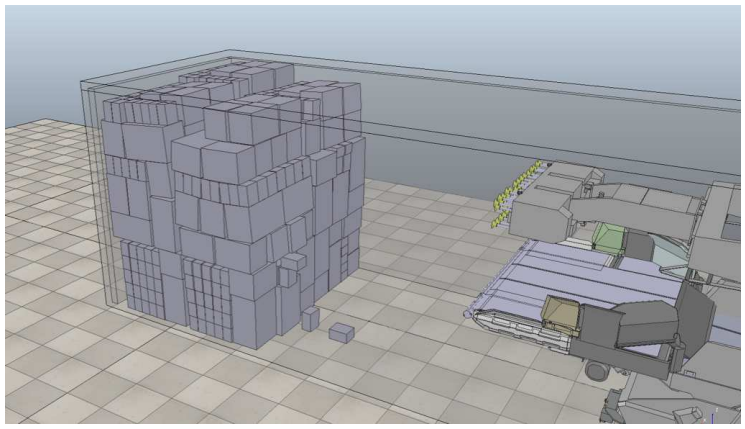
The implementation of the belief space planner for strategy generation is as described in Chapter 5 but with the planning horizon limited.



(a) Environment-A with 329 boxes in total.

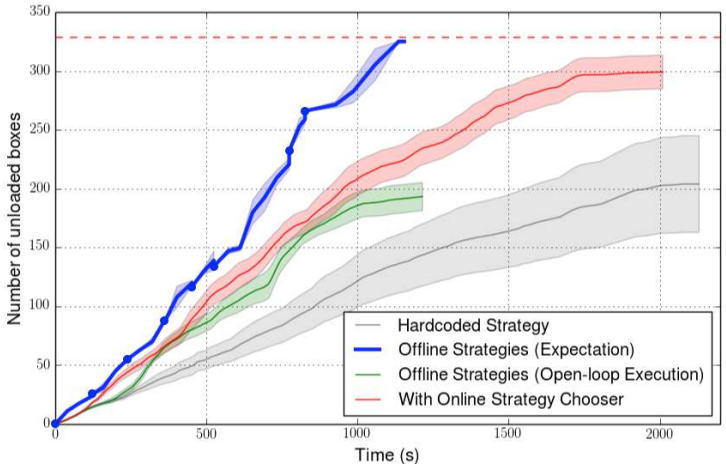


(b) Environment-B1 with 491 boxes in total.

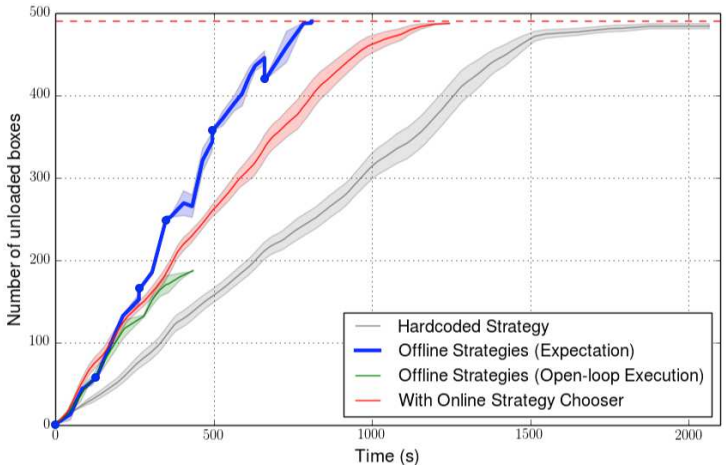


(c) Environment-B2 with 529 boxes in total.

Figure 7.6: Different environments in simulation for experimental validation.



(a) Environment-A.



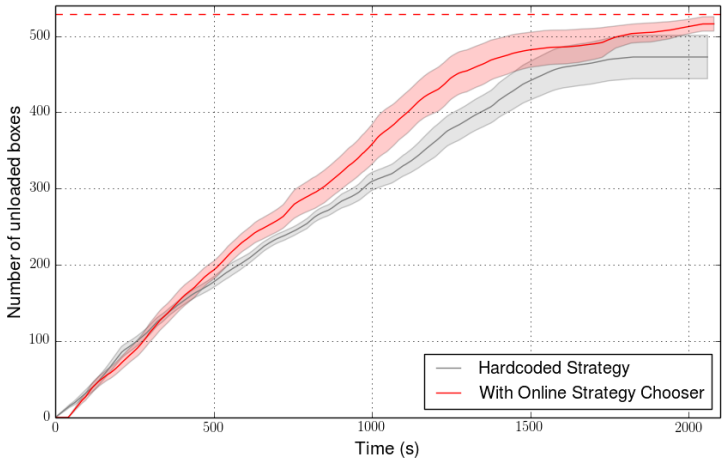
(b) Environment-B1.

Figure 7.7: The numbers of unloaded boxes in simulation for different methods. The red dashed line shows the total number of boxes in each environment. The online strategy chooser for each environment had access to the offline strategies for the corresponding environment.

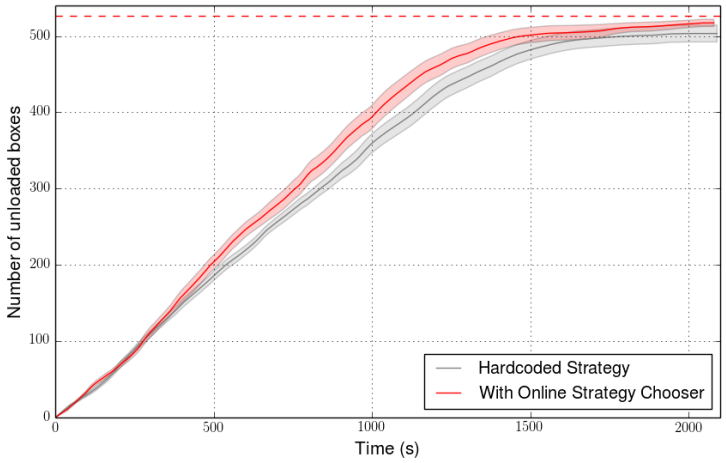
7.5 Experimental Results

In this section, we show results of the proposed PLR framework for full-truck unloading tasks in different environments. We collected the data from 10 runs of each method in each environment. Each run terminates if there is no more box in front of the robot or 10 consecutive macro actions fail (i.e., could not unload any boxes).

Some of environments are presented in Fig. 7.6. Environment-A contains nicely stacked 329 boxes with similar box shapes and masses that are easier to be picked up by the



(a) Environment-B2.



(b) Environment-B2–B6.

Figure 7.8: The number of unloaded boxes in simulation for different methods over five variant environments of Environment-B1. The red dashed line shows the total number of boxes over the variant environments. Note that the online strategy chooser had access to the offline strategies for Environment-B1 but not for its variant environments.

robot's suction pads. Environment-B1 contains complexly stacked 491 heterogeneous boxes with different sets of box shapes and masses, which makes the problem harder. Environment-B2 is a variation of Environment-B1 that contains 529 boxes in a different configuration. We used four more variation environments of Environment-B1 in the experiments.

The strategy generator used 5 particles ($N_{part} = 5$), a planning horizon of 6 ($N_{depth} = 6$), and the maximum number of iterations of 5 ($N_{iter} = 5$) in the experiments. The offline

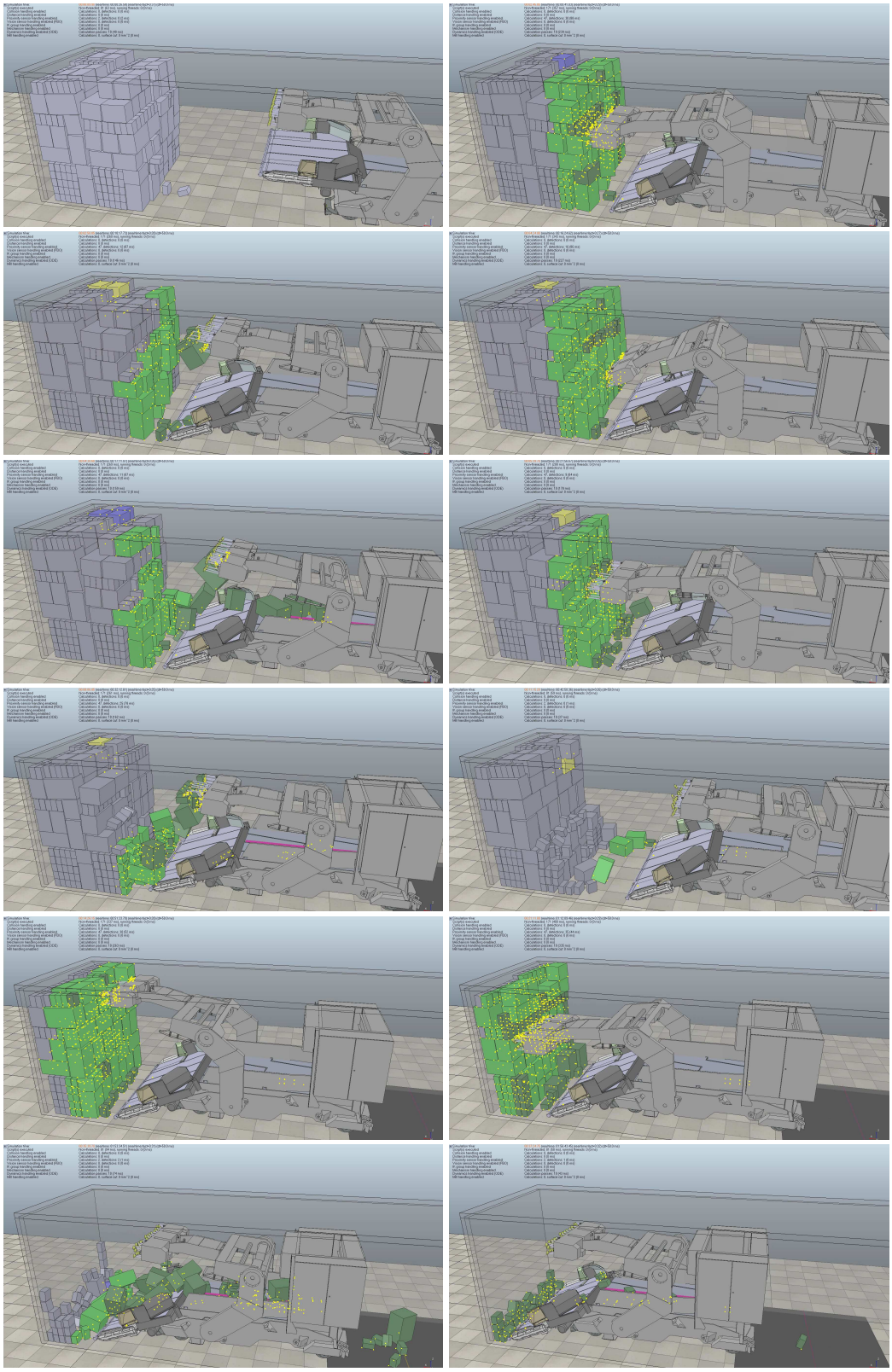


Figure 7.9: Screenshots of full truck unloading for Environment-B2.

strategies are generated for Environment-A and Environment-B1 and used by the online strategy chooser in Environment-A, B1, B2–6.

As for the baseline method, we used a hardcoded strategy that is designed, implemented, and validated by a human expert in the same simulation setting. It is fine tuned to provide sufficiently good performance in general, but due to the nature of hardcoding, it may perform poorly in a different set of environments.

The results of four approaches are presented here: the execution results of the hardcoded strategy, the expected performance of the offline strategies, the open-loop execution results of the offline strategies, and the execution results of the online strategy chooser with the offline strategies. 1) The hardcoded strategy is the above-mentioned decision tree that is manually written by a human expert. Its performance is measured by recurrently executing it (by sending it the strategy executor) until the termination condition is met (the robot reaches the end of the truck or falls into indefinite local minima). 2) The expected performance of the offline strategies can be understood the mostly-likely execution results of the offline strategies but with sporadic full state observations (i.e., removal of all state uncertainty) since the strategy generation is done by finite-horizon belief space planning. 3) The open-loop execution results of the offline strategies are obtained by blindly choosing (i.e., sending to the strategy executor) the offline strategies generated for the respective environments in the sequential order of generation. 4) The online strategy chooser is who has access to the offline strategy library and selects the best-fit strategy to execute at run time given the current observation. The selected strategy (decision tree) is being executed by the strategy executor to the end of its finite horizon, and then the strategy chooser is asked to select another strategy to be executed.

Fig. 7.7 shows the number of unloaded boxes over time in each environment where the offline strategies are generated. In Environment-A, both the expected offline strategies and the online strategy chooser outperformed the hardcoded strategy. The hardcoded strategy frequently executed ineffective actions by just following the hardcoded decision making scheme. Its performance can be good in some environments and can be very bad in other environments because it may fall into local minima and get stuck indefinitely.

Also note that the open-loop execution of the offline strategies are not as effective as the

expected offline strategies or the online strategy chooser. This demonstrates two things: 1) the full-truck unloading process is highly stochastic, so that the actual execution of the offline strategies does not follow the most-likely execution sequence, and thus, 2) it is advantageous to have the online strategy chooser that adapts to the stochastic outcome of the strategy execution in strategy selection at run time.

In Environment-B1, the open-loop execution terminates earlier than in Environment-A. This implies that strategy execution in Environment-B1 experiences higher stochasticity than in Environment-A. The hardcoded strategy is tuned to perform fairly well in Environment-B1, but the online strategy chooser still outperforms it. Compared to the expected (ideal) results of the offline strategies, the online strategy chooser could not achieve as good performance as the expected offline strategies. It would be because of the limited number of offline strategies in the strategy library as well as the incorrect selection of a strategy for the given the observations at run time. For these experiments the strategy chooser used k-Nearest Neighbor algorithm with depth map-based feature representation of the environment. In other words, it did not make use of the information of the box edges or dimensions. These results suggest the future research directions how to improve the generalization quality of the online strategy chooser.

In Fig. 7.8, we present the full-truck unloading results in variant environments for which no offline strategy is generated. The online strategy chooser has access only to the offline strategies for Environment-B1 in this case. While the variant environments have different numbers of boxes in different configurations from Environment-B1, the online strategy chooser could outperform the hardcoded strategy. This demonstrates the benefit of our PLR framework with the offline strategy generator and online strategy chooser in terms of generalization.

The overall procedure of full-truck unloading is presented in Fig. 7.9.

7.6 Summary and Discussion

We presented a hierarchical planning, learning and reasoning system architecture that automatically generate robust motion plans, so-called strategies, and incorporate them

with online adaption power at run time. Especially for strategy generation, we proposed receding horizon control scheme for belief space planning over macro action space. That makes the strategy generalization to be much less complicated and its execution to be more robust. The validation in simulation showed promising results of this approach and also suggested future research topics. We summarize the interesting and important directions for the further work.

Simulator speed A key bottleneck in the offline phase is simulating the outcome of different actions. The larger the number of simulation steps Sim can make per time unit, the more robust the strategies will be as the belief-space planner will be able to explore larger portions of the belief space for the same amount of planning times.

In our implementation, we used a state-of-the-art simulator (V-REP [97]) and initial planning times for strategy generation were in the order of days. This is mainly due to the fact that the simulator is required to run a physics engine testing interactions between all objects in a scene. A possible approach to speed up simulation times is to incorporate domain knowledge that allows to adaptively manage which interactions need to be simulated. For example, when using the scooper-like tool, there is no need to simulate the suction cups on the manipulator-like tool if they are not interacting with the environment. Similarly, interactions between boxes at the end of the trailer which are far from the robot can be set to be static.

Planning with adaptive simulation accuracy Another approach we suggest to handle the long planning times that the simulation introduces is by varying its *accuracy* in the planning algorithm. Planning with adaptive dimensionality has proven an effective tool to tackle complex high-dimensional planning problems (see, e.g., [98, 99, 100]. Here, we suggest to initial plan using the simulator configured to be fast but possibly inaccurate. Only for promising plans should the simulator be invoked with an accurate, yet slow configuration. This is similar to lazy path planning [101, 102, 103] where easy-to-compute estimates of edge weights are used to guide the search algorithm in cases where evaluating true edge weights is computationally expensive.

Active querying The crucial limitation of the strategy generalizer is the requirement to obtain large amounts of supervised data. With a high fidelity simulator, testing

each planned strategy in the library over a wide range of environments will take a prohibitively large amount of time. Also, while using the mapping trained in simulation on the real robot, we expect to encounter environments where further training is needed to predict the correct strategy. In such cases, resetting the environment in the real world (e.g., arranging boxes back to their previous positions) is extremely labor-intensive. Hence, there is a need for the system to query for optimal strategies only on a small number of environments where supervision is extremely useful (e.g., a completely new environment that was never seen before). This problem is commonly referred to as *active learning* in the literature. However, traditional active learning methods cannot be readily applied as the distribution of environments experienced by the system online is non-stationary and changes as the mapping is retrained.

Meta-algorithm for an ensemble of heuristics Given an ensemble of heuristics, there arise very interesting questions for the strategy generator; which heuristic to use to guide the search toward a goal, how much each heuristic should contribute when generating the solution strategy, etc.

When guiding the search, one can switch the reference heuristic in a round-robin fashion as in [96], but also can devise a meta-algorithm that determines in a smarter way which heuristic to use. Dynamic Thompson Sampling (DTS) and Meta-A* were suggested for deterministic planning problems [104, 105] and can possibly be extended to belief space planning for stochastic environments.

At the time the solution policy is to be generated, different heuristics may have different opinion on the best action. Then as in Weighted Majority algorithm [106], a weight for each heuristic can be computed based on its effectiveness in guidance and used to decide the best action for each state. Beyond these examples, there is ample opportunity to incorporate meta-learning techniques with belief space planners using multiple heuristics.

Chapter 8

Conclusion

8.1 Summary of Contributions

In this thesis, we presented a novel belief space planning framework, POMHDP, and its extensions that finds robust motion plans under motion and sensing uncertainty.

POMHDP takes advantage of the strong points of these three approaches: RTDP-Bel for belief space planning with domain knowledge, MHA* for systematic usage of multiple heuristics, and Monte Carlo methods for belief state/history approximation. In other words, POMHDP extends these approaches so that it exhibits better characteristics than them as follows:

1. RTDP-Bel vs. POMHDP: Generative model support, multi-heuristic guidance
   RTDP-Bel requires the access to explicit mathematical models, which restricts its practical usage in the real-world systems. POMHDP can works for both explicit and generative models, which allows it to be applied to real-world or complex systems. In terms of guidance of the forward search, RTDP-Bel depends on only one heuristic, so it may suffer from local minima where the search gets stuck. (Recall that it is often challenging to find a heuristic that captures all the complexity of the environment.) On the other hand, POMHDP makes use of multiple heuristics, so its guidance is more versatile when encountered local minima, and thus, provides

more effective guidance to reach the goal.

2. MHA* vs. POMHDP: Motion and sensing uncertainty
   MHA* is a planning algorithm for deterministic environments. It assumes that the outcome after executing an action is deterministic and the true state is always fully observable, which is not in the real-world systems. POMHDP extends MHA* to belief space planning for stochasitic environments, where the action and observation are not deterministic, so that it can accommodate more realistic problems in structured environments.

3. Monte Carlo methods vs. POMHDP: Sample-efficiency
   Monte Carlo methods use a set of sampled particles to represent a belief state, and operates particle filtering to sample a history, i.e., an action and observation sequence. Monte Carlo-based belief space planners, such as POMCP, do not bootstrap with the domain knowledge, and thus, require a large number of simulations until convergence to the optimal. POMHDP is a Dynamic Programming-based approach that bootstraps with the domain knowledge in the form of heuristics. Given (at least partly) informative multiple heuristics, POMHDP can find a solution with a fewer number of simulations, which is important especially for complex robotic systems with computationally expensive simulation models.

8.2 Future Directions

8.2.1 Finite horizon planning with sub-goal heuristics

For large problems that are not feasible to be solved as a whole, we have to conduct local finite-horizon planning to get approximate partial solutions, which will results in additional suboptimality in the global sense.

There can be several ways to improve the global solution quality. One is to get access to better heuristic estimates for the leaf nodes on the finite-horizon belief tree. This is the basic idea of online-offline combination approach in Section 6, but note that the approx-

imate global solver using local feedback controllers is only applicable to problems with controllable state space.

The other is to exploit the domain knowledge of sub-goals for the given task, where a sub-goal implies a condition that the final goal must satisfy at any case. For example of truck unloading, the final goal is to unload all boxes. Then, one can define a sub-goal such as unloading all boxes within 3 m in depth from the end of the truck. The main benefit of having sub-goals is that it will improve the quality of the local solution and the local solution is guaranteed to be a necessity condition to satisfy the final goal.

Since POMHDP can incorporate multiple heuristics, one can set up each heuristic to guide to each sub-goal without any change to the algorithm. The only change needed is to define the termination condition that takes into account these multiple sub-goals.

8.2.2 Learning observation space representation

Most of the Monte Carlo-based belief space planners, including POMHDP, require the discrete action and observation spaces. Disretized action space will lead to suboptimality in terms of the accumulated action costs to reach the goal, but will not hurt the robustness of execution at run time.

However, observation space discretization is how we interpret the outcome from the environment which is not under our control. Incorrect observation space discretization can lead to meaningless branching of the belief tree, which leads to suboptimal action selection during planning as well as at run time.

One approach to tackle this issue is to incorporate machine learning techniques. Once we get a large set of observations, we can cluster the observations into a finite discrete set of clusters, and then use them to define the discrete observation space.

8.2.3 Meta algorithm for multiple heuristics

Given an ensemble of heuristics, there arise very interesting questions for the strategy generator; which heuristic to use to guide the search toward a goal, how much each

heuristic should contribute when generating the solution strategy, etc.

When guiding the search, one can switch the reference heuristic in a round-robin fashion as in [96], but also can devise a meta-algorithm that determines in a smarter way which heuristic to use. Dynamic Thompson Sampling (DTS) and Meta-A* were suggested for deterministic planning problems [104, 105] and can possibly be extended to belief space planning for stochastic environments.

At the time the solution policy is to be generated, different heuristics may have different opinion on the best action. Then as in Weighted Majority algorithm [106], a weight for each heuristic can be computed based on its effectiveness in guidance and used to decide the best action for each state. Beyond these examples, there is ample opportunity to incorporate meta-learning techniques with belief space planners using multiple heuristics.

Bibliography

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

[2] Mykel J Kochenderfer. *Decision making under uncertainty: theory and application*. MIT press, 2015.

[3] C. Papadimitriou and J. N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[4] O. Madani, S. Hanks, and A. Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 541–548, 1999.

[5] Sandip Aine, Siddharth Swaminathan, Venkatraman Narayanan, Victor Hwang, and Maxim Likhachev. Multi-Heuristic A*. *International Journal of Robotics Research (IJRR)*, 35(1-3):224–243, 2016.

[6] Wee S Lee, Nan Rong, and David Hsu. What makes some POMDP problems easy to approximate? In *Advances in neural information processing systems*, pages 689–696, 2008.

[7] Jonathan Butzke, Krishna Sapkota, Kush Prasad, Brian MacAllister, and Maxim Likhachev. State lattice with controllers: Augmenting lattice-based path planning with controller-based motion primitives. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 258–265. IEEE, 2014.

[8] Ali-akbar Agha-mohammadi, Saurav Agarwal, Sung-Kyun Kim, Suman Chakravorty, and Nancy M. Amato. SLAP: Simultaneous localization and planning under uncertainty for physical mobile robots via dynamic replanning in belief space: Ex-

tended version. *CoRR*, abs/1510.07380, 2015.

[9] Ali-akbar Agha-mohammadi, Suman Chakravorty, and Nancy Amato. FIRM: Sampling-based feedback motion planning under motion uncertainty and imperfect measurements. *International Journal of Robotics Research (IJRR)*, 33(2):268–304, 2014.

[10] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[11] Ira Pohl. First results on the effect of error in heuristic search. *Machine Intelligence*, 5:219–236, 1970.

[12] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. In *Advances in neural information processing systems*, pages 767–774, 2004.

[13] Dimitri P. Bertsekas. *Dynamic programming and optimal control, 3rd Edition*. Athena Scientific, 2005.

[14] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT Press, 2005.

[15] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[16] Stéphane Ross, Joelle Pineau, Sébastien Paquet, and Brahim Chaib-draa. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 32:663–704, 2008.

[17] Edward J Sondik. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations research*, 26(2):282–304, 1978.

[18] Blai Bonet and Héctor Geffner. Solving POMDPs: RTDP-Bel vs. point-based algorithms. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1641–1646, 2009.

[19] Blai Bonet and Hector Geffner. Planning with incomplete information as heuristic

search in belief space. In *International Conference on Artificial Intelligence Planning Systems*, pages 52–61. AAAI Press, 2000.

[20] David E Smith and Daniel S Weld. Conformant graphplan. In *AAAI Innovative Applications of Artificial Intelligence Conference (IAAI)*, pages 889–896, 1998.

[21] Stephen M Majercik and Michael L Littman. Contingent planning under uncertainty via stochastic satisfiability. *Artificial Intelligence*, 147(1-2):119–162, 2003.

[22] Denise Draper, Steve Hanks, and Daniel S Weld. Probabilistic planning with information gathering and contingent execution. In *AIPS*, pages 31–36. Citeseer, 1994.

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

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

[25] Joelle Pineau, Geoffrey Gordon, and Sebastian Thrun. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research (JAIR)*, 27:335–380, 2006.

[26] Trey Smith and Reid Simmons. Heuristic search value iteration for POMDPs. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 520–527, 2004.

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

[28] Michael L Littman, Anthony R Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings 1995*, pages 362–370. Elsevier, 1995.

[29] Guy Shani, Joelle Pineau, and Robert Kaplow. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems*, 27(1):1–51, 2013.

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

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

[32] Eric A Hansen and Shlomo Zilberstein. Heuristic search in cyclic AND/OR graphs. In *AAAI Innovative Applications of Artificial Intelligence Conference (IAAI)*, pages 412–418, 1998.

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

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

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

[36] Blai Bonet and Héctor Geffner. Learning sorting and decision trees with POMDPs. In *International Conference on Machine Learning (ICML)*, pages 73–81. Citeseer, 1998.

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

[38] Héctor Geffner and Blai Bonet. Solving large POMDPs using Real-Time Dynamic Programming. In *AAAI Fall Symposium on POMDPs*. Citeseer, 1998.

[39] Richard E Korf. Real-time heuristic search. *Artificial intelligence*, 42(2-3):189–211, 1990.

[40] Andrew G Barto, Steven J Bradtke, and Satinder P Singh. Learning to act using real-time dynamic programming. *Artificial intelligence*, 72(1-2):81–138, 1995.

[41] Blai Bonet and Hector Geffner. Labeled RTDP: Improving the convergence of Real-Time Dynamic Programming. In *International Conference on Automated Planning and Scheduling (ICAPS)*, volume 3, pages 12–21, 2003.

[42] H Brendan McMahan, Maxim Likhachev, and Geoffrey J Gordon. Bounded Real-Time Dynamic Programming: RTDP with monotone upper bounds and perfor-

mance guarantees. In *International Conference on Machine Learning (ICML)*, pages 569–576. ACM, 2005.

[43] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, pages 2164–2172, 2010.

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

[45] Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *International Conference on Machine Learning (ICML)*, pages 273–280. ACM, 2007.

[46] Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. DESPOT: Online POMDP planning with regularization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1772–1780, 2013.

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

[48] Konstantin M Seiler, Hanna Kurniawati, and Surya PN Singh. An online and approximate solver for POMDPs with continuous action space. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2290–2297. IEEE, 2015.

[49] Zachary Sunberg and Mykel Kochenderfer. POMCPOW: An online algorithm for POMDPs with continuous state, action, and observation spaces. *arXiv:1709.06196*, 2017.

[50] Ali-akbar Agha-mohammadi, Suman Chakravorty, and Nancy Amato. Online replanning in belief space for dynamical systems: Towards handling discrete changes of goal location. In *IEEE International Conference On Robotics and Automation (ICRA): Workshop on Combining Task and Motion Planning*, Karlsruhe, Germany, 2013.

[51] Jur Van Den Berg, Sachin Patil, and Ron Alterovitz. Motion planning under uncertainty using iterative local optimization in belief space. *The International Journal of Robotics Research*, 31(11):1263–1278, 2012.

[52] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. *Technical Report 98-11, Computer Science Department, Iowa State University*, 1998.

[53] Juan Pablo Gonzalez and Anthony Stentz. Planning with uncertainty in position: an optimal and efficient planner. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2435–2442. IEEE, 2005.

[54] Juan Pablo Gonzalez and Anthony Stentz. Planning with uncertainty in position using high-resolution maps. In *2007 IEEE International Conference on Robotics and Automation*, pages 1015–1022. IEEE, 2007.

[55] Samuel Prentice and Nicholas Roy. The belief roadmap: Efficient planning in belief space by factoring the covariance. *International Journal of Robotics Research*, 28(11-12):1448–1465, 2009.

[56] Adam Bry and Nicholas Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 723–730. IEEE, 2011.

[57] David Lenz, Markus Rickert, and Alois Knoll. Heuristic search in belief space for motion planning under uncertainties. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2659–2665. IEEE, 2015.

[58] Kenneth Y Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10(2-4):201–225, 1993.

[59] Srinivas Akella and Matthew T Mason. Parts orienting with partial sensor information. In *1998 IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 557–564. IEEE, 1998.

[60] Nikhil Chavan Dafle, Alberto Rodriguez, Robert Paolini, Bowei Tang, Siddhartha S Srinivasa, Michael Erdmann, Matthew T Mason, Ivan Lundberg, Harald Staab, and Thomas Fuhlbrigge. Extrinsic dexterity: In-hand manipulation with external forces. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1578–1585. IEEE, 2014.

[61] Clemens Eppner, Raphael Deimel, Jos Álvarez-Ruiz, Marianne Maertens, Oliver Brock, et al. Exploitation of environmental constraints in human and robotic grasp-

ing. *International Journal of Robotics Research*, 34(7):1021–1038, 2015.

[62] Jiaji Zhou, Robert Paolini, Aaron M Johnson, J Andrew Bagnell, and Matthew T Mason. A probabilistic planning framework for planar grasping under uncertainty. *IEEE Robotics and Automation Letters*, 2(4):2111–2118, 2017.

[63] Venkatraman Narayanan and Maxim Likhachev. Task-oriented planning for manipulating articulated mechanisms under model uncertainty. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3095–3101. IEEE, 2015.

[64] Joshua A Haustein, Jennifer King, Siddhartha S Srinivasa, and Tamim Asfour. Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3075–3082. IEEE, 2015.

[65] Michael C Koval, Nancy S Pollard, and Siddhartha S Srinivasa. Pre-and post-contact policy decomposition for planar contact manipulation under uncertainty. *The International Journal of Robotics Research*, 35(1-3):244–264, 2016.

[66] Sergey Levine, Nolan Wagener, and Pieter Abbeel. Learning contact-rich manipulation skills with guided policy search. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 156–163. IEEE, 2015.

[67] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[68] Sung-Kyun Kim and Maxim Likhachev. Parts assembly planning under uncertainty with simulation-aided physical reasoning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 4074–4081. IEEE, 2017.

[69] *Coppelia Robotics, V-REP (Virtual Robot Experimentation Platform). [Online]. Available: http://www.coppeliarobotics.com*, 2018.

[70] Fahad Islam, Venkatraman Narayanan, and Maxim Likhachev. Dynamic Multi-Heuristic A*. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2376–2382. IEEE, 2015.

[71] Mike Phillips, Benjamin J Cohen, Sachin Chitta, and Maxim Likhachev. E-Graphs: Bootstrapping planning with experience graphs. In *Robotics: Science and Systems*, 2012.

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

[73] Dan Halperin, Oren Salzman, and Micha Sharir. Algorithmic motion planning. In *Handbook of Discrete and Computational Geometry, Third Edition*, pages 1311–1342. CRC Press, 2017.

[74] Sung-Kyun Kim and Maxim Likhachev. Parts assembly planning under uncertainty with simulation-aided physical reasoning. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 4074–4081, 2017.

[75] Vinitha Ranganeni, Oren Salzman, and Maxim Likhachev. Effective footstep planning for humanoids using homotopy-class guidance. In *Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 500–508, 2018.

[76] Fahad Islam, Oren Salzman, and Maxim Likhachev. Online, interactive user guidance for high-dimensional, constrained motion planning. In *Int. Joint Conference on Artificial Intelligence (IJCAI)*, pages 4921–4928, 2018.

[77] Austin Dionne, Jordan Thayer, and Wheeler Ruml. Deadline-aware search using on-line measures of behavior. In *Symposium on Combinatorial Search (SoCS)*, pages 39–46, 2011.

[78] Judea Pearl. Heuristics: intelligent search strategies for computer problem solving. 1984.

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

[80] Partha P Chakrabarti, Sujoy Ghose, and SC DeSarkar. Admissibility of AO* when heuristics overestimate. *Artificial Intelligence*, 34(1):97–113, 1987.

[81] Sylvain Gelly and David Silver. Monte-Carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

[82] Hanna Kurniawati and Vinay Yadav. An online POMDP solver for uncertainty planning in dynamic environment. In *Robotics Research*, pages 611–629. Springer,

2016.

[83] Sam Prentice and Nicholas Roy. The belief roadmap: Efficient planning in belief space by factoring the covariance. *International Journal of Robotics Research*, 28(11-12), October 2009.

[84] Andrzej Ruszczyński. Risk-averse dynamic programming for Markov decision processes. *Mathematical programming*, 125(2):235–261, 2010.

[85] Anirudha Majumdar and Marco Pavone. How should a robot assess risk? Towards an axiomatic theory of risk in robotics. *arXiv preprint arXiv:1710.11040*, 2017.

[86] Yu Tao, Jan-Peter Muller, and William Poole. Automated localisation of mars rovers using co-registered hirise-ctx-hrsc orthorectified images and wide baseline navcam orthorectified mosaics. *Icarus*, 280:139–157, 2016.

[87] Bob Balaram, Timothy Canham, Courtney Duncan, Håvard F Grip, Wayne Johnson, Justin Maki, Amelia Quon, Ryan Stern, and David Zhu. Mars helicopter technology demonstrator. In *2018 AIAA Atmospheric Flight Mechanics Conference*, page 0023, 2018.

[88] Tamás Kalmár-Nagy, Raffaello D'Andrea, and Pritam Ganguly. Near-optimal dynamics trajectory generation and control of an omnidirectional vehicle. *Robotics and Autonomous Systems*, 46(1):47–64, January 2004.

[89] Ali-akbar Agha-mohammadi, Saurav Agarwal, Aditya Mahadevan, Suman Chakravorty, Daniel Tomkins, Jory Denny, and Nancy Amato. Robust online belief space planning in changing environments: Application to physical mobile robots. In *IEEE Int. Conf. Robot. Autom. (ICRA)*, Hong Kong, China, 2014.

[90] Steven M. LaValle and James J. Kuffner Jr. Randomized kinodynamic planning. *Int. J. Robotics Res.*, 20(5):378–400, 2001.

[91] Maxim Likhachev, Geoffrey Gordon, and Sebastian Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems*, pages 767–774, 2004.

[92] Benjamin J. Cohen, Sachin Chitta, and Maxim Likhachev. Search-based planning for manipulation with motion primitives. *IEEE Int. Conf. on Robotics and Automation*

(ICRA), pages 2902–2908, 2010.

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

[94] Mykel Kochenderfer. *Decision making under uncertainty: theory and application*. MIT press, 2015.

[95] David Silver and Joel Veness. Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems*, pages 2164–2172, 2010.

[96] Sung-Kyun Kim, Oren Salzman, and Maxim Likhachev. POMHDP: Search-based belief space planning using multiple heuristics. *CoRR*, abs/XXXX.YYYY, 2019.

[97] E. Rohmer, S. Singh, and M. Freese. V-REP: a versatile and scalable robot simulation framework. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 1321–1326, 2013.

[98] Kalin Gochev, Alla Safonova, and Maxim Likhachev. Planning with adaptive dimensionality for mobile manipulation. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 2944–2951, 2012.

[99] Kalin Gochev, Alla Safonova, and Maxim Likhachev. Incremental planning with adaptive dimensionality. In *Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2013.

[100] Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015.

[101] Christopher M. Dellin and Siddhartha S. Srinivasa. A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. In *Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 459–467, 2016.

[102] Nika Haghtalab, Simon Mackenzie, Ariel D. Procaccia, Oren Salzman, and Siddhartha S. Srinivasa. The provable virtue of laziness in motion planning. In *Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 106–113, 2018.

[103] Aditya Mandalika, Oren Salzman, and Siddhartha Srinivasa. Lazy receding horizon A* for efficient path planning in graphs with expensive-to-evaluate edges. In

Int. Conf. on Automated Planning and Scheduling (ICAPS), pages 476–484, 2018.

[104] Mike Phillips, Venkatraman Narayanan, Sandip Aine, and Maxim Likhachev. Efficient search with an ensemble of heuristics. In *Int. Joint Conference on Artificial Intelligence (IJCAI)*, pages 784–791, 2015.

[105] Sandip Aine and Maxim Likhachev. Search portfolio with sharing. In *Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 11–19, 2016.

[106] Nick Littlestone and Manfred K Warmuth. The weighted majority algorithm. *Information and computation*, 108(2):212–261, 1994.