

Lecture 4: Infinite-Horizon Discounted Problems (II)

We continue our study on infinite-horizon discounted problems, and address several computational methods in this lecture. They include:

- policy iteration
- modified policy iteration
- asynchronous value iteration
- asynchronous modified policy iteration
- the linear programming approach

We will follow the reference books more closely than we did in earlier lectures. Some of the identical proofs will be sketched or omitted.

A Brief Review

Recall some of the results that we proved in the last lecture:

- T and T_μ are sup-norm contraction mappings, so J^* and J_μ are the unique solution to the DP equations $J = TJ$ and $J = T_\mu J$, respectively, and value iteration $T^k J \rightarrow J^*$ and $T_\mu^k J \rightarrow J_\mu$.
- T and T_μ satisfies the monotonicity property: $J_1 \geq J_2 \Rightarrow TJ_1 \geq TJ_2$, $T_\mu J_1 \geq T_\mu J_2$. This implies in particular that

$$TJ \geq J \Rightarrow J^* \geq J, \quad T_\mu J \geq J \Rightarrow J_\mu \geq J,$$

and

$$TJ \leq J \Rightarrow J^* \leq J, \quad T_\mu J \leq J \Rightarrow J_\mu \leq J.$$

- For a scalar c and a constant vector $e = [1, \dots, 1]'$,

$$T(J + ce) = TJ + \alpha ce \quad T_\mu(J + ce) = T_\mu J + \alpha ce.$$

We will use these results in what follows.

4.1 Policy Iteration

The policy iteration algorithm is similar to the one of the finite-horizon case: we evaluate the cost of a policy, and we then generate a policy with improved performance. For discounted problems, both theory and practice suggest that policy iteration has faster convergence than value iteration.

There are several variants of policy iteration algorithms, which differ in the policy evaluation step. Exact evaluation can be expensive and unnecessary; approximate evaluation – the modified policy iteration algorithm, and asynchronous evaluation algorithms, will be introduced.

Throughout this lecture, we consider deterministic and stationary policies. For such a policy μ , we denote by g_μ the vector of per-stage costs, and P_μ the state transition probability matrix, i.e.,

$$g_\mu = [g(1, \mu(1)), \dots, g(n, \mu(n))]', \quad (P_\mu)_{ij} = p(j|i, \mu(i)).$$

4.1.1 The Exact Algorithm

Policy Iteration Algorithm

1. Start with some policy μ_0 ; and let $k = 0$.
2. (Policy Evaluation) Compute $J_k = J_{\mu_k}$ by finding a solution J_k of

$$g_{\mu_k} + \alpha P_{\mu_k} J = J.$$

3. (Policy Improvement) Obtain a new policy μ_{k+1} such that

$$T_{\mu_{k+1}} J_k = T J_k,$$

and for all states s choose $\mu_{k+1}(s) = \mu_k(s)$ if possible.

4. If $\mu_{k+1} = \mu_k$ then stop; otherwise, increase k and go to step 2.

Comparison to Value Iteration

We do only one value iteration TJ_k per policy iteration. This is computationally much cheaper than value iteration, especially when the action set \mathcal{U} is large, because

- (i) each value iteration TJ involves solving n minimization problems $(TJ)(s), s = 1, \dots, n$, over the action set \mathcal{U} ; and
- (ii) it may take many value iterations $T^k J$ before the corresponding greedy policy actually changes to an improved policy, unlike in policy iteration (see the policy improvement lemma below).

While value iteration is a fixed point iteration for solving a fixed point equation $f(x) = x$, policy iteration has been related to Newton's method for solving a nonlinear equation $f(x) = 0$. See Exercise 1.10 of [Ber01] or pp. 179 of [Put94].

Proposition 1. *The policy iteration algorithm terminates in finite time and returns an optimal policy.*

First we prove the following lemma.

Lemma 1 (Policy Improvement Property). *If μ_k is not optimal, then the policy improvement step generates a policy μ_{k+1} that has costs no worse than μ_{k-1} for all states, and has strict cost improvement for at least one state, i.e.,*

$$J_{\mu_{k+1}} \leq J_{\mu_k} \quad \text{and} \quad \exists s \in \mathcal{S} \text{ s.t. } J_{\mu_{k+1}}(s) < J_{\mu_k}(s). \quad (1)$$

Proof. The first inequality is evident by the monotonicity of $T_{\mu_{k+1}}$ and the relation $T_{\mu_{k+1}} J_{\mu_k} = T J_{\mu_k} \leq J_{\mu_k}$. We prove the second strict inequality by contradiction. Assume that $J_{\mu_{k+1}} = J_{\mu_k}$, then

$$J_{\mu_k} = T_{\mu_{k+1}} J_{\mu_k} = T J_{\mu_k},$$

which implies that $J_{\mu_{k-1}}$ satisfies the optimality equation, and hence μ_{k-1} is optimal, a contradiction. \square

Proof of Prop. 1. By the policy improvement lemma, the policy iteration algorithm does not stop at a non-optimal policy, and does not generate the same non-optimal policy twice. Since in an MDP with a finite number of states and actions, there are only a finite number of deterministic and stationary policies, the policy iteration algorithm must generate an optimal policy in finite time. Since in the policy improvement step, the algorithm keeps the policy unchanged if possible, it must then terminate and return the optimal policy. \square

Convergence Rate

It is shown in [Put94] that under certain conditions, the convergence rate of policy iteration is quadratic, or superlinear, thus superior to the linear convergence rate of value iteration. (See Theorem 6.4.8, Corollary 6.4.10 and Example 6.4.2 of [Put94].)

Computing J_μ

One can solve the equation $J_\mu = g_\mu + \alpha P_\mu J_\mu$ by Gaussian elimination, which has the complexity of $O(n^3)$. The solution is

$$J_\mu = (I - \alpha P_\mu)^{-1} g_\mu. \quad (2)$$

The matrix $I - \alpha P_\mu$ is invertible, and in fact, for any stochastic matrix P , the matrix $I - \alpha P$ is invertible. This is because the spectral radius of P is 1, and thus the spectral radius of αP is strictly less than 1. (The spectral radius of a matrix is defined as the maximal modulus of the eigenvalues of that matrix.) Hence there cannot exist a non-zero vector x such that $(I - \alpha P)x = 0$, indicating that the matrix is non-singular.

One can also verify directly that the inverse of $I - \alpha P$ exists and

$$(I - \alpha P)^{-1} = I + \alpha P + \alpha^2 P^2 + \dots \quad (3)$$

(The right hand side is well-defined, because P^k are again stochastic matrices and thus all of their entries are bounded, and the discounting by α^k makes the summation bounded.) This formula can be verified by observing that

$$(I + \alpha P + \alpha^2 P^2 + \dots + \alpha^n P^n)(I - \alpha P) = I - \alpha P + \alpha P - \alpha^2 P^2 + \dots - \alpha^{n+1} P^{n+1} = I - \alpha^{n+1} P^{n+1}.$$

Thus the limit $(I + \alpha P + \alpha^2 P^2 + \dots)(I - \alpha P)$ is the identity matrix I .

4.1.2 Modified Policy Iteration

Instead of evaluating J_μ exactly, modified policy iteration performs a few iterations of value iteration (for a single policy), $T_\mu^m J$, to approximately evaluate J_μ .

Let $\{m_k\}$ be any sequence of positive numbers.

Modified Policy Iteration Algorithm

1. Start with some policy μ_0 and function J_0 . Let $k = 0$.
2. Do a policy update to choose μ_{k+1} such that

$$T_{\mu_{k+1}} J_k = T J_k,$$

and for all states s , choose $\mu_{k+1}(s) = \mu_k(s)$ if possible.

3. (Approximate Policy Evaluation) Let

$$J_{k+1} = T_{\mu_{k+1}}^{m_k} J_k.$$

Increase k and go back to step 2.

Comparison to Value Iteration

In each policy iteration, we do only one value iteration of the form TJ , and m_k iterations of the form $T_\mu J$, which are much cheaper than value iteration, because they do not involve solving minimization problems as TJ does. Nevertheless, under certain conditions, modified policy iteration is shown to have *faster* convergence rate than value iteration, (where the comparison is between an equal number of value updates in both algorithms). See Corollaries 6.5.7 and 6.5.8 of [Put94].

Choice of m_k

Consider the choices of the number of approximate policy evaluation iterations, m_k :

- The special case where $m_k = 1$ for all k , reduces to value iteration.
- The special case where $m_k = \infty$ for all k (as a conceptual algorithm), reduces to policy iteration.
- m_k can be chosen adaptively based on the difference in two consecutive approximate policy evaluations, $sp(T_{\mu_k}^m J_{k-1} - T_{\mu_k}^{m-1} J_{k-1})$, for instance.
- It is also suggested in [Put94], based on convergence rate analysis, to choose m_k with $m_k \rightarrow \infty$ as $k \rightarrow \infty$.

Proposition 2. *The function J_k and the cost of the policy μ_k generated by the modified policy iteration algorithm converge to J^* .*

Proof. (i) First we assume that J_0 is such that $T_{\mu_0} J_0 \leq J_0$. We will show that

$$T_{\mu_{k-1}} J_{k-1} \leq J_{k-1} \implies T_{\mu_k} J_k \leq J_k \leq T J_{k-1}. \quad (4)$$

This will imply that

$$T J_k \leq J_k \leq T J_{k-1} \leq J_{k-1} \leq T J_{k-2} \leq \dots \quad (5)$$

and we will be able to conclude that (a) J_k is monotonically decreasing and bounded below by J^* , and thus must converge to some limit $\bar{J} \in \mathfrak{R}^n$; (b) $J_k \leq T^k J_0$, so it must be that $\bar{J} = J^*$, because $T^k J_0 \rightarrow J^*$; and (c) $J_{\mu_k} \leq J_k$, and hence J_{μ_k} also converges to J^* .

We now prove the statement (4), and evidently it is sufficient to prove the case where $k = 1$. Indeed, since $T_{\mu_0} J_0 \leq J_0$, we have $T_{\mu_1} J_0 = T J_0 \leq T_{\mu_0} J_0 \leq J_0$, and thus

$$T_{\mu_1} J_0 \leq J_0 \implies T_{\mu_1}^{m_1+1} J_0 \leq T_{\mu_1}^{m_1} J_0 \leq \dots \leq T_{\mu_1} J_0.$$

By the definition of J_1 and μ_1 , this is equivalent to $T_{\mu_1} J_1 \leq J_1 \leq T J_0$. We thus proved (4).

(ii) We now consider an arbitrary J_0 . Since there exists some scalar c sufficiently large such that

$$T_{\mu_0} (J_0 + c e) = T_{\mu_0} J_0 + \alpha c e \leq J_0 + c e,$$

we can define $\tilde{J}_0 = J_0 + c e$ for such a c , which then satisfies $T_{\mu_0} \tilde{J}_0 \leq \tilde{J}_0$. Since a constant shift in J does not change its associated greedy policies, the algorithm starting with J_0 and generating subsequent function and policy pairs (J_k, μ_k) , can be made equivalent to that starting with \tilde{J}_0 and generating (\tilde{J}_k, μ_k) , where J_k and \tilde{J}_k are related by

$$\tilde{J}_k = J_k + \alpha \sum_{i=0}^{k-1} m_i c e.$$

By the first part of the proof, $\tilde{J}_k \rightarrow J^*$, thus $J_k \rightarrow J^*$. By the first part of the proof, $J_{\mu_k} \rightarrow J^*$. \square

Remark 1. For certain undiscounted problems, one may have to impose $T_{\mu_0} J_0 \leq J_0$ as an initial condition on J_0 for the modified policy iteration algorithm to converge. This can be partly seen from part (ii) of the preceding proof for an arbitrary J_0 , in which case we used the contraction property of the DP mappings.

Visualizing the Algorithms

There are figures illustrating conceptually the value iteration, policy iteration and modified policy iteration algorithms. See Figure 1.3.4, pp. 37 of [Ber01], and Figures 6.4.1 pp. 179 and 6.5.2 pp. 189 of [Put94].

4.2 Asynchronous Algorithms

We consider several asynchronous algorithms in this section. In contrast to synchronous algorithms, which update the value or policy for all states at the same time, asynchronous algorithms update the value function or update the policy at only selected states and at selected times.

Asynchronous algorithms have close connections with online simulation-based algorithms or reinforcement learning algorithms, which are naturally asynchronous, since value or policy updates are performed only on limited number of sampled states and based on sampled costs and transitions. The conditions required for the convergence of asynchronous algorithms thus delineate the range of what can be possibly achieved by their simulation-based counterparts, when stochastic noise is introduced.

4.2.1 Asynchronous Value Iteration

The Gauss-Seidel value iteration is like the normal value iteration, except that it computes the function TJ sequentially for each state, and substitutes the new function values at states s in place of $J(s)$, as soon as the former are computed. The algorithm can be used for both value iteration and policy evaluation via computing $T_\mu^m J$.

Gauss-Seidel Value Iteration Algorithm

1. Start with some function J_0 , and let $k=0$.
2. Compute J_{k+1} as follows: for $s = 1, \dots, n$,

$$J_{k+1}(s) = \min_{u \in \mathcal{U}(s)} \left[g(s, u) + \alpha \sum_{j < s} p(j|s, u) J_{k+1}(j) + \alpha \sum_{j \geq s} p(j|s, u) J_k(j) \right]. \quad (6)$$

3. Increase k and go back to step 2.

Proposition 3. *The function J_k generated by the Gauss-Seidel value iteration algorithm converges to J^* .*

The idea of the proof is to show that the mapping associated with the iteration is a contraction mapping with modulus α . Or, in other words, after updated for all states, the function J_{k+1} that we obtain, is closer to J^* than J_k by at least a factor of α . This argument will also be used later to show the convergence of a general asynchronous value iteration algorithm. We omit the proof; for details, see e.g., Proposition 1.3.2 of [Ber01].

There are many variants of the Gauss-Seidel algorithm. For example, between two consecutive iterations, reverse the order of the states to update, or, choose the states in an arbitrary order. Based on the contraction mapping argument, these variants can all be shown to converge.

The convergence rate of the Gauss-Seidel algorithm is shown to be no worse than the value iteration algorithm, and under certain conditions, it is faster than the latter. (See e.g., Proposition 1.3.3 of [Ber01], and Theorem 6.3.7 and 6.3.9, and Example 6.3.2 of [Put94].)

The Gauss-Seidel algorithm can also be viewed as one special case of a general class of algorithms that are based on the so called splitting methods. Indeed, some of the convergence rate analysis that we just mentioned, is based on analyzing the spectral radius of various matrices associated with splitting methods. For details, see Section 6.3.3 of [Put94].

We now introduce a general asynchronous value iteration algorithm.

Asynchronous Value Iteration Algorithm

1. Start with some function J_0 , and let $k = 0$.
2. Select a subset of states S_k , and update J_k on S_k by

$$J_{k+1}(s) = \begin{cases} (TJ_k)(s), & \text{if } s \in S_k, \\ J_k(s), & \text{otherwise.} \end{cases} \quad (7)$$

3. Increase k and go back to step 2.

Proposition 4. *If all states are selected infinitely often, then J_k generated by the asynchronous value iteration algorithm converges to J^* .*

There are a number of ways to prove the proposition, and we will use the contraction property of T .

Proof. Since for any function J ,

$$|(TJ)(s) - J^*(s)| \leq \max_s |(TJ)(s) - J^*(s)| = \|TJ - J^*\| \leq \alpha \|J - J^*\|, \quad (8)$$

we have for all states s , $|J_k(s) - J^*(s)| \leq \|J_{k-1} - J^*\|$, i.e., $\|J_k - J^*\| \leq \|J_{k-1} - J^*\|$. Thus the sequence $\|J_k - J^*\|$ is non-increasing, i.e.,

$$\dots \leq \|J_k - J^*\| \leq \|J_{k-1} - J^*\| \leq \dots \leq \|J_0 - J^*\|. \quad (9)$$

Let $\tau_0 = 0$, and let τ_1 be the time when all states have been selected at least once. Let $\tau_1(s)$ be the latest time when state s is selected in the time interval $(\tau_0, \tau_1]$. Then, by Eqs. (8) and (9),

$$|J_{\tau_1(s)+1}(s) - J^*(s)| \leq \alpha \|J_{\tau_1(s)} - J^*\| \leq \alpha \|J_{\tau_0} - J^*\|.$$

Since s is not selected again after $\tau_1(s)$ in the interval $(\tau_0, \tau_1]$, $J_{\tau_1+1}(s) = J_{\tau_1(s)+1}(s)$, and hence

$$|J_{\tau_1+1}(s) - J^*(s)| \leq \alpha \|J_{\tau_0} - J^*\|.$$

Since all states are selected at least once, by the definition of sup-norm,

$$\|J_{\tau_1+1} - J^*\| \leq \alpha \|J_{\tau_0} - J^*\|.$$

We can repeat this argument by defining τ_n to be the time when all states have been updated at least once since the time τ_{n-1} , and we can conclude that

$$\|J_{\tau_n+1} - J^*\| \leq \alpha^n \|J_{\tau_0} - J^*\| \rightarrow 0.$$

This shows that the non-increasing sequence $\|J_k - J^*\| \rightarrow 0$, and equivalently, $J_k \rightarrow J^*$. \square

Connection of Asynchronous Value Iteration to Q -learning

Consider the computation of $(TJ)(s)$:

$$(TJ)(s) = \min_{u \in \mathcal{U}(s)} \left[g(s, u) + \alpha \sum_{s' \in \mathcal{S}} p(s'|s, u) J(s') \right].$$

It can be equivalently written as

$$(TJ)(s) = \min_{u \in \mathcal{U}(s)} Q(s, u),$$

where the Q function on the joint state and action space is defined by

$$Q(s, u) = g(s, u) + \alpha \sum_{s' \in \mathcal{S}} p(s'|s, u) J(s'), \quad s \in \mathcal{S}, u \in \mathcal{U}(s). \quad (10)$$

When $J = J^*$, we call the associated Q function the optimal Q -function or Q -factor, and denote it by Q^* . That is,

$$Q^*(s, u) = g(s, u) + \alpha \sum_{s' \in \mathcal{S}} p(s'|s, u) J^*(s'), \quad s \in \mathcal{S}, u \in \mathcal{U}(s). \quad (11)$$

The optimality equation $J^* = TJ^*$ is equivalent to the pair of equations, Eq. (11) and

$$J^*(s) = \min_{s' \in \mathcal{U}(s)} Q^*(s, u),$$

or, when written in terms of Q^* only,

$$Q^*(s, u) = g(s, u) + \alpha \sum_{s' \in \mathcal{S}} p(s'|s, u) \min_{u' \in \mathcal{U}(s')} Q^*(s', u'). \quad (12)$$

Obviously, knowing either J^* or Q^* determines the other. However, this intermediate function Q^* turns out to be more convenient to work with in the simulation-based approach for computing J^* .

The value of $Q^*(s, u)$ can be interpreted as the optimal total discounted cost over all policies that apply at the initial state s the action u , or, it can be interpreted as the cost of a policy that applies u at s at the initial stage, and follows an optimal policy at subsequent stages.

The function Q^* can also be interpreted as the optimal cost function for an augmented MDP that has the set of state-action pairs $\{(s, u) | s \in \mathcal{S}, u \in \mathcal{U}(s)\}$ jointly as its state space. (We leave the details of the augmented MDP model, such as the action space and the state transitions, to the readers.) The optimality equation of this augmented MDP is exactly Eq. (12).

Consider a conceptual Q -learning algorithm, where there is no stochastic noise, and we update the Q -function by

$$Q_k(s, u) = g(s, u) + \alpha \sum_{s' \in \mathcal{S}} p(s'|s, u) \min_{u' \in \mathcal{U}(s')} Q_{k-1}(s', u'),$$

for certain selected state-action pairs (s, u) . This can be viewed as the asynchronous value iteration for the augmented MDP, or as the asynchronous value iteration for the original MDP if all the pairs in $\{(s, u) | u \in \mathcal{U}(s)\}$ for a given state s are selected whenever a pair (s, u) for some u is selected.

The Q -learning algorithm (Watkins 1989) for a discounted problem takes the form

$$Q_k(s, u) = (1 - \gamma_k) Q_{k-1}(s, u) + \gamma_k (g(s, u) + \alpha \min_{u' \in \mathcal{U}(s')} Q_{k-1}(s', u'))$$

where s' is a state sampled from the distribution $p(\cdot | s, u)$, and γ_k is a diminishing stepsize parameter, used for averaging out the effect of stochastic noise. Q -learning can be proved to converge for

discounted and certain undiscounted problems (Watkins and Dayan 1992, and Tsitsiklis 1994, among others). Even though it is outside our scope in this lecture to state rigorously all conditions required for the convergence of Q -learning, it is worth to point out that as one of the requirements, all state and action pairs must be encountered infinitely often, and also that there is no condition on the initial Q function (unlike in the case of asynchronous modified policy iteration, to be presented shortly). These are consistent with the convergence analysis on asynchronous value iteration given in this section.

4.2.2 Asynchronous Modified Policy Iteration

Asynchronous modified policy iteration is similar to modified policy iteration, but is substantially more flexible. As a consequence, it needs a crucial initial condition $T_{\mu_0} J_0 \leq J_0$, as will be discussed later.

Asynchronous Modified Policy Iteration Algorithm

1. Start with some function and policy pair (J_0, μ_0) with $T_{\mu_0} J_0 \leq J_0$. Let $k = 0$.
2. Select a subset of states S_k . Either update the function J_k on S_k by

$$J_{k+1}(s) = \begin{cases} (T_{\mu_k} J_k)(s), & \text{if } s \in S_k, \\ J_k(s), & \text{otherwise,} \end{cases} \quad (13)$$

while leaving the policy unchanged by setting $\mu_{k+1} = \mu_k$; or update the policy on S_k by

$$\mu_{k+1}(s) = \begin{cases} u \text{ s.t. } (T_{\mu_{k+1}} J_k)(s) = (T J_k)(s), & \text{if } s \in S_k, \\ \mu_k(s), & \text{otherwise,} \end{cases} \quad (14)$$

while leaving the function unchanged by setting $J_{k+1} = J_k$.

3. Increase k and go back to step 2.

Special Cases of the Algorithm

The asynchronous modified policy iteration algorithm includes all the preceding algorithms, synchronous or asynchronous, as special cases. When a policy update is followed immediately by a value update for the same subset of states, which is then followed by another policy update, and the updates continue in this fashion, we obtain the asynchronous value iteration algorithm. When $S_k = \mathcal{S}$, we obtain the modified policy iteration algorithm, and we obtain the value iteration algorithm, if value updates and policy updates alternate with each other.

Proposition 5. *If the value update (13) and the policy update (14) are executed infinitely often for all states, then (J_k, μ_k) generated by the asynchronous modified policy iteration algorithm satisfies that $J_k, \mu_k \rightarrow J^*$.*

A proof can be found after Proposition 1.3.5 of [Ber01]. We sketch a slightly different proof, (close to that of Williams and Baird 1993), in which we separate those intermediate results that rely on the initial condition $T_{\mu_0} J_0 \leq J_0$, from those that do not. We will see that the initial condition is important for establishing the convergence of the sequence J_k to some limit point. Once the convergence of J_k is established, the rest does not rely on the initial condition on J_0 , and the part of $J_k, \mu_k \rightarrow J^*$ follows from the continuity of the DP mappings and the fact that value and policy updates are executed infinitely often for all states.

Sketch of Proof. Our first step is similar to that in part (i) of the proof for the modified policy iteration algorithm. Here, we show that

$$T_{\mu_{k-1}} J_{k-1} \leq J_{k-1} \implies T_{\mu_k} J_k \leq J_k \leq J_{k-1}. \quad (15)$$

We prove this by induction on k and by considering two cases separately, one case being that the update is on the function, and the other being that the update is on the policy. (We leave the details for readers to check. Alternatively, see the proof of Proposition 1.3.5 of [Ber01].)

The statement (15) then implies that J_k is monotonically decreasing and bounded below by J^* , and hence converges to some limit $\bar{J} \in \mathfrak{R}^n$. We need to show $\bar{J} = T\bar{J}$, which will imply that $\bar{J} = J^*$, and consequently $J_k \rightarrow J^*$ and $J_{\mu_k} \rightarrow J^*$ (since $J_\mu \leq J_k$, or alternatively, use the argument below).

To show $\bar{J} = T\bar{J}$, we will use the following facts: $J_k \rightarrow \bar{J}$, that value update (13) and policy update (14) are executed infinitely often for all states, and that T and T_μ are continuous mappings. (We will only outline the argument, which can be made precise by using ϵ and δ terms.)

To arrive a contradiction, assume that $\bar{J} \neq T\bar{J}$. Then there exists a state s such that $\bar{J}(s) \neq T\bar{J}(s)$. Let t be some time sufficiently large when the policy update is executed for state s and when $\|J_{t'} - \bar{J}\|$ is sufficiently small for all $t' \geq t$. Let $\bar{k} > t$ be the first time after time t when the value update is executed for state s , and let $k < \bar{k}$ be the last time after time t when the policy update is executed for state s . Then $\mu_{\bar{k}}(s) = \mu_k(s)$. We have

$$(T_{\mu_k} J_k)(s) = (T J_k)(s) \approx (T \bar{J})(s),$$

thus

$$J_{\bar{k}+1}(s) = (T_{\mu_{\bar{k}}} J_{\bar{k}})(s) \approx (T_{\mu_{\bar{k}}} \bar{J})(s) \approx (T_{\mu_k} J_k)(s) = (T_{\mu_k} J_k)(s) \approx (T \bar{J})(s).$$

This shows (since value and policy updates are executed infinitely often for all states) that there is a subsequence of J_k converging to a different limit than \bar{J} , a contradiction.

Once we establish $\bar{J} = J^*$, the convergence of the cost of the policies, $J_{\mu_k} \rightarrow J^*$, follows from the continuity of the DP mappings and the fact that the policy update is executed infinitely often for all states. In particular, at time k when the policy update is executed for state s ,

$$(T_{\mu_k} J_k)(s) \approx (T \bar{J})(s),$$

which implies, for a finite action MDP, that μ_k is an optimal action for state s when k is sufficiently large. This is true for all states, thus μ_k is an optimal policy when k is sufficiently large. \square

On the Initial Condition $T_{\mu_0} J_0 \leq J_0$

The assumption $T_{\mu_0} J_0 \leq J_0$ is crucial to establish the convergence of J_k , as can be seen from the proof. For an arbitrary J_0 , non-convergent examples exist (Williams and Baird 1993), unlike in the case of modified policy iteration.

If all the computations TJ and $T_\mu J$ can be done exactly, then the assumption $T_{\mu_0} J_0 \leq J_0$ can be satisfied easily for discounted problems, by either adding a constant to J_0 or letting J_0 be

$$J_0 = \frac{\max_{s,u} g(s,u)}{1-\alpha} e,$$

as can be verified easily.

For simulation-based algorithms, it is almost impossible to satisfy the initial condition $T_{\mu_0} J_0 \leq J_0$, or to maintain the kind of relation for J_k , even if the condition is initially satisfied. This is one of the main difficulties for the convergence of the simulation-based algorithms known as “optimistic policy iteration,” which are conceptually the simulation-based counterpart of asynchronous modified policy iteration.

4.3 Linear Programming

We will follow Section 6.9 of [Put94] on this subject. Here are a few important points: the linear programming problem that is dual to the LP in the last lecture for solving the optimal equation; the interpretation of the dual variables as certain probabilistic quantities, the dual feasible solution as a stationary randomized policy, and the dual objective function as the cost function of a stationary randomized policy; the relation between extreme points or basic feasible solutions of the dual LP and stationary deterministic policies; the formulation of certain constrained discounted problems as the dual linear programming problem with additional constraints.

4.4 Closing Remarks

Contraction mappings play an important role in analyzing discounted problems. Contraction mappings can be defined on general metric spaces, for which there are more general contraction mapping fixed point theorems. Together with different notions of norms, they are powerful tools to analyze infinite-horizon discounted MDPs with countable or general state spaces. Because of them, the discounted problems are analytically well understood. Under very general conditions we can assert that the optimal cost function is the unique solution of the optimality equation. In contrast, in the case of undiscounted problems, beyond the finite space models, various examples are known where the optimality equation does not have a solution, or a solution that can characterize the optimal cost function and an optimal policy.

The discounted cost criterion is essentially a finite-horizon criterion. However, by letting the discount factor α approach 1, the solutions of the discounted problems can be related to the solution of an infinite-horizon undiscounted problem. Analytically, this is an important tool to analyze undiscounted problems, and is called the vanishing discount approach; computationally, this provides an approximate solution method for solving the undiscounted problems. We will study these topics in the coming lectures.

As for computational methods for solving discounted problems, we have seen various iterative methods, such as value iteration and policy iteration, as well as non-iterative methods such as linear programming. Each of them can turn out to be more suitable than the others in certain contexts, particularly, in the case of large scale problems where approximation or simulation-based computation has to be introduced.

Suggested Readings

[Ber01] Vol II., Ch. 1

[Put94] Ch. 6.