Chapter 5

POMDP APPROXIMATION USING SIMULATION AND HEURISTICS

Edwin K. P. Chong

Colorado State University

echong@colostate.edu

C. M. Kreucher

General Dynamics Michigan Research and Development Center

Christopher.Kreucher@gd-ais.com

A. O. Hero III

The University of Michigan

hero@umich.edu

1. Introduction

This chapter discusses a class of approximation methods for sensor management under the partially observable Markov decision process (POMDP) model. Unlike Chapter 3, which focuses on analytic methods for bounding and solving POMDPs, here we discuss methods based on heuristics and simulation. Our aim is to develop methods that are implementable on-line and find nearly optimal policies.

While the methods described herein are more generally applicable, the focus application will be sensor management for target tracking. Information theoretic measures discussed in Chapter 4 and particle filter approximations discussed in Chapter 5 will be implemented to illustrate these approximations.

It is informative to distinguish between *myopic* and *non-myopic* (also known as *dynamic* or *multistage*) resource management, a topic of much current interest (see, e.g., [144, 109, 110]). In myopic resource management, the objective is to optimize performance on a per-decision basis. For example, consider *sensor scheduling* for tracking a single target, where the problem is to select, at each decision epoch, a single sensor to activate. An example sensor-scheduling scheme is CPA (closest point of approach), which selects the sensor that is perceived to be the closest to the target. Another (more sophisticated) example is the method described in [146], where the authors present a sensor management method using alpha-divergence (or Rényi divergence) measures. Their approach is to make the decision that maximizes the expected information gain (which is measured in terms of the alpha-divergence).

Myopic sensor-management schemes may not be ideal when the performance is measured over a horizon of time. In such situations, we need to consider schemes that trade off short-term for long-term performance. We call such schemes non-myopic. Several factors motivate the consideration of non-myopic schemes:

Heterogeneous sensors. If we have sensors with different locations, characteristics, usage costs, and/or lifetimes, the decision of whether or not to use a sensor should consider what the overall performance will be, not whether or not its use maximizes the current performance.

Sensor motion. The future location of a mobile sensor affects how we should act now. To optimize a long-term performance measure, we need to be opportunistic in our choice of sensor decisions.

Target motion. If a target is moving, there is potential benefit in sensing the target before it becomes unresolvable (e.g., too close to other targets or to clutter, or shadowed by large objects). In some scenarios, we may need to identify multiple targets before they cross, to aid in data association.

The rest of this chapter is organized as follows. In Section 2, we give a concrete motivating example that advocates for the use of non-myopic scheduling. Next, in Section 3, we review the basic principles behind Q-value approximation. Then, in Section 4, we illustrate the basic lookahead control framework and describe the constituent components. In Section 6, we describe a host of Q-value approximation methods. Among others, this section includes descriptions of Monte Carlo sampling methods, heuristic approximations, and rollout methods. In Section 7, we provide a set of simulation results on a model problem that illustrate several of the approximate non-myopic scheduling methods described in this chapter. We conclude in Section 8 with some summary remarks.

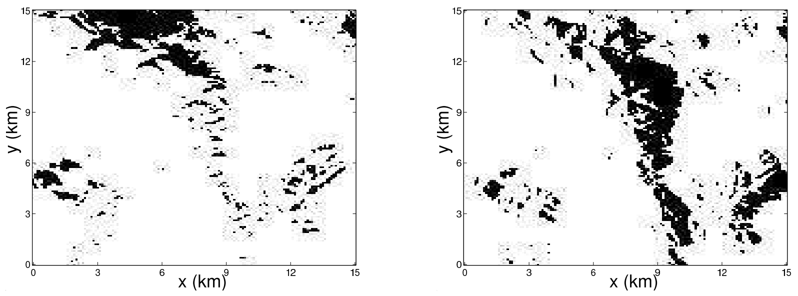
2. Motivating Example

We now present a concrete motivating example that will be used to explain and justify the heuristics and approximations used in this chapter. This example involves a remote sensing application where the goal is to learn the contents of a surveillance region via repeated interrogation.

Consider a single airborne sensor which is able to image a portion of a ground surveillance region to determine the presence or absence of moving ground targets. At each time epoch, the sensor is able to direct an electrically scanned array so as to interrogate a small area on the ground. Each interrogation yields some (imperfect) information about the small area. The objective is to choose the sequence of pointing directions that lead to the best acquisition of information about the surveillance region.

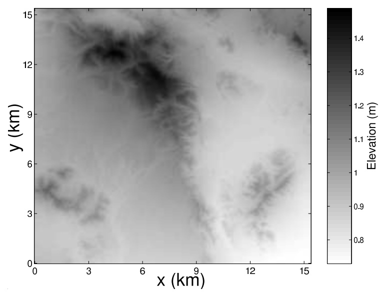
A further complication is the fact that at each time epoch the sensor position causes portions of the ground to be unobservable due obscuration. Obscuration is due to varying degrees of terrain elevation that can block line-of-sight from the sensor to the target on the ground. We assume that given the sensor position and the terrain elevation, the sensor can compute a visibility mask which specifies how well a particular spot on the ground can be seen by the sensor. As an example, in Figure 6.1 we give binary visibility masks that are computed from a sensor positioned (a) below and (b) to the left of the topographically nonhomogeneous surveillance region. As can be seen from the figures, sensor position causes “shadowing” of certain regions. These regions, if measured, would provide no information to the sensor.

This example illustrates a situation where non-myopic scheduling is highly beneficial. Using a known sensor trajectory and known topographical map, the sensor can predict locations that will be obscured in the future. This information can be used to prioritize resources so that they are used on targets that are predicted to become obscured in the future. Extra sensor dwells immediately before obscuration (at the expense of not interrogating other targets) will sharpen the estimate of target location. This sharpened estimate will allow better prediction of where and when the target will emerge from the obscured area. This is illustrated graphically with a six time-step vignette in Figure 6.2.



(a) The visibility mask for a sensor positioned below the region

(b) The visibility mask for a sensor positioned to the left of the region



(c) Elevation map of the surveillance region

Figure 5.1. Visibility masks for a sensor positioned below and left of the surveillance region. We show binary visibility masks (non-visible areas are black and visible areas are white). In general, visibility may be between 0 and 1 indicating areas of reduced visibility, e.g., regions that are partially obscured by foliage (Figure 1 from [144] which is ©2004 IEEE - used with permission).

3. Basic Principle: Q-value Approximation

3.1 Optimal Policy

In general the action chosen at each time k should be allowed to depend on the entire history up to time k (i.e., the action at time k is a random variable that is a function of all observable quantities up to time k). However, it turns out that if an optimal choice of such a sequence of actions exists, then there is an optimal choice of actions that depends only on "belief-state feedback." In other words, it suffices for the action at time k to depend only on the belief state (or information state) b_k at time k. Let $\mathcal{B}$ be the set of distributions over the underlying state space $\mathcal{X}$ (we call $\mathcal{B}$ the *belief-state space*). So what we seek is, at each time k, a mapping $\pi_k^* : \mathcal{B} \to \mathcal{A}$ such that if we perform action

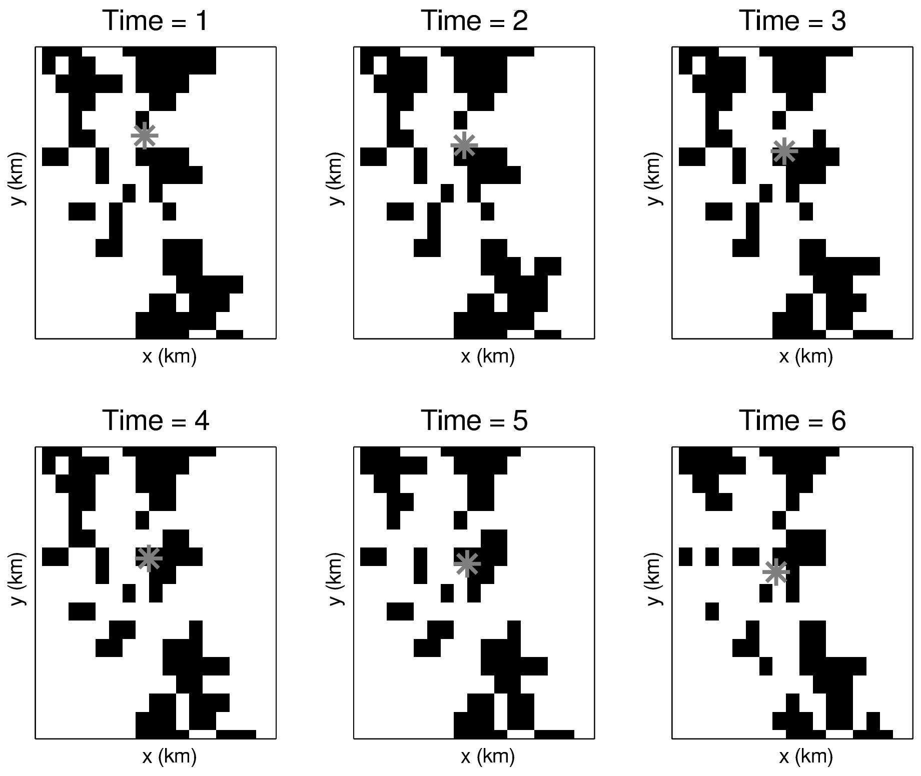


Figure 5.2. A six time step vignette where a target moves through an obscured area. Other targets are present elsewhere in the surveillance region but outside the area shown in the figures. The target is depicted by an asterisk. Areas that are obscured from the sensor point of view are black and areas that are visible are white. Extra dwells just before becoming obscured (time = 1) aid in localization after the target emerges (Figure 2 from [144] which is ©2004 IEEE - used with permission).

$a_k = \pi_k^*(b_k)$, then the resulting objective function is maximized. As usual, we call such a mapping a *policy*.

3.2 *Q*-values

Let $V_H^*(b_0)$ be the optimal objective function value (over horizon H). Then, *Bellman's principle* states that

$$V_H^*(b_0) = \max_a (r(b_0, a) + \mathbb{E}^a[V_{H-1}^*(b_1)|b_0])$$

where $r(b_0, a)$ is the *reward* associated with taking action a at belief state b_0, and b_1 is the random next belief state (with distribution depending on a).

Moreover,

$$\pi_0^*(b_0) = \arg\max_a (r(b_0, a) + \mathbb{E}^a[V_{H-1}^*(b_1)|b_0]).$$

Define the *Q-value* of taking action a at state b_k as

$$Q_{H-k}(b_k, a) = r(b_k, a) + \mathbb{E}^a[V_{H-k-1}^*(b_{k+1})|b_k],$$

where b_{k+1} is the random next belief state. Then, Bellman's principle can be rewritten as

$$\pi_k^*(b_k) = \arg\max_a Q_{H-k}(b_k, a)$$

In other words, the optimal action to take at belief-state b_k (at time k, with a horizon-to-go of $H-k$) is the one with largest Q-value at that belief state. This principle, called *lookahead*, forms the heart of approaches to solving POMDPs.

3.3 Stationary policies

In general, an optimal policy is a function of time k. It turns out that if H is sufficiently large, then the optimal policy is approximately *stationary* (independent of time k). This seems intuitively clear: if the end of the time horizon is a million years away, then how we should act today given a belief-state x is the same as how we should act tomorrow given the same belief state. To put it differently, if H is sufficiently large, the difference between Q_H and Q_{H-1} is negligible. Henceforth we will assume that there is a stationary optimal policy, and this is what we seek. We will use the notation π for stationary policies (with no subscript k).

3.4 Receding horizon

Assuming that H is sufficiently large and that we seek a stationary optimal policy, at any time k we will write:

$$\pi^*(b) = \arg\max_a Q_H(b, a).$$

Notice that the horizon is taken to be fixed at H, regardless of the current time k. This is justified by our assumption that H is so large that at any time k, the horizon is still approximately H time steps away. This approach of taking the horizon to be fixed at H is called *receding horizon control*. For convenience, we will also henceforth drop the subscript H from our notation (unless the subscript is explicitly needed).

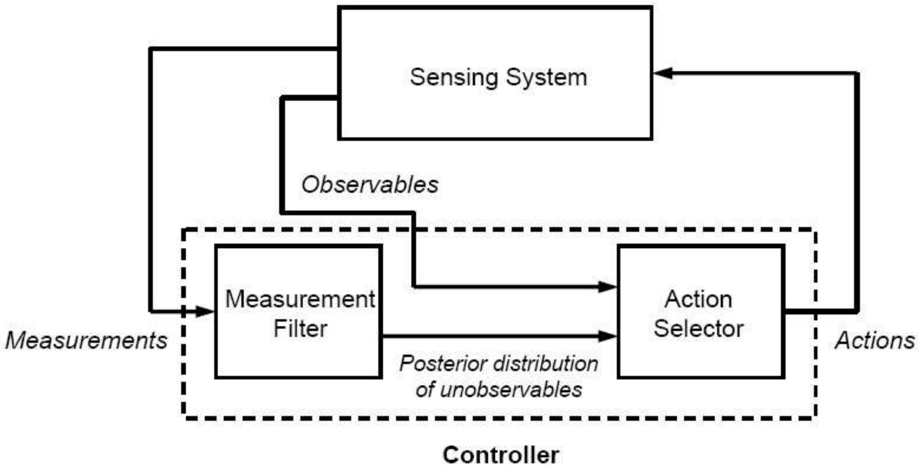


Figure 5.3. Basic lookahead framework.

3.5 Approximating Q-values

Recall that $Q(b, a)$ is simply the reward $r(b, a)$ of taken action a at belief-state b, plus the expected cumulative reward of applying the optimal policy for all future actions. This second term in the Q-value is in general difficult to obtain, especially for a problem with a large belief-state space. For this reason, approximation methods are necessary to obtain Q-values. Note that the quality of an approximation to the Q-value is not so much in the accuracy of the actual values obtained, but in the *ranking* of the actions reflected by their *relative* values.

In Section 6, we describe a variety of methods to approximate Q-values. But before discussing such methods, in the next section we first describe the basic control framework for using Q-values to inform control decisions.

4. Basic Control Architecture

5. Control architecture

By Bellman's principle, knowing the Q-values allows us to make optimal control decisions. In particular, if we are currently at belief-state b, we need only find the action a with the largest $Q(b, a)$. This principle yields a basic control framework that is illustrated in Figure 6.3.

The top-most block represents the sensing system, which we treat as having an input and two forms of output. The input represents actions (external control commands) we can apply to control the sensing system. Actions usually include sensor-resource controls, such as which sensor(s) to activate, at what power level, where to point them, what waveforms to use, and what sensing modes to activate. Actions may also include communication-resource controls, such as the data rate for transmission from each sensor.

The two forms of outputs from the sensing system represent:

1 Fully observable aspects of the internal state of the sensing system (which we call *observables*), and

2 Measurements (observations) of those aspects of the internal state that are not directly observable (which we refer to simply as *measurements*).

We assume that the underlying state-space is the Cartesian product of two sets, one representing unobservables and the other representing observables. Target states are prime examples of unobservables. So, measurements are typically the outputs of sensors, representing observations of target states. Observables include things like sensor locations and orientations, which sensors are activated, battery status readings, etc. In the remainder of this section, we describe the components of our control framework. Our description starts from the architecture of Figure 6.3 and progressively fills in the details.

5.1 Controller

At each decision epoch, the *controller* takes the outputs (measurements and observables) from the tracking system and, in return, generates an action that is fed back to the tracking system. This basic closed-loop architecture is familiar to mainstream control system design approaches.

The controller consists of two main components. The first component is the *measurement filter*, which takes as its input the measurements, and provides as its output the posterior distribution of the unobservable internal states (which we henceforth simply call *unobservables*). In the typical situation where these unobservables are target states, the measurement filter outputs posterior distribution of the target states given the measurement history. We describe the measurement filter further below. The posterior distribution of the unobservables, together with the observables, form the belief state, the posterior distribution of the underlying state.

The second component of the controller is the *action selector*, *Action selector*which takes the belief state and computes an action (which is the output of the controller). The basis for action selection is Bellman's principle, using Q-values. We discuss this below.

5.2 Measurement filter

The measurement filter computes the posterior distribution given measurements. This component is present in virtually any target-tracking system. It turns out that the posterior distribution can be computed iteratively: each time we obtain a new measurement, the posterior distribution can be obtained by updating the previous posterior distribution based on knowing the current action, the transition law, and the observation law. This update is based on Bayes' rule (see Chapter 3).

The measurement filter can be constructed in a number of ways. If we know beforehand that the posterior distribution always resides within a family of distributions that is conveniently parameterized, then all we need to do is keep track of the belief-state parameters. This is the case, for example, if the belief state is Gaussian. Indeed, if the unobservables evolve in a linear fashion, then these Gaussian parameters can be updated using a Kalman filter. See Chapter 11 for a detailed development of the Kalman filter for multiple target tracking with data association. In general, however, it is not practical to keep track of the exact belief state. A variety of options have been explored for belief-state representation and simplification (e.g., [208, 204, 256]). We will have more to say about belief-state simplification in Section 6.10.

Particle filtering is a Monte Carlo sampling method for updating posterior distributions. Instead of maintaining the exact posterior distribution, we maintain a set of representative samples from that distribution. It turns out that this method dovetails naturally with Monte Carlo sampling-based methods for Q-value approximation, as we will describe later in Section 6.7.

5.3 Action selector

As shown in Figure 6.4, the action selector consists of a search (optimization) algorithm that optimizes an objective function, the *Q-function*, with respect to an action. In other words, the Q-function is a function of the action—it maps each action, at a given belief state, to its Q-value. The action that we seek is one that maximizes the Q-function. So, we can think of the Q-function as a kind of "action-utility" function that we wish to maximize.

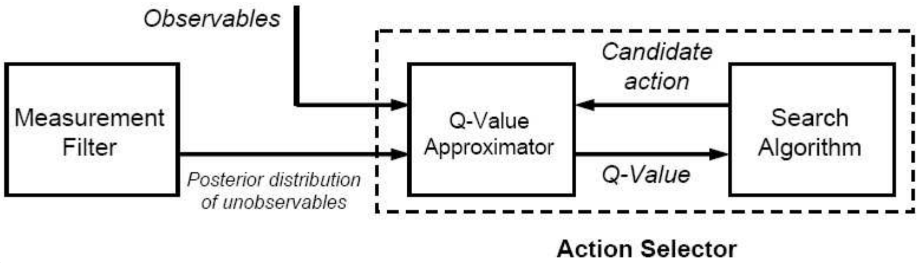


Figure 5.4. Basic components of the action selector.

As is typical, the search algorithm iteratively generates a candidate action and evaluates the Q-function at this action (this numerical quantity is the Q-value), searching over the space of candidate actions for one with the largest Q-value. Methods for obtaining (approximating) the Q-values is described in the next section.

6. Q-value Approximation Methods

6.1 Basic approach

Recall the definition of the Q-value,

$$Q(b, a) = r(b, a) + \mathbb{E}^a[V^*(b')|b], \tag{5.1}$$

where b' is the random next belief state (with distribution depending on a). In all but very special problems, it is impossible to compute the Q-value exactly. In this section, we describe a variety of methods to approximate the Q-value. Because the first term on the right-hand side of (6.1) is usually easy to compute, most approximation methods focus on approximating the second term. As pointed out before, it is important to realize that the quality of an approximation to the Q-value is not so much in the accuracy of the actual values obtained, but in the *ranking* of the actions reflected by their *relative* values.

6.2 Monte Carlo sampling

In general, we can think of Monte Carlo methods simply as the use of computer generated random numbers in computing expectations of random variables through averaging over many samples. With this in mind, it seems natural to consider using Monte Carlo methods to compute the value function directly based on Bellman's equation:

$$V_H^*(b_0) = \max_{a_0}(r(b_0, a_0) + \mathbb{E}^{a_0}[V_{H-1}^*(b_1)|b_0]).$$

Notice that the second term on the right-hand side involves expectations (one per action candidate a_0), which can be computed using Monte Carlo sampling. However, the random variable inside each expectation is itself an objective function value (with horizon $H - 1$), and so it too involves a max of an expectation via Bellman's equation:

$$V_H^*(b_0) = \max_{a_0}\left(r(b_0, a_0) + \mathbb{E}^{a_0}\left[\max_{a_1}(r(b_1, a_1) + \mathbb{E}^{a_1}[V_{H-2}^*(b_2)|b_1])\middle| b_0\right]\right).$$

Notice that we now have two "layers" of max and expectation, one "nested" within the other. Again, we see that the inside expectation involves the value function (with horizon $H - 2$), which yet again can be written as a max of expectations. Proceeding this way, we can write $V_H^*(b_0)$ in terms of H layers of max and expectations. Each expectation that appears in this way of expressing the value function V_H^* can be computed using Monte Carlo sampling. The question that remains is, how computationally burdensome is this task?

Kearns, Mansour, and Ng [134] have provided a method to calculate the computational burden of approximating the value function using Monte Carlo sampling as described above, given some prescribed accuracy in the approximation of the value function. Unfortunately, it turns out that for practical POMDP problems this computational burden is prohibitive, even for modest degrees of accuracy. So, while Bellman's equation suggests a natural Monte Carlo method for approximating the value function, the method is not useful in practice. For this reason, we seek alternative approximation methods. In the next few subsections, we explore some of these methods.

6.3 Relaxation of optimization problem

Some problems that are difficult to solve become drastically easier if we *relax* certain aspects of the problem. For example, by removing a constraint in the problem, the "relaxed" problem may yield to well-known solution methods. This constraint relaxation enlarges the constraint set, and so the solution obtained may no longer be feasible in the original problem. However, the objective function value of the solution *bounds* the optimal objective function value of the original problem.

The Q-value involves the quantity $V^*(b')$, which can be viewed as the optimal objective function value corresponding to some optimization problem. The method of relaxation, if applicable, gives rise to a bound on $V^*(b')$, which then provides an approximation to the Q-value. For example, a relaxation of the original POMDP may result in a bandit problem (see Chapter 7), or may be solvable via linear programming (see Chapter 3). For further discussion on methods based on relaxation, see Chapter 8.

6.4 Heuristic approximation

In some applications, although we are unable to compute Q-values directly, we can use domain knowledge to develop some idea of how it behaves. If so, we can heuristically construct a Q-function based on this knowledge.

Recall from (6.1) that the Q-value is the sum of two terms, where the first term (the immediate reward) is usually easy to compute. Therefore, it often suffices to approximate only the second term in (6.1), which is the mean optimal objective function value starting at the next belief state, which we call the *expected value-to-go* (EVTG). (Note that the EVTG is a function of both b and a, because the distribution of the next belief state is a function of b and a.) In some problems, it is possible to construct a heuristic EVTG based on domain knowledge. If the constructed EVTG properly reflects tradeoffs in the selection of alternative actions, then the ranking of these actions via their Q-values will result in the desired "lookahead."

For example, consider the motivating example of tracking multiple targets with a single sensor. Suppose we can only measure the location of one target per decision epoch. The problem then is to decide which location to measure and the objective function is the aggregate (multi-target) tracking error. The terrain over which the targets are moving is such that the measurement errors are highly location dependent, for example because of the presence of topographical features which cause some areas to be invisible from a future sensor position. In this setting, it is intuitively clear that if we can predict sensor and target motion so that we expect a target is about to be obscured, then we should focus our measurements on that target immediately before the obscuration so that its track accuracy is improved and the overall tracking performance maximized in light of the impending obscuration.

The same reasoning applies in a variety of other situations, including those where targets are predicted to become unresolvable to the sensor (e.g., two targets that cross) or where the target and sensor motion is such that future measurements are predicted to be less reliable (e.g., a bearings-only sensor that is moving away from a target). In these situations, we advocate a heuristic method that replaces the EVTG by a function that captures the long-term benefit of an action in terms of an "opportunity cost" or "regret." That is, we approximate the Q-value as

$$Q(b,a) \approx r(b,a) + wN(b,a), \tag{5.2}$$

where $N(b,a)$ is an easily computed heuristic approximation of the long-term value, and w is a weighting term that allows us to trade the influence of the immediate value and the long-term value. As a concrete example of a useful heuristic, we have used the "gain in information for waiting" as a choice of $N(b,a)$ [145]. Specifically, let $\bar{g}_a^k$ denote the expected myopic gain when taking action a at time k. Furthermore, denote by $p_a^k(\cdot)$ the distribution of myopic gains when taking action a at time k. Then a useful approximation of the long-term value of taking action a is the gain (loss) in information received by waiting until a future time step to take the action,

$$N(b,a) \approx \sum_{m=1}^{M} \gamma^m sgn\big(\bar{g}_a^k - \bar{g}_a^{k+m}\big) D_\alpha\big(p_a^k(\cdot)||p_a^{k+m}(\cdot)\big) \tag{5.3}$$

where M is the number of time steps in the future that are considered.

Each term in the summand of $N(b,a)$ is made up of two components. First, $sgn\big(\bar{g}_a^k - \bar{g}_a^{k+m}\big)$ signifies if the expected reward for taking action a in the future is more or less than the present. A negative value implies that the future is better and that the action ought to be discouraged at present. A positive value implies that the future is worse and that the action ought to be encouraged at present. This may happen, for example, when the visibility of a given target is getting worse with time. The second term, $D_\alpha\big(p_a^k(\cdot)||p_a^{k+m}(\cdot)\big)$, reflects the magnitude of the change in reward using the divergence between the density on myopic rewards at the current time step and at a future time step. A small number implies that the present and future rewards are very similar, and therefore the non-myopic term will have little impact on the decision making.

Therefore, $N(b,a)$ is positive if an action is less favorable in the future (e.g., the target is about to become obscured). This encourages taking actions that are beneficial in the long term, and not just taking actions based on their immediate reward. Likewise, the term is negative if the action is more favorable in the future (e.g., the target is about to emerge from an obscuration). This discourages taking actions now that will have more value in the future.

6.5 Parametric approximation

In situations where a heuristic Q-function is difficult to construct, we may consider methods where the Q-function is approximated by a parametric function (by this we mean that we have a function approximator parameterized by one or more parameters). Let us denote this approximation by $\tilde{Q}(b, \theta)$, where θ is a parameter (to be tuned appropriately). For this approach to be useful, the computation of $\tilde{Q}(b, \theta)$ has to be relatively simple, given b and θ. Typically, we seek approximations for which it is easy to set the value of the parameter θ appropriately, given some information of how the Q-values "should" behave (e.g., from expert knowledge, empirical results, simulation, or on-line observation). This adjustment or tuning of the parameter θ is called *training*.

As in the heuristic approximation approach, the approximation of the Q-function by the parametric function approximator is usually accomplished by approximating the EVTG, or even directly approximating the objective function V^*.[1] In the usual parametric approximation approach, the belief state b is first mapped to a set of *features*. The features are then passed through a parametric function to approximate $V^*(b)$. For example, in the problem of tracking multiple targets with a single sensor, we may extract from the belief state some information on the location of each target relative to the sensor, taking into account the topography. These constitute the features. For each target, we then assign a numerical value to these features, reflecting the measurement accuracy. Finally, we take a linear combination of these numerical values, where the coefficients of this linear combination serve the role of the parameters to be tuned.

The parametric approximation method has some advantages over methods based only on heuristic construction. First, the training process usually involves numerical optimization algorithms, and thus well-established methodology can be brought to bear on the problem. Second, even if we lack immediate expert knowledge on our problem, we may be able to experiment with the system (e.g., by using a simulation model, e.g., a generative model [134]). Such empirical output is useful for training the function approximator. Common training methods found in the literature go by the names of reinforcement learning, Q-learning, neurodynamic programming, and approximate dynamic programming.

The parametric approximation approach may be viewed as a systematic method to implement the heuristic-approximation approach. But note that even in the parametric approach, some heuristics are still needed in the choice of features and in the form of the function approximator. For further reading, see [29].

[1] In fact, given a POMDP, it turns out that the Q-value can be viewed as the objective function value for a related problem; see [29].

6.6 Action-sequence approximations

Let us write the value function (optimal objective function value as a function of belief state) as

$$\begin{aligned} V^*(b) &= \max_{\pi} \mathbb{E}^{\pi}\left[\sum_{k=1}^{H} r(b_k, \pi(b_k)) \middle| b\right] \\ &= \mathbb{E}\left[\max_{a_1,\ldots,a_H : a_k = \pi(b_k)} \sum_{k=1}^{H} r(b_k, a_k) \middle| b\right], \end{aligned} \tag{5.4}$$

where the notation $\max_{a_1,\ldots,a_H:a_k=\pi(b_k)}$ means maximization subject to the constraint that each action a_k is a (fixed) function of the belief state b_k. If we relax this constraint on the actions and allow them to be arbitrary random variables, then we have an upper bound on the value function:

$$\hat{V}_{\text{upper}}(b) = \mathbb{E}\left[\max_{a_1,\ldots,a_H} \sum_{k=1}^{H} r(b_k, a_k) \middle| b\right].$$

In some applications, this upper bound provides a suitable approximation to the value function. The advantage of this approximation method is that in certain situations the computation of the "max" above involves solving a relatively easy optimization problem. This method is called *hindsight optimization* [66, 254].

One implementation of this idea involves averaging over many Monte Carlo simulation runs to compute the expectation above. In this case, the "max" is computed for each simulation run by first generating all the random numbers for that run, and then applying a static optimization algorithm to compute optimal actions $a_1, \ldots, a_H$. It is easy now to see why we call the method "hindsight" optimization: the optimization of the action sequence is done after knowing all uncertainties over time, as if making decisions in hindsight.

As an alternative to relaxing the constraint in (6.4) (that each action a_k is a fixed function of the belief state b_k), suppose we further *restrict* each action to be simply fixed (not random). This restriction gives rise to a lower bound on the value function:

$$\hat{V}_{\text{lower}}(b) = \max_{a_1,\ldots,a_H} \mathbb{E}^{a_1,\ldots,a_H}[r(b_1, a_1) + \cdots + r(b_H, a_H)|b].$$

To use analogous terminology to "hindsight optimization," we may call this method *foresight optimization*—we make decisions before seeing what actually happens, based on our expectation of what will happen. For an application of this method to tracking, see [65].

6.7 Rollout

In this section, we describe the method of *policy rollout* (or simply *rollout*). The basic idea is simple. First let $V^{\pi}(b_0)$ be the objective function value corresponding to policy π. Recall that $V^* = \max_{\pi} V^{\pi}$. In the method of rollout, we assume that we have a candidate policy π_{base} (called the *base policy*), and we simply replace V^* in (6.1) by $V^{\pi_{\text{base}}}$. In other words, we use the following approximation to the Q-value:

$$Q^{\pi_{\text{base}}}(b,a) = r(b,a) + \mathbb{E}^a[V^{\pi_{\text{base}}}(b')|b].$$

We can think of $V^{\pi_{\text{base}}}$ as the performance of applying policy π_{base} in our system. In many situations of interest, $V^{\pi_{\text{base}}}$ is relatively easy to compute, either analytically, numerically, or via Monte Carlo simulation.

It turns out that the policy π defined by

$$\pi(b) = \arg\max_{a} Q^{\pi_{\text{base}}}(b,a) \tag{5.5}$$

is at least as good as π_{base} (in terms of the objective function); in other words, this step of using one policy to define another policy has the property of *policy improvement*. This policy-improvement result is the basis for a method known as *policy iteration*, where we iteratively apply the above policy-improvement step to generate a sequence of policies converging to the optimal policy. However, policy iteration is difficult to apply in problems with large belief-state spaces, because the approach entails explicitly representing a policy and iterating on it (remember that a policy is a mapping with the belief-state space $\mathcal{B}$ as its domain).

In the method of policy rollout, we do not explicitly construct the policy π in (6.5). Instead, at each time step, we use (6.5) to compute the output of the policy at the current belief-state. For example, the term $\mathbb{E}^a[V^{\pi_{\text{base}}}(b')|b]$ can be computed using Monte Carlo sampling. To see how this is done, observe that $V^{\pi_{\text{base}}}(b')$ is simply the mean cumulative reward of applying policy π_{base}, a quantity that can be obtained by Monte Carlo simulation. The term $\mathbb{E}^a[V^{\pi_{\text{base}}}(b')|b]$ is the mean with respect to the random next belief-state b' (with distribution that depends on b and a), again obtainable via Monte Carlo simulation. We provide more details in Section 6.9. In our subsequent discussion of rollout, we will assume by default that the method is implemented using Monte Carlo simulation. For an application of the rollout method to sensor scheduling in target tracking, see [109, 110].

6.8 Parallel rollout

An immediate extension to the method of rollout is to use multiple base policies. So suppose that $\Pi_B = \{\pi^1, \dots, \pi^n\}$ is a set of base policies. Then replace V^* in (6.1) by

$$\hat{V}(b) = \max_{\pi \in \Pi_B} V^{\pi}(b).$$

We call this method *parallel rollout* [62]. Notice that the larger the set Π_B, the tighter $\hat{V}(b)$ becomes as a bound on $V^*(b)$. Of course, if Π_B contains the optimal policy, then $\hat{V} = V^*$. It follows from our discussion of rollout that the policy improvement property also holds when using a lookahead policy based on parallel rollout. As with the rollout method, parallel rollout can be implemented using Monte Carlo sampling.

6.9 Control architecture in the Monte Carlo case

The method of rollout provides a convenient turnkey (systematic) procedure for Monte-Carlo-based decision making and control. Here, we specialize the general control architecture of Section 4 to the use of particle filtering for belief-state updating and a Monte Carlo method for Q-value approximation (e.g., rollout). Particle filtering, which, as discussed in Chapter 5, is a Monte Carlo sampling method for updating posterior distributions, dovetails naturally with Monte Carlo methods for Q-value approximation. An advantage of the Monte Carlo approach is that it does not rely on analytical tractability—it is straightforward in this approach to incorporate sophisticated models for sensor characteristics and target dynamics.

Figure 6.5 shows the control architecture specialized to the Monte Carlo setting. Notice that, in contrast to Figure 6.3, the a particle filter plays the role of the measurement filter, and its output consists of samples of the unobservables. Figure 6.6 shows the action selector in this setting. Contrasting this figure with Figure 6.4, we see that a Monte Carlo simulator plays the role of the Q-value approximator (e.g., through the use of rollout).

As a specific example, consider applying the method of rollout. In this case, the evaluation of the Q-value for any given candidate action relies on a simulation model of the sensing system operating some base policy. This simulation model is a "dynamic" model in the sense that it evaluates the behavior of the sensing system over some horizon of time (which is specified beforehand). The simulator requires as inputs the current observables together with samples of unobservables from the particle filter (to specify initial conditions) and a candidate action. The output of the simulator is a Q-value corresponding to

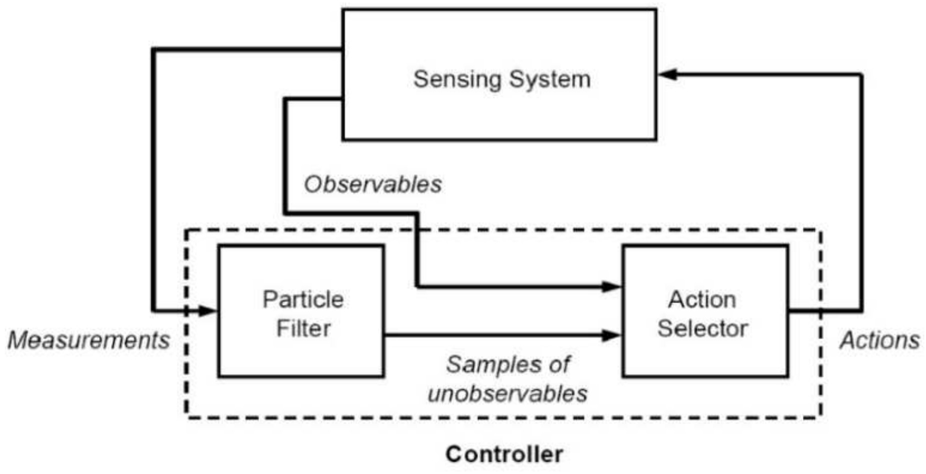


Figure 5.5. Basic control architecture with particle filtering and rollout.

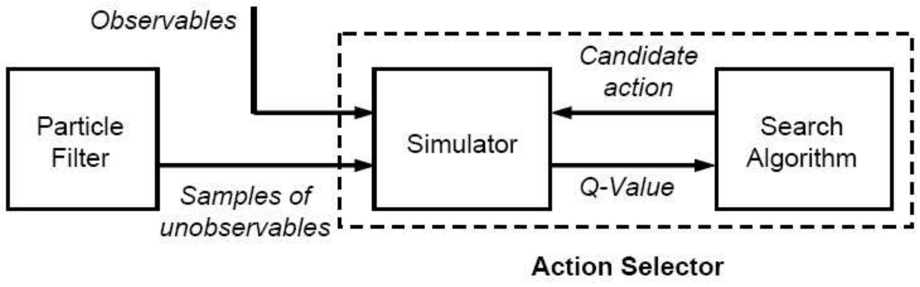


Figure 5.6. Components of the action selector.

the current measurements and observables, for the given candidate action. The output of the simulator represents the mean performance of applying the base policy, depending on the nature of the objective function. For example, the performance measure of the system may be the negative mean of the sum of the cumulative tracking error and the sensor usage cost over a horizon of H time steps, given the current system state and candidate action.

To elaborate on exactly how the Q-value approximation using rollout is implemented, suppose we are given the current observables and a set of samples of the unobservables (from the particle filter). The current observables together with a single sample of unobservables represent a candidate current underlying state of the sensing system. Starting from this candidate current state, we simulate the application of the given candidate action (which then leads to a random next state), followed by application of the base policy for the remainder of the time horizon—during this time horizon, the system state evolves according to the dynamics of the sensing system as encoded within the simulation model. For this single simulation run, we compute the "action utility" of the system (e.g., the negative of the sum of the cumulative tracking error and sensor usage cost over that simulation run). We do this for each sample of the unobservables, and then average over the performance values from these multiple simulation runs. This average is what we output as the Q-value.

The samples of the unobservables from the particle filter that are fed to the simulator (as candidate initial conditions for unobservables) may include all the particles in the particle filter (so that there is one simulation run per particle), or may constitute only a subset of the particles. In principle, we may even run multiple simulation runs per particle.

The above Monte Carlo method for approximating POMDP solutions has some beneficial features. First, it is flexible in that a variety of sensor management scenarios can be tackled using the same framework. This is important because of the wide variety of sensors that may be encountered in practice. Second, the method does not require analytical tractability; in principle, it is sufficient to simulate a system component, whether or not its characteristics are amenable to analysis. Third, the framework is modular in the sense that models of individual system components (e.g., sensor types, target motion) may be treated as "plug-in" modules. Fourth, the approach integrates naturally with existing simulators (e.g., Umbra [100]). Finally, the approach is inherently non-myopic, allowing the trade-off of short-term gains for long-term rewards.

6.10 Belief-state simplification

If we apply the method of rollout to a POMDP, we need a base policy that maps belief states to actions. Moreover, we need to simulate the performance of this policy—in particular, we have to sample future belief states as the system evolves in response to actions resulting from this policy. Because belief states are probability distributions, keeping track of them in a simulation is burdensome.

A variety of methods are available to approximate the belief state. For example, we could simulate a particle filter to approximate the evolution of the belief state (as described in Chapter 5), but even this may be unduly burdensome. As a further simplification, as in Chapter 11 we could use a Gaussian approximation and keep track only of the mean and covariance of the belief state using a Kalman filter or any of its extensions, including *extended Kalman filters* and *unscented Kalman filters* [125]. Naturally, we would expect that the more accurate the approximation of the belief state, the more burdensome the computation.

An extreme special case of the above tradeoff is to use a delta distribution for belief states in our simulation of the future. In other words, in our lookahead simulation, we do away with keeping track of belief states altogether and instead simulate only a *completely observable* version of the system. In this case, we need only consider a base policy that maps underlying states to actions—we could simply apply rollout to this policy, and not have to maintain any belief states in our simulation. Call this method *completely observable (CO) rollout*. It turns out that in certain applications, such as in sensor scheduling for target tracking, a CO-rollout base policy is naturally available (see [109, 110]). Note that we will still need to keep track of (or estimate) the actual belief state of the system, even if we use CO rollout. The benefit of CO rollout is that it allows us to avoid keeping track of (simulated) belief states in our *simulation* of the future evolution of the system.

In designing lookahead methods with a simplified belief state, we must ensure that the simplification does not hide the good or bad effects of actions. In other words, we need to make sure that the resulting Q-value approximation properly ranks current actions. This requires a carefully designed simplification of the belief state together with a base policy that appropriately reflects the effects of taking specific current actions.

For example, suppose that a particular current action results in poor future rewards because it leads to belief states with large variances. Then, if we use the method of CO rollout, we have to be careful to ensure that this detrimental effect of the particular current action be reflected as a cost in the lookahead.

(Otherwise, the effect would not be accounted for properly, because in CO rollout we do not keep track of belief states in our simulation of the future effect of current actions.)

Another caveat in the use of simplified belief states in our lookahead is that the resulting rewards in the lookahead may also be affected (and this may have to be taken into account). For example, consider again the problem of sensor scheduling for target tracking, where the per-step reward is the negative mean of the sum of the tracking error and the sensor usage cost. Suppose that we use a particle filter for tracking (i.e., for keeping track of the actual belief state). However, for our lookahead, we use a Kalman filter to keep track of future belief states in our rollout simulation. In general, the tracking error associated with the Kalman filter is different from that of the particle filter. Therefore, when summed with the sensor usage cost, the relative contribution of the tracking error to the overall reward will be different for the Kalman filter compared to the particle filter. To account for this, we will need to scale the tracking error (or sensor usage cost) in our simulation so that the effect of current actions are properly reflected in the Q-value approximations from the rollout with the simplified belief state calculation.

6.11 Reward surrogation

In applying a POMDP approximation method, it is often useful to substitute the reward function for an alternative (*surrogate*), for a number of reasons. First, we may have a surrogate reward that is much simpler (or more reliable) to calculate than the actual reward (see, e.g., the method of reduction to classification in Section 4 of Chapter 4). Second, it may be desirable to have a single surrogate reward for a range of different actual rewards. For example, information gain is often useful as a surrogate reward in sensing applications, taking the place of a variety of detection and tracking metrics (see Chapter 4). Third, reward surrogation may be necessitated by the use of a belief-state simplification technique. For example, if we use a Kalman filter to update the mean and covariance of the belief state, then the reward can only be calculated using these entities (see Chapter 11).

The use of a surrogate reward can lead to many benefits. But some care must be taken in the design of a suitable surrogate reward. Most important is that the surrogate reward be sufficiently reflective of the true reward that the ranking of actions with respect to the approximate Q-values be preserved. A superficially benign substitution may in fact have unanticipated but significant impact on the ranking of actions. For example, recall the example raised in the previous section on belief-state simplification, where we substitute the tracking error of a particle filter for the tracking error of a Kalman filter. Superficially, this substitute appears to be hardly a "surrogate" at all. However, as pointed out before, the tracking error of the Kalman filter may be significantly different in magnitude from that of a particle filter.

7. Simulation Result

In this section, we illustrate the performance of several of the strategies discussed in this chapter on a common model problem. The model problem has been chosen to have the characteristics of the motivating example given earlier, while remaining simple enough so that the workings of each method are transparent.

In the model problem, there are two targets, each of which is described by a one-dimensional position x. The sensor may measure any one of the 16 cells, which span the possible target locations (see Figure 6.7). The sensor makes three (not necessarily distinct) measurements per time step, receiving binary returns independent from dwell to dwell. In occupied cells, a detection is received with probability P_d (set here at 0.9). In cells that are unoccupied a detection is received with probability P_f (set here at 0.01). At the onset, positions of the targets are known only probabilistically. The belief state for the first target is uniform across sensor cells $\{2 \cdots 6\}$ and for the second target is uniform across sensor cells $\{11 \cdots 15\}$.

The visibility of the cells changes with time to emulate the motivating example. At time 1, all cells are visible to the sensor. At times 2, 3, and 4, cells $\{11 \cdots 15\}$ become obscured. At time 5, all cells are visible again. This model problem reflects the situation where a target is initially visible to the sensor, becomes obscured, and then reemerges from the obscuration.

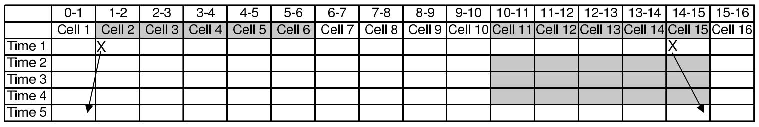


Figure 5.7. The model problem. At the onset, the belief state for target 1 is uniformly distributed across cells $\{2 \cdots 6\}$ and the belief state for target 2 is uniformly distributed across cells $\{11 \cdots 15\}$. At time 1 all cells are visible. At times 2, 3, and 4, cells $\{11 \cdots 15\}$ are obscured. This emulates the situation where one target is initially visible to the sensor, becomes obscured, and then reemerges (Figure 4 from [144] which is ©2004 IEEE - used with permission).

At time 1 a myopic strategy, having no information about the future visibility, will choose to measure cells uniformly from the set $\{2 \cdots 6\} \cup \{11 \cdots 15\}$ as they all have the same expected immediate reward. As a result, target 1 and target 2 will on the average be given equal attention. A non-myopic strategy, on the other hand, will choose to measure cells from $\{11 \cdots 15\}$ as they are soon to become obscured. That is, the policy of looking for target 2 at time 1 followed by looking for target 1 is best.

Figure 6.8 shows the performance of several of the on-line strategies discussed in this chapter on this common model problem. The performance of each scheduling strategy is measured in terms of the mean squared tracking error at each time step. The curves represent averages over $10,000$ realizations of the model problem. Each realization has randomly chosen initial positions of the targets and measurements corrupted by random mistakes as discussed above.

The performance of five different policies is given in Figure 6.8. These policies are described as follows.

- A **random** policy that simply chooses one of the 16 cells randomly for interrogation. This policy provides a worst-case performance and will bound the performance of the other policies.

- A **myopic** policy that takes the action expected to maximize immediate reward. Here the surrogate reward is information gain (see Chapter 4), so the value of an action is estimated by the amount of information it gains. The myopic policy is sub-optimal because it does not consider the long term ramifications of its choices. In particular, at time 1 the myopic strategy has no preference as to which target to measure because both are unobscured and have uncertain position. Therefore, half of the time, target 1 is measured, resulting in an opportunity cost because target 2 is about to disappear.

- The **heuristic EVTG approximation** described in Section 6.4. This policy gives weight to actions expected to be more valuable now than in the future. In particular, actions that correspond to measuring target 2 are given additional value because target 2 is predicted to be obscured in the future. This causes the relative ranking of actions corresponding to measuring target 2 higher than those corresponding to measuring target 1. For this reason, this policy (like the other non-myopic approximations described next) significantly outperforms the myopic strategy. This method has computational burden on the order of T times that of the myopic policy, where T is the horizon length.

- The **rollout** policy described in Section 6.7. The base policy used here is to point the sensor where the target is expected to be. This expectation is computed using the predicted future belief state, which requires the posterior (belief state) to be propagated in time. This is done using a particle filter to represent the posterior. We again use information gain as the surrogate metric to evaluate policies. The computational burden of this method is on the order of NT times that of the myopic policy, where T is the horizon length and N is the number of Monte Carlo trials used in the approximation (here $N = 25$).

- The **completely observable rollout** policy described in Section 6.10. The base policy here is also to point the sensor where the target is expected to be, but is modified to enforce the criterion that the sensor should alternate looking at the two targets. This slight policy modification is necessary due to the delta-function representation of the future belief state. Since the completely observable policy does not require predicting the posterior into the future, it is significantly faster than standard rollout (it is an order of magnitude faster in these simulations). However, completely observable rollout requires a different surrogate reward (one that does not require the posterior like the information gain surrogate metric does). Here we have chosen as a surrogate reward to count the number of detections received, discounting multiple detections of the same target.

8. Summary and Discussion

This chapter has presented approximation methods based on heuristics and simulation for partially observable Markov Decision Processes. We have highlighted via simulation on a simple model problem approaches based on rollout and a particular heuristic based on information gain. We have detailed some of the design choices that go into finding appropriate approximation, including choice of surrogate reward and belief-state representation.

Throughout this chapter we have taken special care to emphasize the limitations of the methods. Broadly speaking, all tractable methods require domain knowledge in the design process. Rollout methods require a base policy specially designed for the problem at hand; relaxation methods require one to identify the proper constraint(s) to remove; heuristic approximations require one to identify an appropriate approximation to the value-to-go function, and so on. That being said, when such domain knowledge is available it can often result in dramatic improvements in system performance over more traditional methods at a fixed computational cost.

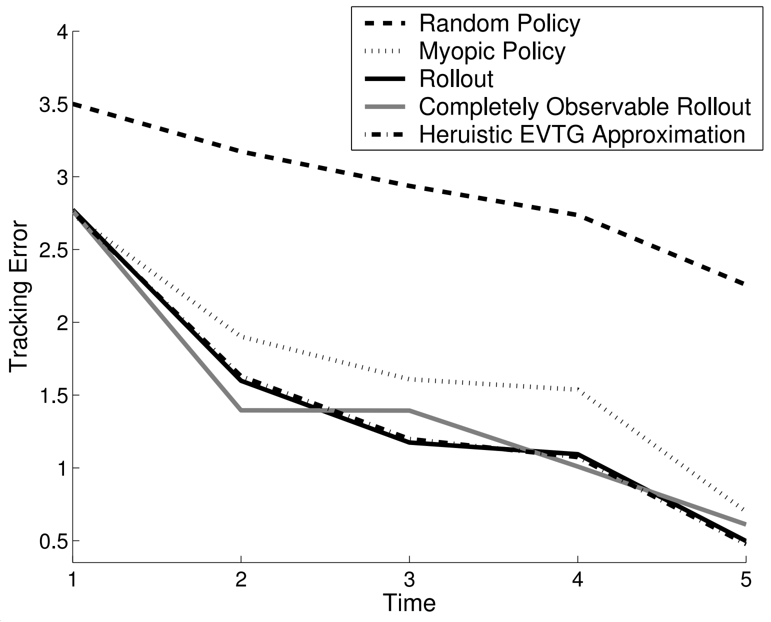


Figure 5.8. The performance of the five policies discussed above. Performance is measured in terms of mean squared tracking error at each time step, averaged over a large number of Monte Carlo trials.

The next two chapters describe the multi-armed bandit (MAB) framework for sensor management. The MAB model can be viewed as a relaxation of the general POMDP model that was the focus of the approximations in this chapter. Multi-target tracking applications will be revisited in Chapters 8 and 11.