20
Sequential Decision Making

The previous chapter on finite state machines provided an introduction into high-level robot behavior specification. Another important topic along this thread is *sequential decision making*[1], where the robot must make a series of decisions to accomplish an objective in an optimal way. This chapter provides an introduction to fundamental topics in decision making, including for problems where there is some uncertainty (e.g. uncertainty about the robot's state or about the environment).

[1] D. Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific, 2019

Sequential Decision Making

In addition to the motion planning and control problems discussed in earlier chapters (which focus on low-level tasks), there exist a broad range of situations where higher-level autonomous decision making is required. For example when deciding whether it is time for a self-driving car to cross an intersection, or whether a robot should first complete task A or task B. Two of the fundamental challenges associated with robotic decision making are that *sequences* of decisions must be made (which requires reasoning about future actions and observations) and that uncertainty may exist in the operating environment. This chapter presents a modeling framework for addressing decision making problems and will also introduce *dynamic programming*, a fundamental approach for solving these problems .

20.1 Deterministic Decision Making Problem

The standard mathematical formulation for decision making problems includes several components: a model of the robot's behavior, a set of admissible controls, and a cost function. This set of components is quite similar to the components used in trajectory optimization problems discussed in previous chapters, however decision making problems are generally represented in *discrete-time* rather than in *continuous-time*[2].

[2] There is a continuous-time formulation, known as the Hamilton–Jacobi–Bellman formulation.

In the deterministic decision making problem, the model of the robot is ex-

pressed in *discrete-time* as:

$$\boldsymbol{x}_{k+1} = f_k(\boldsymbol{x}_k, \boldsymbol{u}_k), \quad k = 0, \ldots, N-1, \tag{20.1}$$

where $\boldsymbol{x}$ is the robot's state, $\boldsymbol{u}$ is the control, f_k defines how the robot's state changes at time step k, and N is an integer that defines a finite planning horizon for the decision making problem. There are generally no restrictions on how the functions f_k are defined, they could come from a physics-based dynamics/kinematics model or even a higher-level state transition model similar to the finite state machine from the previous chapter.

It is also generally assumed that only some control actions are admissible at a given state, which denoted by the set $\mathcal{U}(\boldsymbol{x}_k)$. For example a car may only have an option to turn left or right when it is at an intersection. Therefore the control constraints for the robot at time step k are given by:

$$\boldsymbol{u}_k \in \mathcal{U}(\boldsymbol{x}_k). \tag{20.2}$$

Again, there are generally no restrictions on how the set of admissible control is defined. For example $\mathcal{U}(\boldsymbol{x}_k)$ could be a finite set of actions, it could be a convex region of allowable inputs, etc.

The cost function is assumed to be *additive*, and is defined as:

$$J(\boldsymbol{x}_0, \boldsymbol{u}_0, \ldots, \boldsymbol{u}_{N-1}) = g_N(\boldsymbol{x}_N) + \sum_{k=0}^{N-1} g_k(\boldsymbol{x}_k, \boldsymbol{u}_k), \tag{20.3}$$

where g_N is a terminal state cost function and g_k for $k = 0, \ldots, N-1$ are stage cost functions. These individual cost functions are also not restricted to a particular form (e.g. convex, differentiable, etc.).

Definition 20.1.1 (Deterministic Decision Making Problem)**.** *The deterministic decision making problem can be expressed for the system model* (20.1)*, control constraints* (20.2)*, and cost function* (20.3) *as:*

$$J^*(\boldsymbol{x}_0) = \min_{\boldsymbol{u}_k \in \mathcal{U}(\boldsymbol{x}_k),\ k=0,\ldots,N-1} J(\boldsymbol{x}_0, \boldsymbol{u}_0, \ldots, \boldsymbol{u}_{N-1}). \tag{20.4}$$

Notice that this problem is used to compute an *open-loop* control sequence $\{\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{N-1}\}$ given an initial condition $\boldsymbol{x}_0$, which is similar to the trajectory optimization problems seen in earlier chapters. However, this problem is generally quite hard to solve since there is no guarantee that the model (20.1) and cost function (20.3) have any particular structure that can be leveraged to make the optimization problem amenable to numerical optimization algorithms. While it is theoretically possible to solve the problem through a brute force search over all possible combinations of sequences $\{\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{N-1}\}$, this leads to a combinatorial explosion of options and is therefore not possible in practical settings (except of course for very small problems).

20.1.1 Principle of Optimality (Deterministic)

Fortunately, there is in fact an underlying structure to the deterministic decision making problem that can be leveraged to make the problem easier to solve. This structure is commonly referred to as the *principle of optimality*.

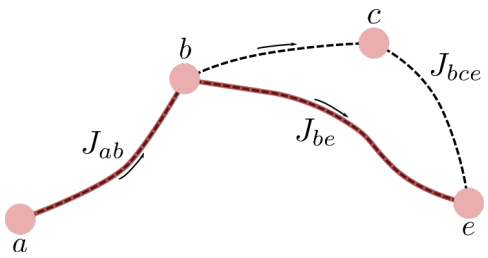

Figure 20.1: Starting from point a, let the red path $a - b - e$ be the optimal path from a to e, with a total cost of $J^*_{ae} = J_{ab} + J_{be}$. The principle of optimality in this case says that the path $b - e$ must therefore be the optimal path when starting from point b. This can be proven by contradiction, since if the path $b - c - e$ had a lower cost than path $b - e$ (i.e. $J_{bce} < J_{be}$), then the original path $a - b - e$ cannot be optimal!

The principle of optimality for deterministic systems is that for a sequence of optimal decisions, the *tail* of the optimal sequence is also optimal for a *tail subproblem*. For a concrete example see Figure 20.1. This can greatly simplify the overall problem, since you can "reuse" optimal paths for different scenarios. More formally, the principle of optimality is given by the following theorem:

Theorem 20.1.2 (Principle of Optimality (Deterministic)). *Let* $\{\boldsymbol{u}_0^*, \boldsymbol{u}_1^* \ldots, \boldsymbol{u}_{N-1}^*\}$ *be an optimal control sequence to the deterministic decision making problem* (20.4) *with a given initial condition* $\boldsymbol{x}_0^*$*, such that the resulting optimal state sequence is* $\{\boldsymbol{x}_0^*, \boldsymbol{x}_1^* \ldots, \boldsymbol{x}_N^*\}$*. Then, the tail sequence* $\{\boldsymbol{u}_k^*, \ldots, \boldsymbol{u}_{N-1}^*\}$ *is an optimal control sequence when starting from* $\boldsymbol{x}_k^*$ *and minimizing the cost from time k to time N*

$$J_{tail}(\boldsymbol{x}_k, \boldsymbol{u}_k, \ldots, \boldsymbol{u}_{N-1}) = g_N(\boldsymbol{x}_N) + \sum_{m=k}^{N-1} g_m(\boldsymbol{x}_m, \boldsymbol{u}_m).$$

To see how the principle of optimality can be applied to simplify the decision making problem, consider the scenario in Figure 20.2. In this case it is desired to find an optimal path from point b to point f, and it is assumed that optimal paths from c, d, and e to f are already known. A brute force search over all possible paths in this problem would require nine paths to be evaluated:

$$\{b-c-f, \quad b-c-d-f, \quad b-c-d-e-f, \quad b-d-c-f, \quad b-d-f,$$
$$b-d-e-f, \quad b-e-d-c-f, \quad b-e-d-f, \quad b-e-f\}.$$

However, by leveraging the principle of optimality the number of candidate paths is reduced to three:

$$b-c-f, \quad b-d-f, \quad b-e-f.$$

In other words, the principle of optimality allows the search to be performed over *immediate* decisions by also concatenating the optimal tail decisions! This procedure is generally implemented backward in time, for example in Figure 20.2 the point f (the goal) is first evaluated, then the points c, d, and e, and then finally the point b.

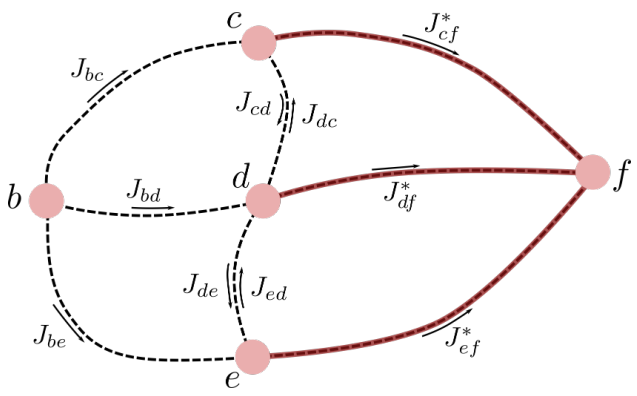

Figure 20.2: Suppose the optimal paths from points c, d and e to f are known (shown in red). By using the principle of optimality, an optimal path from point b to f can be found by *only* searching over paths from b to c, d, and e, and determining the lowest cost from the candidates $\{J_{bc} + J^*_{cf}, J_{bd} + J^*_{df}, J_{be} + J^*_{ef}\}$. In other words, the *optimal tails* can be leveraged to reduce the total number of paths that need to be considered when finding an optimal path from b to f!

20.1.2 *Dynamic Programming (Deterministic)*

The dynamic programming (DP) algorithm *globally* solves the deterministic decision making problem (20.4) by leveraging the principle of optimality[3]. The dynamic programming algorithm is given in Algorithm 16, where it can be seen that a backward-in-time recursion is used and at each step a *local* optimization is performed (this local optimization is referred to as the *Bellman* equation), leveraging the optimal *tail* costs from the previous iteration. The output

Algorithm 16: Dynamic Programming (Deterministic)

$J^*_N(\boldsymbol{x}_N) = g_n(\boldsymbol{x}_N)$, for all $\boldsymbol{x}_N$
for $k = N-1$ **to** 0 **do**
  $J^*_k(\boldsymbol{x}_k) = \min_{\boldsymbol{u}_k \in \mathcal{U}(\boldsymbol{x}_k)} g_k(\boldsymbol{x}_k, \boldsymbol{u}_k) + J^*_{k+1}(f_k(\boldsymbol{x}_k, \boldsymbol{u}_k))$, for all $\boldsymbol{x}_k$
return $J^*_0(\cdot), \ldots, J^*_N(\cdot)$

of the dynamic programming algorithm is a set of costs $J^*_k(\boldsymbol{x}_k)$ for each time step $k = 0, \ldots, N$ and states $\boldsymbol{x}_k$, which provide the optimal *tail* cost for the *tail* subproblem.

Given an initial condition $\boldsymbol{x}_0$, the optimal control sequence $\{\boldsymbol{u}^*_0, \ldots, \boldsymbol{u}^*_{N-1}\}$ that solves the deterministic decision making problem can be computed with a "forward pass", where:

$$\boldsymbol{u}^*_0 = \arg\min_{\boldsymbol{u}_0 \in \mathcal{U}(\boldsymbol{x}_0)} g_0(\boldsymbol{x}_0, \boldsymbol{u}_0) + J^*_1(f_0(\boldsymbol{x}_0, \boldsymbol{u}_0)).$$

The next state is then computed as $\boldsymbol{x}^*_1 = f_0(\boldsymbol{x}_0, \boldsymbol{u}^*_0)$, and the process is repeated:

$$\boldsymbol{u}^*_1 = \arg\min_{\boldsymbol{u}_1 \in \mathcal{U}(\boldsymbol{x}^*_1)} g_1(\boldsymbol{x}^*_1, \boldsymbol{u}_1) + J^*_2(f_1(\boldsymbol{x}^*_1, \boldsymbol{u}_1)),$$

until the full trajectory and optimal control is specified.

Note that in practice the DP algorithm is not practical for continuously values states $\boldsymbol{x}$, since an infinite number of states would have to be iterated over. Therefore one possible modification to handle continuously valued states is to

[3] Note that the principle of optimality is a fundamental property that is actually utilized in almost all decision making algorithms, including reinforcement learning.

quantize the state space into a finite set of states (other approaches, such as interpolation, are also possible). Also, it is interesting to note that the addition of control constraints can actually simplify the procedure, since it restricts the number of possible options that need to be considered!

Example 20.1.1 (Deterministic Dynamic Programming)**.** Consider the environment shown in Figure 20.3, where the goal is to start at point a and reach point h while incurring the smallest cost. In this problem the state is represented as the current location (i.e. a, b, etc.), and the control constraints are encoded by the arrows indicating possible directions of travel (e.g. at point c it is possible to either go right or up, but not down or left). The cost of traversing between two points is also denoted in Figure 20.3.

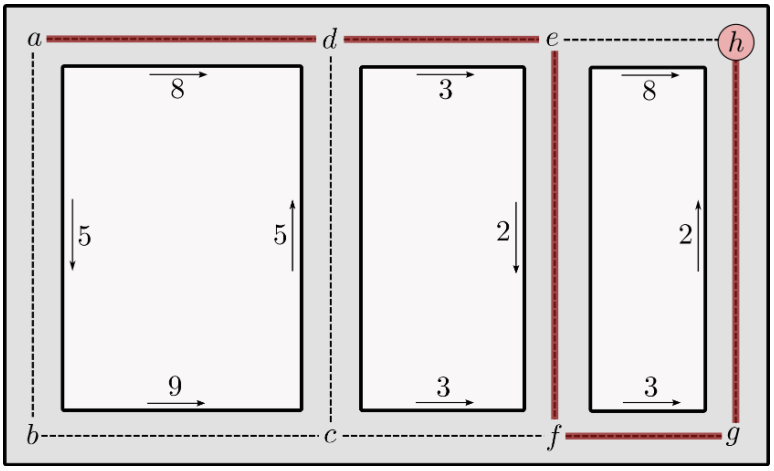

Figure 20.3: A deterministic decision making problem where the goal is to move from point a to point h while incurring the minimal amount of cost. The red path indicates the optimal path. This problem is solved by dynamic programming in Example 20.1.1.

To implement the DP algorithm, the final point h is chosen as $\boldsymbol{x}_N$, and the DP recursion begins with:

$$J_N^*(h) = 0,$$

since there is no cost to stay at point h. Moving backward in time, it can be seen that the possible states $\boldsymbol{x}_{N-1}$ that can transition to $\boldsymbol{x}_N = h$ are the points h, e, and g (assuming it is possible to stay at h with no cost). Therefore in the first step of the DP recursion:

$$\begin{aligned} J_{N-1}^*(h) &= 0 + J_N^*(h) = 0, \quad u_{N-1}^*(h) = \text{stay}. \\ J_{N-1}^*(e) &= 8 + J_N^*(h) = 8, \quad u_{N-1}^*(e) = \text{right}, \\ J_{N-1}^*(g) &= 2 + J_N^*(h) = 2, \quad u_{N-1}^*(g) = \text{up}, \end{aligned}$$

Note that $J_k^*(h) = 0$ for all $k \leq N$, and therefore it will not be explicitly included in the following steps. In the next step:

$$\begin{aligned} &J_{N-2}^*(e) = 8 + J_{N-1}^*(h) = 8, \quad u_{N-2}^*(e) = \text{right}, \\ &J_{N-2}^*(g) = 2, \quad u_{N-2}^*(g) = \text{up}, \\ &J_{N-2}^*(d) = 3 + J_{N-1}^*(e) = 11, \quad u_{N-2}^*(d) = \text{right}, \\ &J_{N-2}^*(f) = 3 + J_{N-1}^*(g) = 5, \quad u_{N-2}^*(f) = \text{right}, \end{aligned}$$

At this point, these optimal tail costs can be considered to be the optimal costs associated with control actions that lead from e, g, d, or f to the end point h in *two* steps! Continuing on:

$$
\begin{aligned}
&J^*_{N-3}(e) = \min\{8 + J^*_{N-2}(h),\, 2 + J^*_{N-2}(f)\} = 7, \quad u^*_{N-3}(e) = \text{down},\\
&J^*_{N-3}(g) = 2, \quad u^*_{N-3}(g) = \text{up},\\
&J^*_{N-3}(d) = 3 + J^*_{N-2}(e) = 11, \quad u^*_{N-3}(d) = \text{right},\\
&J^*_{N-3}(f) = 5, \quad u^*_{N-3}(f) = \text{right},\\
&J^*_{N-3}(a) = 8 + J^*_{N-2}(d) = 19, \quad u^*_{N-3}(a) = \text{right},\\
&J^*_{N-3}(c) = \min\{5 + J^*_{N-2}(d),\, 3 + J^*_{N-2}(f)\} = 8, \quad u^*_{N-3}(c) = \text{right}.
\end{aligned}
$$

Interestingly, it can be seen that it is now possible to accomplish the objective (i.e. go from point a to h) in 3 time steps (i.e. on path $a - d - e - f$) and incur an optimal cost of 19. However it turns out that an even lower cost is achievable if the number of time steps is increased further! Continuing the DP recursion:

$$
\begin{aligned}
&J^*_{N-4}(e) = 7, \quad u^*_{N-4}(e) = \text{down},\\
&J^*_{N-4}(g) = 2, \quad u^*_{N-4}(g) = \text{up},\\
&J^*_{N-4}(d) = 3 + J^*_{N-3}(e) = 10, \quad u^*_{N-4}(d) = \text{right},\\
&J^*_{N-4}(f) = 5, \quad u^*_{N-4}(f) = \text{right},\\
&J^*_{N-4}(a) = 8 + J^*_{N-3}(d) = 19, \quad u^*_{N-4}(a) = \text{right}\\
&J^*_{N-4}(c) = \min\{5 + J^*_{N-3}(d),\, 3 + J^*_{N-3}(f)\} = 8, \quad u^*_{N-4}(c) = \text{right},\\
&J^*_{N-4}(b) = 9 + J^*_{N-3}(c) = 17, \quad u^*_{N-4}(b) = \text{right},
\end{aligned}
$$

and finally with one more iteration:

$$
\begin{aligned}
&J^*_{N-5}(e) = 7, \quad u^*_{N-5}(e) = \text{down},\\
&J^*_{N-5}(g) = 2, \quad u^*_{N-5}(g) = \text{up},\\
&J^*_{N-5}(d) = 10, \quad u^*_{N-5}(d) = \text{right},\\
&J^*_{N-5}(f) = 5, \quad u^*_{N-5}(f) = \text{right},\\
&J^*_{N-5}(a) = \min\{8 + J^*_{N-4}(d),\, 5 + J^*_{N-4}(b)\} = 18, \quad u^*_{N-5}(a) = \text{right}\\
&J^*_{N-5}(c) = \min\{5 + J^*_{N-4}(d),\, 3 + J^*_{N-4}(f)\} = 8, \quad u^*_{N-5}(c) = \text{right},\\
&J^*_{N-5}(b) = 9 + J^*_{N-4}(c) = 17, \quad u^*_{N-5}(b) = \text{right}.
\end{aligned}
$$

Additional iterations are not included in this example because the costs and optimal decisions will no longer change with longer horizons (see for yourself!). Therefore it can be seen that with a sufficiently long horizon ($N \geq 5$), the optimal path from a to h is $a - d - e - f - g - h$ and incurs a cost of 18. Not this process has actually given a lot more information than what was original asked for. In particular, given *any* starting point and *any* horizon it is straightforward to generate an optimal control sequence! For example, if you wanted to start at point c and get to h in $N = 3$ steps you could immediately see that the optimal path is $c - f - g - h$ and the optimal cost is 8.

20.2 Stochastic Decision Making Problem

In the stochastic decision making problem it is assumed that there is some *uncertainty* in the robot's behavior or in the environment. This uncertainty is captured in the stochastic discrete-time robot model:

$$\boldsymbol{x}_{k+1} = f_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k), \quad k = 0, \ldots, N-1, \tag{20.5}$$

where $\boldsymbol{w}_k$ represents a stochastic disturbance term. Additionally, it is assumed that this disturbance has a known conditional probability distribution $P_k(\boldsymbol{w}_k \mid \boldsymbol{x}_k, \boldsymbol{u}_k)$. Note that it is assumed that the disturbance is only dependent on the current state $\boldsymbol{x}_k$ and control $\boldsymbol{u}_k$, and not states from earlier in the robot's history. This is another example of the Markov assumption, which was similarly used to develop the algorithms for localization and filtering in previous chapters.

Another main difference between the stochastic decision making problem and the deterministic problem is that a control *policy* is computed in the stochastic case. A control policy, usually denoted $\boldsymbol{u} = \pi(\boldsymbol{x})$, is a function that maps the state $\boldsymbol{x}$ to a control $\boldsymbol{u}$, and therefore defines a closed-loop controller (whereas in the deterministic setting an open-loop sequence was computed). Generally speaking, the search for control *policies* makes the problem more difficult to solve, but is typically required in stochastic settings because uncertainty would lead to undesirable behavior under open-loop control plans. Specifically, in the stochastic decision making problem the policies $\pi = \{\pi_0, \ldots, \pi_{N-1}\}$ are computed, which define the controls by $\boldsymbol{u}_k = \pi_k(\boldsymbol{x}_k)$.

Of course the cost function is also modified to handle the uncertainty. In particular, a *risk-neutral* formulation is used (i.e. minimize the cost *on average*), where the cost is defined by the *expected* value:

$$J_\pi(\boldsymbol{x}_0) = E_{\boldsymbol{w}}\left[g_N(\boldsymbol{x}_N) + \sum_{k=0}^{N-1} g_k(\boldsymbol{x}_k, \pi(\boldsymbol{x}_k), \boldsymbol{w}_k)\right], \tag{20.6}$$

where the expectation is over the stochastic variables $\boldsymbol{w}$. The stochastic decision making problem can now be stated as:

Definition 20.2.1 (Stochastic Decision Making Problem)**.** *The stochastic decision making problem can be expressed for the system model* (20.5)*, control constraints* (20.2)*, and cost function* (20.6) *as:*

$$J^*(\boldsymbol{x}_0) = \min_\pi \; J_\pi(\boldsymbol{x}_0). \tag{20.7}$$

20.2.1 Principle of Optimality (Stochastic)

The principle of optimality can again be applied in the stochastic setting, and the intuition is identical to the deterministic case (however the proof is slightly different because the reasoning is in terms of probability distributions). The principle of optimality in the stochastic setting is stated formally as:

Theorem 20.2.2 (Principle of Optimality (Stochastic)). *Let $\pi^* = \{\pi_0^*, \pi_1^* \ldots, \pi_{N-1}^*\}$ be an optimal policy for the stochastic decision making problem* (20.7)*, and assume the state $\boldsymbol{x}_k$ is reachable. Then, the tail policy sequence $\{\pi_k^*, \ldots, \pi_{N-1}^*\}$ is an optimal policy sequence when starting from $\boldsymbol{x}_k$ to minimize the cost from time k to time N.*

Again, by leveraging the principle of optimality the decision making problem can be simplified to making immediate decisions by concatenating optimal tail policies.

20.2.2 *Dynamic Programming (Stochastic)*

The dynamic programming algorithm for the stochastic setting is also quite similar to DP for deterministic problems, and is given in Algorithm 17. Once

Algorithm 17: Dynamic Programming (Stochastic)

$J_N(\boldsymbol{x}_N) = g_n(\boldsymbol{x}_N)$, for all $\boldsymbol{x}_N$
for $k = N-1$ **to** 0 **do**
  $J_k(\boldsymbol{x}_k) = \min_{\boldsymbol{u}_k \in \mathcal{U}(\boldsymbol{x}_k)} E_{\boldsymbol{w}_k}\big[g_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k) + J_{k+1}(f_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k))\big]$, for all $\boldsymbol{x}_k$
return $J_0(\cdot), \ldots, J_N(\cdot)$

Algorithm 17 is run, the optimal policy is defined by:

$$\pi_k^*(\boldsymbol{x}_k) = \arg\min_{\boldsymbol{u}_k \in \mathcal{U}(\boldsymbol{x}_k)} E_{\boldsymbol{w}_k}\big[g_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k) + J_{k+1}(f_k(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k))\big].$$

Example 20.2.1 (Stochastic Dynamic Programming). Consider an inventory control problem, where the available stock of a particular item is the state $x_k \in \mathbb{N}$, the ability to add to the inventory is the control $u_k \in \mathbb{N}$, and the demand for the item is a stochastic variable $w_k \in \mathbb{N}$. The dynamics of the available stock is modeled as:

$$x_{k+1} = \max\{0, x_k + u_k - w_k\},$$

which models the fact that demand reduces available stock but can also never be negative. Additionally, consider the control constraints:

$$x_k + u_k \leq 2,$$

which limits the amount of additional inventory that can be added based on the current available stock to ensure that $x_k \leq 2$. The demand w_k is assumed to be modeled probabilistically with a distribution:

$$p(w_k = 0) = 0.1, \quad p(w_k = 1) = 0.7, \quad p(w_k = 2) = 0.2.$$

Finally, the cost is given for a horizon of $N = 3$ as:

$$E\big[\sum_{k=0}^{2} u_k + (x_k + u_k - w_k)^2\big],$$

which penalizes ordering new stock at each time step and also having available stock at the next time step (i.e. having to store stock).

The dynamic programming algorithm can then be applied, starting with the end costs:

$$J_3(x_3) = 0,$$

and then recursively computing:

$$J_2(0) = \min_{u_2 \in \{0,1,2\}} E\left[u_2 + (u_2 - w_2)^2\right] = \min_{u_2 \in \{0,1,2\}} u_2 + 0.1u_2^2 + 0.7(u_2 - 1)^2 + 0.2(u_2 - 2)^2 = 1.3,$$

$$J_2(1) = \min_{u_2 \in \{0,1\}} E\left[u_2 + (1 + u_2 - w_2)^2\right] = 0.3,$$

$$J_2(2) = E\left[(2 - w_2)^2\right] = 1.1,$$

where the last cost is easily evaluated since the constraint makes $u_2 = 0$ the only feasible choice. The optimal stage policies associated with this step are:

$$\pi_2^*(0) = 1,$$
$$\pi_2^*(1) = 0,$$
$$\pi_2^*(2) = 0.$$

In the next step:

$$J_1(0) = \min_{u_1 \in \{0,1,2\}} E\left[u_1 + (u_1 - w_1)^2 + J_2(\max\{0, u_1 - w_1\})\right] = 2.5,$$

$$J_1(1) = \min_{u_1 \in \{0,1,\}} E\left[u_1 + (1 + u_1 - w_1)^2 + J_2(\max\{0, 1 + u_1 - w_1\})\right] = 1.5,$$

$$J_1(2) = E\left[(2 - w_1)^2 + J_2(\max\{0, 2 - w_1\})\right] = 1.68,$$

with optimal stage policies:

$$\pi_1^*(0) = 1,$$
$$\pi_1^*(1) = 0,$$
$$\pi_1^*(2) = 0.$$

Finally, in the last step:

$$J_0(0) = \min_{u_0 \in \{0,1,2\}} E\left[u_0 + (u_0 - w_0)^2 + J_1(\max\{0, u_0 - w_0\})\right] = 3.7,$$

$$J_0(1) = \min_{u_0 \in \{0,1,\}} E\left[u_0 + (1 + u_0 - w_0)^2 + J_1(\max\{0, 1 + u_0 - w_0\})\right] = 2.7,$$

$$J_0(2) = E\left[(2 - w_0)^2 + J_1(\max\{0, 2 - w_0\})\right] = 2.818,$$

with optimal stage policies:

$$\pi_0^*(0) = 1,$$
$$\pi_0^*(1) = 0,$$
$$\pi_0^*(2) = 0.$$

Interestingly, the best scenario occurs with an initial stock of one, rather than have no stock or too much stock. Also, the policy ends up being the same at all time steps: if you have no stock you add one item, otherwise you do nothing.

20.3 Challenges and Extensions of Dynamic Programming

Dynamic programming is a powerful algorithm, but suffers from several practical considerations: the "curse of dimensionality", the "curse of modeling", and the "curse of time". The curse of dimensionality arises because of a exponential growth of the computational and storage requirements based on the dimension of the state. For example if the state has dimension one (i.e. $\boldsymbol{x} \in \mathbb{R}$) and can take on 100 different values, then at each step of the algorithm the Bellman equation must be solved 100 times. While this may be possible from a practical perspective, if $\boldsymbol{x} \in \mathbb{R}^3$ this would lead to 100^3 solves of the Bellman equation! Additionally, extensions to the problems presented in this chapter where the full state is not *known* (e.g. because you can only measure some parts of the state), the problem also become intractable. The curse of modeling results from the complexity of modeling stochastic systems. In particular, it can be very hard to obtain expressions for transition probabilities for real world systems! Lastly, the curse of time is that the data of the problem may not be known ahead of time (such that the DP algorithm can be run offline). Therefore it may be required to solve the DP algorithm online when the data becomes available, or when the data changes and the problem needs to be resolved.

20.3.1 Reinforcement Learning

The practical challenges related to dynamic programming motivated the development of *suboptimal* dynamic programming approaches, which more commonly are referred to as *reinforcement learning* approaches. The goal of these approaches is to make *approximations* to the original problem that make it more practical for specific settings, such as with high-dimensional states, when the model is not known, and more. Broadly speaking, there are two main categories of approximations. The first category includes approximations in the value space (i.e. where the optimal cost function is approximated). The second category includes approximations in the policy space (i.e. where the policy is approximated by a neural network whose weights are optimized over).