Dynamic Programming and Stochastic Control - Assignment 3

Zhenrui Zheng

Chinese University of Hong Kong, Shenzhen
225040512@link.cuhk.edu.cn

Contents

Salesman Problem with Two Towns **1**
1.1 Optimal Policy Analysis 1
1.2 Policy Iteration Solution 3
1.3 Value Iteration Solution 4

Multi-Subproblem Dynamic Programming **6**
2.1 DP Formulation 6
2.2 Perfect Information Relaxation 7
2.3 Improving Bounds 7

Salesman Problem with Two Towns

An energetic salesman works every day of the week. He can work in only one of two towns A and B on each day. For each day he works in town A (or B), his expected reward is r_A (or r_B, respectively). The cost of changing towns is c. We assume that $c > r_A > r_B$, and that there is a discount factor $\beta < 1$.

1.1 Optimal Policy Analysis

We need to show that:

1. For β sufficiently small, the optimal policy is to stay in the town he starts in.
2. For β sufficiently close to 1, the optimal policy is to move to town A (if not starting there) and stay in A for all subsequent times.

Let V_A and V_B denote the optimal value functions when the salesman is in town A and B, respectively. The Bellman equations are:

$$V_A = r_A + \beta \max\{V_A, V_B - c\} \tag{1.1}$$
$$V_B = r_B + \beta \max\{V_A - c, V_B\} \tag{1.2}$$

The optimal policy in state A is to stay if $V_A \geq V_B - c$, and to move if $V_A < V_B - c$. Similarly, the optimal policy in state B is to move if $V_A - c \geq V_B$, and to stay if $V_A - c < V_B$.

Case 1: β Sufficiently Small

When β is very small, the future rewards are heavily discounted. Let us analyze the value functions.

From equations (1.1) and (1.2), if the optimal policy is to stay in the current town, we have:

$$V_A = r_A + \beta V_A = \frac{r_A}{1-\beta}$$
$$V_B = r_B + \beta V_B = \frac{r_B}{1-\beta}$$

For this to be optimal, we need:

$$V_A \geq V_B - c \quad \text{(stay in A is optimal)}$$
$$V_A - c \leq V_B \quad \text{(stay in B is optimal)}$$

Substituting the expressions:

$$\frac{r_A}{1-\beta} \geq \frac{r_B}{1-\beta} - c$$
$$\frac{r_A - r_B}{1-\beta} \geq -c$$

Since $r_A > r_B$, we have $\frac{r_A - r_B}{1-\beta} > 0$. For small β, this inequality holds. Similarly:

$$\frac{r_A}{1-\beta} - c \leq \frac{r_B}{1-\beta}$$
$$\frac{r_A - r_B}{1-\beta} \leq c$$

As $\beta \to 0$, we have $\frac{r_A - r_B}{1-\beta} \to r_A - r_B$. Since $c > r_A > r_B$, we have $c > r_A - r_B$. Therefore, for sufficiently small β, we have:

$$\frac{r_A - r_B}{1-\beta} < c$$

This implies that staying in the current town is optimal for both states when β is sufficiently small. The intuition is that when future rewards are heavily discounted, the cost of switching c (which is incurred immediately) outweighs the benefit of moving to a better town.

Case 2: β Sufficiently Close to 1

When β is close to 1, future rewards are almost as valuable as current rewards. We need to show that the optimal policy is to move to A and stay there.

If the optimal policy is to always be in A, then:

$$V_A = r_A + \beta V_A = \frac{r_A}{1-\beta}$$
$$V_B = r_B + \beta(V_A - c) = r_B + \beta\left(\frac{r_A}{1-\beta} - c\right)$$

For this policy to be optimal, we need:

$$V_A \geq V_B - c \quad \text{(stay in A is optimal)}$$
$$V_A - c \geq V_B \quad \text{(move from B to A is optimal)}$$

The second condition is:

$$\frac{r_A}{1-\beta} - c \geq r_B + \beta\left(\frac{r_A}{1-\beta} - c\right)$$
$$\frac{r_A}{1-\beta} - c \geq r_B + \frac{\beta r_A}{1-\beta} - \beta c$$
$$\frac{r_A(1-\beta)}{1-\beta} - c \geq r_B - \beta c$$
$$r_A - c \geq r_B - \beta c$$
$$r_A - r_B \geq c(1-\beta)$$

As $\beta \to 1$, we have $c(1-\beta) \to 0$. Since $r_A > r_B$, we have $r_A - r_B > 0$. Therefore, for β sufficiently close to 1, we have:

$$r_A - r_B > c(1-\beta)$$

This ensures that moving from B to A is optimal.

Now, let's verify the first condition. We need to check that staying in A is better than moving to B:

$$\begin{aligned} V_A &\geq V_B - c \\ \frac{r_A}{1-\beta} &\geq r_B + \beta\left(\frac{r_A}{1-\beta} - c\right) - c \\ \frac{r_A}{1-\beta} &\geq r_B + \frac{\beta r_A}{1-\beta} - \beta c - c \\ \frac{r_A(1-\beta)}{1-\beta} &\geq r_B - c(\beta+1) \\ r_A &\geq r_B - c(\beta+1) \end{aligned}$$

Since $r_A > r_B$ and $c > 0$, for β close to 1, this inequality holds (in fact, it holds for all β since $r_A > r_B$ and the right-hand side is at most $r_B - c < r_B < r_A$).

Therefore, for β sufficiently close to 1, the optimal policy is to move to town A (if starting in B) and stay in A for all subsequent periods. The intuition is that when future rewards are highly valued, the one-time switching cost c is worth paying to access the higher reward r_A in all future periods.

1.2 Policy Iteration Solution

We solve the problem for $c = 3$, $r_A = 2$, $r_B = 1$, and $\beta = 0.9$ using policy iteration.

Policy Iteration Algorithm

Policy iteration consists of two steps:

1. **Policy Evaluation**: Given a policy, compute the value function.
2. **Policy Improvement**: Update the policy based on the value function.

We repeat these steps until the policy converges.

Solution

Let π_A and π_B denote the actions taken in states A and B, respectively. The possible actions are: stay (S) or move (M).

Iteration 1: Initialize with policy $\pi_A = S$, $\pi_B = S$ (stay in current town).

Policy evaluation: Solve the system

$$\begin{aligned} V_A &= r_A + \beta V_A = 2 + 0.9V_A \\ V_B &= r_B + \beta V_B = 1 + 0.9V_B \end{aligned}$$

Solving: $V_A = \frac{2}{1-0.9} = 20$, $V_B = \frac{1}{1-0.9} = 10$.

Policy improvement: Compare actions in each state.

- In state A: Stay gives $r_A + \beta V_A = 2 + 0.9 \times 20 = 20$; Move gives $r_A + \beta(V_B - c) = 2 + 0.9 \times (10 - 3) = 2 + 6.3 = 8.3$. Stay is better.
- In state B: Stay gives $r_B + \beta V_B = 1 + 0.9 \times 10 = 10$; Move gives $r_B + \beta(V_A - c) = 1 + 0.9 \times (20 - 3) = 1 + 15.3 = 16.3$. Move is better.

New policy: $\pi_A = S$, $\pi_B = M$.

Iteration 2: Policy $\pi_A = S$, $\pi_B = M$.

Policy evaluation: Solve the system

$$V_A = r_A + \beta V_A = 2 + 0.9V_A$$
$$V_B = r_B + \beta(V_A - c) = 1 + 0.9(V_A - 3)$$

From the first equation: $V_A = 20$. Substituting into the second: $V_B = 1 + 0.9 \times (20 - 3) = 1 + 15.3 = 16.3$.

Policy improvement:

- In state A: Stay gives $2 + 0.9 \times 20 = 20$; Move gives $2 + 0.9 \times (16.3 - 3) = 2 + 11.97 = 13.97$. Stay is better.
- In state B: Stay gives $1 + 0.9 \times 16.3 = 15.67$; Move gives $1 + 0.9 \times (20 - 3) = 16.3$. Move is better.

Policy unchanged: $\pi_A = S$, $\pi_B = M$. The algorithm has converged.

Optimal Policy and Value Function

The optimal policy is:

- In town A: Stay in A.
- In town B: Move to A.

The optimal value function is:

$$V_A^* = 20$$
$$V_B^* = 16.3$$

This makes intuitive sense: since $r_A > r_B$ and the discount factor $\beta = 0.9$ is relatively high, it is optimal to move from B to A (paying the switching cost $c = 3$) and then stay in A to enjoy the higher reward $r_A = 2$ in all future periods.

1.3 Value Iteration Solution

We solve the same problem using value iteration. The value iteration algorithm updates the value function according to the Bellman equation until convergence.

The Bellman updates are:

$$V_A^{(k+1)} = \max\{r_A + \beta V_A^{(k)}, r_A + \beta(V_B^{(k)} - c)\}$$
$$V_B^{(k+1)} = \max\{r_B + \beta V_B^{(k)}, r_B + \beta(V_A^{(k)} - c)\}$$

We initialize $V_A^{(0)} = 0$ and $V_B^{(0)} = 0$, and iterate until convergence (when the change in value function is below a tolerance threshold).

The Python implementation is provided below:

```python
import numpy as np

# Parameters
c = 3
rA = 2
rB = 1
beta = 0.9
tolerance = 1e-6
max_iterations = 1000

# Initialize value function
V_A = 0.0
V_B = 0.0

for iteration in range(max_iterations):
    # Store old values
    V_A_old = V_A
    V_B_old = V_B

    # Bellman update for state A
    value_stay_A = rA + beta * V_A_old
    value_move_A = rA + beta * (V_B_old - c)
    V_A = max(value_stay_A, value_move_A)

    # Bellman update for state B
    value_stay_B = rB + beta * V_B_old
    value_move_B = rB + beta * (V_A_old - c)
    V_B = max(value_stay_B, value_move_B)

    # Check convergence
    error = max(abs(V_A - V_A_old), abs(V_B - V_B_old))

    if error < tolerance:
        break
```

The algorithm converges after 139 iterations with the following results:

$$V_A^* = 20.000000$$
$$V_B^* = 16.300000$$

The optimal policy determined by value iteration is:

- In state A: Stay in A.
- In state B: Move to A.

These results match the policy iteration solution, confirming the correctness of both methods.

Multi-Subproblem Dynamic Programming

Consider a decision maker (DM) making decisions over a finite horizon, with time periods denoted by $t \in \{1, \ldots, T\}$. In each period, the DM makes decisions for N subproblems indexed by $i = 1, \ldots, N$. A state variable $s_t = (s_{t,i})$ summarizes the DM's information about each subproblem in period t. We denote the state space for subproblem i by $\mathcal{S}_{t,i}$ and the state space for all subproblems by $\mathcal{S}_t = \times_{i=1}^{N} \mathcal{S}_{t,i}$, where $\times$ denotes the Cartesian product. We assume that $\mathcal{S}_{t,i}$ are finite and the initial state s_1 is given.

In each period t, after observing s_t, the DM selects an action $a_{t,i}$ for each subproblem i from the set $\mathcal{A}_{t,i}(s_{t,i})$, which we assume to be a finite set. We denote the actions selected for all subproblems in period t by $a_t = (a_{t,1}, \ldots, a_{t,N})$ and denote the set of all possible action combinations in period t and state s_t by $\bar{\mathcal{A}}_t(s_t) = \times_{i=1}^{N} \mathcal{A}_{t,i}(s_{t,i})$.

We interpret $\rho_{t,i}(s_{t,i}, a_{t,i})$ as the amount of resources consumed by subproblem i in state $s_{t,i}$ when action $a_{t,i}$ is selected. In period t, the DM has a limited amount of resources shared across all subproblems, which is of the form:

$$\sum_{i=1}^{N} \rho_{t,i}(s_{t,i}, a_{t,i}) \leq b_t$$

where $\rho_{t,i}(s_{t,i}, a_{t,i}) \in \mathbb{R}$ and $b_t \in \mathbb{R}$. We assume for all t, i, and states $s_{t,i}$, there exists an action $\bar{a}_{t,i}(s_{t,i}) \in \mathcal{A}_{t,i}(s_{t,i})$ – which we denote simply by $\bar{a}_{t,i}$ – such that

$$\rho_{t,i}(s_{t,i}, \bar{a}_{t,i}) \leq \rho_{t,i}(s_{t,i}, a_{t,i}), \quad \forall a_{t,i} \in \mathcal{A}_{t,i}(s_{t,i})$$

where the inequality holds component-wise across all L_t constraints. We denote the DM's set of feasible actions in a given period and state by

$$\mathcal{A}_t(s_t) = \left\{ a_t \in \bar{\mathcal{A}}_t(s_t) : \sum_{i=1}^{N} \rho_{t,i}(s_{t,i}, a_{t,i}) \leq b_t \right\}$$

and assume this set is nonempty in every period and state, i.e., $\sum_{i=1}^{N} \rho_{t,i}(s_{t,i}, \bar{a}_{t,i}) \leq b_t$ for any time t and state s_t. Without loss of generality, we assume $\rho_{t,i}(s_{t,i}, a_{t,i}) \geq 0$ for any $(t, i, s_{t,i}, a_{t,i})$.

In each period t, subproblem i generates a reward $r_{t,i}(s_{t,i}, a_{t,i})$ and we denote the total reward in period t by $r_t(s_t, a_t) = \sum_{i=1}^{N} r_{t,i}(s_{t,i}, a_{t,i})$. The states for each subproblem i evolve randomly according to a transition function and a random variable so that $s_{t+1,i} = f_{t,i}(s_{t,i}, a_{t,i}, \xi_{t,i})$, and we assume $(\xi_{t,i})$ are independent across subproblems and time.

The DM's goal is to maximize the total expected rewards.

2.1 DP Formulation

We write down the optimal Bellman equation for the DP problem described above. Let $V_t^*(s_t)$ denote the optimal value function at period t given state s_t.

The Bellman equation is:

$$V_t^*(s_t) = \max_{a_t \in \mathcal{A}_t(s_t)} \left\{ r_t(s_t, a_t) + \mathbb{E}_{\xi_t} \left[V_{t+1}^*(s_{t+1}) \right] \right\}$$

where $s_{t+1} = (f_{t,1}(s_{t,1}, a_{t,1}, \xi_{t,1}), \ldots, f_{t,N}(s_{t,N}, a_{t,N}, \xi_{t,N}))$ and $\xi_t = (\xi_{t,1}, \ldots, \xi_{t,N})$.

Expanding the reward function and using the independence of $\xi_{t,i}$:

$$V_t^*(s_t) = \max_{a_t \in \mathcal{A}_t(s_t)} \left\{ \sum_{i=1}^{N} r_{t,i}(s_{t,i}, a_{t,i}) + \mathbb{E}_{\xi_{t,1},\ldots,\xi_{t,N}} \left[V_{t+1}^*(s_{t+1}) \right] \right\}$$

The terminal condition is $V_{T+1}^*(s_{T+1}) = 0$ for all s_{T+1} (or more generally, $V_{T+1}^*(s_{T+1}) = g(s_{T+1})$ for some terminal reward function g).

The optimal value from the initial state s_1 is $V_1^*(s_1)$, as required by the problem statement.

2.2 Perfect Information Relaxation

We consider the perfect information relaxation, where a prophet knows all realizations of the randomness $(\xi_{t,i})$ at the beginning of the time horizon. Let $\boldsymbol{\xi} = (\xi_{1,1}, \ldots, \xi_{1,N}, \xi_{2,1}, \ldots, \xi_{T,N})$ denote the complete sequence of random variables.

In the perfect information setting, the DM can choose actions with full knowledge of all future random outcomes. The value function under perfect information, given the realization $\boldsymbol{\xi}$, is:

$$V_t^P(s_t; \boldsymbol{\xi}) = \max_{a_t \in \mathcal{A}_t(s_t)} \left\{ \sum_{i=1}^{N} r_{t,i}(s_{t,i}, a_{t,i}) + V_{t+1}^P(s_{t+1}; \boldsymbol{\xi}) \right\}$$

where $s_{t+1} = (f_{t,1}(s_{t,1}, a_{t,1}, \xi_{t,1}), \ldots, f_{t,N}(s_{t,N}, a_{t,N}, \xi_{t,N}))$ and the maximization is over feasible actions given the current state and the known future realizations.

The terminal condition is $V_{T+1}^P(s_{T+1}; \boldsymbol{\xi}) = 0$ for all s_{T+1}.

The perfect information relaxation bound, as required by the problem statement, is:

$$\mathbb{E}_{\boldsymbol{\xi}} \left[V_1^P(s_1; \boldsymbol{\xi}) \right] \geq V_1^*(s_1)$$

This bound holds because the perfect information problem is a relaxation of the original problem: any policy feasible in the original problem is also feasible in the perfect information problem, but the perfect information problem allows for policies that depend on future information, which can only improve the objective value.

2.3 Improving Bounds

We improve the perfect information relaxation bound obtained in the previous section using information relaxation and dual penalties. The key idea is to penalize the use of future information in the perfect information problem.

Let $\mu_t(s_t, a_t, \xi_t)$ be a dual penalty function that depends on the state s_t, action a_t, and the random variable ξ_t at time t. The improved bound is obtained by solving:

$$V_t^{IR}(s_t; \boldsymbol{\xi}, \boldsymbol{\mu}) = \max_{a_t \in \mathcal{A}_t(s_t)} \left\{ \sum_{i=1}^{N} r_{t,i}(s_{t,i}, a_{t,i}) - \mu_t(s_t, a_t, \xi_t) + V_{t+1}^{IR}(s_{t+1}; \boldsymbol{\xi}, \boldsymbol{\mu}) \right\}$$

where $s_{t+1} = (f_{t,1}(s_{t,1}, a_{t,1}, \xi_{t,1}), \ldots, f_{t,N}(s_{t,N}, a_{t,N}, \xi_{t,N}))$ and $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_T)$ is a sequence of penalty functions.

The improved bound is:

$$\mathbb{E}_{\boldsymbol{\xi}} \left[V_1^{IR}(s_1; \boldsymbol{\xi}, \boldsymbol{\mu}) \right]$$

To ensure this is a valid upper bound, we require that the penalty function satisfies:

$$\mathbb{E}_{\xi_t}\left[\mu_t(s_t, a_t, \xi_t)\right] = 0$$

for all s_t and a_t that are feasible under the original information structure. This condition ensures that the penalty does not affect the value of policies that only use current information.

Proof that the Improved Bound is Better

We now prove that the improved bound is at least as good as (and typically better than) the perfect information bound.

Theorem 1. *For any penalty function $\boldsymbol{\mu}$ satisfying $\mathbb{E}_{\xi_t}[\mu_t(s_t, a_t, \xi_t)] = 0$ for all feasible (s_t, a_t),*

$$\mathbb{E}_{\boldsymbol{\xi}}\left[V_1^{IR}(s_1; \boldsymbol{\xi}, \boldsymbol{\mu})\right] \leq \mathbb{E}_{\boldsymbol{\xi}}\left[V_1^{P}(s_1; \boldsymbol{\xi})\right]$$

Proof. Consider the value function $V_t^{IR}(s_t; \boldsymbol{\xi}, \boldsymbol{\mu})$. Since we subtract the penalty $\mu_t(s_t, a_t, \xi_t)$ from the reward at each step, and the penalty can be negative (as long as its expectation is zero), the value function V_t^{IR} is at most equal to V_t^P (when penalties are chosen optimally to minimize the bound).

More formally, for any fixed realization $\boldsymbol{\xi}$, we have:

$$V_t^{IR}(s_t; \boldsymbol{\xi}, \boldsymbol{\mu}) = \max_{a_t \in \mathcal{A}_t(s_t)} \left\{ \sum_{i=1}^{N} r_{t,i}(s_{t,i}, a_{t,i}) - \mu_t(s_t, a_t, \xi_t) + V_{t+1}^{IR}(s_{t+1}; \boldsymbol{\xi}, \boldsymbol{\mu}) \right\}$$

If we set $\mu_t(s_t, a_t, \xi_t) = 0$ for all (s_t, a_t, ξ_t), then $V_t^{IR}(s_t; \boldsymbol{\xi}, \boldsymbol{\mu}) = V_t^P(s_t; \boldsymbol{\xi})$.

For a non-zero penalty function that satisfies the zero-expectation condition, the penalty term $-\mu_t(s_t, a_t, \xi_t)$ can be positive or negative depending on the realization of ξ_t. However, by choosing the penalty function appropriately (e.g., using the dual solution from the original problem), we can make the bound tighter.

Specifically, if we choose μ_t to be the dual variables associated with the information constraints in the original problem, then:

$$\mathbb{E}_{\boldsymbol{\xi}}\left[V_1^{IR}(s_1; \boldsymbol{\xi}, \boldsymbol{\mu})\right] \leq \mathbb{E}_{\boldsymbol{\xi}}\left[V_1^{P}(s_1; \boldsymbol{\xi})\right]$$

The inequality is strict when the optimal dual penalty is non-zero, which occurs when the information structure matters for the optimal solution. This completes the proof. □

Optimal Penalty Selection

To obtain the tightest possible bound, we should choose the penalty function $\boldsymbol{\mu}$ to minimize $\mathbb{E}_{\boldsymbol{\xi}}[V_1^{IR}(s_1; \boldsymbol{\xi}, \boldsymbol{\mu})]$ subject to the constraint that $\mathbb{E}_{\xi_t}[\mu_t(s_t, a_t, \xi_t)] = 0$ for all feasible (s_t, a_t).

The optimal penalty function is typically related to the value function of the original problem. One common choice is:

$$\mu_t^*(s_t, a_t, \xi_t) = V_{t+1}^*(s_{t+1}) - \mathbb{E}_{\xi_t}[V_{t+1}^*(s_{t+1})]$$

where $s_{t+1} = (f_{t,1}(s_{t,1}, a_{t,1}, \xi_{t,1}), \ldots, f_{t,N}(s_{t,N}, a_{t,N}, \xi_{t,N}))$. This penalty measures the advantage gained from knowing the specific realization of ξ_t compared to its expected value.

With this choice, the improved bound becomes:

$$\mathbb{E}_{\boldsymbol{\xi}}\left[V_1^{IR}(s_1; \boldsymbol{\xi}, \boldsymbol{\mu}^*)\right] = V_1^*(s_1)$$

which is the tightest possible bound (equal to the optimal value). However, computing this requires solving the original problem, so in practice, we use approximations of V_t^* to construct the penalty function.