ECE276B: Planning & Learning in Robotics
Lecture 3: The Dynamic Programming Algorithm

Lecturer:

Nikolay Atanasov: natanasov@ucsd.edu

Teaching Assistants:

Tianyu Wang: tiw161@eng.ucsd.edu

Yongxi Lu: yol070@eng.ucsd.edu

UC San Diego

JACOBS SCHOOL OF ENGINEERING

Electrical and Computer Engineering

Dynamic Programming

- **Objective**: construct an optimal policy π^* (independent of x_0):

$$\pi^* = \arg\max_{\pi \in \Pi} J_0^{\pi}(x_0), \quad \forall x_0 \in \mathcal{X}$$

- **Dynamic programming (DP)**: a collection of algorithms that can compute optimal closed-loop policies given a known MDP model of the environment.
  - Idea: use value functions to structure the search for good policies
  - Generality: can handle non-convex and non-linear problems
  - Complexity: **polynomial in the number of states and actions**
  - Efficiency: much more efficient than the brute-force approach of evaluating all possible strategies. May be better suited for large state spaces than other methods such as direct policy search and linear programming.

- **Value function** $V_t^{\pi}(x)$: estimates how good (in terms of expected cost/return) it is to be in state x at time t and follow controls from a given policy π

Principle of Optimality

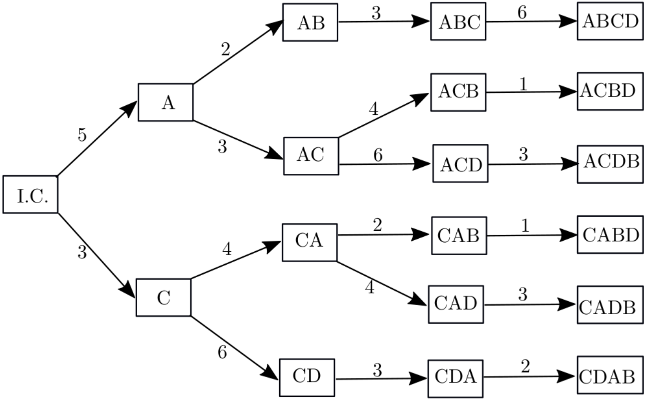
- Let $\pi^*_{0:T-1}$ be an optimal closed-loop policy
- Consider a **subproblem**, where the state is x_t at time t and we want to minimize:

$$J_t^\pi(x_t) = \mathbb{E}_{x_{t+1:T}}\left[\gamma^{T-t} g_T(x_T) + \sum_{\tau=t}^{T-1} \gamma^{\tau-t} g_\tau(x_\tau, \pi_\tau(x_\tau)) \;\middle|\; x_t\right]$$

- **Principle of optimality**: the truncated policy $\pi^*_{t:T-1}$ is optimal for the subproblem starting at time t
- **Intuition**: Suppose $\pi^*_{t:T-1}$ were not optimal for the subproblem. Then, there would exist a policy yielding a lower cost on at least some portion of the state space.

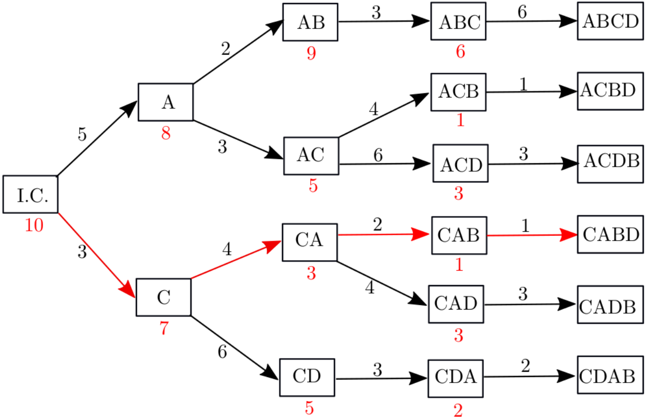
Example: Deterministic Scheduling Problem

- Consider a deterministic scheduling problem where 4 operations A, B, C, D are used to produce a product
- Rules: Operation A must occur before B, and C before D
- Cost: there is a transition cost between each two operations:



Example: Deterministic Scheduling Problem

- The DP algorithm is applied backwards in time. First, construct an optimal solution at the last stage and then work backwards.
- The optimal cost-to-go at each state of the scheduling problem is denoted with red text below the state:



The Dynamic Programming Algorithm

Algorithm 1 Dynamic Programming

Input: MDP $(\mathcal{X}, \mathcal{U}, p_f, g, \gamma)$, initial state $x_0 \in \mathcal{X}$, and horizon T

$V_T(x) = g_T(x), \quad \forall x \in \mathcal{X}$

for $t = (T-1) \ldots 0$ **do**

$\quad Q_t(x,u) \leftarrow g_t(x,u) + \gamma \mathbb{E}_{x' \sim p_f(\cdot|x,u)} \left[V_{t+1}(x') \right], \quad \forall x \in \mathcal{X}, u \in \mathcal{U}(x)$

$\quad V_t(x) = \min_{u \in \mathcal{U}(x)} Q_t(x,u), \qquad \forall x \in \mathcal{X}$

$\quad \pi_t(x) = \arg\min_{u \in \mathcal{U}(x)} Q_t(x,u), \qquad \forall x \in \mathcal{X}$

return policy $\pi_{0:T-1}$ and value function V_0

Theorem: Optimality of the DP Algorithm

The policy $\pi_{0:T-1}$ and value function V_0 returned by the DP algorithm are optimal for the finite-horizon optimal control problem.

The Dynamic Programming Algorithm

- At each recursion step, the optimization needs to be performed over all possible values of $x \in \mathcal{X}$ because we do not know a priori which states will be visited
- This point-wise optimization for each $x \in \mathcal{X}$ is what gives us a policy π_t, i.e., a function specifying the optimal control for **every** state $x \in \mathcal{X}$
- Consider a discrete-space example with $N_x = 10$ states, $N_u = 10$ control inputs, planning horizon $T = 4$, and given x_0:
  - There are $N_u^T = 10^4$ different open-loop strategies
  - There are $N_u^{N_x(T-1)+1} = 10^{31}$ different closed-loop strategies
  - For each stage t and each state x_t, the DP algorithm goes through the N_u control inputs to determine the optimal input. In total, there are $N_u N_x(T-1) + N_u = 310$ such operations.

Proof of Dynamic Programming Optimality

- **Claim**: The policy $\pi_{0:T-1}$ and value function V_0 returned by the DP algorithm are optimal
- Let $J_t^*(x)$ be the optimal cost for the $(T-t)$-stage problem that starts at time t in state x.
- Proceed by induction
- **Base-case**: $J_T^*(x) = g_T(x) = V_T(x)$
- **Hypothesis**: Assume that for $t+1$, $V_{t+1}(x) = J_{t+1}^*(x)$ for all $x \in \mathcal{X}$
- **Induction**: Show that $J_t^*(x_t) = V_t(x_t)$ for all $x_t \in \mathcal{X}$

Proof of Dynamic Programming Optimality

$$
\begin{aligned}
J_t^*(x_t) &= \min_{\pi_{t:T-1}} \mathbb{E}_{x_{t+1:T}|x_t}\left[g_t(x_t,\pi_t(x_t)) + \gamma^{T-t} g_T(x_T) + \sum_{\tau=t+1}^{T-1} \gamma^{\tau-t} g_\tau(x_\tau,\pi_\tau(x_\tau))\right] \\
&\overset{(1)}{=} \min_{\pi_{t:T-1}} g_t(x_t,\pi_t(x_t)) + \mathbb{E}_{x_{t+1:T}|x_t}\left[\gamma^{T-t} g_T(x_T) + \sum_{\tau=t+1}^{T-1} \gamma^{\tau-t} g_\tau(x_\tau,\pi_\tau(x_\tau))\right] \\
&\overset{(2)}{=} \min_{\pi_{t:T-1}} g_t(x_t,\pi_t(x_t)) + \gamma\mathbb{E}_{x_{t+1}|x_t}\left[\mathbb{E}_{x_{t+2:T}|x_{t+1}}\left[\gamma^{T-t-1} g_T(x_T) + \sum_{\tau=t+1}^{T-1} \gamma^{\tau-t-1} g_\tau(x_\tau,\pi_\tau(x_\tau))\right]\right] \\
&\overset{(3)}{=} \min_{\pi_t}\left\{g_t(x_t,\pi_t(x_t)) + \gamma\mathbb{E}_{x_{t+1}|x_t}\left[\min_{\pi_{t+1:T-1}} \mathbb{E}_{x_{t+2:T}|x_{t+1}}\left[\gamma^{T-t-1} g_T(x_T) + \sum_{\tau=t+1}^{T-1} \gamma^{\tau-t-1} g_\tau(x_\tau,\pi_\tau(x_\tau))\right]\right]\right\} \\
&\overset{(4)}{=} \min_{\pi_t}\left\{g_t(x_t,\pi_t(x_t)) + \gamma\mathbb{E}_{x_{t+1}\sim p_f(\cdot|x_t,\pi_t(x_t))}\left[J_{t+1}^*(x_{t+1})\right]\right\} \\
&\overset{(5)}{=} \min_{u_t\in\mathcal{U}(x_t)}\left\{g_t(x_t,u_t) + \gamma\mathbb{E}_{x_{t+1}\sim p_f(\cdot|x_t,u_t)}\left[V_{t+1}(x_{t+1})\right]\right\} \\
&= V_t(x_t), \quad \forall x_t \in \mathcal{X}
\end{aligned}
$$

Proof of Dynamic Programming Optimality

(1) Since $g_t(x_t, \pi_t(x_t))$ is not a function of $x_{t+1:T}$

(2) Using conditional probability $p(x_{t+1:T}|x_t) = p(x_{t+2:T}|x_{t+1}, x_t)p(x_{t+1}|x_t)$ and the Markov assumption

(3) The minimization can be split since the term $g_t(x_t, \pi_t(x_t))$ does not depend on $\pi_{t+1:T-1}$. The expectation $\mathbb{E}_{x_{t+1}|x_t}$ and $\min_{\pi_{t+1:T}}$ can be exchanged since the functions $\pi_{t+1:T-1}$ make the cost small for all initial conditions., i.e., independently of x_{t+1}.

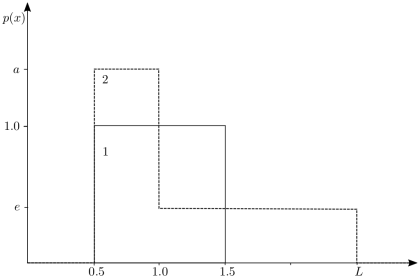
- (1)-(3) is the *principle of optimality*

(4) By definition of $J^*_{t+1}(\cdot)$ and the motion model $x_{t+1} \sim p_f(\cdot \mid x_t, u_t)$

(5) By the induction hypothesis

Is Expected Value a Good Choice for the Cost?

- The expected value is a useful metric but does not take higher order statistics (e.g., variance) into account.
- However, if variance is included into the cost function, the problem becomes much more complicated and we cannot simply apply the DP algorithm.
- It is easy to generate examples, in which two pdfs/policies have the same expectation but very different variance.
- Consider the following two pdfs with $a = \frac{4(L-1)}{(2L-1)}$ and $e = \frac{1}{(L-1)(2L-1)}$



Is Expected Value a Good Choice for the Cost?

- Both pdfs have the same mean: $\int xp(x)dx = 1$
- The variance of first pdf is:

$$Var(x) = \mathbb{E}\left[x^2\right] - [\mathbb{E}x]^2 = \int_{0.5}^{1.5} x^2 dx - 1 = \frac{1}{12}$$

- The variance of the second pdf is:

$$Var(x) = \int_{0.5}^{1.0} ax^2 dx + \int_{0.5}^{L} ex^2 dx - 1 = \frac{L}{6}$$

- Both pdfs have the same mean but, as $L \to \infty$, the variance of the second pdf becomes arbitrarily large. Hence, the first pdf would be preferable.

Example: Chess Strategy Optimization

- State: $x_t \in \mathcal{X} := \{-2, -1, 0, 1, 2\}$ – the difference between our and the opponent's score at the end of game t
- Input: $u_t \in \mathcal{U} := \{timid, bold\}$
- Dynamics: with $p_d > p_w$:

$$\begin{aligned} p_f(x_{t+1} = x_t \mid u_t = timid, x_t) &= p_d \\ p_f(x_{t+1} = x_t - 1 \mid u_t = timid, x_t) &= 1 - p_d \\ p_f(x_{t+1} = x_t + 1 \mid u_t = bold, x_t) &= p_w \\ p_f(x_{t+1} = x_t - 1 \mid u_t = bold, x_t) &= 1 - p_w \end{aligned}$$

- Cost: $J_t^*(x_t) = \mathbb{E}\left[g_2(x_2) + \sum_{t=\tau}^{1} \underbrace{g_\tau(x_\tau, u_\tau)}_{=0}\right]$ with

$$g_2(x_2) = \begin{cases} -1 & \text{if } x_2 > 0 \\ -p_w & \text{if } x_2 = 0 \\ 0 & \text{if } x_2 < 0 \end{cases}$$

Dynamic Programming Applied to the Chess Problem

- Initialize: $V_2(x_2) = \begin{cases} -1 & \text{if } x_2 > 0 \\ -p_w & \text{if } x_2 = 0 \\ 0 & \text{if } x_2 < 0 \end{cases}$

- Recursion: for all $x_t \in \mathcal{X}$ and $t = 1, 0$:

$$V_t(x_t) = \min_{u_t \in \mathcal{U}} \left\{ g_t(x_t, u_t) + \mathbb{E}_{x_{t+1}|x_t,u_t} \left[V_{t+1}(x_{t+1}) \right] \right\}$$

$$= \min \left\{ \underbrace{p_d V_{t+1}(x_t) + (1 - p_d) V_{t+1}(x_t - 1)}_{\text{timid}}, \underbrace{p_w V_{t+1}(x_t + 1) + (1 - p_w) V_{t+1}(x_t - 1)}_{\text{bold}} \right\}$$

DP Applied to the Chess Problem ($t = 1$)

- $x_1 = 1$:

$$\begin{aligned} V_1(1) &= -\max\{p_d + (1-p_d)p_w, p_w + (1-p_w)p_w\} \overset{\text{since}}{\underset{p_d > p_w}{=\!=\!=}} \\ &= -p_d - (1-p_d)p_w \\ \pi_1^*(1) &= \textit{timid} \end{aligned}$$

- $x_1 = 0$:

$$\begin{aligned} V_1(0) &= -\max\{p_d p_w + (1-p_d)0, p_w + (1-p_w)0\} = -p_w \\ \pi_1^*(0) &= \textit{bold} \end{aligned}$$

- $x_1 = -1$:

$$\begin{aligned} V_1(-1) &= -\max\{p_d 0 + (1-p_d)0, p_w p_w + (1-p_w)0\} = -p_w^2 \\ \pi_1^*(-1) &= \textit{bold} \end{aligned}$$

DP Applied to the Chess Problem ($t = 0$)

- $x_0 = 0$:

$$\begin{aligned} V_0(0) &= -\max\{p_d V_1^*(0) + (1-p_d)V_1^*(-1), p_w V_1^*(1) + (1-p_w)V_1^*(-1)\} \\ &= -\max\left\{p_d p_w + (1-p_d)p_w^2, p_w(p_d + (1-p_d)p_w) + (1-p_w)p_w^2\right\} \\ &= -p_d p_w - (1-p_d)p_w^2 - (1-p_w)p_w^2 \\ \pi_0^*(0) &= \textit{bold} \end{aligned}$$

- Thus, as we saw before, the optimal strategy is to play timid iff ahead in the score

Converting Time-lag Problems to the Standard Form

- A system that involves time lag:

$$x_{t+1} = f_t(x_t, x_{t-1}, u_t, u_{t-1}, w_t)$$

can be converted to the standard form via **state augmentation**

- Let $y_t := x_{t-1}$ and $s_t := u_{t-1}$ and define the augmented dynamics:

$$\tilde{x}_{t+1} := \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ s_{t+1} \end{bmatrix} = \begin{bmatrix} f_t(x_t, y_t, u_t, s_t, w_t) \\ x_t \\ u_t \end{bmatrix} =: \tilde{f}_t(\tilde{x}_t, u_t, w_t)$$

- Note that this procedure works for an arbitrary number of time lags but the dimension of the state space grows and increases the computational burden exponentially ("curse of dimensionality")

Converting Correlated Disturbance Problems to the Standard Form

- Disturbances w_t that are correlated across time (colored noise) can be modeled as:

$$w_t = C_t y_{t+1}$$
$$y_{t+1} = A_t y_t + \xi_t$$

  where A_t, C_t are known and ξ_t are independent random variables

- **Augmented state**: $\tilde{x}_t := (x_t, y_t)$ with dynamics:

$$\tilde{x}_{t+1} = \begin{bmatrix} x_{t+1} \\ y_{t+1} \end{bmatrix} = \begin{bmatrix} f_t(x_t, u_t, C_t(A_t y_t + \xi_t)) \\ A_t y_t + \xi_t \end{bmatrix} =: \tilde{f}_t(\tilde{x}_t, u_t, \xi_t)$$

- **State estimator**: note that y_t must be observed at time t, which can be done using a state estimator