Next Up Previous Contents

Next: 4.5 Asynchronous Dynamic Programming **Up:** 4. Dynamic Programming **Previous:** 4.3 Policy Iteration **Contents**

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^π occurs only in the limit. Must we wait for exact convergence, or can we stop short of that? The example in Figure 4.2 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 backup of each state). This algorithm is called *value iteration*. It can be written as a particularly simple backup operation that combines the policy improvement and truncated policy evaluation steps:

$$\begin{aligned} V_{k+1}(s) &= \max_a E\left\{r_{t+1} + \gamma V_k(s_{t+1}) \mid s_t = s, a_t = a\right\} \qquad (4.10) \\ &= \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V_k(s')\right], \end{aligned}$$

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 backup is identical to the policy evaluation backup (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: Figure 3.4a shows the backup diagram for policy evaluation and Figure 3.7a shows the backup diagram for value iteration. These two are the natural backup operations for computing V^π 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. Figure 4.5 gives a complete value iteration algorithm with this kind of termination condition.

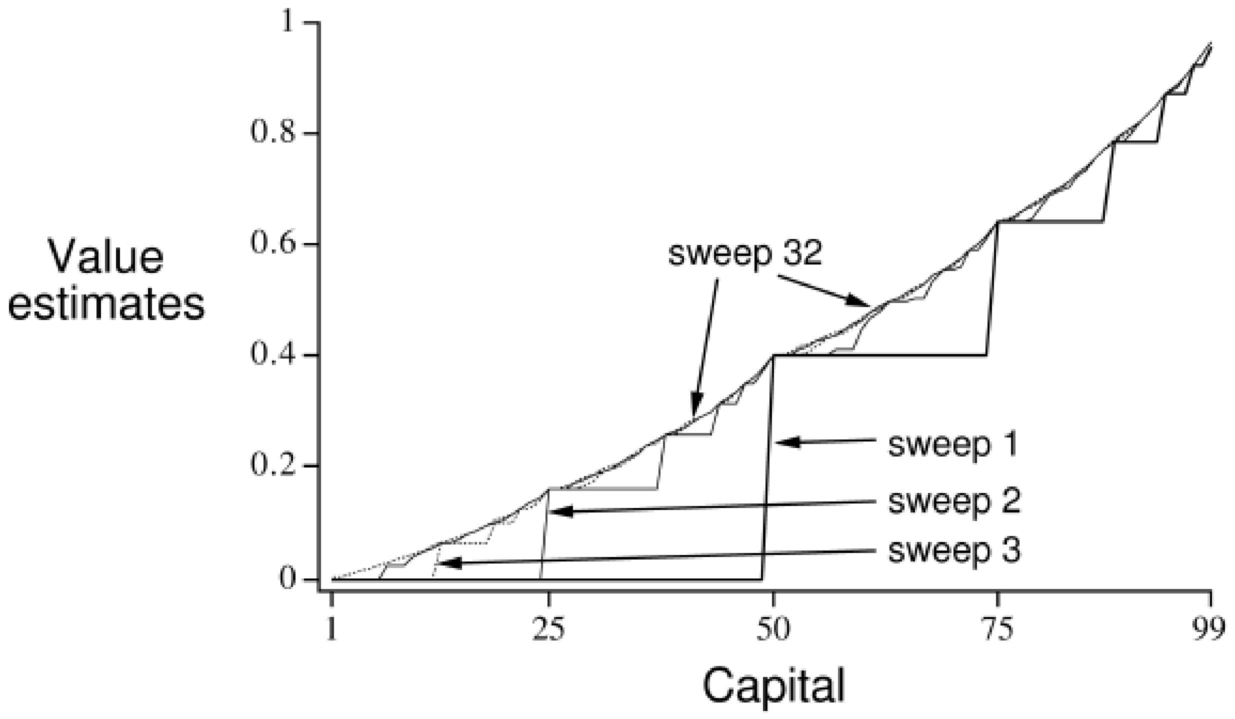
Value iteration effectively combines, in each of its sweeps, one sweep of policy evaluation and one sweep of policy improvement. Faster convergence is often achieved by interposing multiple policy evaluation sweeps between each policy improvement sweep. In general, the entire class of truncated policy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

```
Initialize $V$ arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi$, such that
    $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
```

Figure 4.5: Value iteration.

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 denote the probability of the coin coming up heads. If p is known, then the entire problem is known and it can be solved, for instance, by value iteration. Figure 4.6 shows the change in the value function over successive sweeps of value iteration, and the final policy found, for the case of $p = 0.4$.

■



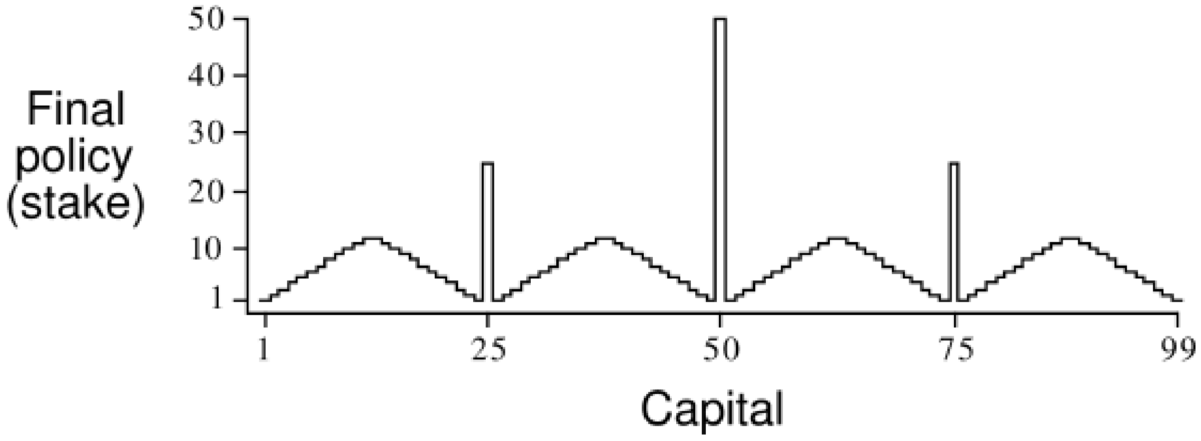


Figure 4.6: The solution to the gambler's problem for $p=0.4$. The upper graph shows the value function found by successive sweeps of value iteration. The lower graph shows the final policy.

Exercise 4.7 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 4.8 (programming) Implement value iteration for the gambler's problem and solve it for $p=0.25$ and $p=0.55$. 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 4.6. Are your results stable as $\theta \rightarrow 0$?

Exercise 4.9 What is the analog of the value iteration backup (4.10) for action values, $Q_{k+1}(s,a)$?

Next Up Previous Contents

Next: 4.5 Asynchronous Dynamic Programming **Up:** 4. Dynamic Programming **Previous:** 4.3 Policy Iteration **Contents**

Mark Lee 2005-01-04