Dynamic Programming and Stochastic Control - Assignment 1

Zhenrui Zheng

Chinese University of Hong Kong, Shenzhen

225040512@link.cuhk.edu.cn

Contents

Problem 1: Dynamic Pricing in Discrete Time

1.1 Derivation of Value Function

Recall that in a Markov Decision Process with random state transition, the optimal action $a^*(s)$ in state s is defined as:

$$a^*(s) = \arg\max_{a \in A} Q(s,a) = \arg\max_{a \in A} \mathbb{E}_{s' \sim P(s'|s,a)} \left[r(s', s, a) + V(s') \right]$$

For the given problem, the state of k^{th} day is the remaining unsold items x_k, and the action is the price u_k. Thus we can write the optimal action as:

$$u_k^{*1} = u^*(k, x_k) = \arg\max_{u \in \mathbb{R}_+} \mathbb{E}_{x_{k+1} \sim P(x_{k+1}|x_k,u)} \left[g(x_{k+1}, x_k, u_k) + V_{k+1}(x_{k+1}) \right]$$

Where $g(x_{k+1}, x_k, u_k)$ is the profit from selling $x_k - x_{k+1}$ items at price u, that is:

$$g(x_{k+1}, x_k, u_k) = u_k(x_k - x_{k+1})$$

and $V_{k+1}(x_{k+1})$ is the optimal expected value function of the next state $(k+1, x_{k+1})$. Since we already have:

$$q_k(u_k) = \alpha \exp(-u_k)$$

Thus the action value function $Q(x_k, u_k)$ for any $x_k > 0$ can be expanded (from its expectation form) as:

$$\begin{aligned} Q(x_k, u_k) &= \alpha \exp(-u_k) \left[g(x_k - 1, x_k, u_k) + V_{k+1}(x_k - 1) \right] \\ &\quad + (1 - \alpha \exp(-u_k)) \left[g(x_k, x_k, u_k) + V_{k+1}(x_k) \right] \\ &= \alpha \exp(-u_k) \left[u_k + V_{k+1}(x_k - 1) \right] \\ &\quad + (1 - \alpha \exp(-u_k)) \left[V_{k+1}(x_k) \right] \end{aligned}$$

To find the optimal u_k, we take the derivative of $Q(x_k, u_k)$ with respect to u_k and set it to zero:

$$\begin{aligned} \frac{\partial Q}{\partial u_k} &= -\alpha \exp(-u_k) \left[u_k + V_{k+1}(x_k - 1) \right] \\ &\quad + \alpha \exp(-u_k) \cdot 1 \\ &\quad + \alpha \exp(-u_k) \left[V_{k+1}(x_k) \right] \\ &= \alpha \exp(-u_k) \left[1 - u_k - V_{k+1}(x_k - 1) + V_{k+1}(x_k) \right] = 0 \\ \implies & \alpha \exp(-u_k^*) \left[u_k^* + V_{k+1}(x_k - 1) - V_{k+1}(x_k) \right] = \alpha \exp(-u_k^*) \\ & u_k^* = 1 + V_{k+1}(x_k) - V_{k+1}(x_k - 1) \end{aligned}$$

Thus,

$$\begin{aligned} V_k(x_k) = Q(x_k, u_k^*) &= \alpha \exp(-u_k^*) \left[u_k^*(x_k) + V_{k+1}(x_k - 1) - V_{k+1}(x_k)\right] + V_{k+1}(x_k) \\ &= \alpha \exp(-u_k^*) + V_{k+1}(x_k) \end{aligned}$$

1.2 Proof of Monotonicity

Assume that $V_k(0) = 0$ and $V_N(x_N) = 0$, let's prove the closed form of $V_k(x_k)$ by induction.

Assume that for some $k < N$, $V_k(x_k)$ has the closed form:

$$V_k(x_k) = \begin{cases} (N-k)\alpha \exp(-1) & \text{if } x_k \geq N - k \\ \sum_{i=k}^{N-x_k} \alpha \exp(-u_i^*(x_k)) + x_k \alpha \exp(-1) & \text{if } 0 < x_k < N - k \\ 0 & \text{if } x_k = 0 \end{cases}$$

For V_{k-1}, there are four cases:

1. $x_{k-1} = 0$
   $V_{k-1}(x_{k-1}) = 0$ according to our assumption.

2. $0 < x_{k-1} < N - k$

$$\begin{aligned} V_{k-1}(x_{k-1}) &= \alpha \exp(-u_{k-1}^*(x_{k-1})) + V_k(x_{k-1}) \\ &= \alpha \exp(-u_{k-1}^*(x_{k-1})) + \sum_{i=k}^{N-x_{k-1}} \alpha \exp(-u_i^*(x_{k-1})) + x_{k-1}\alpha \exp(-1) \\ &= \sum_{i=k-1}^{N-x_{k-1}} \alpha \exp(-u_i^*(x_{k-1})) + x_{k-1}\alpha \exp(-1) \end{aligned}$$

3. $x_{k-1} = N - k$
   Let's first derive $u_{k-1}^*(x_{k-1})$. Assume that for some $k < N - 1$, $u_k^*(N - k - 1) = 1$, then:

$$\begin{aligned} u_{k-1}^*(N-k) &= 1 + V_k(N-k) - V_k(N-k-1) \\ &= 1 + (N-k)\alpha \exp(-1) \\ &\quad - \left(\sum_{i=k}^{N-x_{k-1}+1} \alpha \exp(-u_i^*(x_{k-1} - 1)) + (x_{k-1} - 1)\alpha \exp(-1) \right) \\ &= 1 + (N-k)\alpha \exp(-1) - (\alpha \exp(-1) + (N-k-1)\alpha \exp(-1)) \\ &= 1 \end{aligned}$$

   Which also holds for $k = N - 1$, $u_{N-1}^*(0) = 1$. Thus,

$$\begin{aligned} V_{k-1}(x_{k-1}) &= \alpha \exp(-u_{k-1}^*(x_{k-1})) + V_k(x_{k-1}) \\ &= \alpha \exp(-u_{k-1}^*(N-k)) + (N-k)\alpha \exp(-1) \\ &= \alpha \exp(-1) + (N-k)\alpha \exp(-1) \\ &= (N - (k-1))\alpha \exp(-1) \end{aligned}$$

[1] For simplicity, we use u_k instead of $u_k(x_k)$ when there is no ambiguity.

4. $x_{k-1} \geq N - (k-1) \quad (x_{k-1} > N - k)$

$$\begin{aligned} u_{k-1}^*(x_{k-1}) &= 1 + V_k(x_{k-1}) - V_k(x_{k-1} - 1) \\ &= 1 + (N-k)\alpha \exp(-1) - (N-k)\alpha \exp(-1) \quad (x_{k-1} - 1 \geq N - k) \\ &= 1 \end{aligned}$$

Thus,

$$\begin{aligned} V_{k-1}(x_{k-1}) &= \alpha \exp(-u_{k-1}^*(x_{k-1})) + V_k(x_{k-1}) \\ &= \alpha \exp(-1) + (N-k)\alpha \exp(-1) \\ &= (N - (k-1))\alpha \exp(-1) \end{aligned}$$

Therefore, we have proved that the induction relation holds. For $k = N$, It's trivial to prove that the assumption also holds by substitution. We have therefore proved the closed form of $V_k(x_k)$.

Then, we can prove the monotonically non-increasing property of $u_k^*(x_k)$.

$$\begin{aligned} u_k^*(x_k) &= 1 + V_{k+1}(x_k) - V_{k+1}(x_k - 1) \\ u_k^*(x_k) - u_k^*(x_k - 1) &= V_{k+1}(x_k) - 2V_{k+1}(x_k - 1) + V_{k+1}(x_k - 2) \end{aligned}$$

Which actually requires us to prove that $V_{k+1}(x_k) - 2V_{k+1}(x_k - 1) + V_{k+1}(x_k - 2) \leq 0$, that is, V_k is a concave function. Since we have already derived the piecewise form of V_k, we can prove its concavity by case analysis on x_k, expanding the expressions of V_{k+1}, and using a little bit induction assumption. However, this would be quite tedious and the proof is straightforward to those who understand the problem structure.

Problem 2: Label correcting with negative arc lengths

2.1 Proof of Correctness

Let's prove that the modified algorithm terminates with a shortest path from s to t. The proof is divided into two parts: firstly, we prove that the algorithm must terminate; secondly, we prove that upon termination, the computed distance d_t is indeed the shortest path distance.

Part A: Termination

Let d_j be the label of node j, representing the length of a computed path from s to j. The algorithm initializes $d_s = 0$ and $d_j = \infty$ for all $j \neq s$.

Lemma 1. *For any node j, the label d_j is non-increasing throughout the execution of the algorithm.*

Proof. A label d_j is updated only in the Modified Step 2, where it is set to $d_i + a_{ij}$. The condition for this update is $d_i + a_{ij} < \min\{d_j, \text{UPPER} - u_j\}$, which implies $d_i + a_{ij} < d_j$. Thus, any update to d_j strictly decreases its value. □

Lemma 2. *The label d_j of any node j is always greater than or equal to the shortest path distance from s to j, denoted $\delta(s, j)$.*

Proof. We prove this by induction on the number of label updates.

Base Case: Initially, $d_s = 0 = \delta(s, s)$ and $d_j = \infty \geq \delta(s, j)$ for $j \neq s$, the property holds.

Induction: Assume that before any update, $d_k \geq \delta(s, k)$ for all nodes k. Consider an update to d_j where d_j is set to $d_i + a_{ij}$. By the inductive hypothesis, $d_i \geq \delta(s, i)$. By the triangle inequality for shortest paths, $\delta(s, j) \leq \delta(s, i) + a_{ij}$. Thus, the new value for d_j satisfies:

$$d_j^{\text{new}} = d_i + a_{ij} \geq \delta(s, i) + a_{ij} \geq \delta(s, j)$$

The property is maintained after the update. □

Theorem 1. *The algorithm terminates.*

Proof. The algorithm terminates when the set OPEN is empty. A node j is added to OPEN only when its label d_j decreases. Therefore, we need to prove that each d_j can only be decreased a finite number of times.

From the previous lemma, we know that $d_j \geq \delta(s, j)$, and d_j can only decrease. If we assume that arc lengths a_{ij} are integers, then the number of possible values for d_j between its initial value and its lower bound $\delta(s, j)$ is finite. Therefore, the label of each node j can only be updated a finite number of times, and thus can only be added to the OPEN set a finite number of times. Since the number of nodes is also finite, the algorithm must terminate. □

Part B: Correctness

We now prove that upon termination, the algorithm finds a shortest path. That is, the final label d_t equals the shortest path distance $\delta(s, t)$.

Theorem 2. *If the graph G contains no negative cycles and there is a path from s to t, the modified label correcting algorithm terminates with $d_t = \delta(s, t)$.*

Proof. We know from Lemma 2 that $d_j \geq \delta(s, j)$ for all nodes j throughout the algorithm. Thus, at termination, $d_t \geq \delta(s, t)$. We only need to prove that $d_t \leq \delta(s, t)$. We prove this by contradiction. Assume the algorithm terminates with $d_t > \delta(s, t)$. Let $P^* = (s = v_0, v_1, \ldots, v_k = t)$ be a shortest path from s to t. Since $d_{v_0} = d_s = 0 = \delta(s, s)$ and we assume $d_{v_k} = d_t > \delta(s, t)$, there must be a first node v_m on this path for which the final label d_{v_m} satisfies $d_{v_m} > \delta(s, v_m)$. Since v_m is the first such node on P^*, the label for the preceding node, v_{m-1}, must be correct at termination. That is, $d_{v_{m-1}} = \delta(s, v_{m-1})$. Because the algorithm has terminated, the set OPEN is empty. Which implies that node v_{m-1} must have been removed from OPEN at some point after its label was last updated. Let's consider the last time v_{m-1} was removed from OPEN. At that time, its label was its final, optimal value, $d_{v_{m-1}} = \delta(s, v_{m-1})$. During the processing of v_{m-1}, the algorithm examined the arc (v_{m-1}, v_m). Since the final label d_{v_m} is not $\delta(s, v_m)$, the relaxation step for v_m via v_{m-1} must have failed. This means the condition for the update was not met:

$$d_{v_{m-1}} + a_{v_{m-1}, v_m} \geq \min\{d_{v_m}, \text{UPPER} - u_{v_m}\}$$

(where d_{v_m} and UPPER are the values at that time). Since labels only decrease, the final values also satisfy this inequality. Substituting $d_{v_{m-1}} = \delta(s, v_{m-1})$ and noting that $\delta(s, v_{m-1}) + a_{v_{m-1}, v_m} = \delta(s, v_m)$ because (v_{m-1}, v_m) is an arc on a shortest path, we get:

$$\delta(s, v_m) \geq \min\{d_{v_m}, \text{UPPER} - u_{v_m}\}$$

This implies two conditions must hold:

1. $\delta(s, v_m) \geq d_{v_m}$
2. $\delta(s, v_m) \geq \text{UPPER} - u_{v_m}$

Let's analyze the first condition. We already know from Lemma 2 that $d_{v_m} \geq \delta(s, v_m)$. Combining this with $\delta(s, v_m) \geq d_{v_m}$ implies that $d_{v_m} = \delta(s, v_m)$. This contradicts our assumption that v_m was the first node on the path with $d_{v_m} > \delta(s, v_m)$. The pruning

condition UPPER $- u_j$ does not discard the shortest path: UPPER is the length of some $s - t$ path, so UPPER $\geq \delta(s,t)$. The value u_{v_m} is an underestimate of the distance from v_m to t, so $u_{v_m} \leq \delta(v_m,t)$. Therefore,

$$\text{UPPER} - u_{v_m} \geq \delta(s,t) - \delta(v_m,t)$$

Since v_m is on a shortest path to t, we have $\delta(s,t) = \delta(s,v_m) + \delta(v_m,t)$. Substituting this gives:

$$\text{UPPER} - u_{v_m} \geq (\delta(s,v_m) + \delta(v_m,t)) - \delta(v_m,t) = \delta(s,v_m)$$

This shows that $\delta(s,v_m) \leq \text{UPPER} - u_{v_m}$ is always true. The pruning step only prevents an update if $\delta(s,v_m) \geq \text{UPPER} - u_{v_m}$, which can only happen if equality holds. This pruning is safe because if $d_i + a_{ij} + u_j \geq \text{UPPER}$, the path through (i,j) cannot lead to a better solution than the one that defined the current UPPER.

To this end, we have proved that the assumption that a node on the shortest path has a suboptimal label at termination is false. Thus, all nodes on the shortest path, including t, must have their optimal labels. That is, upon termination, $d_t = \delta(s,t)$. □

2.2 Label Correcting Algorithm as a Special Case

We can see that the Label Correcting Algorithm given in class is a special case of the modified algorithm through comparing their update rules:

- **Standard LCA Update Rule:**

$$d_i + a_{ij} < \min\{d_j, \text{UPPER}\}$$

- **Modified LCA Update Rule:**

$$d_i + a_{ij} < \min\{d_j, \text{UPPER} - u_j\}$$

it's straightforward that setting u_j to zero in modified LCA will lead to standard LCA. But it's also worth noting that setting $u_j = 0$ implies that $\delta(j,t) >= 0$ for any node j, which is always true only when arc length ≥ 0, which is consistent with the conditions for standard LCA.

Problem 3: Object Partition as Shortest Path Problem

We can model this problem as finding the shortest path in a directed acyclic graph (DAG). Let's construct the graph $G = (V, E)$ as follows:

- **Nodes:** We define a set of $N + 1$ nodes, $V = \{v_0, v_1, \ldots, v_N\}$. Each node v_k represents a potential boundary point in our partition. Specifically, node v_k represents making a partition after the k-th object. v_0 represents before the 1st object.
- **Edges:** For every pair (i, j) such that $0 \leq i < j \leq N$, we draw a directed edge from node v_i to node v_j. This edge (v_i, v_j) represents forming a single cluster containing objects $\{i + 1, i + 2, \ldots, j\}$.
- **Edge Weights:** The weight of edge (v_i, v_j), denoted as $w(v_i, v_j) = a_{i+1,j}$, is the cost of generating the corresponding cluster.
- **Node Distance:** The distance of node v_i, denoted as d_{v_i}, can be seen as the minimum cost to generate a valid partition for objects 1 i. Thus, the distance of node v_N is the minimum cost for partitioning all objects.

Therefore, the problem of finding the minimum cost grouping is equivalent to finding the **shortest path** from node v_0 to node v_N in the constructed graph G. We can either solve it using any shortest path algorithm, or in a DP style. For the latter, the state transition is:

$$d_{v_i} = \min_{0 \leq j < i} \{d_{v_j} + a_{j+1,i}\}$$

and $d_{v_0} = 0$. Iteration over all nodes v_1 v_N will give the minimum cost for partitioning all objects. To recover the actual grouping, one can record the predecessor for each node in shortest-path solution, or record the $\arg\min_j \{d_{v_j} + a_{j+1,i}\}$ for each i during Iteration.

Algorithm 1 Optimal Clustering DP

Input: Number of objects N, and costs a_{ij} for $1 \leq i \leq j \leq N$.
Output: The minimum total cost and the corresponding partition.
Initialize arrays $C[0 \dots N]$ and $P[1 \dots N]$.
$C[0] \leftarrow 0$
for $j \leftarrow 1$ **to** N **do**
  $C[j] \leftarrow \infty$
  for $i \leftarrow 0$ **to** $j - 1$ **do**
    if $C[i] + a_{i+1,j} < C[j]$ **then**
      $C[j] \leftarrow C[i] + a_{i+1,j}$
      $P[j] \leftarrow i$
    end if
  end for
end for
{The minimum cost is $C[N]$. Now, reconstruct the clusters.}
Initialize an empty list of clusters.
$j \leftarrow N$
while $j > 0$ **do**
  $i \leftarrow P[j]$
  Add cluster $\{i + 1, \dots, j\}$ to the clusters.
  $j \leftarrow i$
end while
return $C[N]$ and clusters.

Problem 4: Dynamic Programming with Discount Factor

To derive the given DP algorithm, we will use the Principle of Optimality. We define the value function $V_k(x_k)$ as the minimum expected discounted cost-to-go from stage k to stage N, given that the system is in state x_k at time k.

According to the definition, $V_k(x_k)$ should be:

$$V_k(x_k) = \min_{\mu_k,\ldots,\mu_{N-1}} E_{w_k,\ldots,w_{N-1}} \left(\sum_{j=k}^{N-1} \alpha^{j-k} g_j(x_j, \mu_j(x_j), w_j) + \alpha^{N-k} g_N(x_N) \right)$$

This definition implies that $V_k(x_k)$ represents the minimum expected future cost, where each instantaneous cost g_j is discounted by α^{j-k} relative to stage k.

For the terminal stage $k = N$, the summation $\sum_{j=N}^{N-1}$ is empty. According to our definition of $V_N(x_N)$:

$$V_N(x_N) = \min E\left(\alpha^{N-N} g_N(x_N)\right) = g_N(x_N)$$

Which matches the first equation provided in the problem statement.

For any stage $k < N$, we apply the Principle of Optimality. The optimal cost $V_k(x_k)$ can be expressed by considering the immediate cost at stage k and the optimal future cost from stage $k+1$.

$$V_k(x_k) = \min_{u_k \in U_k(x_k)} E_{w_k} \left[\alpha^{k-k} g_k + \min_{\mu_{k+1},\ldots,\mu_{N-1}} E_{w_{k+1},\ldots,w_{N-1}} \left(\sum_{j=k+1}^{N-1} \alpha^{j-k} g_j + \alpha^{N-k} g_N \right) \right]$$

Where we omit the dependence of $g_k(x_k, u_k, w_k)$ for simplicity. In which, the second term in the expectation:

$$\min_{\mu_{k+1},\ldots,\mu_{N-1}} E_{w_{k+1},\ldots,w_{N-1}} \left(\sum_{j=k+1}^{N-1} \alpha^{j-k} g_j + \alpha^{N-k} g_N \right)$$

We can factor out α from each term in the sum and the terminal cost:

$$= \min_{\mu_{k+1},\ldots,\mu_{N-1}} E_{w_{k+1},\ldots,w_{N-1}} \left(\alpha \sum_{j=k+1}^{N-1} \alpha^{j-(k+1)} g_j + \alpha \alpha^{N-(k+1)} g_N \right)$$

$$= \alpha \min_{\mu_{k+1},\ldots,\mu_{N-1}} E_{w_{k+1},\ldots,w_{N-1}} \left(\sum_{j=k+1}^{N-1} \alpha^{j-(k+1)} g_j + \alpha^{N-(k+1)} g_N \right)$$

Which matches exactly with previously defined $V_{k+1}(x_{k+1})$. Thus, we can re-write the expression as:

$$\begin{aligned} V_k(x_k) &= \min_{u_k \in U_k(x_k)} E_{w_k}\left[\alpha^{k-k} g_k + \alpha V_{k+1}(x_{k+1})\right] \\ &= \min_{u_k \in U_k(x_k)} E_{w_k}\left[g_k(x_k, u_k, w_k) + \alpha V_{k+1}(f_k(x_k, u_k, w_k))\right] \end{aligned}$$

Which matches exactly with the given DP algorithm.

Problem 5: Dynamic Programming with Conditional Termination

To reformulate the problem into the basic problem framework, we need to define a new augmented state, augmented system dynamics, and augmented cost functions that capture the termination logic and costs.

Let the original state be x_k. We augment the state x_k with a binary flag s_k indicating whether the system has terminated. Thus the new state is $\tilde{x}_k = (x_k, s_k)$. When $s_k =$ TRUE, the specific value of x_k is no longer relevant for future evolution or costs. We can introduce a dummy state x_{dummy} for this component.

Then, the new system dynamics can be written as:

$$\tilde{x}_{k+1} = \begin{cases} (x_{\text{dummy}}, \text{TRUE}) & \text{if } s_k = \text{TRUE or } w_k = \bar{w} \text{ or } u_k = \bar{u} \\ (f_k(x_k, u_k, w_k), \text{FALSE}) & \text{otherwise} \end{cases}$$

The new control policy is:

$$\tilde{u}_k(\tilde{x}_k) = \begin{cases} u_{\text{dummy}} & \text{if } s_k = \text{TRUE} \\ u_k(x_k) & \text{otherwise} \end{cases}$$

Where u_{dummy} acts as a padding strategy, only used to ensure the consistency of the formula.

The new cost function is:

$$\tilde{g}_k(\tilde{x}_k, \tilde{u}_k, w_k) = \begin{cases} 0 & \text{if } s_k = \text{TRUE} \\ g_k(x_k, u_k, w_k) + T & \text{if } s_k = \text{FALSE and } (w_k = \bar{w} \text{ or } u_k = \bar{u}) \\ g_k(x_k, u_k, w_k) & \text{otherwise} \end{cases}$$

To this end, we have reformulated the problem into the basic problem framework. One can directly use the formula from the basic problem to solve this problem with a termination state:

$$E\left[\tilde{g}_N(\tilde{x}_N) + \sum_{k=i}^{N-1} \tilde{g}_k(\tilde{x}_k, \tilde{u}_k(\tilde{x}_k), w_k)\right]$$

Problem 6: Unbounded Knapsack Problem

To handle the product maximization, we transform it into a sum maximization by taking the natural logarithm of the objective function. Since $p_j(m_j) \in (0, 1]$ (assuming non-zero success probabilities), $\log(p_j(m_j)) \leq 0$. Maximizing the sum of these logarithms is equivalent to maximizing their product. Let $g_j(m_j) = \log(p_j(m_j))$. The transformed problem is:

$$\text{Maximize } \sum_{j=1}^{N} g_j(m_j)$$

$$\text{Subject to } \sum_{j=1}^{N} c_j m_j \leq A, \quad m_j \in \{0, 1, 2, \ldots\}$$

Which is a classic unbounded knapsack problem.

Let $dp(j, k)$ represent the maximum total log-reliability for the first j stages (i.e., stages $1, 2, \ldots, j$), given that a total cost of exactly k has been spent on spare components for these j stages.

- j: The current stage index, ranging from 0 to N.
- k: The total cost spent on spare components for stages 1 through j, ranging from 0 to A.

Then we have base cases:

- $dp(0, 0) = 0$: represents the state before considering any stages, where no cost has been spent and the accumulated log-reliability is 0 (an empty product is 1).
- $dp(0, k) = -\infty$ for $k = 1, \ldots, A$: Any state with a non-zero cost before processing any stages is unreachable. We use $-\infty$ to denote an unreachable state or a state with zero reliability (since $\log(0) = -\infty$).

The state transition would be:

$$dp(j, k) = \max_{0 \leq m_j \leq \lfloor k/c_j \rfloor} \{dp(j-1, k - c_j m_j) + g_j(m_j)\}$$

The maximization is performed over all valid integer values of m_j such that $c_j m_j \leq k$ (to ensure $k - c_j m_j \geq 0$) and $m_j \geq 0$. We initialize $dp(j, k) = -\infty$ for all j, k before computation. We only consider terms where $dp(j-1, k - c_j m_j) \neq -\infty$.

After filling the DP table up to stage N, the maximum total log-reliability for the entire device, considering all possible budget allocations up to A, is:

$$\max_{0 \le k \le A} dp(N, k)$$

The maximum overall device reliability is then obtained by exponentiating this value:

$$\text{Maximum Reliability} = e^{\max_{0 \le k \le A} dp(N,k)}$$

To reconstruct the optimal solution, one can store the optimal m_j for each j, k during iteration, and then backtrack to find the optimal solution.

Problem 7: Monotonicity Property of Dynamic Programming

We need to prove two statements:

1. If $J_{N-1}(x) \leq J_N(x)$ for all $x \in S$, then $J_k(x) \leq J_{k+1}(x)$ for all $x \in S$ and $k \in \{0, \ldots, N-1\}$.
2. If $J_{N-1}(x) \geq J_N(x)$ for all $x \in S$, then $J_k(x) \geq J_{k+1}(x)$ for all $x \in S$ and $k \in \{0, \ldots, N-1\}$.

The core of the proof relies on the monotonicity of the Bellman operator:

For a time-invariant system, we can define the Bellman operator T as: $T(J)(x) = \min_{u \in U(x)} E_w[g(x,u,w) + J(f(x,u,w))]$ The DP recursion can then be written as $J_k(x) = T(J_{k+1})(x)$.

Lemma 3. *If two functions $J(y)$ and $\bar{J}(y)$ satisfy $J(y) \leq \bar{J}(y)$ for all $y \in S$, then $T(J)(x) \leq T(\bar{J})(x)$ for all $x \in S$.*

Proof. Assume $J(y) \leq \bar{J}(y)$ for all $y \in S$. For any given state $x \in S$ and control $u \in U(x)$, we have: $g(x,u,w) + J(f(x,u,w)) \leq g(x,u,w) + \bar{J}(f(x,u,w))$. Taking the expectation with respect to w (which preserves inequalities): $E_w[g(x,u,w) + J(f(x,u,w))] \leq E_w[g(x,u,w) + \bar{J}(f(x,u,w))]$. Let $Q_J(x,u) = E_w[g(x,u,w) + J(f(x,u,w))]$ and $Q_{\bar{J}}(x,u) = E_w[g(x,u,w) + \bar{J}(f(x,u,w))]$. Thus, $Q_J(x,u) \leq Q_{\bar{J}}(x,u)$ for all $u \in U(x)$.

Now, consider the definition of the Bellman operator: $T(J)(x) = \min_{u \in U(x)} Q_J(x,u), T(\bar{J})(x) = \min_{u \in U(x)} Q_{\bar{J}}(x,u)$. Let u^* be an optimal control for $T(\bar{J})(x)$, so $T(\bar{J})(x) = Q_{\bar{J}}(x,u^*)$. Since $Q_J(x,u) \leq Q_{\bar{J}}(x,u)$ for all $u \in U(x)$, it follows that $Q_J(x,u^*) \leq Q_{\bar{J}}(x,u^*)$. By the definition of the minimum, $T(J)(x) \leq Q_J(x,u^*)$. Combining these inequalities, we get: $T(J)(x) \leq Q_J(x,u^*) \leq Q_{\bar{J}}(x,u^*) = T(\bar{J})(x)$. Therefore, the monotonicity of Bellman operator T is guaranteed. □

Lemma 4. *If $J_{N-1}(x) \leq J_N(x)$ for all $x \in S$, then $J_k(x) \leq J_{k+1}(x)$ for all $x \in S$ and $k \in \{0, \ldots, N-1\}$.*

Proof. We will prove this by backward induction on k.

Inductive Hypothesis: Assume that for some $k \in \{0, \ldots, N-2\}$, we have $J_{k+1}(x) \leq J_{k+2}(x)$ for all $x \in S$.

Inductive Step: We need to show that $J_k(x) \leq J_{k+1}(x)$ for all $x \in S$. From the Bellman equation, we have:

$$J_k(x) = T(J_{k+1})(x)$$
$$J_{k+1}(x) = T(J_{k+2})(x)$$

By the inductive hypothesis, $J_{k+1}(y) \leq J_{k+2}(y)$ for all $y \in S$. Since the Bellman operator T is monotonic, applying T to both sides of the inequality $J_{k+1} \leq J_{k+2}$ preserves the inequality: $T(J_{k+1})(x) \leq T(J_{k+2})(x)$. Substituting the definitions of $J_k(x)$ and $J_{k+1}(x)$ from the Bellman equation: $J_k(x) \leq J_{k+1}(x)$. This completes the induction. □

Lemma 5. *If $J_{N-1}(x) \geq J_N(x)$ for all $x \in S$, then $J_k(x) \geq J_{k+1}(x)$ for all $x \in S$ and $k \in \{0, \ldots, N-1\}$.*

Proof. This can be proven using the exact same steps as above.

Inductive Hypothesis: Assume that for some $k \in \{0, \ldots, N-2\}$, we have $J_{k+1}(x) \geq J_{k+2}(x)$ for all $x \in S$.

Inductive Step: We need to show that $J_k(x) \geq J_{k+1}(x)$ for all $x \in S$. From the Bellman equation, we have:

$$J_k(x) = T(J_{k+1})(x)$$
$$J_{k+1}(x) = T(J_{k+2})(x)$$

By the inductive hypothesis, $J_{k+1}(y) \geq J_{k+2}(y)$ for all $y \in S$. Since the Bellman operator T is monotonic, applying T to both sides of the inequality $J_{k+1} \geq J_{k+2}$ preserves the inequality: $T(J_{k+1})(x) \geq T(J_{k+2})(x)$. Substituting the definitions of $J_k(x)$ and $J_{k+1}(x)$ from the Bellman equation: $J_k(x) \geq J_{k+1}(x)$. This completes the induction. □

To this end, we have proved the monotonicity property of DP.

Problem 8: Multiplicative Cost Dynamic Programming

In the framework of a basic finite-horizon optimal control problem, we consider a system with state dynamics given by:

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

where x_k is the state at stage k, u_k is the control chosen at stage k, and w_k is a random disturbance occurring at stage k. The initial state x_0 is given.

The problem specifies a multiplicative cost function:

$$E_w[g_N(x_N) \prod_{k=1}^{N} g_k(x_k, u_k, w_k)]$$

To provide a well-posed DP-like algorithm within the "basic problem framework", we interpret the cost function as follows:

- $g_N(x_N)$ is the terminal cost, dependent only on the final state x_N.
- The product term $\prod_{k=0}^{N-1} g_k(x_k, u_k, w_k)$ represents the product of stage costs, where $g_k(x_k, u_k, w_k)$ is the cost factor incurred at stage k due to state x_k, control u_k, and disturbance w_k. The original problem's indexing ($k = 1$ to N in the product) is adjusted to $k = 0$ to $N-1$ to align with standard finite-horizon DP where controls $u_0, \ldots, u_{N-1}$ are chosen.

Thus, the total cost function to be minimized is interpreted as:

$$J = E_{w_0,\ldots,w_{N-1}} \left[g_N(x_N) \cdot \prod_{k=0}^{N-1} g_k(x_k, u_k, w_k) \right]$$

The objective is to find a policy $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$ such that $u_k = \mu_k(x_k)$ minimizes J. The problem states that $g_k(x_k, u_k, w_k) \geq 0$ for all x_k, u_k, w_k, which is crucial for the well-definedness of the minimization problem with a multiplicative cost.

8.1 Dynamic Programming Algorithm

We develop a dynamic programming algorithm using a backward pass approach. Let $J_k(x_k)$ denote the minimum expected cost from stage k to the terminal stage N, given that the system is in state x_k at stage k.

1. Terminal Stage (N)

At the terminal stage N, no further controls are chosen. The remaining cost is simply the terminal cost $g_N(x_N)$.

$$J_N(x_N) = g_N(x_N)$$

for all possible states x_N.

2. Iteration for Stages $k = N-1, N-2, \ldots, 0$ (Backward Pass)

For each stage k (from $N-1$ down to 0) and for each possible state x_k: The decision is to choose a control u_k. This choice incurs an immediate multiplicative cost factor $g_k(x_k, u_k, w_k)$ and transitions the system to a new state $x_{k+1} = f_k(x_k, u_k, w_k)$. The minimum expected future cost from x_{k+1} is $J_{k+1}(x_{k+1})$. Since the cost is multiplicative and all $g_k \geq 0$, the total expected cost from stage k onwards, given x_k and u_k, is the expectation of the product of the current cost factor and the future minimum expected cost. The Bellman equation for this problem is:

$$J_k(x_k) = \min_{u_k \in U_k(x_k)} E_{w_k}[g_k(x_k, u_k, w_k) \cdot J_{k+1}(f_k(x_k, u_k, w_k))]$$

where $U_k(x_k)$ is the set of admissible controls at state x_k at stage k. The optimal control policy $\mu_k(x_k)$ for state x_k at stage k is stored as:

$$\mu_k(x_k) = \arg \min_{u_k \in U_k(x_k)} E_{w_k}[g_k(x_k, u_k, w_k) \cdot J_{k+1}(f_k(x_k, u_k, w_k))]$$

After computing $J_k(x_k)$ and $\mu_k(x_k)$ for all stages $k = N-1, \ldots, 0$ and all relevant states x_k, the minimum expected total cost from the initial state x_0 is $J_0(x_0)$. The optimal policy is given by the sequence of functions $\pi^* = \{\mu_0, \mu_1, \ldots, \mu_{N-1}\}$.

One might be tempted to use a seemingly viable transformation by taking the logarithm of the total cost J and attempting to convert it directly into a standard additive DP problem:

$$\begin{aligned}\log(J) &= \log\left(E_{w_0,\ldots,w_{N-1}}\left[g_N(x_N) \cdot \prod_{k=0}^{N-1} g_k(x_k, u_k, w_k)\right]\right) \\ &= E_{w_0,\ldots,w_{N-1}}\left[\log(g_N(x_N)) + \sum_{k=0}^{N-1} \log(g_k(x_k, u_k, w_k))\right]\end{aligned}$$

however, this is incorrect because the logarithm is a non-linear function that does not commute with the expectation operator, meaning $\log(E[X]) \neq E[\log X]$.

Problem 9: Flexible Employment

The original solution title was "Optimal Job Reception Problem", but the term "灵活就业" suddenly came to my mind. Quite interesting anyway.

Let $P(w_j)$ be the probability of receiving a job offer with salary w_j, c be the unemployment benefit, and $\alpha < 1$ be the discount factor. Define the following value functions:

- $V(\bar{s}_j)$: The maximum expected discounted income if the worker is employed at salary w_j.
- $V(s_j)$: The maximum expected discounted income if the worker is unemployed and receives an offer w_j.
- V_U: The maximum expected discounted income if the worker is unemployed and has not yet received an offer (i.e., before the current period's offer is revealed).

9.1 Part (a)

If the worker accepts a job with salary w_j, she (why is "she"?) will earn w_j indefinitely. The total discounted income is an infinite geometric series:

$$V(\bar{s}_j) = w_j + \alpha w_j + \alpha^2 w_j + \cdots = \sum_{k=0}^{\infty} \alpha^k w_j = \frac{w_j}{1-\alpha}$$

When being unemployed and offred w_j, The worker has two options:

1. **Accept w_j:** She receives w_j in the current period and transitions to the employed state $\bar{s}_j$ in the next period. The total value is $w_j + \alpha V(\bar{s}_j)$. Substituting $V(\bar{s}_j)$:

$$w_j + \alpha \frac{w_j}{1-\alpha} = w_j \left(1 + \frac{\alpha}{1-\alpha}\right) = w_j \left(\frac{1-\alpha+\alpha}{1-\alpha}\right) = \frac{w_j}{1-\alpha}$$

2. **Reject w_j:** She receives unemployment benefit c in the current period and remains unemployed, waiting for a new offer in the next period. The total value is $c + \alpha V_U$.

Thus, the Bellman equation for $V(s_j)$ is:

$$V(s_j) = \max\left(\frac{w_j}{1-\alpha}, \quad c + \alpha V_U\right)$$

Where V_U is the expected value of $V(s_k)$ over all possible offers w_k:

$$V_U = \sum_{k=1}^{n} P(w_k) V(s_k)$$

Substituting $V(s_k)$:

$$V_U = \sum_{k=1}^{n} P(w_k) \max\left(\frac{w_k}{1-\alpha}, \quad c + \alpha V_U\right)$$

This is an implicit equation for V_U. Due to the monotonicity of the max function and $\alpha < 1$, this equation has a unique fixed-point solution for V_U.

The worker will accept the job offer w_j if the value of accepting is greater than or equal to the value of rejecting:

$$\frac{w_j}{1-\alpha} \geq c + \alpha V_U$$

Multiplying both sides by $(1-\alpha)$:

$$w_j \geq (1-\alpha)(c + \alpha V_U)$$

Let $\bar{w} = (1-\alpha)(c + \alpha V_U)$. The optimal strategy is to accept a job offer w_j if and only if $w_j \geq \bar{w}$. This proves the existence of such a threshold.

9.2 Part (b)

Case 1: Firing probability $p_i = p$ for all i

If employed at w_j, the worker earns w_j in the current period. In the next period, with probability $1-p$, she remains employed at w_j, and with probability p, she is fired and returns to the unemployed state (before receiving a new offer).

$$V(\bar{s}_j) = w_j + \alpha\left[(1-p)V(\bar{s}_j) + pV_U\right]$$

Solving for $V(\bar{s}_j)$:

$$\begin{aligned} V(\bar{s}_j) - \alpha(1-p)V(\bar{s}_j) &= w_j + \alpha p V_U \\ V(\bar{s}_j)(1 - \alpha(1-p)) &= w_j + \alpha p V_U \\ V(\bar{s}_j) &= \frac{w_j + \alpha p V_U}{1 - \alpha(1-p)} \end{aligned}$$

The Bellman equation for $V(s_j)$ remains:

$$V(s_j) = \max\left(w_j + \alpha V(\bar{s}_j), \quad c + \alpha V_U\right)$$

The worker accepts w_j if $w_j + \alpha V(\bar{s}_j) \geq c + \alpha V_U$. Substitute $V(\bar{s}_j)$:

$$\begin{aligned} w_j + \alpha\frac{w_j + \alpha p V_U}{1 - \alpha(1-p)} &\geq c + \alpha V_U \\ w_j\left(1 + \frac{\alpha}{1 - \alpha(1-p)}\right) &\geq c + \alpha V_U - \frac{\alpha^2 p V_U}{1 - \alpha(1-p)} \\ w_j\left(\frac{1 - \alpha + \alpha p + \alpha}{1 - \alpha + \alpha p}\right) &\geq c + V_U\left(\frac{\alpha(1 - \alpha + \alpha p) - \alpha^2 p}{1 - \alpha + \alpha p}\right) \\ w_j\left(\frac{1 + \alpha p}{1 - \alpha + \alpha p}\right) &\geq c + V_U\left(\frac{\alpha - \alpha^2 + \alpha^2 p - \alpha^2 p}{1 - \alpha + \alpha p}\right) \\ w_j\left(\frac{1 + \alpha p}{1 - \alpha + \alpha p}\right) &\geq c + V_U\left(\frac{\alpha(1-\alpha)}{1 - \alpha + \alpha p}\right) \end{aligned}$$

Multiplying both sides by $(1-\alpha+\alpha p)$:

$$w_j(1+\alpha p) \geq c(1-\alpha+\alpha p) + V_U\alpha(1-\alpha)$$
$$w_j \geq \frac{c(1-\alpha+\alpha p) + V_U\alpha(1-\alpha)}{1+\alpha p}$$

Let $\bar{w}' = \frac{c(1-\alpha+\alpha p)+V_U\alpha(1-\alpha)}{1+\alpha p}$. This $\bar{w}'$ is a constant threshold (once V_U is determined). Therefore, the result from part (a) still holds: there exists a single threshold $\bar{w}'$, and the worker accepts w_j if and only if $w_j \geq \bar{w}'$. V_U is still determined by its fixed-point equation, incorporating the new $V(\bar{s}_k)$.

Case 2: Firing probability p_i depends on i

The firing probability is now p_j, specific to salary w_j.

$$V(\bar{s}_j) = w_j + \alpha\left[(1-p_j)V(\bar{s}_j) + p_jV_U\right]$$

Solving for $V(\bar{s}_j)$:

$$V(\bar{s}_j) = \frac{w_j + \alpha p_j V_U}{1-\alpha(1-p_j)}$$

$V(s_j)$:

$$V(s_j) = \max\left(w_j + \alpha V(\bar{s}_j), \quad c + \alpha V_U\right)$$

The worker accepts w_j if $w_j + \alpha V(\bar{s}_j) \geq c + \alpha V_U$. Substituting $V(\bar{s}_j)$ and performing the same algebraic steps as in Case(a) (replacing p with p_j):

$$w_j \geq \frac{c(1-\alpha+\alpha p_j) + V_U\alpha(1-\alpha)}{1+\alpha p_j}$$

Let $RHS_j = \frac{c(1-\alpha+\alpha p_j)+V_U\alpha(1-\alpha)}{1+\alpha p_j}$.

In this case, the condition for accepting an offer w_j is $w_j \geq RHS_j$. The critical observation is that RHS_j is no longer a constant threshold. Instead, it depends on p_j, which is specific to the offer w_j. Which implies the decision depends on both salary and job security.

The worker may weigh the offered salary w_j against its associated firing probability p_j. A high salary w_j with a high p_j might be less attractive than a slightly lower salary w_k with a very low p_k. The optimal strategy is to accept an offer w_j if its value $w_j + \alpha V(\bar{s}_j)$ (which incorporates p_j) is greater than or equal to the value of continuing to search, $c + \alpha V_U$.