

Policy iteration often converges in surprisingly few iterations, as illustrated in the example of Jack’s car rental and in the example in Figure 4.1. The bottom-left diagram of Figure 4.1 shows the value function for the equiprobable random policy, and the bottom-right diagram shows a greedy policy for this value function. The policy improvement theorem assures us that these policies are better than the original random policy. In this case, however, these policies are not just better, but optimal, proceeding to the terminal states in the minimum number of steps. In this example, policy iteration would find the optimal policy after just one iteration.

*Exercise 4.4* The policy iteration algorithm on page 80 has a subtle bug in that it may never terminate if the policy continually switches between two or more policies that are equally good. This is okay for pedagogy, but not for actual use. Modify the pseudocode so that convergence is guaranteed.  $\square$

*Exercise 4.5* How would policy iteration be defined for action values? Give a complete algorithm for computing  $q_*$ , analogous to that on page 80 for computing  $v_*$ . Please pay special attention to this exercise, because the ideas involved will be used throughout the rest of the book.  $\square$

*Exercise 4.6* Suppose you are restricted to considering only policies that are  $\varepsilon$ -soft, meaning that the probability of selecting each action in each state,  $s$ , is at least  $\varepsilon/|\mathcal{A}(s)|$ . Describe qualitatively the changes that would be required in each of the steps 3, 2, and 1, in that order, of the policy iteration algorithm for  $v_*$  on page 80.  $\square$

*Exercise 4.7 (programming)* Write a program for policy iteration and re-solve Jack’s car rental problem with the following changes. One of Jack’s employees at the first location rides a bus home each night and lives near the second location. She is happy to shuttle one car to the second location for free. Each additional car still costs \$2, as do all cars moved in the other direction. In addition, Jack has limited parking space at each location. If more than 10 cars are kept overnight at a location (after any moving of cars), then an additional cost of \$4 must be incurred to use a second parking lot (independent of how many cars are kept there). These sorts of nonlinearities and arbitrary dynamics often occur in real problems and cannot easily be handled by optimization methods other than dynamic programming. To check your program, first replicate the results given for the original problem.  $\square$

## 4.4 Value Iteration

One drawback to policy iteration is that each of its iterations involves policy evaluation, which may itself be a protracted iterative computation requiring multiple sweeps through the state set. If policy evaluation is done iteratively, then convergence exactly to  $v_\pi$  occurs only in the limit. Must we wait for exact convergence, or can we stop short of that? The example in Figure 4.1 certainly suggests that it may be possible to truncate policy evaluation. In that example, policy evaluation iterations beyond the first three have no effect on the corresponding greedy policy.

In fact, the policy evaluation step of policy iteration can be truncated in several ways without losing the convergence guarantees of policy iteration. One important special

case is when policy evaluation is stopped after just one sweep (one update of each state). This algorithm is called *value iteration*. It can be written as a particularly simple update operation that combines the policy improvement and truncated policy evaluation steps:

$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_k(s')], \end{aligned} \quad (4.10)$$

for all  $s \in \mathcal{S}$ . For arbitrary  $v_0$ , the sequence  $\{v_k\}$  can be shown to converge to  $v_*$  under the same conditions that guarantee the existence of  $v_*$ .

Another way of understanding value iteration is by reference to the Bellman optimality equation (4.1). Note that value iteration is obtained simply by turning the Bellman optimality equation into an update rule. Also note how the value iteration update is identical to the policy evaluation update (4.5) except that it requires the maximum to be taken over all actions. Another way of seeing this close relationship is to compare the backup diagrams for these algorithms on page 59 (policy evaluation) and on the left of Figure 3.4 (value iteration). These two are the natural backup operations for computing  $v_\pi$  and  $v_*$ .

Finally, let us consider how value iteration terminates. Like policy evaluation, value iteration formally requires an infinite number of iterations to converge exactly to  $v_*$ . In practice, we stop once the value function changes by only a small amount in a sweep. The box below shows a complete algorithm with this kind of termination condition.

#### Value Iteration, for estimating $\pi \approx \pi_*$

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

Initialize  $V(s)$ , for all  $s \in \mathcal{S}^+$ , arbitrarily except that  $V(\text{terminal}) = 0$

Loop:

```
|  $\Delta \leftarrow 0$ 
| Loop for each  $s \in \mathcal{S}$ :
|    $v \leftarrow V(s)$ 
|    $V(s) \leftarrow \max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma V(s')]$ 
|    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
```

until  $\Delta < \theta$

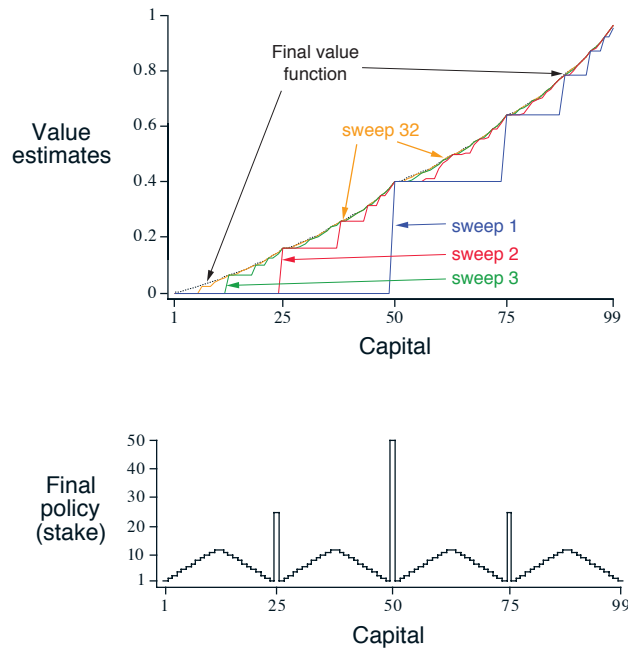
Output a deterministic policy,  $\pi \approx \pi_*$ , such that

$$\pi(s) = \operatorname{argmax}_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma V(s')]$$

Value iteration effectively combines, in each of its sweeps, one sweep of policy evaluation and one sweep of policy improvement. Faster convergence is often achieved by interposing multiple policy evaluation sweeps between each policy improvement sweep. In general, the entire class of truncated policy iteration algorithms can be thought of as sequences of sweeps, some of which use policy evaluation updates and some of which use value iteration updates. Because the max operation in (4.10) is the only difference between

these updates, this just means that the max operation is added to some sweeps of policy evaluation. All of these algorithms converge to an optimal policy for discounted finite MDPs.

**Example 4.3: Gambler’s Problem** A gambler has the opportunity to make bets on the outcomes of a sequence of coin flips. If the coin comes up heads, he wins as many dollars as he has staked on that flip; if it is tails, he loses his stake. The game ends when the gambler wins by reaching his goal of \$100, or loses by running out of money. On each flip, the gambler must decide what portion of his capital to stake, in integer numbers of dollars. This problem can be formulated as an undiscounted, episodic, finite MDP. The state is the gambler’s capital,  $s \in \{1, 2, \dots, 99\}$  and the actions are stakes,  $a \in \{0, 1, \dots, \min(s, 100 - s)\}$ . The reward is zero on all transitions except those on which the gambler reaches his goal, when it is +1. The state-value function then gives the probability of winning from each state. A policy is a mapping from levels of capital to stakes. The optimal policy maximizes the probability of reaching the goal. Let  $p_h$  denote the probability of the coin coming up heads. If  $p_h$  is known, then the entire problem is known and it can be solved, for instance, by value iteration. Figure 4.3 shows the change in the value function over successive sweeps of value iteration, and the final policy found, for the case of  $p_h = 0.4$ . This policy is optimal, but not unique. In fact, there is a whole family of optimal policies, all corresponding to ties for the argmax action selection with respect to the optimal value function. Can you guess what the entire family looks like?



**Figure 4.3:** The solution to the gambler’s problem for  $p_h = 0.4$ . The upper graph shows the value function found by successive sweeps of value iteration. The lower graph shows the final policy. ■

*Exercise 4.8* Why does the optimal policy for the gambler’s problem have such a curious form? In particular, for capital of 50 it bets it all on one flip, but for capital of 51 it does not. Why is this a good policy? □

*Exercise 4.9 (programming)* Implement value iteration for the gambler’s problem and solve it for  $p_h = 0.25$  and  $p_h = 0.55$ . In programming, you may find it convenient to introduce two dummy states corresponding to termination with capital of 0 and 100, giving them values of 0 and 1 respectively. Show your results graphically, as in Figure 4.3. Are your results stable as  $\theta \rightarrow 0$ ? □

*Exercise 4.10* What is the analog of the value iteration update (4.10) for action values,  $q_{k+1}(s, a)$ ? □