2 Dynamic Programming – Finite Horizon

2.1 Introduction

Dynamic Programming (DP) is a general approach for solving *multi-stage* optimization problems, or *optimal planning* problems. The underlying idea is to use *backward recursion* to reduce the computational complexity.

DP has been widely applied to problems of optimal control, graph search, multistage planning, etc. This method was popularized in the 50's and 60's, mainly through the work of Richard Bellman.

Our interest here is in optimal control problems that involve:
- possibly stochastic systems
- discrete time variable
- a finite number of states and actions (choices)
- additive cost structure

The standard model for such problems is *Markov Decision Processes* (MDPs).

We start in this chapter to describe the MDP model and DP for *finite horizon* problem. The next chapter deals with the infinite horizon case.

References:

Standard references on DP and MDPs are:

D. Bertsekas, *Dynamic Programming and Optimal Control*, Vol.1+2, 3rd. ed.

M.L. Puterman, *Markov Decision Processes*, 1994.

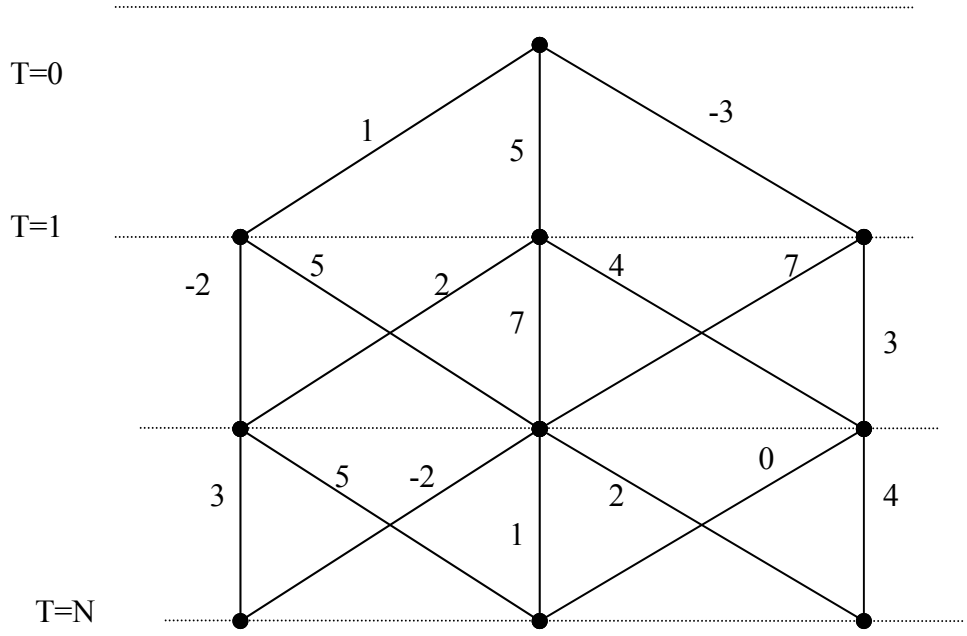
The basic theory is also covered in Bertsekas & Tsistiklis (1996). The books of Sutton and Barto (1998) and Powell (2007) present less rigorous introductions.

.

2.2 A Simple Graph-Search Example

The simplest demonstration of dynamic programming is probably finding longest (or shortest) paths in a graph.

Suppose we wish to find the longest distance we can traverse from a given node in a given graph in N steps.



The number of different *paths* is exponential in N, so trying them all is not feasible. Instead, we can use backward recursion to compute the maximal distance from each node to the end step, staring with T=N.

Using this recursion, the longest distance is easily seen to be 14 (and the shortest is 0).

2.3 The MDP Model

A Markov Decision Process defines an optimization problem with two ingredients: (1) a controlled dynamic system, and (2) a cost (or reward) structure.

Controlled System Dynamics

The dynamic system we consider is specified by:

1. The time axis: $\mathbf{T} = \{0,1,\ldots,N\}$ (a discrete-time, finite horizon problem).
2. A finite state space S.
3. A finite action set $A(s)$ for each state $s \in S$.
4. State transition structure $P = \{p(s' \mid s,a) : s,s' \in S, a \in A(s)\}$

The state transition matrix defines a *controlled Markov chain* over the given state and action sets. That is,

$$prob(s_{t+1} = s' \mid s_t = s, a_t = a) = p(s' \mid s,a)$$

and we also assume the Markov property:

$$prob(s_{t+1} = s' \mid s_t, a_t) = prob(s_{t+1} = s' \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0)$$

It is naturally required that

$$p(s' \mid s,a) \geq 0 \text{, and } \sum_{s' \in S} p(s' \mid s,a) = 1 \text{ for all } s \in S, a \in A(s).$$

Remarks:

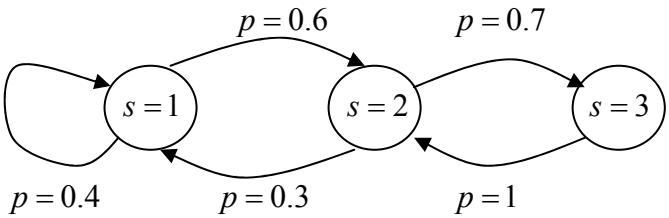
1. It is will be convenient in theoretical developments to suppress the dependence of $A(s)$ on s, and assume a single actions set A which applies to all states. This entails no loss of generality, since we can always take $A = \bigcup_s A(s)$, with irrelevant actions at each states duplicating the effect of existing ones.

2. The model defined above is *stationary* (or time invariant), as the dynamics does not depend on time. In general, the system dynamics (and also the state and action sets) can depend on the time index. That is, for each $t \in \mathbf{T}$ we have
$$P_t = \{p_t(s' \mid s,a) : s \in S_t, a \in A_t, s' \in S_{t+1}\}$$
and
$$prob(s_{t+1} = s' \mid s_t = s, a_t = a) = p_t(s' \mid s,a).$$
In the present chapter we will proceed with the non-stationary model.

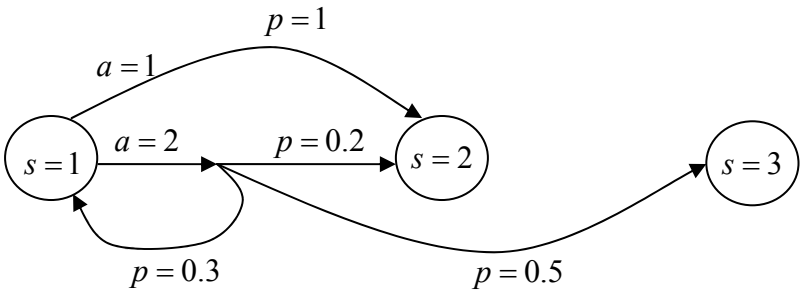
3. The state dynamics can be equivalently defined via a *state equation* of the form $s_{t+1} = f_t(s_t, a_t, n_t)$, where each n_t is a random variable which is independent of the past (namely of (s_k, a_k, n_k) for $k \leq t-1$).

Graphical Notation:

Recall that a (stationary) Markov chain is defined by the state transition matrix $P = \{p(s' \mid s) : s, s' \in S\}$. The state transition probabilities are often demonstrated via a state transition diagram, such as:



A graphic description of a controlled Markov chain is a bit more complicated because of the additional action variable. We then obtain the following diagram (drawn for state s=1 only):



This means that $p(s' = 2 \mid s = 1, a = 1) = 1$, $p(s' = 1 \mid s = 1, a = 2) = 0.3$, etc.

Reward Structure

The reward structure for an MDP is specified by:

5. An immediate reward function $r_t = \{r_t(s,a) : s \in S_t, a \in A_t\}$ for each $t \in \mathbf{T}$. The reward obtained at time $t \in \mathbf{T}$ is therefore $R_t = r_t(s_t, a_t)$.

6. A performance measure, or *optimality criterion*. The most common one for the finite-horizon problem is the *expected total reward:*

$$J = E\Big(\sum_{t=0}^{N} R_t\Big) = E\Big(\sum_{t=0}^{N} r_t(s_t, a_t)\Big)$$

Our goal is to maximize J, by an appropriate selection of actions.

Remarks:

1. *Terminal rewards:* At $t = N$, we often have $r_N(s,a) = r_N(s)$. We refer to $r_N(s)$ as the *terminal reward*. We will henceforth assume that this is the case.

2. *Stationary models:* For a stationary model, we have $r_t(s,a) = r(s,a)$ for every $t \in \mathbf{T}$.

3. *Cost functions*: In some case it is more natural to work with a cost function $c_t(s,a)$ instead of a reward function. Our goal then is to *minimize* the total expected cost $J = E(\sum_{t=0}^{N} C_t)$.
   The two formulations are of course equivalent: we can simply define $r_t = -c_t$ to transform the cost formulation into a reward formulation.

4. *More general reward functions:* In some cases the obtained reward may depend on the next state as well: $R_t = \tilde{r}_t(s_t, a_t, s_{t+1})$. For control purposes, when we only consider the expected value of the reward, we can reduce this reward function to the usual one by defining
$$r_t(s,a) \triangleq \sum_{s' \in S} p(s' \mid s,a)\tilde{r}_t(s,a,s') \equiv E(R_t \mid s_t = s, a_t = a)$$

5. *Random rewards:* The reward R_t may also be random, namely a random variable whose *distribution* depends on (s_t, a_t). This can also be reduced to our model for control purposes by looking at the expected value of R_t, namely $r_t(s,a) = E(R_t \mid s_t = s, a_t = a)$.

Control Policies

- A *general* control policy π is a mapping from each possible history $h_t = (s_0, a_0, \ldots, s_{t-1}, a_{t-1}, s_t)$ to $a_t = \pi_t(h_t)$.

- A *Markov* control policy π depends on the current state and time only: $a_t = \pi_t(s_t)$.

- A *stationary* control policy chooses the action depending on the current state alone: $a_t = \pi(s_t)$. Such policies will play a major role in infinite-horizon problems.

- We denote the sets of general / Markov / stationary policies by Π / Π_M / Π_S, respectively

- We can also define *randomized* versions of the above. In a randomized control policy, the action a_t is chosen according to a probability distribution $\pi_t(a \mid h_t)$ over A_t. Randomized policies will not be required for the time being (but will be used in the learning problem).

Given a control policy π and initial state $s_0 = s$, together with the system description, we obtain a probability distribution over the state-action sequence $h_N = (s_0, a_0, \ldots, s_{N-1}, a_{N-1}, s_N, a_N)$. We denote this probability distribution by $P^{\pi,s}(\cdot)$ or $P^{\pi}(\cdot \mid s_0 = s)$. The corresponding expectation operator is denoted as $E^{\pi,s}(\cdot)$ or $E^{\pi}(\cdot \mid s_0 = s)$

Remarks:

1. For most (non-learning) problems, Markov policies suffice to achieve the optimum.

2. Implicit in these definitions of control policies is the assumption that the current state s_t is fully observed. If this is not the case we consider the problem of a Partially Observed MDP (POMDP), which is more involved.

2.4 Finite-Horizon Dynamic Programming

Recall that we consider the expected total reward criterion, which we denote as

$$\begin{aligned} J(\pi, s) &= E^{\pi}\left(\sum_{t=0}^{N} R_t \mid s_0 = s\right) \\ &= E^{\pi}\left(\sum_{t=0}^{N-1} r_t(s_t, a_t) + r_N(s_N) \mid s_0 = s\right) \end{aligned}$$

Here π is the control policy used, and s is a given initial state. We wish to maximize $J(\pi, s)$ over all control policies, and find an optimal policy π^* that achieves the maximal reward $J^*(s)$. Thus,

$$J^*(s) \triangleq J(\pi^*, s) = \max_{\pi \in \Pi} J(\pi, s)$$

A. The Principle of Optimality

The celebrated "principle of optimality" applies to a large class of multi-stage optimization problems, and is at the heart of DP. As a general principle, it states that

The tail of an optimal policy is optimal for the "tail" problem.

For a more precise statement which applies to our model, let $\pi^* = (\pi_0, \ldots, \pi_{N-1})$ denote an optimal Markov policy. Take any state $s_k = s'$ which has a positive probability under π^*, namely $P^{\pi^*, s}(s_k = s') > 0$. Then the tail policy $(\pi_k, \ldots, \pi_{N-1})$ is optimal for the "tail" cost $J_k(\pi, s') = E^{\pi}\left(\sum_{t=k}^{N} R_t \mid s_k = s'\right)$.

B. Dynamic Programming for Policy Evaluation

As a "warmup", let us evaluate the reward of a given policy.

Let $\pi = (\pi_1, \ldots, \pi_{N-1})$ be a given Markov policy. Define the following reward-to-go function, or value function:

$$V_k^\pi(s) = E^\pi \left(\sum_{t=k}^N R_t \mid s_k = s \right)$$

Observe that $V_0^\pi(s) = J(\pi, s)$.

Lemma 1 $V_k^\pi(s)$ may be computed by the backward recursion:

$$V_k^\pi(s) = \left\{ r_k(s,a) + \sum_{s' \in S_{k+1}} p_k(s' \mid s, a)\, V_{k+1}^\pi(s') \right\}_{a=\pi_k(s)}, \quad k = N-1, \ldots, 0$$

starting with $V_N^\pi(s) = r_N(s)$.

Proof: Observe that:

$$\begin{aligned} V_k^\pi(s) &= E^\pi \left(R_k + \sum_{t=k+1}^N R_t \,\middle|\, s_k = s, a_k = \pi_k(s) \right) \\ &= E^\pi \left(E^\pi \left(R_k + \sum_{t=k+1}^N R_t \,\middle|\, s_k = s, a_k = \pi_k(s), s_{k+1} \right) \middle|\, s_k = s, a_k = \pi_k(s) \right) \\ &= E^\pi \left(r(s_k, a_k) + V_{k+1}^\pi(s_{k+1}) \,\middle|\, s_k = s, a_k = \pi_k(s) \right) \\ &= r_k(s, \pi_k(s)) + \sum_{s' \in S_{k+1}} p_k(s' \mid s, \pi_k(s))\, V_{k+1}^\pi(s') \end{aligned}$$

□

Remark: With the more general reward function $\tilde{r}_t(s, a, s')$, the recursion takes the form

$$V_k^\pi(s) = \left\{ \sum_{s' \in S_{k+1}} p_k(s' \mid s, a)[r_k(s, a, s') + V_{k+1}^\pi(s')] \right\}_{a=\pi_k(s)}$$

A similar observation applies to all Dynamic Programming equations below.

C. The Dynamic Programming Algorithm

We next define the *optimal* value function:

$$V_k^*(s) = \max_{\pi} E^{\pi}\left(\sum_{t=k}^{N} R_t \mid s_k = s\right)$$

Note that the maximum should be taken over "tail" policies $\pi^k = (\pi_k, \ldots, \pi_{N-1})$ that start from time k. Obviously, $V_0^* = J^*$.

Theorem 2 (Finite-horizon Dynamic Programming)
(i) Backward recursion: $V_N^*(s) = r_N(s)$, and

$$V_k^*(s) = \max_{a \in A_k}\left\{r_k(s,a) + \sum_{s' \in S_{k+1}} p_k(s' \mid s,a) V_{k+1}^*(s')\right\}, \qquad k = N-1, \ldots, 0$$

(ii) Optimal policy: Any Markov policy π^* that satisfies

$$\pi_k^*(s) \in \arg\max_{a \in A_k}\left\{r_k(s,a) + \sum_{s' \in S_{k+1}} p_k(s' \mid s,a) V_{k+1}^*(s')\right\}, \qquad k = 0, \ldots, N-1$$

is an optimal control policy. Furthermore, π^* maximizes $V_0^{\pi}(s)$ simultaneously for *every* initial state $s_0 = s$.

Proof :
(i) $V_N^*(s) = r_N(s)$ follows from the definition. We proceed by backward induction. Assume that the required equality holds for $k' \geq k+1$. We need to show that $V_k^*(s) = Z_k(s)$, where $Z_k(s) \triangleq \max_{a \in A_k}\left\{r_k(s,a) + \sum_{s' \in S_{k+1}} p_k(s' \mid s,a) V_{k+1}^*(s')\right\}$.

To establish $V_k^*(s) \leq Z_k(s)$, it is enough to show that $V_k^{\pi^k}(s) \leq Z_k(s)$ for any (general, randomize) "tail" policy π^k. Consider then any "tail" policy $\pi^k = (\pi_k, \ldots \pi_{N-1})$. Note that $a_k \sim \pi_k(a \mid s_k)$, $a_{k+1} \sim \pi_{k+1}(a \mid s_k, a_k, s_{k+1})$, etc. For each $s \in S_k$ and $a \in A_k$, let $(\pi^k \mid s,a)$ denote the policy from time $k+1$ onwards which is obtained from π^k when $s_k = s, a_k = a$. Proceeding as in page 2.9 (value iteration for a fixed policy), it follows that

$$V_k^{\pi^k}(s) = \sum_{a \in A_k} \pi_k(a \mid s)\left\{r_k(s,a) + \sum_{s' \in S_{k+1}} p_k(s' \mid s,a) V_{k+1}^{(\pi^k \mid s,a)}(s')\right\}$$

But since V_{k+1}^* is optimal we have that $V_{k+1}^*(s') \geq V_{k+1}^{(\pi^k \mid s,a)}(s')$, so that

$$V_k^{\pi^k}(s) \leq \sum_{a \in A_k} \pi_k(a \mid s)\left\{r_k(s,a) + \sum_{s' \in S_{k+1}} p_k(s' \mid s,a) V_{k+1}^*(s')\right\} \leq Z_k(s).$$

We next show that $V_k^*(s) \geq Z_k(s)$. For that purpose, it is enough to find a policy π^k so that $V_k^{\pi^k}(s) \geq Z_k(s)$. Define π^k as follows: Choose $a_k = \bar{a}$ so that

$$\bar{a} \in \arg\max_{a \in A_k}\left\{r_k(s,a) + \sum_{s' \in S_{k+1}} p_k(s' \mid s,a) V_{k+1}^*(s')\right\}$$

and then after observing $s_{k+1} = s'$ proceed with the policy $\pi = \pi^{k+1}$ which maximizes $V_{k+1}^{\pi}(s')$. For this policy we obtain:

$$V_k^{\pi^k}(s) = r_k(s,\bar{a}) + \sum_{s' \in S_{k+1}} p(s' \mid s,\bar{a}) V_{k+1}^{\pi^{k+1}}(s')$$
$$= r_k(s,\bar{a}) + \sum_{s' \in S_{k+1}} p(s' \mid s,\bar{a}) V_{k+1}^{*}(s') = Z$$

This completes the proof of (i).

(ii) (Outline – exercise). Let π^* be the (Markov) policy defined in (ii). Using value iteration for this policy, prove by backward induction that $V_k^{\pi^*} = V_k^*$. □

Note that optimal control policy obtained as above is a deterministic *Markov* policy.

To summarize:

- The optimal value function can be computed by backward recursion. This recursive equation is known as the ***Dynamic Programming Equation***, ***Optimality Equation***, or ***Bellman's Equation***.
- Computation of the value function in this way is known as the value iteration algorithm, or just ***value iteration***.
- The value function is computed for *all* states at each stage.
- An optimal policy is easily derived from the optimal value.
- The optimization in each stage is performed in the *action* space. The total number of minimization operations needed is $N \times |S|$ – each over $|A|$ choices. This replaces "brute force" optimization in policy space, with tremendous computational savings as the number of Markov policies is $|A|^{N|S|}$.

D. Some Operator Notation

Define the operator T_k^* over the space of functions $V : S_k \to \mathbb{R}$ (equivalently: the space of vectors $V \in \mathbb{R}^{|S_k|}$),

$$(T_k^* V)(s) = \max_{a \in A_k} \left\{ r_k(s,a) + \sum_{s' \in S_k} p_k(s' \mid s,a) V(s') \right\}$$

Similarly, for a given decision function $\pi_k : S_k \to A_k$,

$$(T_k^{\pi_k} V)(s) = r_k(s, \pi_k(s)) + \sum_{s' \in S_k} p_k(s' \mid s, \pi_k(s)) V(s')$$

Note that $(T_k^* V)(s) = \max_{\pi_k} \{T_k^{\pi_k} V(s)\}$.

We will now use the shorthand vector notation $T_k^* V = \max_{\pi_k} T_k^{\pi_k} V$, and rewrite the Dynamic Programming equations in this notation. This gives:

Policy evaluation: $V_k^{\pi^k} = T_k^{\pi_k} V_{k+1}^{\pi^{k+1}}$.

Optimality equation: $V_k^* = T_k^* V_{k+1}^*$.

Optimal policy: $\pi_k^* \in \arg\max_{\pi_k} T_k^{\pi_k} V_{k+1}^*$

E. The Q function

Define

$$Q_k(s,a) = r_k(s,a) + \sum_{s' \in S_k} p_k(s' \mid s,a) V_k^*(s')$$

This is known as the (optimal) state-action value function, or simply as the *Q*-function. $Q_k(s,a)$ is the expected total reward from stage k onward, if we choose $a_k = a$ and then proceed optimally.

It is easily seen that

$$V_k^*(s) = \max_{a \in A_k} Q_k(s,a)$$

and

$$\pi_k^*(s) \in \arg\max_{a \in A_k} Q_k(s,a)$$

Furthermore, in terms of *Q* the optimality equation reads:

$$Q_k(s,a) = r_k(s,a) + \sum_{s' \in S} p_k(s' \mid s,a) \max_{a' \in A_k} Q_{k+1}^*(s',a')$$

The *Q* function allows an immediate computation of the optimal actions at each stage. It will form the basis for *Q-learning algorithm*, which is one of the basic Reinforcement Learning algorithms.