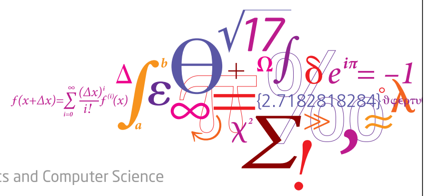
02465: Introduction to reinforcement learning and control

Dynamical Programming

Tue Herlau

DTU Compute, Technical University of Denmark (DTU)



DTU Compute
Department of Applied Mathematics and Computer Science

Lecture Schedule

Syllabus: `https://02465material.pages.compute.dtu.dk/02465public`
Help improve lecture by giving feedback on DTU learn

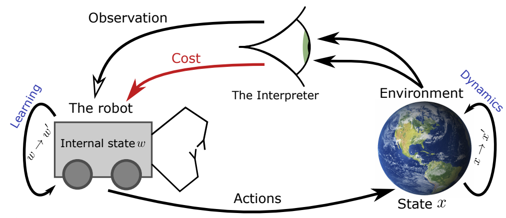
Reading material:

- [Her24, Chapter 5-6.2] Formalization of the decision problem and the DP algorithm

Learning Objectives

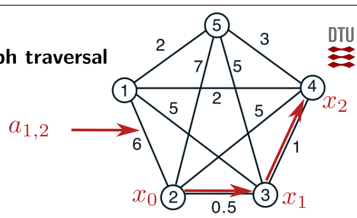
- Dynamical Programming
- Principle of optimality
- Optimal policy/value function using DP

The decision problem



State The configuration of the environment x
Action What we do u
Cost/reward A number which depends on the state and action

Example: Shortest path graph traversal



Find shortest path from starting node $x_0 = 2$ to final node $t = 5$

State Current node $x_k = 4$
Actions next possible node: $u_k \in \{1, 2, \dots, 5\}$
Dynamics Deterministic, known

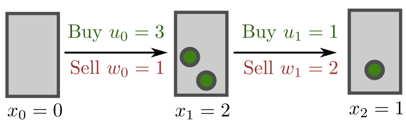
$$x_{k+1} = f(x_k = 4, u_k = 5) = 5$$

Cost Sum of edge weights

$$\sum_{k=0}^{N-1} a_{x_k, u_k} + \begin{cases} 0 & \text{if } x_N = t \\ \infty & \text{otherwise} \end{cases}$$

We want optimal path $\{2, 3, 4, 5\}$

Inventory control



- We order a quantity of an item at period $k = 0, \dots, N$ so as to meet a stochastic demand

  x_k stock available at the beginning of the kth period,
  $u_k \geq 0$ stock ordered (and immediately delivered) at the beginning of the kth period.
  $w_k \geq 0$ Demand during the k'th period

- Dynamics: $x_{k+1} = x_k + u_k - w_k$
- Cost per new unit c; cost to hold x_k units is $r(x_k)$

$$r(x_k) + cu_k$$

- Select actions $u_0, \dots, u_{N-1}$ to minimize cost

We want proven optimal rule for ordering

Basic control setup: Environment dynamics

Finite time Problem starts at time 0 and terminates at *fixed* time N. Indexed as $k = 0, 1, \ldots, N$.

State space The states x_k belong to the **state space** $\mathcal{S}_k$

Control The available controls u_k belong to the **action space** $\mathcal{A}_k(x_k)$, which may depend on x_k

Dynamics

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

Disturbance/noise A random quantity w_k with distribution

$$w_k \sim P_k(W_k | x_k, u_k)$$

Cost and control

Agent observe x_k, agent choose u_k, environment generates w_k

Cost At each stage k we obtain cost

$$g_k(x_k, u_k, w_k), \quad k = 0, \ldots, N-1 \quad \text{and} \quad g_N(x_k) \text{ for } k = N.$$

Action choice Chosen as $u_k = \mu_k(x_k)$ using a function $\mu_k : \mathcal{S}_k \to \mathcal{A}_k(x_k)$

$$\mu_k(x_k) = \{\text{Action to take in state } x_k \text{ in period } k\}$$

Policy The collection $\pi = \{\mu_0, \mu_1, \ldots, \mu_{N-1}\}$

Rollout of policy Given x_0, select $u_k = \mu_k(x_k)$ to obtain a **trajectory** $x_0, u_0, x_1, \ldots, x_N$ and **accumulated cost**

$$g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k)$$

Expected cost/value function

Expected cost Given π, x_0 it is the average cost of all trajectories:

$$J_\pi(x_0) = \mathbb{E}\left[g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k)\right]$$

Optimal policy Given x_0, an optimal policy π^* is one that minimizes the cost

$$\pi^*(x_0) = \arg\min_{\pi=\{\mu_0,\ldots,\mu_{N-1}\}} J_\pi(x_0)$$

Optimal cost function The optimal cost, given x_0, is denoted $J^*(x_0)$ and is defined as

$$J^*(x_0) = \min_{\pi=\{\mu_0,\ldots,\mu_{N-1}\}} J_\pi(x_0)$$

J_π is the key quantity in control/reinforcement learning

Open versus closed loop

Our goal is to find the policy π which minimize:

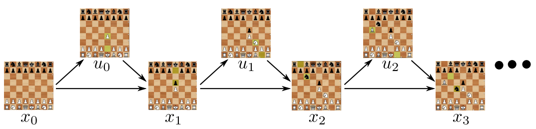
$$J_\pi(x_0) = \mathbb{E}\left[g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k)\right]$$

Closed-loop minimization Select u_k last-minute as $u_k = \mu_k(x_k)$ when information x_k is available

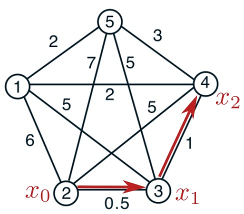
Open-loop minimization Select actions $u_0, \ldots, u_{N-1}$ at $k = 0$

- Open-loop minimization is simpler

Open or closed loop



- If environment is stochastic, we need a closed-loop controller
- If environment is deterministic, we know the position x_k with certainty given $u_0, \ldots, u_{k-1}$. Therefore, there is no advantage in delaying choice



Quiz: Chess and DP

Suppose the game of chess was formulated as dynamical programming (N, $\mathcal{S}_k$, $\mathcal{A}_k$, etc.) with the intention of obtaining a good policy μ_k using dynamical programming.
This will lead to several practical problems, however, focusing just on the potential problems listed below, which one will be a main obstacle?

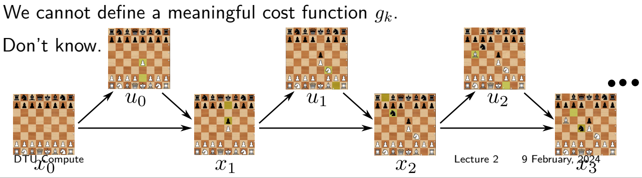
a. The policy function μ_k will require too much memory to store

b. Given a state x_k, it is not practical to define the action spaces $\mathcal{A}_k(x_k)$

c. It will require too much space to store the state space $\mathcal{S}_2$.

d. We cannot define a meaningful cost function g_k.

e. Don't know.



Summary: Discrete stochastic decision problem

- The states are $x_0, \dots, x_N$, and the controls are $u_0, \dots, u_{N-1}$
- $w_k \sim P_k(W_k = w_k | x_k, u_k)$, $k = 0, \dots, N-1$ are random disturbances
- The system evolves as

$$x_{k+1} = f_k(x_k, \mu_k(x_k), w_k), \quad k = 0, \dots, N-1$$

- At time k, the possible states/actions are $x_k \in S_k$ and $u_k \in \mathcal{A}_k(x_k)$
- Policy is a sequence of functions $\pi = \{\mu_0, \dots, \mu_{N-1}\}$, $\mu_k : S_k \mapsto \mathcal{A}_k(x_k)$
- The cost starting in x_0 is:

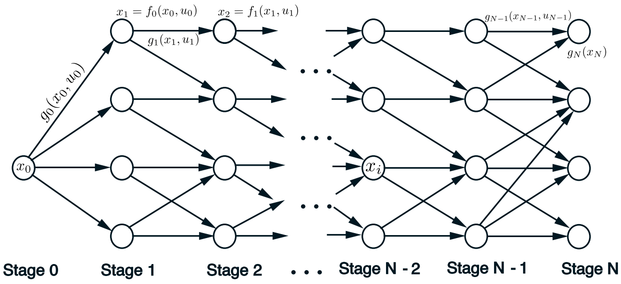
$$J_\pi(x_0) = \mathbb{E}\left[g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k) \right]$$

- **The control problem**: Given x_0, determine optimal policy by minimizing

$$\pi^*(x_0) = \underset{\pi = \{\mu_0, \dots, \mu_{N-1}\}}{\arg\min} J_\pi(x_0)$$

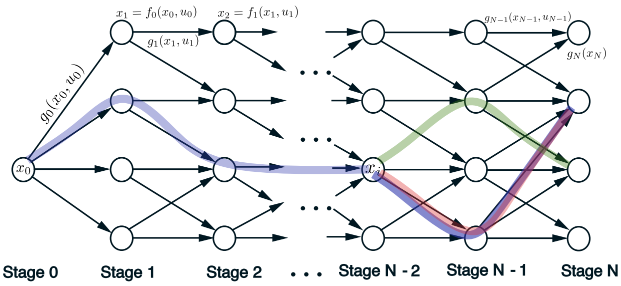
Graph representation

Starting in x_0, decision problem can be seen as traversing a graph



- Nodes are states, edges are possible transitions, cost is sum of edges
- In deterministic case, actions are edges and a policy is just a path

Principle of optimality (PO), deterministic case



The **blue line** is a path corresponding to an **optimal** policy

$$J^*(x_0) = J_{\pi^*}(x_0) = \min_\pi J_\pi(x_0)$$

Suppose at stage i optimal path $\pi^* = \left\{\mu_0^*, \mu_1^*, \dots, \mu_{N-1}^*\right\}$ pass through x_i

- **PO:** The **tail policy** $\left\{\mu_i^*, \mu_{i+1}^*, \dots, \mu_{N-1}^*\right\}$ is optimal from x_i to x_N
- **Why?** Suppose **alternative tail policy** $\{\mu_i', \dots, \mu_{N-1}'\}$ is better than

Definitions

For any policy $\pi = \{\mu_0, \mu_1, \dots, \mu_{N-1}\}$

- For any $k = 0, \dots, N-1$, $\pi^k = \{\mu_k, \mu_{k+1}, \dots, \mu_{N-1}\}$ is a **tail policy**
- For any x_k the **cost of the tail policy** is

$$J_{k,\pi}(x_k) = \mathbb{E}\left\{ g_N(x_N) + \sum_{i=k}^{N-1} g_i(x_i, \mu_i(x_i), w_i) \right\}$$

- And the **optimal cost of a tail policy** starting in x_k

$$J_k^*(x_k) = \min_{\pi^k} J_{k,\pi_k}(x_k)$$

- Note that $J_0^*(x_0) = J^*(x_0)$

Proof of PO in deterministic case

$$\begin{aligned}
J_{\pi^*}(x_0) &= g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k^*(x_k)) = \\
&\left(\sum_{k=0}^{i-1} g_k(x_k, \mu_k^*(x_k))\right) + \left(g_N(x_N) + \sum_{k=i}^{N-1} g_k(x_k, \mu_k^*(x_k))\right) \\
&\geq \left(\sum_{k=0}^{i-1} g_k(x_k, \mu_k^*(x_k))\right) + g_N(x_N') + \sum_{k=i}^{N-1} g_k(x_k', \mu_k'(x_k')) \\
&= J_{\pi = (\mu_0, \dots, \mu_{i-1}, \pi_k')}
\end{aligned}$$

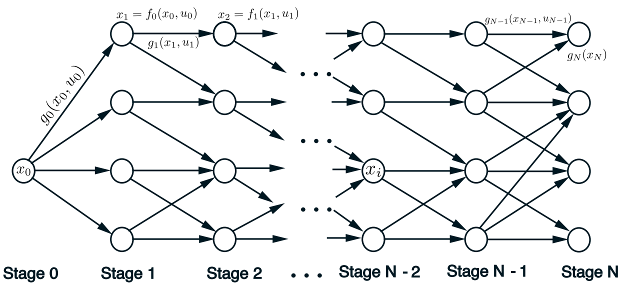
If the optimal tail policy π_i' had a lower tail cost than the tail of optimal policy this means:

$$g_N(x_N) + \sum_{k=i}^{N-1} g_k(x_k, \mu_k^*(x_k)) > g_N(x_N') + \sum_{k=i}^{N-1} g_k(x_k', \mu_k'(x_k'))$$

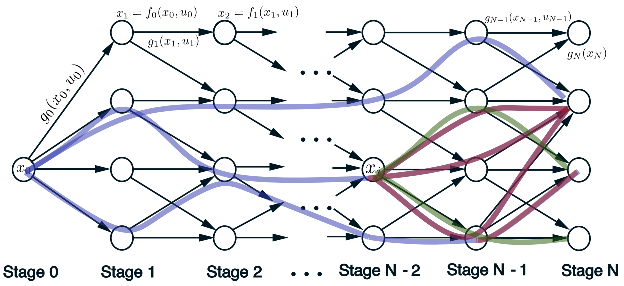
and so the combined policy $(\mu_0, \dots, \mu_{i-1}, \pi_i')$ would have lower cost than optimal policy π^*

The stochastic case

Consider the stochastic case. Trajectories are now random



The stochastic case



- Consider **tail policy** of π^*: $J_{i,\pi^*}(x_0)$
- Suppose **optimal tail policy** $J_i^*(x_i)$ is an improvement
- It seems true the combined policy is an improvement over π^* [Her24, appendix A]

Principle of optimality

Consider a general, stochastic/discrete finite-horizon decision problem

The principle of optimality

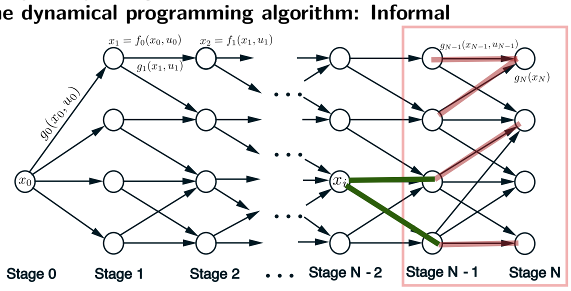
Let $\pi^* = \{\mu_0^*, \mu_1^*, \dots, \mu_{N-1}^*\}$ be an optimal policy for the problem, and assume that when using π^*, a given state x_i occurs at stage i with positive probability.
Suppose $\tilde{\pi}_k^*$ is the optimal tail policy obtained by minimizing the tail cost starting from x_i

$$J_{k,\pi}(x_i) = \mathbb{E}\left\{ g_N(x_N) + \sum_{i=k}^{N-1} g_i(x_i, \mu_i(x_i), w_i) \right\}.$$

Then the truncated policy $\{\mu_i^*, \mu_{i+1}^*, \dots, \mu_{N-1}^*\}$ of π^* is optimal for the tail problem

$$J_{k,\tilde{\pi}_{*,k}}(x_k) = J_{k,\pi^*}(x_k).$$

The dynamical programming algorithm: Informal



- Suppose we know the optimal tail policy at stage $k+1$ for all x_{k+1}
- Cost of optimal path π_k^* from k to N is the cost of optimal path $x_k \to x_{k+1}$ and then $x_{k+1} \to x_N$
- The later part is the same as $J_{k+1}^*(x_{k+1})$ by the **PO**
- We find optimal cost by minimizing

$$J_k^*(x_k) = \min_{u_k \in \mathcal{A}_k(x_k)} \left[g_k(x_k, u_k) + J_{k+1}^*(x_{k+1}) \right], \quad \mu_k(x_k) = u_k^*$$

The Dynamical Programming algorithm

The Dynamical Programming algorithm

For every initial state x_0, the optimal cost $J^*(x_0)$ is equal to $J_0(x_0)$, and optimal policy π^* is $\pi^* = \{\mu_0, \dots, \mu_{N-1}\}$, computed by the following algorithm, which proceeds backward in time from $k = N$ to $k = 0$ and for each $x_k \in S_k$ computes

$$J_N(x_N) = g_N(x_N) \quad (1)$$

$$J_k(x_k) = \min_{u_k \in \mathcal{A}_k(x_k)} \mathbb{E}_{w_k} \{ g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k)) \} \quad (2)$$

$$\mu_k(x_k) = u_k^* \quad (u_k^* \textit{ is the } u_k \textit{ which minimizes the above expression}). \quad (3)$$

- There are N μ's and $N+1$ J's. This will also be the case in the code
- In the deterministic case:

$$J_k(x_k) = \min_{u_k \in \mathcal{A}_k(x_k)} \{ g_k(x_k, u_k) + J_{k+1}(f_k(x_k, u_k)) \}$$

Example: Inventory control

- Consider the inventory control problem where we plan over $N = 3$ stages
- Customers can buy $w_k = 0$ to $w_k = 2$ units and we can order $u_k = 0$ to $u_k = 2$ units
- We assume the stock can hold from 0 to 2 units (no excess stock; no backlog)

$$x_{k+1} = f_k(x_k, u_k, w_k) = x_k + u_k - w_k \text{ (threshold s.t. } 0 \le x_{k+1} \le 2)$$

- The cost to buy an item is 1 plus quadratic penalty for excess stock and unmet demand:

$$u_k + (x_k + u_k - w_k)^2$$

- There is no terminal cost $g_N(x_N) = 0$
- The demand has distribution

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

Implementation

```python
# inventory.py
class InventoryDPModel(DPModel):
    def __init__(self, N=3):
        super().__init__(N=N)

    def A(self, x, k): # Action space A_k(x)
        return {0, 1, 2}

    def S(self, k): # State space S_k
        return {0, 1, 2}

    def g(self, x, u, w, k): # Cost function g_k(x,u,w)
        return u + (x + u - w) ** 2

    def f(self, x, u, w, k): # Dynamics f_k(x,u,w)
        return max(0, min(2, x + u - w ))

    def Pw(self, x, u, k): # Distribution over random disturbances
        return {0:.1, 1:.7, 2:0.2}

    def gN(self, x):
        return 0
```

Option 1: Pen-and-paper

First step: $J_3(x_3) = 0$ (for all x_3)

Step $k = 2$ For $x_2 = 0$

$$J_2(0) = \min_{u_2=0,1,2} \mathbb{E}_{w_2} \left\{ u_2 + (u_2 - w_2)^2 \right\}$$
$$= \min_{u_2=0,1,2} \left[u_2 + 0.1\,(u_2)^2 + 0.7\,(u_2-1)^2 + 0.2\,(u_2-2)^2 \right]$$
$$= \min_{u_2=0,1,2} \{0.7\cdot 1 + 0.2\cdot 4,\ 1 + 0.1\cdot 1 + 0.2\cdot 1,\ 2 + 0.1\cdot 4 + 0.7\cdot 1\}$$
$$= \min_{u_2=0,1,2} \{1.5, 1.3, 3.1\}$$

Therefore $\mu_2^*(0) = 1$ and $J_2^*(0) = 1.3$

Until nails bleed Keep at it for $x_2 = 1, 2$ and then for $k = 1$ and finally $k = 0$...

Quiz: Manual DP

Suppose that for a given k:

- $\mathcal{A}_k(x_k) = \{0, 1\}$, $f_k(x_k, u_k, w_k) = x_k + u_k w_k$
- $g_k(x_k, u_k, w_k) = -x_k u_k$, $J_{k+1}(x_{k+1}) = x_{k+1}$
- $\mathbb{E}[w_k] = 1$

What is the value of $J_k(x_k = 1)$?. **Tip:**

$$J_k(x_k) = \min_{u_k \in \mathcal{A}_k(x_k)} \mathbb{E}_{w_k} \{ g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k)) \}$$

a. $J_k(1) = -2$
b. $J_k(1) = -1$
c. $J_k(1) = 0$
d. $J_k(1) = 1$
e. $J_k(1) = 2$
f. Don't know.

Option 2: Computer

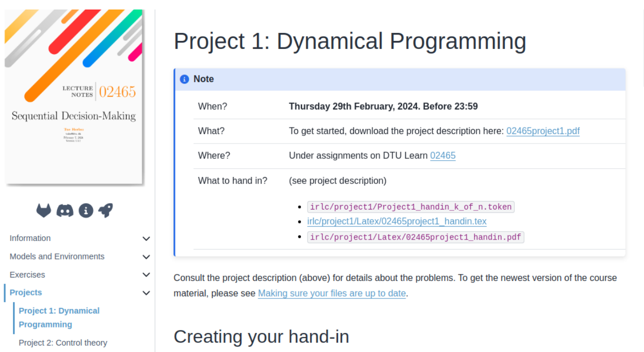
```python
    # inventory.py
    inv = InventoryDPModel()
    J,pi = DP_stochastic(inv)
    print(f"Inventory control optimal policy/value functions")
    for k in range(inv.N):
        print(", ".join([f" J_{k}(x_{k}={i}) = {J[k][i]:.2f}" for i in inv.S(k)] ) )
    for k in range(inv.N):
        print(", ".join([f"pi_{k}(x_{k}={i}) = {pi[k][i]}" for i in inv.S(k)] ) )
```

```
Inventory control optimal policy/value functions
 J_0(x_0=0) = 3.70,  J_0(x_0=1) = 2.70,  J_0(x_0=2) = 2.82
 J_1(x_1=0) = 2.50,  J_1(x_1=1) = 1.50,  J_1(x_1=2) = 1.68
 J_2(x_2=0) = 1.30,  J_2(x_2=1) = 0.30,  J_2(x_2=2) = 1.10
pi_0(x_0=0) = 1, pi_0(x_0=1) = 0, pi_0(x_0=2) = 0
pi_1(x_1=0) = 1, pi_1(x_1=1) = 0, pi_1(x_1=2) = 0
pi_2(x_2=0) = 1, pi_2(x_2=1) = 0, pi_2(x_2=2) = 0
```

lecture_02_optimal_dp_g1.py
lecture_02_frozen_long_slippery.py

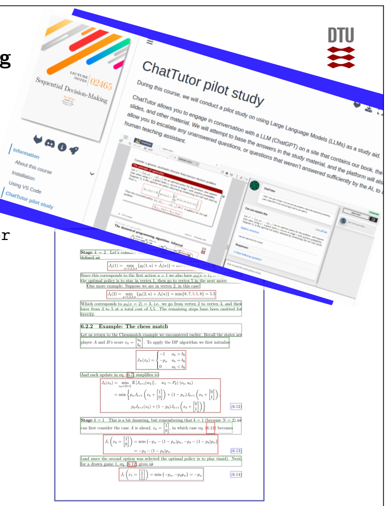
Part 1 of the project

- you *should* be all set!



Experiment on AI in teaching

- How can AI improve studying?
  - Log in: Ask Marius/Me.
  - Feedback very appreciated on Discord
  - File issues on `https://github.com/tuhe/chattutor`
- Completely voluntary.
  - Discord/TAs are still the main feedback channels
  - Waiting for a Piazza license
- Data and privacy
  - We will note store identifying information after a year
  - **Anonymized** data may be used for research purposes



Recruiting usability testers

- Friday Feb. 16th from 12.00 - 15.30
- 30 minutes sessions
- 7 students
  - 5 who have not tried ChatTutor
  - 2 who have already tried it
  - Free lunch!
- `marius@ddsa.dk`

Tue Herlau.
Sequential decision making.
(Freely available online), 2024.