# CHAPTER 12
# Dynamic Programming

**Problem 12.1**

As the discount factor $\gamma$ approaches 1, the computation of the cost-to-go function $J^{\pi}(i)$ becomes longer because of the corresponding increase in the number of time steps involved in the computation.

**Problem 12.2**

(a) Let $\pi$ be an arbitrary policy, and suppose that this policy chooses action $a \in A_i$ at time step 0. We may then write

$$J^{\pi}(i) = \sum_{a \in A_i} p_a \left( c(i, a) + \sum_{j=1}^{N} p_{ij}(a) W^{\pi}(j) \right)$$

where $p_a$ is the probability of choosing action $a$, $c(i, a)$ is the expected cost, $p_{ij}(a)$ is the probability of transition from state $i$ to state $j$ under action $a$, $W^{\pi}(j)$ is the expected cost-to-go function from time step $n = 1$ onward, and $j$ is the state at that time step. We now note that

$$W^{\pi}(j) \leq \gamma \ J(j)$$

which follows from the observation that if the state at time step $n = 1$ is $j$, then the situation at that time step is the same as if the process had started in state $j$ with the exception that all the returns are multiplied by the discount factor $\gamma$. Hence, we have

$$\begin{aligned} J^{\pi}(i) &\geq p_a \left( c(i, a) + \gamma \sum_j p_{ij}(a) J(j) \right) \\ &\geq p_a \min_{a \in A_i} \left( c(i, a) + \gamma \sum_j p_{ij}(a) J(j) \right) \\ &= \min_a \left( c(i, a) + \gamma \sum_j p_{ij}(a) J(j) \right) \end{aligned}$$

which implies that

$$J^{\pi}(i) \min_{a \in A_i} \geq \left( c(i, a) + \gamma \sum_j p_{ij}(a) J(j) \right) \tag{1}$$

(b) Suppose we next go the other way by choosing $a_0$ with

$$c(i, a_0) + \gamma \sum_j p_{ij}(a_0) = \min_a \left( c(i, a) + \gamma \sum_j p_{ij}(a) J(j) \right) \quad \textbf{(2)}$$

Let $\pi$ be the policy that chooses $a_0$ at time step 0 and, if the next state is $j$, the process is viewed as originating in state $j$ following a policy $\pi_j$ such that

$$J^{\pi_j} \leq J(j) + \varepsilon$$

Where $\varepsilon$ is a small positive number. Hence

$$\begin{aligned} J^{\pi_j} &= c(i, a_0) + \sum_{j=1}^{N} p_{ij}(a_0) J^{\pi_j}(j) \\ &\leq c(i, a_0) + \sum_{j=1}^{N} p_{ij}(a_0) J(j) + \gamma\varepsilon \end{aligned} \quad (3)$$

Since $J(i) \leq J^{\pi}(i)$, (3) implies that

$$J(i) \leq c(i, a_0) + \gamma \sum_{j=1}^{N} p_{ij}(a_0) J(j) + \gamma\varepsilon$$

Hence, from (2) it follows that

$$J(i) \leq \min_{a \in A_i} \left( c(i, a) + \gamma \sum_j p_{ij}(a) J(j) \right) + \gamma\varepsilon \quad (4)$$

(c) Finally, since $\varepsilon$ is arbitrary, we immediately deduce from (1) and (4) that the optimum cost-to-go function

$$J^*(i) = \min_a \left( c(i, a) + \gamma \sum_j p_{ij}(a) J^*(j) \right) \quad (5)$$

which is the desired result

**Problem 12.3**

Writing the system of $N$ simultaneous equations (12.22) of the text in matrix form:

$$\mathbf{J}^{\mu} = \mathbf{c}(\mu) + \gamma \mathbf{P}(\mu)\mathbf{J}^{\mu} \tag{1}$$

where

$$\mathbf{J}^{(\mu)} = \left[ J^{(\mu)}(1),\ J^{(\mu)}(2),\ \ldots\ J^{(\mu)}(N) \right]^T$$

$$\mathbf{c}(\mu) = \left[ \mathbf{C}(1, \mu),\ \mathbf{C}(2, \mu),\ \ldots\ \mathbf{C}(N, \mu) \right]^T$$

$$\mathbf{P}(\mu) = \begin{bmatrix} p_{11}(\mu) & p_{12}(\mu) & \ldots & p_{1N}(\mu) \\ p_{21}(\mu) & p_{22}(\mu) & \ldots & p_{2N}(\mu) \\ \vdots & \vdots & & \vdots \\ p_{N1}(\mu) & p_{N2}(\mu) & \ldots & p_{NN}(\mu) \end{bmatrix}$$

Rearranging terms in 1), we may write

$$\mathbf{I} - \gamma \mathbf{P}(\mu)\mathbf{J}^{(\mu)} = \mathbf{c}(\mu)$$

where **I** is the *N*-by-*N* identity matrix. For the solution $\mathbf{J}^{\mu}$ to be unique we require that the *N*-by-*N* matrix (**I** - γ**P**(μ)) have an inverse matrix for all possible values of the discount factor γ.

**Problem 12.4**

Consider an admissible policy $\{\mu_0, \mu_1, ...\}$, a positive integer *K*, and cost-to-go function *J*. Let the costs of the first *K* stages be accumulated, and add the terminal cost $\gamma^K J(X_K)$, thereby obtaining the total expected cost

$$\mathbf{E}\left[ \gamma^K J(X_K) + \sum_{n=0}^{K-1} \gamma^n g(X_n, \mu_n(X_n), X_{n-1}) \right]$$

where **E** is the expectational operator, To minimize the total expected cost, we start with $\gamma^K J(X_K)$ and perform *K* iterations of the dynamic programming algorithm, as shown by

$$J_n(X_n) = \min_{\mu_n} \mathbf{E}[g_n(X_n, \mu_n(X_n), X_{n-1}) + J_{n+1}(X_{n+1})] \tag{1}$$

with the initial condition

$$J_K(X) = \gamma^K J(X)$$

Now consider the function $V_n$ defined by

$$V_n(X) = \frac{J_{K-n}(X)}{\gamma^{K-n}} \quad \text{for all } n \text{ and } X \tag{2}$$

The function $V_n(X)$ is the optimal $K$-stage cost $J_0(X)$. Hence, the dynamic programming algorithm of (1) can be rewritten in terms of the function $V_n(X)$ as follows:

$$V_{n+1}(X_0) = \min_{\mu} \mathbf{E}[g(X_0, \mu(X_0), X_1) + \gamma V_n(X_1)]$$

with the initial condition

$V_0(X) = J(X)$

which has the same mathematical form as that specified in the problem.

**Problem 12.5**

An important property of dynamic programming is the *monotonicity* property described by

$$J^{\mu_{n+1}} \leq J^{\mu_n}$$

This property follows from the fact that if the terminal cost $g_K$ for $K$ stages is changed to a uniformly larger cost $\bar{g}_K$, that is,

$$\bar{g}_K(X_K) \geq g_K(X_K) \quad \text{for all } X_K,$$

then the last stage cost-to-go function $J_{K-1}(X_{K-1})$ will be uniformly increased. In more general terms, we may state the following.

> Given two cost-to-go functions $J_{K'+1}$ and $\bar{J}_{K+1}$ with
>
> $$\bar{J}_{K+1}(X_{K+1}) \geq J_{K+1}(X_{K+1}) \quad \text{for all } X_{K+1},$$
>
> we find that for all $X_K$ and $\mu_K$ the following relation holds
>
> $$\mathbf{E}[g_K(X_K, \mu_K, X_{K+1}) + J_{K+1}(X_{K+1})] \leq \mathbf{E}[g_K(X_K, \mu_K, X_{K+1}) + \bar{J}_{K+1}(X_{K+1})]$$

This relation merely restates the monotonicity property of the dynamic programming algorithm.

**Problem 12.6**

According to (12.24) of the text the $Q$-factor for state-action pair ($i$, $a$) and stationary policy $\mu$ satisfies the condition

$$Q^{\mu}(i, \mu(i)) = \min_{a} Q^{\mu}(i, a) \qquad \text{for all } i$$

This equation emphasizes the fact that the policy $\mu$ is *greedy* with respect to the cost-to-go function $J^{\mu}(i)$.

**Problem 12.7**

Figure 1, shown below, presents an interesting interpretation of the policy iteration algorithm. In this interpretation, the policy evaluation step is viewed as the work of a *critic* that evaluates the performance of the current policy; that is, it calculates an estimate of the cost-to-go function $J^{\mu_n}$. The policy improvement step is viewed as the work of a *controller* or *actor* that accounts for the latest evaluation made by her critic and acts out the improved policy $\mu_{n+1}$. In short, the critic looks after policy evaluation and the controller (actor) looks after policy improvement and the iteration between them goes on.

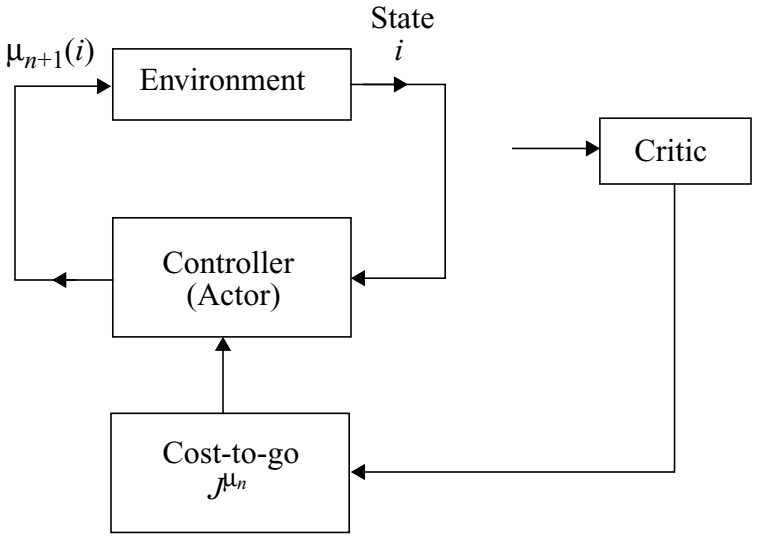


Figure 1: Problem 12.7

**Problem 12.8**

From (12.29) in the text, we find that for each possible state, the value iteration algorithm requires $NM$ iterations, where $N$ is the number of states and $M$ is the number of admissible actions. Hence, the total number of iterations for all $N$ states in $N^2M$.

**Problem 12.9**

To reformulate the value-iteration algorithm in terms of $Q$-factors, the only change we need to make is in step 2 of Table 12.2 in the text. Specifically, we rewrite this step as follows:

For $n = 0, 1, 2, \ldots$, compute

$$Q(i, a) = c(i, a) + \gamma \sum_{j=1}^{N} p_{ij}(a) J_n(j)$$

$$J_{n+1}(i) = \min_a Q(i, a)$$

**Problem 12.10**

The policy-iteration algorithm alternates between two steps: policy evaluation, and policy improvement. In other words, an optimal policy is computed directly in the policy iteration algorithm. In contrast, no such thing happens in the value iteration algorithm.

Another point of difference is that in policy iteration the cost-to-go function is recomputed on each iteration of the algorithm. This burdensome computational difficulty is avoided in the value-iteration algorithm.

**Problem 12.14**

From the definition of $Q$-factor given in (12.24) in the text and Bellman's optimality equation (12.11), we immediately see that

$$J^*(i) = \min_a Q(i, a)$$

where the minimization is performed over all possible actions $a$.

**Problem 12.15**

The value-iteration algorithm requires knowledge of the state transition probabilities. In contrast, $Q$-learning operates without this knowledge. But through an interactive process, $Q$-learning learns estimates of the transition probabilities in an implicit manner. Recognizing the intimate relationship between value iteration and $Q$-learning, we may therefore view $Q$-learning as an adaptive version of the value-iteration algorithm.

**Problem 12.16**

Using Table 12.4 in the text, we may construct the signal-flow graph in Figure 1 for the *Q*-learning algorithm:

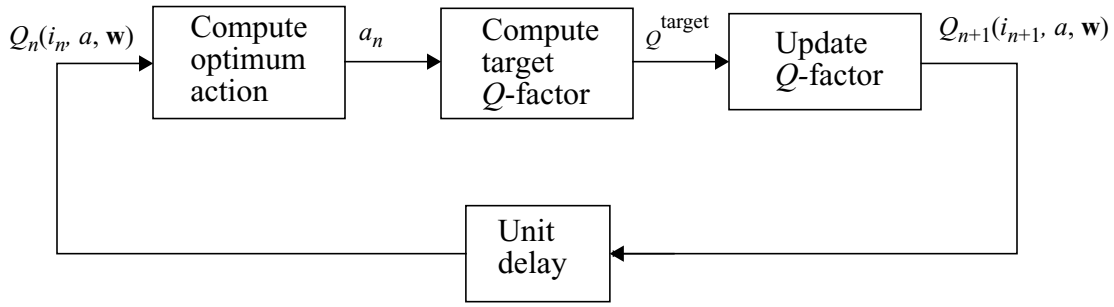


Figure 1: Problem 12.16

**Problem 12.17**

The whole point of the *Q*-learning algorithm is that it eliminates the need for knowing the state transition probabilities. If knowledge of the state transition probabilities is available, then the *Q*-learning algorithm assumes the same form as the value-iteration algorithm.