16 *Approximate POMDP Techniques*

16.1 Motivation

In previous chapters, we have studied two main frameworks for action selection under uncertainty: MDPs and POMDPs. Both frameworks address non-deterministic action outcomes, but they differ in their ability to accommodate sensor limitations. Only the POMDP algorithms can cope with uncertainty in perception, whereas MDP algorithms assume that the state is fully observable. However, the computational expense of exact planning in POMDPs renders exact methods inapplicable to just about any practical problem in robotics.

This chapter describes POMDP algorithms that scale. As we shall see in this chapter, both MDP and POMDP are extreme ends of a spectrum of possible probabilistic planning and control algorithms. This chapter reviews a number of approximate POMDP techniques that fall in between MDPs and POMDPs. The algorithms discussed here share with POMDPs the use of value iteration in belief space. However, they approximate the value function in a number of ways. By doing so, they gain immense speed-ups over the full POMDP solution.

The techniques surveyed in this chapter have been chosen because they characterize different styles of approximation. In particular, we will discuss the following three algorithms:

- *QMDP* is a hybrid between MDPs and POMDPs. This algorithm generalizes the MDP-optimal value function defined over states, into a POMDP-style value function over beliefs. QMDP would be exact under the—usually false—assumption that after one step of control, the state becomes fully observable. Value iteration in QMDPs is of the same complexity as in MDPs.

```
1:  Algorithm QMDP($b = (p_1, \ldots, p_N)$):
2:      $\hat{V}$ = MDP_discrete_value_iteration() // see page 502
3:      for all control actions $u$ do
4:          $Q(x_i, u) = r(x_i, u) + \sum_{j=1}^{N} \hat{V}(x_j)\, p(x_j \mid u, x_i)$
5:      endfor
6:      return $\operatorname{argmax}_u \sum_{i=1}^{N} p_i\, Q(x_i, u)$
```

Table 16.1 The QMDP algorithm computes the expected return for each control action u, and then selects the action u that yields the highest value. The value function used here is MDP-optimal, hence dismisses the state uncertainty in the POMDP.

- The *augmented MDP*, or *AMDP*. This algorithm projects the belief state into a low-dimensional sufficient statistic and performs value iteration in this lower-dimensional space. The most basic implementation involves a representation that combines the most likely state and the degree of uncertainty, measured by entropy. The planning is therefore only marginally less efficient than planning in MDPs, but the result can be quite an improvement!
- *Monte Carlo POMDP*, or *MC-POMDP*. This is the particle filter version of the POMDP algorithm, where beliefs are approximated using particles. By constructing a belief point set dynamically—just like the PBVI algorithm described towards the end of the previous chapter—MC-POMDPs can maintain a relatively small set of beliefs. MC-POMDPs are applicable to continuous-valued states, actions, and measurements, but they are subject to the same approximations that we have encountered in all particle filter applications in this book, plus some additional ones that are unique to MC-POMDPs.

These algorithms cover some of the primary techniques for approximating value functions in the emerging literature on probabilistic planning and control.

16.2 QMDPs

QMDPs are an attempt to combine the best of MDPs and POMDPs. Value functions are easier to compute for MDPs than for POMDPs, but MDPs rely on the assumption that the state is fully observable. A QMDP is computationally just about as efficient as an MDP, but it returns a policy that is defined over the belief state.

The mathematical "trick" is relatively straightforward. The MDP algorithm discussed in Chapter 14 provides us with a state-based value function that is optimal under the assumption that the state is fully observable. The resulting value function $\hat{V}$ is defined over world states. The QMDP generalizes this value to the belief space through the mathematical expectation:

$$\hat{V}(b) \quad = \quad E_x[\hat{V}(x)] \quad = \quad \sum_{i=1}^{N} p_i \, \hat{V}(x_i) \tag{16.1}$$

Here we use our familiar notation $p_i = b(x_i)$. Thus, this value function is linear, with the parameters

$$u_i \quad = \quad \hat{V}(x_i) \tag{16.2}$$

This linear function is exactly of the form used by the POMDP value iteration algorithm. Hence the value function over the belief space is given by the following linear equation:

$$\hat{V}(b) \quad = \quad \sum_{i=1}^{N} p_i \, u_i \tag{16.3}$$

The MDP value function provides a *single* linear constraint in belief space. This enables us to apply the algorithm **policy_POMDP** in Table 15.2, with a single linear constraint.

The most basic version of this idea leads to the algorithm **QMDP** shown in Table 16.1. Here we use a slightly different notation than in Table 15.2: Instead of caching away one linear function for each action u and letting **policy_POMDP** determine the action, our formulation of **QMDP** directly computes the optimal value function through a function Q. The value of $Q(x_i, u)$, as calculated in line 4 in Table 16.1, is the MDP-value of the control u in state x_i. The generalization to belief states then follows in line 6, where the expectation is taken over the belief state. Line 6 also maximizes over all actions, and returns the control action with the highest expected value.

The insight that the MDP-optimal value function can be generalized to belief space enables us to arbitrarily combine MDP and POMDP backups.

In particular, the MDP-optimal value function $\hat{V}$ can be used as input to the POMDP algorithm in Table 15.1 (page 529). With T further POMDP backups, the resulting policy can actively engage in information gathering—as long as the information shows utility within the next T time steps. Even for very small values of T, we usually obtain a robust probabilistic control algorithm that is computationally vastly superior to the full POMDP solution.

16.3 Augmented Markov Decision Processes

16.3.1 The Augmented State Space

The augmented MDP, or *AMDP*, is an alternative to the QMDP algorithm. It too approximates the full POMDP value function. However, instead of ignoring state uncertainty beyond a small time horizon T, the AMDP compresses the belief state into a more compact representation, and then performs full POMDP-style probabilistic backups.

The fundamental assumption in AMDPs is that the belief space can be summarized by a lower-dimensional "sufficient" statistic f, which maps belief distributions into a lower dimensional space. Values and actions are calculated from this statistic $f(b)$ instead of the original belief b. The more compact the statistic, the more efficient the resulting value iteration algorithm.

In many situations a good choice of the statistic is the tuple

$$\bar{b} = \begin{pmatrix} \underset{x}{\text{argmax}}\; b(x) \\ H_b(x) \end{pmatrix} \tag{16.4}$$

Here $\text{argmax}_x\; b(x)$ is the most likely state under the belief distribution b, and

$$H_b(x) = -\int b(x) \log b(x)\, dx \tag{16.5}$$

AUGMENTED STATE SPACE

is the *entropy* of the belief. This space will be called the *augmented state space*, since it augments the state space by a single value, the entropy of the belief distribution. Calculating a value function over the augmented state space, instead of the belief space, makes for a huge change in complexity. The augmented state avoids the high dimensionality of the belief space, which leads to enormous savings when computing a value function (from worst case doubly exponential to low-degree polynomial).

The augmented state representation is mathematically justified if $f(b)$ is a sufficient statistic of b with regards to the estimation of value:

$$V(b) \quad = \quad V(f(b)) \tag{16.6}$$

for all beliefs b the robot may encounter. In practice, this assumption will rarely hold true. However, the resulting value function might still be good enough for a sensible choice of control.

Alternatively, one might consider different statistics, such as the moments of the belief distribution (mean, variance, ...), the eigenvalues and vectors of the covariance, and so on.

16.3.2 The AMDP Algorithm

The AMDP algorithm performs value iteration in the augmented state space. To do so, we have to overcome two obstacles. First, the exact value update is non-linear for our augmented state representation. This is because the entropy is a non-linear function of the belief parameters. It therefore becomes necessary to approximate the value backup. AMDPs discretize the augmented state, representing the value function $\hat{V}$ by a look-up table. We already encountered such an approximation when discussing MDPs. In AMDPs, this table is one dimension larger than the state space table used by MDPs.

The second obstacle pertains to the transition probabilities and the payoff function in the augmented state space. We are normally given probabilities such as the motion model $p(x' \mid u, x)$, the measurement model $p(z \mid x)$, and the payoff function $r(x, u)$. But for value iteration in the augmented state space we need to define similar functions over the augmented state space.

AMDPs use a "trick" for constructing the necessary functions. The trick is to learn transition probabilities and payoffs from simulations. The learning algorithm is based on a frequency statistic, which counts how often an augmented belief $\bar{b}$ transitions to another belief $\bar{b}'$ under a control u, and what average payoff this transition induces.

Table 16.2 states the basic algorithm **AMDP_value_iteration**. The algorithm breaks down into two phases. In a first phase (lines 2–19), it constructs a transition probability table $\hat{\mathcal{P}}$ for the transition from an augmented state $\bar{b}$ and a control action u to a possible subsequent augmented state $\bar{b}'$. It also constructs a payoff function $\hat{\mathcal{R}}$ which measures the expected immediate payoff r when u is chosen in the augmented state $\bar{b}$.

These functions are estimated through a sampling procedure, in which we generate n samples for each combination of $\bar{b}$ and u (line 8). For each of

```
1:  Algorithm AMDP_value_iteration( ):
2:      for all $\bar{b}$ do                                   // learn model
3:          for all $u$ do
4:              for all $\bar{b}$ do                           // initialize model
5:                  $\hat{\mathcal{P}}(\bar{b}, u, \bar{b}') = 0$
6:              endfor
7:              $\hat{\mathcal{R}}(\bar{b}, u) = 0$
8:              repeat $n$ times                               // learn model
9:                  generate $b$ with $f(b) = \bar{b}$
10:                 sample $x \sim b(x)$                       // belief sampling
11:                 sample $x' \sim p(x' \mid u, x)$           // motion model
12:                 sample $z \sim p(z \mid x')$               // measurement model
13:                 calculate $b' = B(b, u, z)$                // Bayes filter
14:                 calculate $\bar{b}' = f(b')$               // belief state statistic
15:                 $\hat{\mathcal{P}}(\bar{b}, u, \bar{b}') = \hat{\mathcal{P}}(\bar{b}, u, \bar{b}') + \frac{1}{n}$   // learn transitions prob's
16:                 $\hat{\mathcal{R}}(\bar{b}, u) = \hat{\mathcal{R}}(\bar{b}, u) + \frac{r(u,s)}{n}$   // learn payoff model
17:             endrepeat
18:         endfor
19:     endfor
20:     for all $\bar{b}$                                      // initialize value function
21:         $\hat{V}(\bar{b}) = r_{\min}$
22:     endfor
23:     repeat until convergence                               // value iteration
24:         for all $\bar{b}$ do
25:             $\hat{V}(\bar{b}) = \gamma \max_u \left[ \hat{\mathcal{R}}(u, \bar{b}) + \sum_{\bar{b}'} \hat{V}(\bar{b}') \, \hat{\mathcal{P}}(\bar{b}, u, \bar{b}') \right]$
26:     endfor
27:     return $\hat{V}, \hat{\mathcal{P}}, \hat{\mathcal{R}}$                     // return value fct & model
```

```
1:  Algorithm policy_AMDP($\hat{V}, \hat{\mathcal{P}}, \hat{\mathcal{R}}, b$):
2:      $\bar{b} = f(b)$
3:      return $\operatorname{argmax}_u \left[ \hat{\mathcal{R}}(u, \bar{b}) + \sum_{\bar{b}'} \hat{V}(\bar{b}') \, \hat{\mathcal{P}}(\bar{b}, u, \bar{b}') \right]$
```

Table 16.2 Top: The value iteration algorithm for augmented MDPs. Bottom: The algorithm for selecting a control action.

these Monte Carlo simulations, the algorithm first generates a belief b for which $f(b) = \bar{b}$. This step is tricky (in fact, it is ill-defined): In the original AMDP model, the creators simply choose to set b to a symmetric Gaussian with parameters chosen to match $\hat{b}$. Next, the AMDP algorithm samples a pose x, a successor pose x', and a measurement z, all in the obvious ways. It then applies the Bayes filter to generate a posterior belief $B(b, u, z)$, for which it calculates the augmented statistics (line 14). The tables $\hat{\mathcal{P}}$ and $\hat{\mathcal{R}}$ are then updated in lines 15 and 16, using simple frequency counts weighted (in the case of the payoff) with the actual payoff for this Monte Carlo sample.

Once the learning is completed, AMDP continues with value iteration. This is implemented in lines 20-26. As usual, the value function is initialized by a large negative value. Iteration of the backup equation in line 25 leads to a value function defined over the augmented state space.

When using AMDPs, the state tracking usually takes place over the original belief space. For example, when using AMDPs for robot motion, one might use MCL for tracking the belief over the robot's pose. The algorithm **policy_AMDP** in Table 16.2 shows how to extract a policy action from the AMDP value function. It extracts the augmented state representation from the full belief in line 2, and then simply chooses the control action that maximizes the expected value (line 3).

16.3.3 Mathematical Derivation of AMDPs

The derivation of AMDP is relatively straightforward, under the assumption that f is a sufficient statistic of the belief state b; i.e., the world is *Markov* relative to the state $f(b)$. We start with an appropriate modification of the standard POMDP-style backup in Equation (15.2). Let f be the function that extracts the statistic $\bar{b}$ from b, hence $\bar{b} = f(b)$ for arbitrary beliefs b. Assuming that f is a sufficient statistic, the POMDP value iteration equation (15.2) can be defined over the AMDP state space

$$V_T(\bar{b}) = \gamma \max_u \left[r(\bar{b}, u) + \int V_{T-1}(\bar{b}')\, p(\bar{b}' \mid u, \bar{b})\, d\bar{b}' \right] \tag{16.7}$$

where $\bar{b}$ refers to the low-dimensional statistic of b defined in (16.4). Here $V_{T-1}(\bar{b}')$ and $V_T(\bar{b})$ are realized through look-up tables.

This equation contains the probability $p(\bar{b}' \mid u, \bar{b})$, which needs further explanation. Specifically, we have

$$p(\bar{b}' \mid u, \bar{b}) = \int p(\bar{b}' \mid u, b)\, p(b \mid \bar{b})\, db \tag{16.8}$$

$$
\begin{aligned}
&= \int\int p(\bar{b}' \mid z, u, b)\, p(z \mid b)\, p(b \mid \bar{b})\, dz\, db \\
&= \int\int p(\bar{b}' = f(B(b,u,z))\, p(z \mid b)\, p(b \mid \bar{b})\, dz\, db \\
&= \int\int p(\bar{b}' = f(B(b,u,z)) \int p(z \mid x') \\
&\qquad \int p(x' \mid u, x)\, b(x)\, dx\, dx'\, dz\;\; p(b \mid \bar{b})\, db
\end{aligned}
$$

This transformation exploited the fact that the posterior belief b' is uniquely determined once we know the prior belief b, the control u, and the measurement z. This same "trick" was already exploited in our derivation of POMDPs. It enabled us to replace the distribution over posterior beliefs by the Bayes filter result $B(b,u,z)$. In the augmented state space, we therefore could replace $p(\bar{b}' \mid z, u, b)$ by a point-mass distribution centered on $f(B(b,u,z))$.

Our learning algorithm in Table 16.2 approximates this equation through Monte Carlo sampling. It replaced each of the integrals by a sampler. The reader should take a moment to establish the correspondence: Each of the nested integrals in (16.8) maps directly to one of the sampling steps in Table 16.2.

Along the same lines, we can derive an expression for the expected payoff $r(\bar{b}, u)$:

$$
\begin{aligned}
(16.9)\qquad r(\bar{b}, u) &= \int r(b,u)\, p(b \mid \bar{b})\, db \\
&= \int\int r(x,u)\, b(x)\, dx\, p(b \mid \bar{b})\, db
\end{aligned}
$$

Once again, the algorithm **AMDP_value_iteration** approximates this integral using Monte Carlo sampling. The resulting learned payoff function resides in the lookup $\hat{\mathcal{R}}$. The value iteration backup in lines 20–26 of **AMDP_value_iteration** is essentially identical to the derivation of MDPs.

As noted above, this Monte Carlo approximation is only legitimate when $\bar{b}$ is a sufficient statistics of b and the system is Markov with respect to $\bar{b}$. In practice, this is usually *not* the case, and proper sampling if augmented states would therefore have to be conditioned on past actions and measurements—a nightmare! The AMDP algorithm ignores this, by simply generating b's with $f(b) = \bar{b}$. Our example above involved a symmetric Gaussian whose parameters matched $\bar{b}$. An alternative—which deviates from the original AMDP algorithm but would be mathematically more sound—would involve

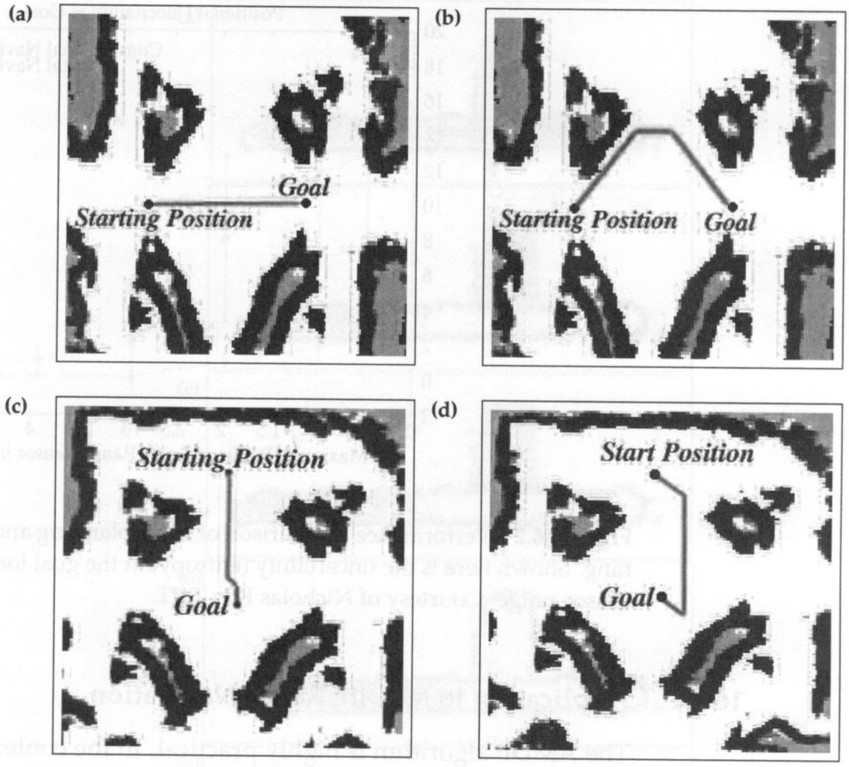


Figure 16.1 Examples of robot paths in a large, open environment, for two different configurations (top row and bottom row). The diagrams (a) and (c) show paths generated by a conventional dynamic programming path planner that ignores the robot's perceptual uncertainty. The diagrams (b) and (d) are obtained using the augmented MDP planner, which anticipates uncertainty and avoids regions where the robot is more likely to get lost. Courtesy of Nicholas Roy, MIT.

simulation of entire traces of belief states using the motion and measurement models, and using subsequent pairs of simulated belief states to learn $\hat{\mathcal{P}}$ and $\hat{\mathcal{R}}$. Below, when discussing MC-POMDPs, we will encounter such a technique. MC-POMDPs sidestep this issue by using simulation to generate plausible pairs of belief states.

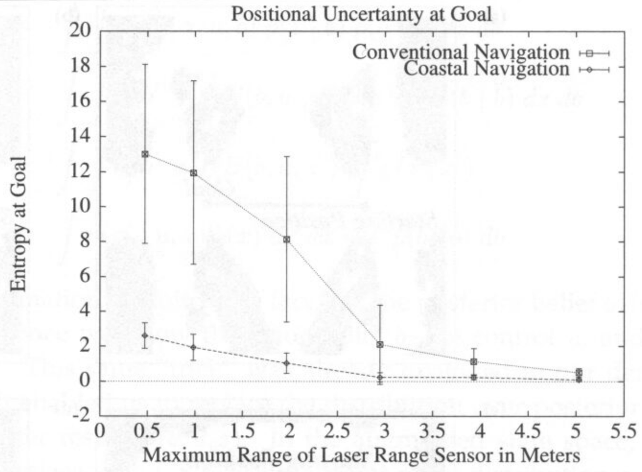


Figure 16.2 Performance comparison of MDP planning and Augmented MDP planning. Shown here is the uncertainty (entropy) at the goal location as a function of the sensor range. Courtesy of Nicholas Roy, MIT.

16.3.4 Application to Mobile Robot Navigation

The AMDP algorithm is highly practical. In the context of mobile robot navigation, AMDPs enable a robot to consider its general level of "confusion" in its action choice. This pertains not just to the momentary uncertainty, but also future expected uncertainties a robot may experience through the choice of its actions.

Our example involves a robot navigating in a known environment. It was already given as an example in the introduction to this book; see Figure 1.2 on page 7. Clearly, the level of confusion depends on where the robot navigates. A robot traversing a large featureless area is likely to gradually lose information as to where it is. This is reflected in the conditional probability $p(\bar{b}' \mid u, \bar{b})$, which with high likelihood increases the entropy of the belief in such areas. In areas populated with localization features, e.g., near the walls with distinctive features, the uncertainty is more likely to decrease. The AMDP anticipates such situations and generates policies that minimize the time of arrival while simultaneously maximizing the certainty at the time of arrival at a goal location. Since the uncertainty is an estimate of the true positioning error, it is a good measure for the chances of actually arriving at the desired location.

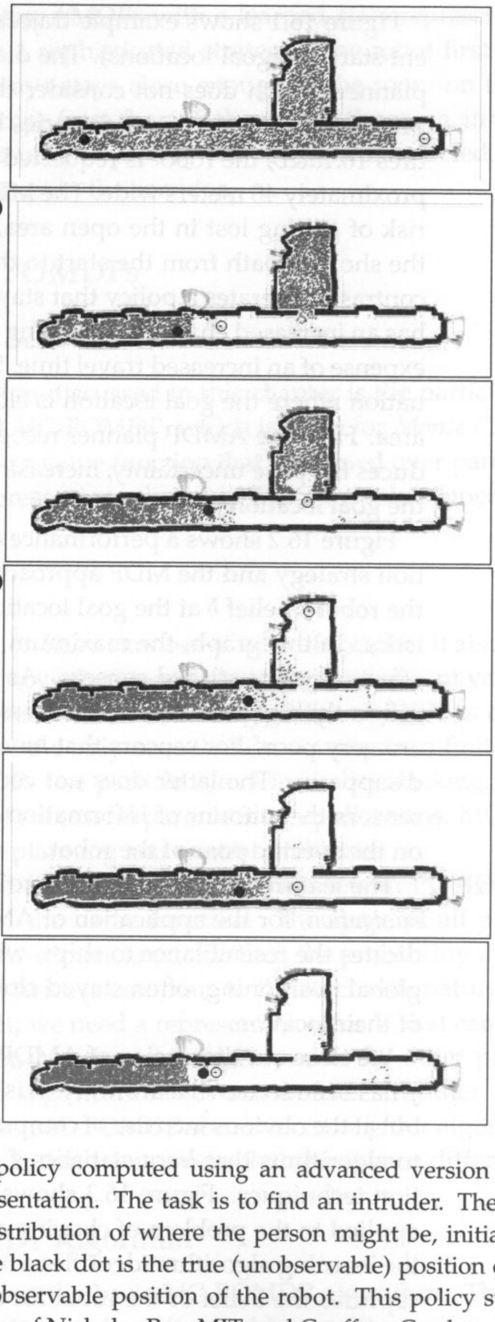


Figure 16.3 The policy computed using an advanced version of AMDP, with a learned state representation. The task is to find an intruder. The gray particles are drawn from the distribution of where the person might be, initially uniformly distributed in (a). The black dot is the true (unobservable) position of the person. The open circle is the observable position of the robot. This policy succeeds with high likelihood. Courtesy of Nicholas Roy, MIT, and Geoffrey Gordon, CMU.

Figure 16.1 shows example trajectories for two constellations (two different start and goal locations). The diagrams on the left correspond to a MDP planner, which does not consider the robot's uncertainty. The augmented MDP planner generates trajectories like the ones shown on the right. In Figures 16.1a&b, the robot is requested to move through a large open area, approximately 40 meters wide. The MDP algorithm, not aware of the increased risk of getting lost in the open area, generates a policy that corresponds to the shortest path from the start to the goal location. The AMDP planner, in contrast, generates a policy that stays close to the obstacles, where the robot has an increased chance of receiving informative sensor measurements at the expense of an increased travel time. Similarly, Figure 16.1c&d considers a situation where the goal location is close to the center of the featureless, open area. Here the AMDP planner recognizes that passing by known objects reduces the pose uncertainty, increasing the chances of successfully arriving at the goal location.

Figure 16.2 shows a performance comparison between the AMDP navigation strategy and the MDP approach. In particular, it depicts the entropy of the robot's belief b at the goal location, as a function of the sensor characteristics. In this graph, the maximum perceptual range is varied, to study the effect of impoverished sensors. As the graph suggests, the AMDP has significantly higher chances of success. The difference is largest if the sensors are very poor. For sensors that have a long range, the difference ultimately disappears. The latter does not come as a surprise, since with good range sensors the amount of information that can be perceived is less dependent on the specific pose of the robot.

COASTAL NAVIGATION

The feature to anticipate and avoid uncertainty has led to the name *coastal navigation*, for the application of AMDPs to robot navigation. This name indicates the resemblance to ships, which, before the advent of satellite-based global positioning, often stayed close to the coastline so as to not lose track of their location.

We close our discussion of AMDPs by noting that the choice of the statistic f has been somewhat arbitrary. It is possible to add more features as needed, but at the obvious increase of computational complexity. Recent work has led to algorithms that *learn* statistics f, using non-linear dimensionality reduction techniques. Figure 16.3 shows the result of such a learning algorithm, applied to the problem of clearing a building from a moving intruder. Here the learning algorithm identifies a 6-dimensional state representation, which captures the belief of the robot for any plausible pursuit strategy. The gray particles represent the robot's belief about the intruder's location. As this example illustrates, AMDPs with a learned state representation succeed in generating quite a sophisticated strategy: The robot first clears part of the corridor, but always stays close enough to the room on the top that an intruder cannot escape from there. It then clears the room, in time short enough to prevent an intruder to pass by the corridor undetected. The robot finally continues its pursuit in the corridor.

16.4 Monte Carlo POMDPs

16.4.1 Using Particle Sets

The final algorithm discussed in this chapter is the particle filter solution to POMDPs, called *MC-POMDP*, which is short for *Monte Carlo POMDP*. MC-POMDPs acquire a value function that is defined over particle sets. Let $\mathcal{X}$ be a particle set representing a belief b. Then the value function is represented as a function

$$V : \mathcal{X} \longrightarrow \Re \tag{16.10}$$

This representation has a number of advantages, but it also creates a number of difficulties. A key advantage is that we can represent value functions over arbitrary state spaces. In fact, among all the algorithms discussed thus far, MC-POMDPs are the only ones that do not require a finite state space. Further, MC-POMDPs use particle filters for belief tracking. We have already seen a number of successful particle filter applications. MC-POMDPs extend particle filters to planning and control problems.

The primary difficulty in using particle sets in POMDPs pertains to the representation of the value function. The space of all particle sets of any given size M is M-dimensional. Further, the probability that any particle set is ever observed twice is zero, due to the stochastic nature of particle generation. As a result, we need a representation for V that can be updated using some particle set, but then provides a value for other particle sets, which the MC-POMDP algorithm never saw before. In other words, we need a *learning algorithm*. MC-POMDP use a *nearest neighbor* algorithm using locally weighted interpolation when interpolating between different beliefs.

16.4.2 The MC-POMDP Algorithm

Table 16.3 lays out the basic MC-POMDP algorithm. The MC-POMDP algorithm required a number of nested loops. The innermost loop, in lines 6

```
1:  Algorithm MC-POMDP($b_0$, $V$):
2:      repeat until convergence
3:          sample $x \sim b(x)$                              // initialization
4:          initialize $\mathcal{X}$ with $M$ samples of $b(x)$
5:          repeat until episode over
6:              for all control actions $u$ do                // update value function
7:                  $Q(u) = 0$
8:                  repeat $n$ times
9:                      select random $x \in \mathcal{X}$
10:                     sample $x' \sim p(x' \mid u, x)$
11:                     sample $z \sim p(z \mid x')$
12:                     $\mathcal{X}' = \textbf{Particle\_filter}(\mathcal{X}, u, z)$
13:                     $Q(u) = Q(u) + \frac{1}{n}\,\gamma\,[r(x,u) + V(\mathcal{X}')]$
14:                 endrepeat
15:             endfor
16:             $V(\mathcal{X}) = \max_u Q(u)$                // update value function
17:             $u^* = \operatorname{argmax}_u Q(u)$           // select greedy action
18:             sample $x' \sim p(x' \mid u, x)$              // simulate state transition
19:             sample $z \sim p(z \mid x')$
20:             $\mathcal{X}' = \textbf{Particle\_filter}(\mathcal{X}, u, z)$   // compute new belief
21:             set $x = x'$; $\mathcal{X} = \mathcal{X}'$    // update state and belief
22:         endrepeat
23:     endrepeat
24:     return $V$
```

Table 16.3 The MC-POMDP algorithm.

through 16 in Table 16.3, updates the value function V for a specific belief $\mathcal{X}$. It does so by simulating for each applicable control action u, the set of possible successor beliefs. This simulation takes place in lines 9 through 12. From that, it gathers a local value for each of the applicable actions (line 13). The value function update takes place in line 16, in which V is simply set to the maximum of all Q_u's.

Following this local backup is a step in which MC-POMDPs simulate the physical system, to generate a new particle set $\mathcal{X}$. This simulation takes place in lines 17 through 21. In our example, the update always selects the greedy action (line 17); however, in practice it may be advantageous to occasionally select a random action. By transitioning to a new belief $\mathcal{X}$, the MC-POMDP value iteration performs the update for a different belief state. By iterating through entire episodes (outer loops in lines 2 through 5), the value function is eventually updated everywhere.

The key open question pertains to representation of the function V. MC-POMDP uses a local learning algorithm reminiscent of nearest neighbor. This algorithm grows a set of reference beliefs $\mathcal{X}_i$ with associated values V_i. When a query arrives with a previously unseen particle set $\mathcal{X}_{\text{query}}$, MC-POMDP identifies the K "nearest" particle sets in its memory. To define a suitable concept of nearness for particle sets requires additional assumptions. In the original implementation, MC-POMDP convolves each particle with a Gaussian with small, fixed covariance, and then measures the KL-divergence between the resulting mixtures of Gaussians. Leaving details aside, this step makes it possible to determine K nearest reference particle sets $\mathcal{X}_1, \ldots, \mathcal{X}_K$, with an associated measure of distance, denoted $d_1, \ldots, d_K$ (we note that KL-divergence is not a distance in the technical sense, since it is asymmetric). The value of the query set $\mathcal{X}_{\text{query}}$, is then obtained through the following formula

$$(16.11)\qquad V(\mathcal{X}_{\text{query}}) \;=\; \eta \sum_{k=1}^{K} \frac{1}{d_k}\, V_k$$

with $\eta = \left[\sum_k \frac{1}{d_k}\right]^{-1}$. Here $\mathcal{X}_k$ is the k-th reference belief in the set of K nearest neighbors, and d_k is the associated distance to the query set. This interpolation formula, known as *Shepard's interpolation*, explains how to calculate $V(\mathcal{X}')$ in line 13 of Table 16.3.

SHEPARD'S INTERPOLATION

The update in line 16 involves an implicit case differentiation. If the reference set contains already K particle sets whose distance is below a user-defined threshold, the corresponding V-values are simply updated in proportion to their contribution in the interpolation:

$$V_k \longleftarrow V_k + \alpha\,\eta\,\frac{1}{d_k}\,(\max_u\ Q(u) - V_k) \tag{16.12}$$

where α is a learning rate. The expression $\max_u\ Q(u)$ is the "target" value for the function V, and $\eta\,\frac{1}{d_k}$ is the contribution of the k-th reference particle set under Shepard's interpolation formula.

If there are less than K particle sets whose distance falls below the threshold, the query particle set is simply added into the reference set, with the associated value $V = \max_u\ Q(u)$. In this way, the set of reference particle sets grow over time. The value of K and the user-specified distance threshold determine the smoothness of the MC-POMDP value function. In practice, selecting appropriate values will take some thought, since it is easy to exceed the memory of an ordinary PC with the reference set, when the threshold is chosen too tightly.

16.4.3 Mathematical Derivation of MC-POMDPs

The MC-POMDP algorithm relies on a number of approximations: the use of particle sets constitutes one such approximation. Another one is the local learning algorithm for representing V, which is clearly approximate. A third approximation is due to the Monte Carlo backup step of the value function. Each of these approximations jeopardizes convergence of the basic algorithm.

The mathematical justification for using particle filters was already provided in Chapter 4. The Monte Carlo update step follows from the general POMDP update Equation (15.43) on page 532, which is restated here:

$$V_T(b) \quad = \quad \gamma\,\max_u\left[r(b,u) + \int V_{T-1}(B(b,u,z))\;p(z \mid u,b)\;dz\right] \tag{16.13}$$

The Monte Carlo approximation is now derived in a way entirely analogous to our AMDP derivation. We begin with the measurement probability $p(z \mid u, b)$, which resolves as follows:

$$p(z \mid u,b) \quad = \quad \int\!\!\int p(z \mid x')\;p(x' \mid u,x)\;b(x)\;dx\;dx' \tag{16.14}$$

Similarly, we obtain for $r(b,u)$:

$$r(b,u) \quad = \quad \int r(x,u)\;b(x)\;dx \tag{16.15}$$

This enables us to re-write (16.13) as follows:

$$
\begin{aligned}
\text{(16.16)} \quad V_T(b) &= \gamma \max_u \left[\int r(x,u)\, b(x)\, dx \right. \\
&\quad \left. + \int V_{T-1}(B(b,u,z)) \left[\int\int p(z \mid x')\, p(x' \mid u,x)\, b(x)\, dx\, dx' \right] dz \right] \\
&= \gamma \max_u \int\int\int [r(x,u) + V_{T-1}(B(b,u,z))\, p(z \mid x')\, p(x' \mid u,x)] \\
&\qquad b(x)\, dx\, dx'\, dz
\end{aligned}
$$

The Monte Carlo approximation to this integral is now a multi-variable sampling algorithm, which requires us to sample $x \sim b(x)$, $x' \sim p(x' \mid u, x)$ and $z \sim p(z \mid x')$. Once we have x, x', and z, we can compute $B(b, u, z)$ via the Bayes filter. We then compute $V_{T-1}(B(b, u, z))$ using the local learning algorithm, and $r(x, u)$ by simply looking it up. We note that all these steps are implemented in 7 through 14 of Table 16.3, with the final maximization carried out in line 16.

The local learning algorithm, which plays such a central role in MC-POMDPs, may easily destroy any convergence we might otherwise obtain with the Monte Carlo algorithm. We will not attempt to characterize the conditions under which local learning may give accurate approximations, but instead simply state that care has to be taken setting its various parameters.

16.4.4 Practical Considerations

From the three POMDP approximations provided in this chapter, MC-POMDP is the least developed and potentially the least efficient one. Its approximation relies on a learning algorithm for representing the value function. Implementing an MC-POMDP algorithm can be tricky. A good understanding of the smoothness of the value function is required, as is the number of particles one seeks to employ.

The original implementation of the MC-POMDP algorithm led to the results shown in Figure 16.4. A robot, shown in Figure 16.4a, is placed near a graspable object located on the floor near the robot, which it can detect using a camera. However, initially the object is placed outside the robot's perceptual field. A successful policy will therefore exhibit three stages: A search stage, in which the robot rotates until it senses the object; a motion stage in which the robot centers itself relative to the object so that it can grasp it; and a final grasping action. The combination of active perception and goal-directed behavior make this a relatively challenging probabilistic control problem.

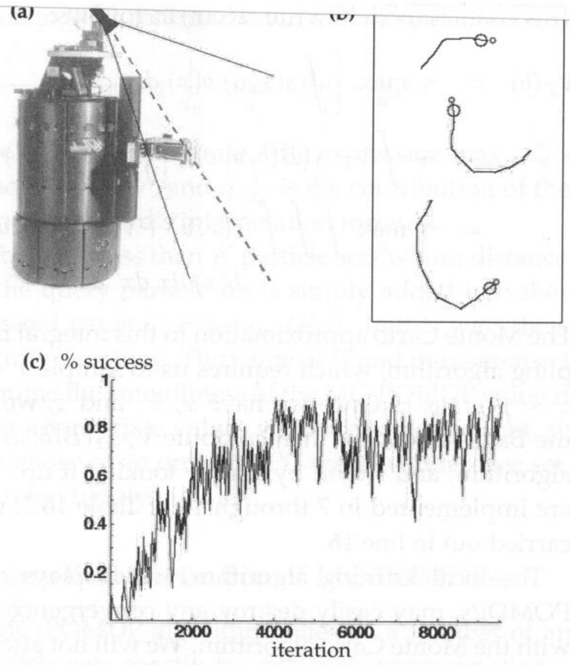


Figure 16.4 A robotic find and fetch task: (a) The mobile robot with gripper and camera, holding the target object. (b) 2-D trajectory of three successful policy executions, in which the robot rotates until it sees the object, and then initiates a successful grasp action (c) success rate as a function of number of planning steps, evaluated in simulation.

Figure 16.4b shows example episodes, in which the robot turned, moved, and grasped successfully. The trajectories shown there are projected motion directories in 2-D. Quantitative results are shown in Figure 16.4c, which plots the success rate as a function of update iterations of the MC-POMDP value iteration. 4,000 iterations of value backup require approximately 2 hours computation time, on a low-end PC, at which point the average performance levels off at 80%. The remaining 20% failures are largely due to configurations in which the robot fails to position itself to grasp the object—in part a consequence of the many approximations in MC-POMDPs.

16.5 Summary

In this section, we introduced three approximate probabilistic planning and control algorithms, with varying degrees of practical applicability. All three algorithms relied on approximations of the POMDP value function. However, they differed in the nature of their approximations.

- The QMDP framework considers uncertainty only for a single action choice: It is based on the assumption that after the immediate next control action, the state of the world suddenly becomes observable. The full observability made it possible to use the MDP-optimal value function. QMDP generalizes the MDP value function to belief spaces through the mathematical expectation operator. As a result, planning in QMDPs is as efficient as in MDPs, but the value function generally overestimates the true value of a belief state.

- Extensions of the QMDP algorithm combine the MDP-optimal value function with a sequence of POMDP backups. When combined with T POMDP backup steps, the resulting policy considers information-gathering actions within the horizon of T, and then relies on the QMDP assumption of a fully observable state. The larger the horizon T is, the closer the resulting policy to the full POMDP solution.

- The AMDP algorithm pursues a different approximation: It maps the belief into a lower-dimensional representation, over which it then performs exact value iteration. The "classical" representation consists of the most likely state under a belief, along with the belief entropy. With this representation, AMDPs are like MDPs with one added dimension in the state representation that measures the global degree of uncertainty of the robot.

- To implement an AMDP, it becomes necessary to *learn* the state transition and the reward function in the low-dimensional belief space. AMDPs achieve this by an initial phase, in which statistics are cached into look-up tables, representing the state transition and reward function. Thus, AMDPs operate over a learned model, and are only accurate to the degree that the learned model is accurate.

- AMDPs applied to mobile robot navigation in known environments is called *coastal navigation*. This navigation technique anticipates uncertainty, and selects motion that trades off overall path length with the uncertainty accrued along a path. The resulting trajectories differ significantly from any non-probabilistic solution: A "coastally" navigating robot stays away from areas in which chances of getting permanently lost are high. Being temporarily lost is acceptable, if the robot can later re-localize with sufficiently high probability.

- The MC-POMDP algorithm is the particle filter version of POMDPs. It calculates a value function defined over sets of particles. To implement such a value function, MC-POMDPs had to resort to a local learning technique, which used a locally weighted learning rule in combination with a proximity test based on KL-divergence. MC-POMDPs then apply Monte Carlo sampling to implement an approximate value backup. The resulting algorithm is a full-fledged POMDP algorithm whose computational complexity and accuracy are both functions of the parameters of the learning algorithm.

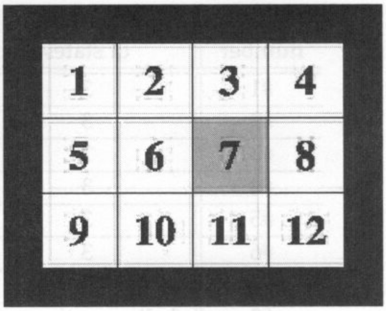
The key lesson to take away from this chapter is that there exists a number of approximations whose computational complexity is much closer to MDPs, but that still consider state uncertainty. No matter how crude the approximation, algorithms that consider state uncertainty tend to be significantly more robust than algorithms that entirely ignore state uncertainty. Even a single new element in the state vector—which measures global uncertainty in a one-dimensional way—can make a huge difference in the performance of a robot.

16.6 Bibliographical Remarks

The literature on approximate POMDP problem solving was already extensively discussed in the last chapter (15.7). The QMDP algorithm described in this chapter is due to Littman et al. (1995). The AMDP algorithm for a fixed augmented state representation was developed by Roy et al. (1999). Later, Roy et al. (2004) extended it to a learned state representation. Thrun (2000a) devised the Monte Carlo POMDP algorithm.

16.7 Exercises

1. In this question, you are asked to design an AMDP that solves a simple navigation problem. Consider the following environment with 12 discrete states.



Initially, the robot is placed at a random location, chosen uniformly among all 12 states. Its goal is to advance to state 7. At any point in time, the robot goes north, east, west, or south. Its only sensor is a bumper: When it hits an obstacle, the bumper triggers and the robot does not change states. The robot cannot sense what state it is in, and it cannot sense the direction of its bumper. There is no noise in this problem, just the initial location uncertainty (which we will assume to be uniform).

(a) How many states will an AMDP minimally have to possess? Describe them all.

(b) How many of those states are reachable from the initial AMDP state? Describe them all.

(c) Now assume the robot starts at state 2 (but it still does not know, so its internal belief state will be different). Draw the state transition diagram between all AMDP states that can be reached within four actions.

(d) For this specific type problem (noise-free sensors and robot motion, finite state, action, and measurement space), can you think of a more compact representation than the one used by AMDPs, which is still sufficient to find the optimal solution?

(e) For this specific type problem (noise-free sensors and robot motion, finite state, action, and measurement space), can you craft a state space for which AMDPs will fail to find the optimal solution?

2. In the previous chapter, we learned about the *tiger problem* (Exercise 1 on page 544). What modification of this problem will enable QMDP to come up with an optimal solution? Hint: There are multiple possible answers.

3. In this question, we would like you to determine the *size* of the belief state space. Consider the following table:

problem number	number of states	sensors sensors	state transition	initial state
#1	3	perfect	noise-free	known
#2	3	perfect	noisy	known
#3	3	noise-free	noise-free	unknown (uniform)
#4	3	noisy	noise-free	known
#5	3	noisy	noise-free	unknown (uniform)
#6	3	none	noise-free	unknown (uniform)
#7	3	none	noisy	known
#8	1-dim continuum	perfect	noisy	known
#9	1-dim continuum	noisy	noisy	known
#10	2-dim continuum	noisy	noisy	unknown (uniform)

A perfect sensor always provides the full state information. A noise-free sensor may provide partial state information, but it does so without any randomness. A noisy sensor may be partial and is also subject to noise. A noise-free state transition is deterministic, whereas a stochastic state transition is called noisy. Finally, we only distinguish two types of initial conditions, one in which the initial state is known with absolute certainty, and one in which it is entirely unknown and the prior over states is uniform.

Your question: What is the size of the reachable belief space for all 10 problems above? Hint: It may be finite or infinite, and in the infinite case you should be able to tell what dimension the belief state has.

4. We want you to brainstorm about the failure modes of an AMDP planner. In particular, the AMDP *learns* state transition and reward functions. Brainstorm what can go wrong with this when such learned models are used for value iteration. Identify at least three different types of problems, and discuss them in detail.