Course Notes On Dynamic Optimization (Fall 2023)
Lecture 1A: intro to DP

Instructor: Daniel Russo

Email: djr2174@gsb.columbia.edu

Graduate Instructor: David Cheikhi
Email: d.cheikhi@columbia.edu

September 9, 2023

These notes are based of scribed notes from a previous edition of the class. I have done some follow up light editing, but there may be typos or errors.

This course course is about decision-making across time and under uncertainty. Two core challenges distinguish this form other areas of optimization

1. **Inter-temporal tradeoffs:** the decision-maker may wish to forego immediate benefit now in order to position themselves to accrue greater rewards in the future.
2. **The role of information:** As time passes, new information is revealed to the decision-maker, and this may impact future decisions. The DM cannot commit to a specific sequence of actions upfront. Instead they select a policy: a contingent plan for how future actions will be selected depending on what new information is revealed in the interim.

As a rule of thumb, before deciding whether to approach a problem using the tools covered in this course, you should ask yourself whether (1) and (2) are essential ingredients.

The main intellectual aspect that distinguishes this topic from other areas of optimization is the role of information. When making a decision, the decision-maker must account for the fact that their actions now have an impact on ability to take succesful actions in the future and that

1 Initial Examples

Shortest Path Problem The first example considering a simple problem with deterministic dynamics . Let's consider the following tree. The question is how would we solve the shortest path problem going from root s to a leaf in $\{C, D, E, F\}$.

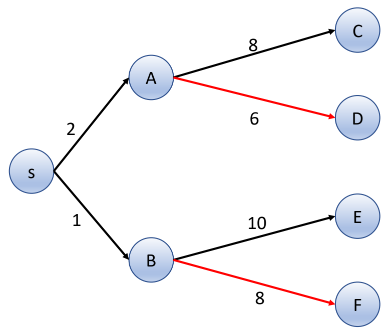

Figure 1: **Shortest path problem example:** optimal intermediate paths are drawn in red.

To do so, a natural approach would be to first find the shortest path from every intermediate node to leaves and continuing by backward iteration. In figure ??, going from A to a leaf can be done optimally with a cost of 6 while going from B to the leaf is done optimally with a cost of 8.

Finally, we get the optimal solution path by assessing whether it is better for us to incur a cost of 2, then 6 (by walking through A), or 1 then 8 (through B). The solution is clearly, by going through A and we obtain, that way, the optimal path from s to leaves.

This example can seem straightforward but it already uses a key principle that we will develop through out this lesson : *the shortest path should follow the shortest route on every subgraph.* This general principle is referred to as Bellman's principle of optimality.

Inventory Control with backlogged demand In the previous example, the problem was completely deterministic. Dynamic Programming usually enables us coping with uncertainty. Let's consider the problem of inventory where the demand is stochastic. We denote

- N : length of the selling horizon,
- x_k : inventory on hand at period k,
- $u_k \geqslant 0$: order of inventory at period k,
- $w_k \geqslant 0$: demand for the product at period k (assumed to be independent and identically distributed from a known distribution).

The state evolution is given by,

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

Our objective is to minimize the following cost function,

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

where the expectation is taken over w_k's.

An important question at this point is : *what are we minimizing over?* We distinguish between two types of problems:

- **Open-loop control:** decisions $u_0, \dots, u_{N-1}$ are fixed a priori and cannot be modified during the selling season.
- **Closed-loop control:** decisions are made sequentially and at the last possible moment. For instance in our problem, we choose u_k at epoch k. In this framework, our decisions can rely on historical data. Formally, we minimize over policies μ_k that decides what to do after observing x_k. Our general policy $\pi = (\mu_0, \mu_1, \dots, \mu_{N-1})$ enable us to generate the action u_k at epoch k according to the relation $u_k = \mu_k(x_k)$.

This class is almost exclusively focused on closed-loop control. The performance gap between the optimal closed-loop policy and the the optimal opten-loop one of often called the **value of information**. Most examples in this course are ones where the value of information is large.

2 Finite Horizon Optimal Control: general problem formulation

In this section we formulate a general framework modeling a class of dynamic stochastic problems. We study a controlled dynamic system of the form:

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

where,

- $x_k \in S_k$ represents the "state" of the system,
- $u_k \in U(x_k)$ is the "control" or the "action,
- w_k is a random disturbance. The family $(w_k)_{0 \leqslant k \leqslant N-1}$ is assumed to be independent.

The goal is to solve the optimization problem

$$\min_{\pi} \mathbb{E}^{\pi}\Big[\sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) + g_N(x_N)\Big]. \tag{4}$$

The minimization in problem (4) is over policies $\pi = (\mu_0, \dots, \mu_n)$ where $\mu(x_k) \in U(x_k)$[1].

Remark (The uncontrolled case). Suppose there is no control decision, i.e. that $U(x)$ is a singleton for each x. Then we can drop u from our notation and simply study the stochastic recursion

$$x_{k+1} = f_k(x_k, w_k).$$

Because the w_k's are independent, this defines a Markov process. If $f_k = f$ for each k, and law of w_k does not depend on the time period k, it is time-homogenous Markov process. In fact **Any Markov process could be written in this form.** It is mostly a stylistic choice whether one models the system by specifying transition probabilities or by specifying a system function f and distribution of the disturbances.

A similar comment applies to the controlled case.

Remark. Shou

[1]Such a policy is called a deterministic Markov Policy. That sufficiency of that such policies is a basic result in DP.

3 The Dynamic Programming Algorithm and cost-to-go functions

For any policy $\pi = (\mu_0, \dots, \mu_{N-1})$, denote by, $\pi^k = (\mu_k, \dots, \mu_{N-1})$ the "tail policy". For each epoch k we can now define the optimal remaining "tail cost" under policy π as follow. For every $x \in S_k$,

$$J_k^{\pi^k}(x) := E^{\pi}\Big[\sum_{i=k}^{N-1} g_i(x_i, u_i, w_i) + g_N(x_N)\Big| x_k = x\Big]. \tag{5}$$

The optimal cost-to-go is then defined for every period k and every $x \in S_k$ as

$$J_k^*(x) = \inf_{\pi^k} J_k^{\pi^k}(x). \tag{6}$$

Based on these notions we can now define and analyze the Dynamic Programming Algorithm. Assuming all state and action spaces are finite (to keep things simple), we initialize the following quantities,

$$J_N(x) = g_N(x) \quad \forall x \in S_N, \tag{7}$$

and we define by backward induction for k ranging from $N-1$ to 0,

$$J_k(x) = \min_{u \in U(x)} \mathbb{E}[g_k(x, u, w_k) + J_{k+1}(f_k(x, u, w_k))] \quad \forall x \in S_k. \tag{8}$$

Remark. Here it seems to define a multi-objective problem depending on x and k. But we will find a policy π that is optimal simultaneously for every subproblem.

We now argue why the DP algorithm gives us an optimal policy. To avoid dealing with technicalities, we make the following assumption.

Assumption 1. *(Overly restrictive regularity condition) The state space and action space are finite. For any k, s, and u, the random variable $g_k(x, u, w_k)$ is integrable.*

Proposition 1. *Under the Assumption 1,*

$$J_k = J_k^*, \quad \forall k, \tag{9}$$

and by considering $\mu_k^*(x)$ that achieves the minimum in the right hand side term of (8). We can define an optimal policy $\pi^* = (\mu_0^*, \dots, \mu_{N-1}^*)$. satisfying for each period k,

$$J_k^{\pi_k^*}(x) = J_k^*(x) \quad \forall x \in S_k. \tag{10}$$

Before proving the Proposition 1, we need to first introduce the following lemma. The lemma is instructive in its own right and reveals a connection between the DP recursion and the law of iterated expectations.

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

$$J_k^{\pi^k}(x) = \mathbb{E}[g_k(x, \mu_k(x), w_k) + J_{k+1}^{\pi^{k+1}}(f_k(x, \mu_k(x), w_k))] \tag{11}$$

.

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

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

By the tower property of conditional expectation,

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

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 equation (3). □

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

$$\begin{aligned}J_{N-1}^{\pi^{N-1}}(x) &= \mathbb{E}[g_{N-1}(x, \mu_{N-1}(x), w_{N-1}) + J_N^{\pi^N}(f_{N-1}(x, \mu_{N-1}(x), w_{N-1}))] && \text{(by (11))} \\ &= \mathbb{E}[g_{N-1}(x, \mu_{N-1}(x), w_{N-1}) + J_N(f_{N-1}(x, \mu_{N-1}(x), w_{N-1}))] && \text{(by } J_N^{\mu^N}(x) = J_N(x)\text{)} \\ &\geqslant \min_{u \in U_{N-1}(x)} \mathbb{E}[g_{N-1}(x, u, w_{N-1}) + J_N(f_{N-1}(x, u, w_{N-1}))] \\ &= J_{N-1}(x) && \text{(by (8)).}\end{aligned}$$

Observe that we will have $J_{N-1}^{\pi^{N-1}}(x) = J_{N-1}(x)$ when $\mu_{N-1}(x) = \mu_{N-1}^*(x)$. Next we will use induction to prove the rest of the Proposition 1. Suppose that $J_k^{\pi^k}(x) \geqslant J_k(x)$. Following the same argument, we can show that

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

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

4 Did we just solve dynamic optimization? On the curse of dimensionality

Curse of dimensionality. There are three main requirements to implementing the DP algorithm

1. **The ability to average over draws of the w_k:** A way to think about is the practice is, could you develop an (empirical) simulator that would allow you to start the system at some

arbitrary state, and simulate outcomes under a specified policy?

2. **the ability to optimize over u.** Implementing the DP algorithm requires solving one-period optimization problems (. . . with an objective function that, through the cost-to-go function, faithfully reflects the long-term implications of a decision.) This is never an issue if the set of possible actions is small, but can be a barrier otherwise.

3. **The ability to iterate over all possible states of the world.** This restriction turns out to be especially limiting. Even the state variable seems to be a succinct description of the state of the system, the number of possible settings of the state variable is often scales exponentially in relevant problem parameters.

   - Imagine a robotic sensor collects 50 measurements which we model as the state of the system. Assume value of each measurement $x_{k,i}$ has a continuous value between [0,1]. We discretize $x_{k,i}$ such that $x_{k,i} \in \{0.1, 0.2, \cdots, 1\}$. Then the total number of states is 10^{50}. Each J_k is a 10^{50} dimension vector, which is too large to store, much less compute.