

## Chapter 4

# Dynamic Programming

The term dynamic programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process (MDP). Classical DP algorithms are of limited utility in reinforcement learning both because of their assumption of a perfect model and because of their great computational expense, but they are still important theoretically. DP provides an essential foundation for the understanding of the methods presented in the rest of this book. In fact, all of these methods can be viewed as attempts to achieve much the same effect as DP, only with less computation and without assuming a perfect model of the environment.

Starting with this chapter, we usually assume that the environment is a finite MDP. That is, we assume that its state, action, and reward sets,  $\mathcal{S}$ ,  $\mathcal{A}$ , and  $\mathcal{R}$ , are finite, and that its dynamics are given by a set of probabilities  $p(s', r | s, a)$ , for all  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}(s)$ ,  $r \in \mathcal{R}$ , and  $s' \in \mathcal{S}^+$  ( $\mathcal{S}^+$  is  $\mathcal{S}$  plus a terminal state if the problem is episodic). Although DP ideas can be applied to problems with continuous state and action spaces, exact solutions are possible only in special cases. A common way of obtaining approximate solutions for tasks with continuous states and actions is to quantize the state and action spaces and then apply finite-state DP methods. The methods we explore in Part II are applicable to continuous problems and are a significant extension of that approach.

The key idea of DP, and of reinforcement learning generally, is the use of value functions to organize and structure the search for good policies. In this chapter we show how DP can be used to compute the value functions defined in Chapter 3. As discussed there, we can easily obtain optimal policies once we have found the optimal value functions,  $v_*$  or  $q_*$ , which satisfy the Bellman optimality equations:

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_*(s')], \end{aligned} \tag{4.1}$$

or

$$\begin{aligned} q_*(s, a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a\right] \\ &= \sum_{s', r} p(s', r | s, a) [r + \gamma \max_{a'} q_*(s', a')], \end{aligned} \tag{4.2}$$

for all  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}(s)$ , and  $s' \in \mathcal{S}^+$ . As we shall see, DP algorithms are obtained by turning Bellman equations such as these into assignments, that is, into update rules for improving approximations of the desired value functions.

## 4.1 Policy Evaluation (Prediction)

First we consider how to compute the state-value function  $v_\pi$  for an arbitrary policy  $\pi$ . This is called *policy evaluation* in the DP literature. We also refer to it as the *prediction problem*. Recall from Chapter 3 that, for all  $s \in \mathcal{S}$ ,

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \end{aligned} \tag{from (3.9)}$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \tag{4.3}$$

$$= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left[ r + \gamma v_\pi(s') \right], \tag{4.4}$$

where  $\pi(a|s)$  is the probability of taking action  $a$  in state  $s$  under policy  $\pi$ , and the expectations are subscripted by  $\pi$  to indicate that they are conditional on  $\pi$  being followed. The existence and uniqueness of  $v_\pi$  are guaranteed as long as either  $\gamma < 1$  or eventual termination is guaranteed from all states under the policy  $\pi$ .

If the environment's dynamics are completely known, then (4.4) is a system of  $|\mathcal{S}|$  simultaneous linear equations in  $|\mathcal{S}|$  unknowns (the  $v_\pi(s)$ ,  $s \in \mathcal{S}$ ). In principle, its solution is a straightforward, if tedious, computation. For our purposes, iterative solution methods are most suitable. Consider a sequence of approximate value functions  $v_0, v_1, v_2, \dots$ , each mapping  $\mathcal{S}^+$  to  $\mathbb{R}$  (the real numbers). The initial approximation,  $v_0$ , is chosen arbitrarily (except that the terminal state, if any, must be given value 0), and each successive approximation is obtained by using the Bellman equation for  $v_\pi$  (4.4) as an update rule:

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left[ r + \gamma v_k(s') \right], \end{aligned} \tag{4.5}$$

for all  $s \in \mathcal{S}$ . Clearly,  $v_k = v_\pi$  is a fixed point for this update rule because the Bellman equation for  $v_\pi$  assures us of equality in this case. Indeed, the sequence  $\{v_k\}$  can be shown in general to converge to  $v_\pi$  as  $k \rightarrow \infty$  under the same conditions that guarantee the existence of  $v_\pi$ . This algorithm is called *iterative policy evaluation*.

To produce each successive approximation,  $v_{k+1}$  from  $v_k$ , iterative policy evaluation applies the same operation to each state  $s$ : it replaces the old value of  $s$  with a new value obtained from the old values of the successor states of  $s$ , and the expected immediate rewards, along all the one-step transitions possible under the policy being evaluated. We call this kind of operation an *expected update*. Each iteration of iterative policy evaluation updates the value of every state once to produce the new approximate value function

$v_{k+1}$ . There are several different kinds of expected updates, depending on whether a state (as here) or a state–action pair is being updated, and depending on the precise way the estimated values of the successor states are combined. All the updates done in DP algorithms are called *expected* updates because they are based on an expectation over all possible next states rather than on a sample next state. The nature of an update can be expressed in an equation, as above, or in a backup diagram like those introduced in Chapter 3. For example, the backup diagram corresponding to the expected update used in iterative policy evaluation is shown on page 59.

To write a sequential computer program to implement iterative policy evaluation as given by (4.5) you would have to use two arrays, one for the old values,  $v_k(s)$ , and one for the new values,  $v_{k+1}(s)$ . With two arrays, the new values can be computed one by one from the old values without the old values being changed. Alternatively, you could use one array and update the values “in place,” that is, with each new value immediately overwriting the old one. Then, depending on the order in which the states are updated, sometimes new values are used instead of old ones on the right-hand side of (4.5). This in-place algorithm also converges to  $v_\pi$ ; in fact, it usually converges faster than the two-array version, as you might expect, because it uses new data as soon as they are available. We think of the updates as being done in a *sweep* through the state space. For the in-place algorithm, the order in which states have their values updated during the sweep has a significant influence on the rate of convergence. We usually have the in-place version in mind when we think of DP algorithms.

A complete in-place version of iterative policy evaluation is shown in pseudocode in the box below. Note how it handles termination. Formally, iterative policy evaluation converges only in the limit, but in practice it must be halted short of this. The pseudocode tests the quantity  $\max_{s \in \mathcal{S}} |v_{k+1}(s) - v_k(s)|$  after each sweep and stops when it is sufficiently small.

#### Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input  $\pi$ , the policy to be evaluated

Algorithm parameter: a small threshold  $\theta > 0$  determining accuracy of estimation

Initialize  $V(s)$  arbitrarily, for  $s \in \mathcal{S}$ , and  $V(\text{terminal})$  to 0

Loop:

$\Delta \leftarrow 0$

Loop for each  $s \in \mathcal{S}$ :

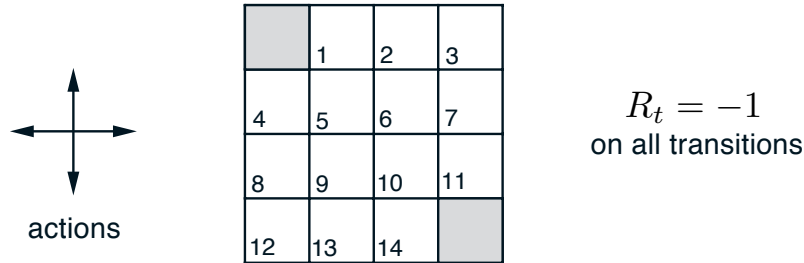
$v \leftarrow V(s)$

$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) [r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until  $\Delta < \theta$

**Example 4.1** Consider the  $4 \times 4$  gridworld shown below.



The nonterminal states are  $\mathcal{S} = \{1, 2, \dots, 14\}$ . There are four actions possible in each state,  $\mathcal{A} = \{\text{up}, \text{down}, \text{right}, \text{left}\}$ , which deterministically cause the corresponding state transitions, except that actions that would take the agent off the grid in fact leave the state unchanged. Thus, for instance,  $p(6, -1 | 5, \text{right}) = 1$ ,  $p(7, -1 | 7, \text{right}) = 1$ , and  $p(10, r | 5, \text{right}) = 0$  for all  $r \in \mathcal{R}$ . This is an undiscounted, episodic task. The reward is  $-1$  on all transitions until the terminal state is reached. The terminal state is shaded in the figure (although it is shown in two places, it is formally one state). The expected reward function is thus  $r(s, a, s') = -1$  for all states  $s, s'$  and actions  $a$ . Suppose the agent follows the equiprobable random policy (all actions equally likely). The left side of Figure 4.1 shows the sequence of value functions  $\{v_k\}$  computed by iterative policy evaluation. The final estimate is in fact  $v_\pi$ , which in this case gives for each state the negation of the expected number of steps from that state until termination. ■

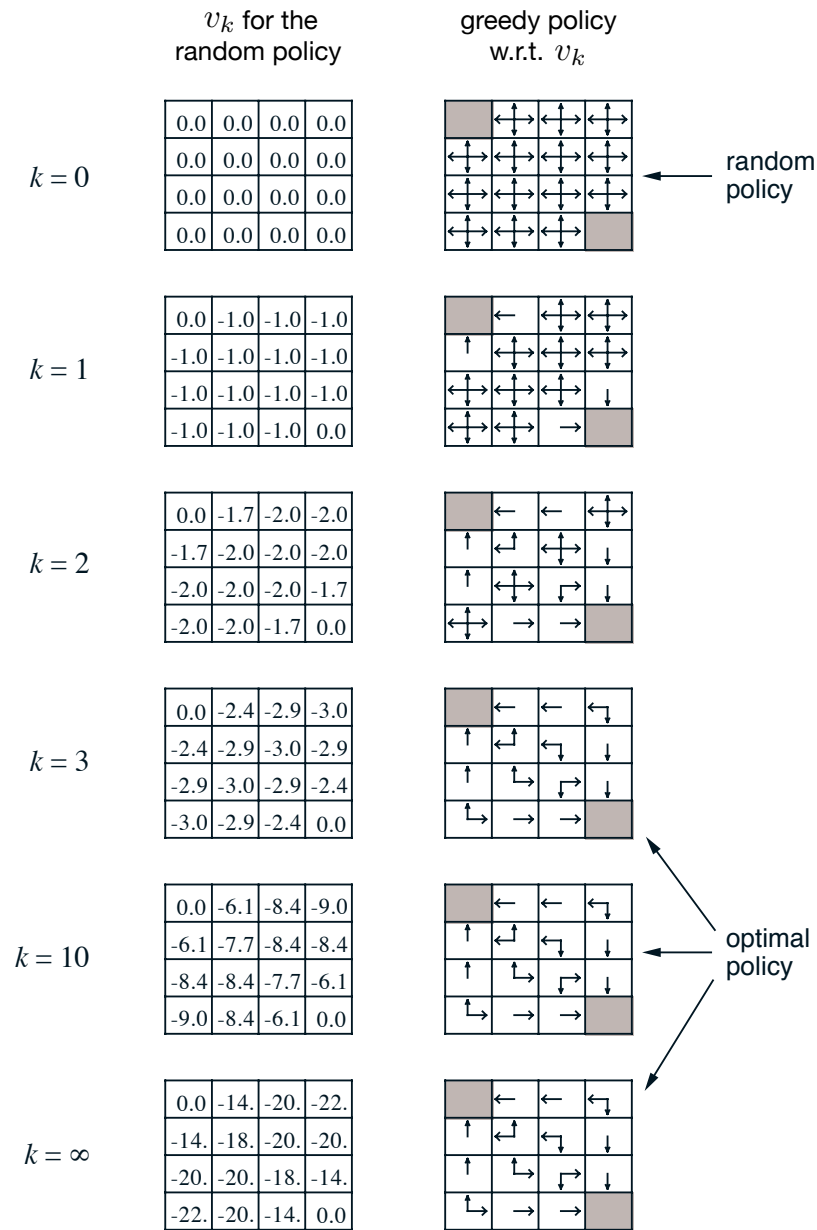
*Exercise 4.1* In Example 4.1, if  $\pi$  is the equiprobable random policy, what is  $q_\pi(11, \text{down})$ ? What is  $q_\pi(7, \text{down})$ ? □

*Exercise 4.2* In Example 4.1, suppose a new state 15 is added to the gridworld just below state 13, and its actions, **left**, **up**, **right**, and **down**, take the agent to states 12, 13, 14, and 15, respectively. Assume that the transitions *from* the original states are unchanged. What, then, is  $v_\pi(15)$  for the equiprobable random policy? Now suppose the dynamics of state 13 are also changed, such that action **down** from state 13 takes the agent to the new state 15. What is  $v_\pi(15)$  for the equiprobable random policy in this case? □

*Exercise 4.3* What are the equations analogous to (4.3), (4.4), and (4.5), but for *action-value* functions instead of state-value functions? □

## 4.2 Policy Improvement

Our reason for computing the value function for a policy is to help find better policies. Suppose we have determined the value function  $v_\pi$  for an arbitrary deterministic policy  $\pi$ . For some state  $s$  we would like to know whether or not we should change the policy to deterministically choose an action  $a \neq \pi(s)$ . We know how good it is to follow the current policy from  $s$ —that is  $v_\pi(s)$ —but would it be better or worse to change to the new policy? One way to answer this question is to consider selecting  $a$  in  $s$  and thereafter



**Figure 4.1:** Convergence of iterative policy evaluation on a small gridworld. The left column is the sequence of approximations of the state-value function for the random policy (all actions equally likely). The right column is the sequence of greedy policies corresponding to the value function estimates (arrows are shown for all actions achieving the maximum, and the numbers shown are rounded to two significant digits). The last policy is guaranteed only to be an improvement over the random policy, but in this case it, and all policies after the third iteration, are optimal.

following the existing policy,  $\pi$ . The value of this way of behaving is

$$\begin{aligned} q_\pi(s, a) &\doteq \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_\pi(s')]. \end{aligned} \quad (4.6)$$

The key criterion is whether this is greater than or less than  $v_\pi(s)$ . If it is greater—that is, if it is better to select  $a$  once in  $s$  and thereafter follow  $\pi$  than it would be to follow  $\pi$  all the time—then one would expect it to be better still to select  $a$  every time  $s$  is encountered, and that the new policy would in fact be a better one overall.

That this is true is a special case of a general result called the *policy improvement theorem*. Let  $\pi$  and  $\pi'$  be any pair of deterministic policies such that, for all  $s \in \mathcal{S}$ ,

$$q_\pi(s, \pi'(s)) \geq v_\pi(s). \quad (4.7)$$

Then the policy  $\pi'$  must be as good as, or better than,  $\pi$ . That is, it must obtain greater or equal expected return from all states  $s \in \mathcal{S}$ :

$$v_{\pi'}(s) \geq v_\pi(s). \quad (4.8)$$

Moreover, if there is strict inequality of (4.7) at any state, then there must be strict inequality of (4.8) at that state.

The policy improvement theorem applies to the two policies that we considered at the beginning of this section: an original deterministic policy,  $\pi$ , and a changed policy,  $\pi'$ , that is identical to  $\pi$  except that  $\pi'(s) = a \neq \pi(s)$ . For states other than  $s$ , (4.7) holds because the two sides are equal. Thus, if  $q_\pi(s, a) > v_\pi(s)$ , then the changed policy is indeed better than  $\pi$ .

The idea behind the proof of the policy improvement theorem is easy to understand. Starting from (4.7), we keep expanding the  $q_\pi$  side with (4.6) and reapplying (4.7) until we get  $v_{\pi'}(s)$ :

$$\begin{aligned} v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\ &= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] && \text{(by (4.6))} \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] && \text{(by (4.7))} \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}[R_{t+2} + \gamma v_\pi(S_{t+2}) \mid S_{t+1}, A_{t+1} = \pi'(S_{t+1})] \mid S_t = s] \\ &= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \mid S_t = s] \\ &\vdots \\ &\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \mid S_t = s] \\ &= v_{\pi'}(s). \end{aligned}$$

So far we have seen how, given a policy and its value function, we can easily evaluate a change in the policy at a single state. It is a natural extension to consider changes at

all states, selecting at each state the action that appears best according to  $q_\pi(s, a)$ . In other words, to consider the new *greedy* policy,  $\pi'$ , given by

$$\begin{aligned}\pi'(s) &\doteq \operatorname{argmax}_a q_\pi(s, a) \\ &= \operatorname{argmax}_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \operatorname{argmax}_a \sum_{s', r} p(s', r \mid s, a) \left[ r + \gamma v_\pi(s') \right],\end{aligned}\tag{4.9}$$

where  $\operatorname{argmax}_a$  denotes the value of  $a$  at which the expression that follows is maximized (with ties broken arbitrarily). The greedy policy takes the action that looks best in the short term—after one step of lookahead—according to  $v_\pi$ . By construction, the greedy policy meets the conditions of the policy improvement theorem (4.7), so we know that it is as good as, or better than, the original policy. The process of making a new policy that improves on an original policy, by making it greedy with respect to the value function of the original policy, is called *policy improvement*.

Suppose the new greedy policy,  $\pi'$ , is as good as, but not better than, the old policy  $\pi$ . Then  $v_\pi = v_{\pi'}$ , and from (4.9) it follows that for all  $s \in \mathcal{S}$ :

$$\begin{aligned}v_{\pi'}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r \mid s, a) \left[ r + \gamma v_{\pi'}(s') \right].\end{aligned}$$

But this is the same as the Bellman optimality equation (4.1), and therefore,  $v_{\pi'}$  must be  $v_*$ , and both  $\pi$  and  $\pi'$  must be optimal policies. Policy improvement thus must give us a strictly better policy except when the original policy is already optimal.

So far in this section we have considered the special case of deterministic policies. In the general case, a stochastic policy  $\pi$  specifies probabilities,  $\pi(a|s)$ , for taking each action,  $a$ , in each state,  $s$ . We will not go through the details, but in fact all the ideas of this section extend easily to stochastic policies. In particular, the policy improvement theorem carries through as stated for the stochastic case. In addition, if there are ties in policy improvement steps such as (4.9)—that is, if there are several actions at which the maximum is achieved—then in the stochastic case we need not select a single action from among them. Instead, each maximizing action can be given a portion of the probability of being selected in the new greedy policy. Any apportioning scheme is allowed as long as all submaximal actions are given zero probability.

The last row of Figure 4.1 shows an example of policy improvement for stochastic policies. Here the original policy,  $\pi$ , is the equiprobable random policy, and the new policy,  $\pi'$ , is greedy with respect to  $v_\pi$ . The value function  $v_\pi$  is shown in the bottom-left diagram and the set of possible  $\pi'$  is shown in the bottom-right diagram. The states with multiple arrows in the  $\pi'$  diagram are those in which several actions achieve the maximum in (4.9); any apportionment of probability among these actions is permitted. For any such policy, its state values  $v_{\pi'}(s)$  can be seen by inspection to be either  $-1$ ,  $-2$ , or  $-3$ , for all states  $s \in \mathcal{S}$ , whereas  $v_\pi(s)$  is at most  $-14$ . Thus,  $v_{\pi'}(s) \geq v_\pi(s)$ , for all

$s \in \mathcal{S}$ , illustrating policy improvement. Although in this case the new policy  $\pi'$  happens to be optimal, in general only an improvement is guaranteed.

### 4.3 Policy Iteration

Once a policy,  $\pi$ , has been improved using  $v_\pi$  to yield a better policy,  $\pi'$ , we can then compute  $v_{\pi'}$  and improve it again to yield an even better  $\pi''$ . We can thus obtain a sequence of monotonically improving policies and value functions:

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*,$$

where  $\xrightarrow{\text{E}}$  denotes a policy *evaluation* and  $\xrightarrow{\text{I}}$  denotes a policy *improvement*. Each policy is guaranteed to be a strict improvement over the previous one (unless it is already optimal). Because a finite MDP has only a finite number of deterministic policies, this process must converge to an optimal policy and the optimal value function in a finite number of iterations.

This way of finding an optimal policy is called *policy iteration*. A complete algorithm is given in the box below. Note that each policy evaluation, itself an iterative computation, is started with the value function for the previous policy. This typically results in a great increase in the speed of convergence of policy evaluation (presumably because the value function changes little from one policy to the next).

#### Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

##### 1. Initialization

$V(s) \in \mathbb{R}$  and  $\pi(s) \in \mathcal{A}(s)$  arbitrarily for all  $s \in \mathcal{S}$ ;  $V(\text{terminal}) \doteq 0$

##### 2. Policy Evaluation

Loop:

$\Delta \leftarrow 0$

Loop for each  $s \in \mathcal{S}$ :

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s',r} p(s', r | s, \pi(s)) [r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until  $\Delta < \theta$  (a small positive number determining the accuracy of estimation)

##### 3. Policy Improvement

*policy-stable*  $\leftarrow$  true

For each  $s \in \mathcal{S}$ :

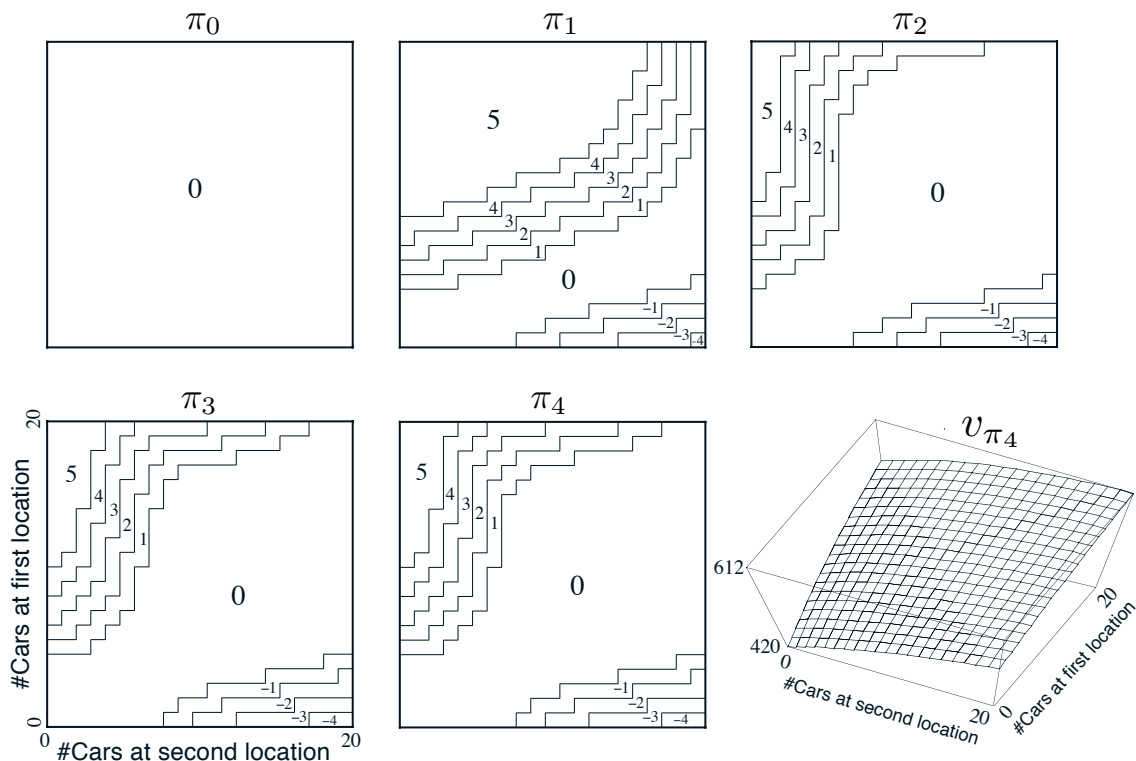
*old-action*  $\leftarrow$   $\pi(s)$

$\pi(s) \leftarrow \arg \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$

If *old-action*  $\neq$   $\pi(s)$ , then *policy-stable*  $\leftarrow$  false

If *policy-stable*, then stop and return  $V \approx v_*$  and  $\pi \approx \pi_*$ ; else go to 2

**Example 4.2: Jack’s Car Rental** Jack manages two locations for a nationwide car rental company. Each day, some number of customers arrive at each location to rent cars. If Jack has a car available, he rents it out and is credited \$10 by the national company. If he is out of cars at that location, then the business is lost. Cars become available for renting the day after they are returned. To help ensure that cars are available where they are needed, Jack can move them between the two locations overnight, at a cost of \$2 per car moved. We assume that the number of cars requested and returned at each location are Poisson random variables, meaning that the probability that the number is  $n$  is  $\frac{\lambda^n}{n!}e^{-\lambda}$ , where  $\lambda$  is the expected number. Suppose  $\lambda$  is 3 and 4 for rental requests at the first and second locations and 3 and 2 for returns. To simplify the problem slightly, we assume that there can be no more than 20 cars at each location (any additional cars are returned to the nationwide company, and thus disappear from the problem) and a maximum of five cars can be moved from one location to the other in one night. We take the discount rate to be  $\gamma = 0.9$  and formulate this as a continuing finite MDP, where the time steps are days, the state is the number of cars at each location at the end of the day, and the actions are the net numbers of cars moved between the two locations overnight. Figure 4.2 shows the sequence of policies found by policy iteration starting from the policy that never moves any cars.



**Figure 4.2:** The sequence of policies found by policy iteration on Jack’s car rental problem, and the final state-value function. The first five diagrams show, for each number of cars at each location at the end of the day, the number of cars to be moved from the first location to the second (negative numbers indicate transfers from the second location to the first). Each successive policy is a strict improvement over the previous policy, and the last policy is optimal. ■