B9120 Dynamic Programming Lecture 1 - 01/27/2020

Introduction to Dynamic Programming

Lecturer: Daniel Russo *Scribe: Judy Gan, Omar Mouchtaki*

- Formulation of Finite Horizon Problems.
- The Dynamic Porgramming Algorithm
- Optimal Stopping and the optimality of myopic policies

1 Motivating Examples

Shortest Path Problem: This first example is a simple deterministic problem that provides intuition for cenral ideas in Dynamic Programming. Consider the graph shown in Figure 1. The goal is to find the shortest path from the root s to any leaf in the set $\{C, D, E, F\}$.

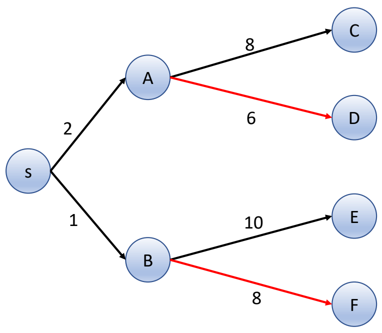

Figure 1: Shortest path problem. Numbers above the arrows indicate costs. Optimal intermediate paths are drawn in red.

A natural solution approach here would be to first find the shortest path from every intermediate node to any leaf node and then continue by backward iteration. In figure 1, notice that the shortest path from A to any of it's leaves is $A \to D$ with a cost of 6 while that from B is $B \to F$ with a cost of 8. The shortest path from S can then be found by comparing the costs of $S \to A \to D$ with a total cost of 8 (2 from $S \to A$ then 6 from $A \to D$) and the cost of $S \to B \to F$ with a total cost of 9 (1 from $S \to B$ then 8 from $B \to F$). In this case, the shortest path is $S \to A \to D$.

This simple example illustrates a key principle of dynamic programming for the shortest path problem: *the shortest path on a graph is the shortest path for every sub-graph.* In general, this is referred to as Bellman's principle of optimality. In Bellman's own words:

> An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

Inventory Control with backlogged demand: The previous problem was completely deterministic. Dynamic programming, however, can also be used to solve stochastic problems. Let's consider the problem of inventory control with stochastic demands. Starting from some initial inventory level, we need to optimize over ordering decisions at every time step, given the demand at each epoch is random. For this setting, we consider a fixed planning horizon and aim to minimize the cumulative expected sum of holding costs, backlogging costs, and ordering cost. We allow for demands to backlog, that is, unfulfilled demand at any time step is not lost. We also assume zero lead times - orders to replenish inventory made in any time period are filled in the same period itself. Denote,

- N: selling horizon,
- x_k: inventory in period k,
- $u_k \geq 0$: order in period k,
- $w_k \geq 0$: demand for the product in period k which we assume is independent and identically distributed from a known distribution.

The state evolution is given by,

$$x_{k+1} = x_k + u_k - w_k \quad \text{for } k = 0, 1 \ldots, N-1. \tag{1}$$

Note that as we allow the demand to backlog, there is no constraint on the inventory to be positive. Our objective is to minimize the following cost function,

$$\mathbb{E}_{\{w_k\}}\Big[\sum_{k=1}^{N-1} \underbrace{hx_k^+}_{\text{holding cost}} + \underbrace{bx_k^-}_{\text{backlogging cost}} + \underbrace{cu_k}_{\text{ordering cost}} - \underbrace{R(x_N)}_{\text{salvage value}}\Big], \tag{2}$$

where the expectation is taken over the random demand. An important question here is to ask: *what exactly are we minimizing over?* We distinguish between two types of solution strategies.

- **Open-loop control:** In this setting we optimize over the decisions $\{u_0, \ldots, u_{N-1}\}$ apriori and fix them throughout the selling season.
- **Close-loop control:** Here, decisions are made sequentially relying on historical data. Formally, we minimize over policies μ_k that take action u_k after observing all the information up to epoch k including x_k. Essentially we are searching over the set of policies, $\pi = (\mu_0, \mu_1, \ldots, \mu_{N-1})$, which take action $u_k = \mu_k(x_k)$.

2 Formulation of Finite Horizon Optimal Control Problems

In this section we present a general framework for modeling a class of dynamic stochastic problems. Assume the states, $\{x_k\}$, evolve as the following dynamic system:

$$x_{k+1} = f_k(x_k, u_k, w_k) \quad \text{for } k = \{0, 1, \ldots, N-1\}. \tag{3}$$

Here, S_k represents the state space with $x_k \in S_k$ being the state of the system at time k. We take $U(x)$ to be the space of allowable actions in state x and $u_k \in U(x_k)$ to be the control action selected at state x_k. Finally, $\{w_k\}$ denotes the sequence of random disturbances which we assume to be independent. The goal then is to solve the following optimization problem:

$$\min_{\pi} \ \mathbb{E}_{\pi}\Big[\sum_{k=1}^{N-1} g_k(x_k, u_k, w_k) + g_N(x_N)\Big] \tag{4}$$

where $g_k(\cdot)$ denote the instantaneous costs incurred at each time step k[1]. We consider the closed loop setting where the minimization in problem (4) is over deterministic Markov policies $\pi = (\mu_0, \dots, \mu_n)$ with $\mu(x_k) \in U(x_k)$. That the set of such policies suffice to optimally solve this problem is a basic result in dynamic programming.

3 The Dynamic Programming Algorithm

We present in this section a simple but powerful algorithm to solve the problem described above. We first define some basic quantities below.

Cost-to-go functions. For any policy $\pi = (\mu_0, \dots, \mu_{N-1})$, let $\pi^k = (\mu_k, \dots, \mu_{N-1})$ denote the "tail policy" (i.e. the policy from k^{th} period onwards). For each epoch k, we define the "cost-to-go" under policy π as:

$$J_k^{\pi^k}(x) := E^{\pi^k}\Big[\sum_{i=k}^{N-1} g_i(x_i, u_i, w_i) + g_N(x_N)\Big| x_k = x\Big],$$

for every starting state $x \in S_k$. The optimal cost-to-go function for every period k and for all $x \in S_k$ can then be defined as:

$$J_k^*(x) = \inf_{\pi^k} J_k^{\pi^k}(x).$$

For simplicity, we assume throughout the state and action spaces to be finite.

Remark *In this formulation, the policy with minimal cost-to-go seems to depend on the initial state x. But we will show that there is a single policy that is optimal simultaneously for every starting state.*

The dynamic programming algorithm is

- Initialize: $J_N(x) = g_N(x) \quad \forall\, x \in S_N$.
- For each $k \in \{N-1, \dots, 0\}$,

$$J_k(x) = \min_{u \in U(x)} \mathbb{E}[g_k(x, u, w_k) + J_{k+1}(f_k(x, u, w_k))] \quad \forall\, x \in S_k. \tag{5}$$

We now argue why the dynamic programming algorithm shown above gives an optimal policy. Consider the following proposition.

Proposition 1. *For the finite horizon control problem presented above, we have that*

$$J_k = J_k^*, \quad \forall\ k \in \{0, 1, \dots, N\}, \tag{6}$$

and by considering $\mu_k^*(x)$ that achieves the minimum in Equation (5), we can define an optimal policy $\pi^* = (\mu_0^*, \dots, \mu_{N-1}^*)$, which satisfies that for each period k,

$$J_k^{\pi_k^*}(x) = J_k^*(x) \quad \forall x \in S_k.$$

Before proving Proposition 1, we first introduce the following lemma.

Lemma 2 (Dynamic Programming Recursion for Policy Evaluation)**.** *For any policy $\pi = (\mu_0, ..., \mu_{N-1})$,*

$$J_k^{\pi^k}(x) = \mathbb{E}[g_k(x, \mu_k(x), w_k) + J_{k+1}^{\pi^{k+1}}(f_k(x, \mu_k(x), w_k))]. \tag{7}$$

[1] We allow $g(\cdot)$ to generally depend on (x_k, u_k, w_k).

Proof. Define $c_i = g_i(x_i, \mu_i(x_i), w_i)$ for $\{i = k, k+1, \dots, N-1\}$ and $c_N = g_N(x_N)$. By the definition of the cost-to-go function under the "tail policy" π^k,

$$J_k^{\pi^k}(x) = \mathbb{E}[c_k + \cdots + c_N | x_k = x].$$

Using the tower property of conditional expectation, we get

$$\begin{aligned}
\mathbb{E}[c_k + \cdots + c_N | x_k = x] &= \mathbb{E}[\mathbb{E}[c_k + c_{k+1} + \cdots + c_N | x_{k+1}, x_k] | x_k = x] \\
&= \mathbb{E}[c_k + \mathbb{E}[c_{k+1} + \cdots + c_N | x_{k+1}, x_k] | x_k = x] \\
&= \mathbb{E}[c_k + J_{k+1}^{\pi^{k+1}}(x_{k+1}) | x_k = x] \\
&= \mathbb{E}[g_k(x, \mu_k(x), w_k) + J_{k+1}^{\pi^{k+1}}(f_k(x, \mu_k(x), w_k))]
\end{aligned}$$

where the second last equality uses the definition of $J_{k+1}^{\pi^{k+1}}(x_{k+1})$ and the last equality uses the definition of c_k and the state evolution recursion given in Equation (3). □

We use Lemma 2 to prove Proposition 1.

Proof of Proposition 1. Since we don't perform any action at the last period N, for any policy $\pi = (\mu_0, \cdots, \mu_N)$, let $J_N^{\pi^N}(x) = J_N(x) = g_N(x)\ \forall x \in S_N$. Recall that $\pi^k(\cdot)$ denotes the tail policy starting period k. For k = N-1,

$$\begin{aligned}
J_{N-1}^{\pi^{N-1}}(x) &= \mathbb{E}[g_{N-1}(x, \mu_{N-1}(x), w_{N-1}) + J_N^{\pi^N}(f_{N-1}(x, \mu_{N-1}(x), w_{N-1}))] && \text{(by Lemma (2))} \\
&= \mathbb{E}[g_{N-1}(x, \mu_{N-1}(x), w_{N-1}) + J_N(f_{N-1}(x, \mu_{N-1}(x), w_{N-1}))] && \text{(by } J_N^{\pi^N}(x) = J_N(x)) \\
&\geq \min_{u \in U_{N-1}(x)} \mathbb{E}[g_{N-1}(x, u, w_{N-1}) + J_N(f_{N-1}(x, u, w_{N-1}))] \\
&= J_{N-1}(x) && \text{(by (5)).}
\end{aligned}$$

Recall that J_k is the cost-to-go function computed by the dynamic programming algorithm. Thus, for any policy π_{N-1}, we have $J_{N-1}^{\pi^{N-1}} \geq J_{N-1}$ (that is, the DP solution is lower bounds the cost-to-go function for any policy). Observe that $J_{N-1}^{\pi^{N-1}}(x) = J_{N-1}(x)$ when $\mu_{N-1}(x) = \mu_{N-1}^*(x)$. Next we will use induction to prove the rest of the Proposition 1. Suppose that $J_k^{\pi^k}(x) \geq J_k(x)$ and, that equality holds for the policy $(\mu_k^*, \cdots, \mu_{N-1}^*)$. Following the same argument, we can show that

$$\begin{aligned}
J_{k-1}^{\pi^{k-1}}(x) &= \mathbb{E}[g_{k-1}(x, \mu_{k-1}(x), w_{k-1}) + J_k^{\pi^k}(f_{k-1}(x, \mu_{k-1}(x), w_{k-1}))] \\
&\geq \mathbb{E}[g_{k-1}(x, \mu_{k-1}(x), w_{k-1}) + J_k(f_{k-1}(x, \mu_{k-1}(x), w_{k-1}))] \\
&\geq \min_{u \in U_{k-1}(x)} \mathbb{E}[g_{k-1}(x, u, w_{k-1}) + J_k(f_{k-1}(x, u, w_{k-1}))] \\
&= J_{k-1}(x).
\end{aligned}$$

Moreover, $J_{k-1}^{\pi^{k-1}}(x) = J_{k-1}(x)$ if $\mu_i(x) = \mu_i^*(x),\ \forall i \in \{k-1, \dots, N-1\}$. Hence we proved that for any policy, π, $J_k^{\pi^k}(x) \geq J_k(x),\ \forall k = \{0, \dots, N\}$, and the equality holds if $\pi = (\mu_0^*, \cdots, \mu_{N-1}^*)$. □

4 Curse of dimensionality and perspective on the class.

In principle, Proposition 1 already gives us the optimal policy for a very broad class of problems. Unfortunately, the algorithm requires iterating over and storing vectors J_k of length equal to the number of states in the problem. Due to a phenomenon known as the 'curse of dimensionality', most realistic problems involve enormous sate spaces that render this infeasible.

Consider the following simple example. A robot's sensors produce 50 measurements that are stored in a vector $x \in [0,1]^50$. (We think of x as the state of the system) Although the elements of x are continuous, for simplicity we can discretize them such that $x_i \in \{0.1, 0.2, \cdots, 1\}$. Then, the total number of states is 10^{50} and each J_k is a vector with dimension 10^{50}.

Since most of the dynamic programming problems can't be solved exactly due to the curse of dimensionality, this course will place an unusual emphasize two aspects aspects of DP:

1. Structural results: we'll look at some canonical dynamic programming problems in which problem specific structures simplify computation and offer insight.

2. Approximate algorithms: along with studying classic algorithms for tractable state spaces, we will emphasize ideas from the approximate dynamic programming and reinforcement learning literatures that sometimes succeed with enormous state spaces.

5 Optimal Stopping Problem: Asset selling example

One of the classic OR problems with special structure is the optimal stopping problem which has generated considerable interest in theory as well as practice (specially for applications in finance). For example, the price of an option is usually based on the optimal value that a person can extract if they exercise it at the optimal stopping time.

In this section, we will formulate an optimal stopping problem in an asset selling example. We will highlight the special structural results and use the DP algorithm to solve it exactly even though the state space is continuous.

Asset selling with irrevocable decisions: Assuming a discrete time setting, $k = \{0, 1, \ldots, N-1\}$, consider the setting where a person has an asset to sell by time N. At each epoch, they receive an offer w_k drawn independently from some distribution. The decision at every time epoch is to either accept the offer and invest the money at a fixed rate of interest $r > 0$ or reject and wait for the next offer with the goal to maximize the expected final revenue. Note, for this setting we assume that a rejected offer is lost.

We fix some notation. Let x_k to denote the state at time k. The state should have the information of whether the asset has been already sold or not (we cannot resell the asset) as well as the current offer. We incorporate this by defining the state in the following way:

$$x_{k+1} = \begin{cases} \text{sold} & \text{if } U_k = \text{Accept or } x_k = \text{sold} \\ w_k & \text{o.w.} \end{cases}$$

$\forall \{k = 0, \ldots, N-1\}$. Setting $x_0 = 0$ as a dummy variable, the state space is $S \subset \mathbb{R} \cup \{\text{sold}\}$. Similarly, the action space can be defined as:

$$U_k(x_k) = \begin{cases} \emptyset & \text{if } x_k = \text{sold} \\ \{\text{Accept, Reject}\} & \text{o.w.} \end{cases}$$

The revenue for each period is defined as,

$$g_k(x_k, u_k, w_k) = \begin{cases} 0 & \text{if } u_k \neq \text{Accept} \\ (1+r)^{N-k} x_k & \text{if } u_k = \text{Accept} \end{cases}$$

with the revenue for the final state being,

$$g_N(x_N) = \begin{cases} 0 & \text{if } x_N = \text{sold} \\ x_N & \text{o.w.} \end{cases}$$

DP Algorithm: Following the DP algorithm described in the previous section, set $J_N^*(x) = g_N(x)$. For $k = \{N-1, N-2, \ldots, 0\}$, set

$$J_k^*(x) = \begin{cases} \max\{(1+r)^{N-k}x, \mathbb{E}[J_{k+1}^*(w_k)]\} & \text{if } x \neq \text{sold} \\ 0 & \text{if } x = \text{sold} \end{cases}$$

Optimal Policy: Given the structure of the value-to-go functions, $J_k^*(x)$ (it is the maximum of the termination value $(1+r)^{N-k}x$ and the continuation value $\mathbb{E}[J_{k+1}^*(w_k)]$), the optimal policy can be easily computed as the following threshold policy:

$$\mu_k^*(x_k) |\, (x_k \neq \text{sold}) = \begin{cases} \text{Accept} & \text{if } x_k \geq \alpha_k \\ \text{Reject} & \text{if } x_k \leq \alpha_k \end{cases},$$

where the thresholds, $\alpha_k = \mathbb{E}[J_{k+1}^*(w_k)]/(1+r)^{N-k}$, depend on time k[2]. We remark that α_k's obey their own recursion. Since we must accept the last offer, $\alpha_N = -\infty$. For $k = \{N-1, \ldots, 0\}$,

$$\alpha_k = \frac{1}{1+r}\mathbb{E}[\max\{w_k, \alpha_{k+1}\}].$$

Another important point to note is that for an infinite horizon problem with i.i.d offers, the optimal policy is stationary and the optimal threshold α^* is the solution to the fixed point equation:

$$\alpha = \frac{1}{1+r}\mathbb{E}[\max\{w_1, \alpha\}]$$

Asset selling with offers retained We modify the previous setting with two changes.

- The offers $w_0, \ldots, w_{N-1}$ are i.i.d and non-negative.
- The rejected offers are not lost. At any period k, we can choose the highest offer received so far.

To accommodate this setting, we define the state such that,

$$x_{k+1} = \begin{cases} \text{sold} & \text{if } U_k = \text{Accept or } x_k = \text{sold} \\ \max\{x_k, w_k\} & \text{o.w.} \end{cases}$$

$\forall\, k = \{0, \ldots, N-1\}$. The action space and functions g_k's stay the same. Given this setting, we have the following proposition.

Proposition 3. *An optimal policy for asset selling with offers retained is a stationary policy, $\pi^* = (\mu^*, \mu^*, \cdots, \mu^*)$, where for $x \neq$ sold,*

$$\mu^*(x_k) = \begin{cases} \textit{Accept} & \textit{if } x_k \geq \frac{1}{1+r}\mathbb{E}_{w_k}[\max\{x_k, w_k\}] \\ \textit{Reject} & \textit{o.w.} \end{cases}.$$

An important point to note here is that as $\max\{x_k, w_k\} = x_{k+1}$, Proposition 3 shows that one-step look-ahead policy is optimal when the offers can be retained.

Proof. First define $S_k^* = \{x \,|\, x \geq (1+r)^{-(N-k)}\mathbb{E}[J_{k+1}^*(\max\{x, w_k\})]\}$, which is the set of offers that makes it optimal to stop at time k. Next we will prove some important monotonicity results. Clearly, for $x \neq$ sold, we can set $J_N^*(x) = x$. For $k = N-1$ and $x \neq$ sold,

$$\begin{aligned} J_{N-1}^*(x) &= \max\{(1+r)x, \mathbb{E}[\max\{w_{N-1}, x\}]\} \\ &\geq (1+r)x \\ &= (1+r)J_N^*(x). \end{aligned}$$

[2]This is intuitive as we need to sell by the end of the horizon.

By induction, assume that $J^*_{k+1}(x) \geq (1+r)J^*_{k+2}(x)$. Then,

$$\begin{aligned}
J^*_k(x) &= \max\{(1+r)^{N-k}x, \mathbb{E}[J^*_{k+1}(\max\{x, w_k\})]\} \\
&\geq \max\{(1+r)^{N-k}x, (1+r)\mathbb{E}[J^*_{k+2}(\max\{x, w_k\})]\} \\
&= (1+r)\max\{(1+r)^{N-(k+1)}x, \mathbb{E}[J^*_{k+2}(\max\{x, w_k\})]\} \\
&= (1+r)J^*_{k+1}(x).
\end{aligned}$$

Proof to be completed next class

$\square$