OPTIMAL PLANNING

When the MDP parameters ($P^a_{ss'}$, $R^a_{ss'}$) are known, the Bellman (optimality) equations may be applied iteratively to efficiently compute optimal value functions, from which we may determine optimal policies. These iterative algorithms fall under a broader class of algorithms known as dynamic programming.

Policy Evaluation

Let π be an arbitrary policy. Define a sequence of functions $V_0, V_1, V_2, \ldots$ according to

$$V_{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V_k(s') \right]$$

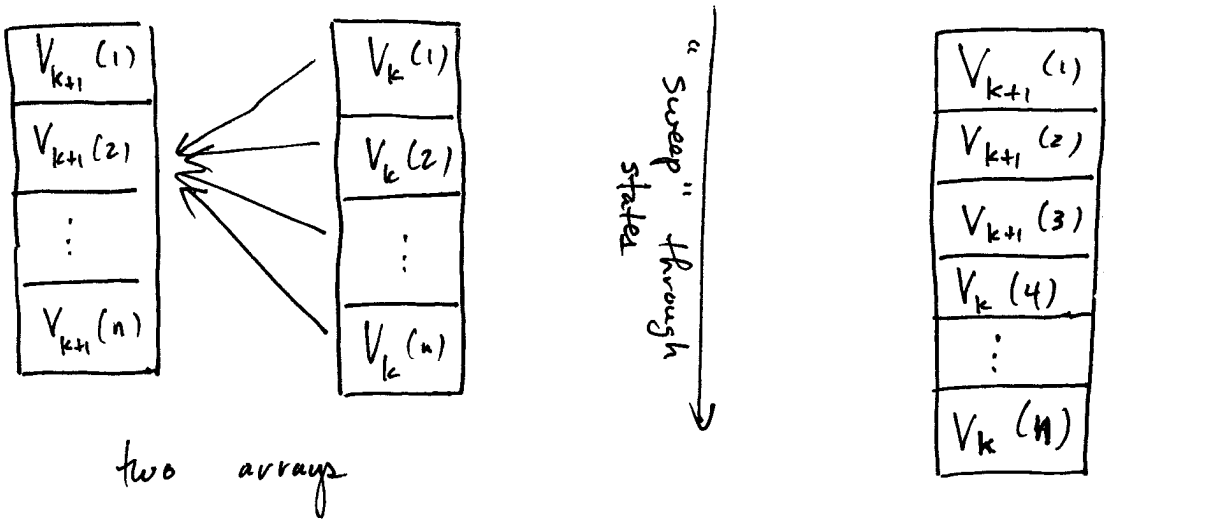
Then $\{V_k\}$ converges to V^π. When $|S|$ is large, this is much more efficient than solving the linear system of equations.

V_0 may be arbitrary, except that we require

$$V(\text{terminal state}) = 0$$

for episodic tasks.

To implement this algorithm, either one array or two may be used — convergence is gaurantee in either case.



Policy Improvement

If we can compute V^π, the we can compute

$$Q^\pi(s,a) = \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s') \right]$$

If $Q^\pi(s,a) > V^\pi(s)$ for some a, then it can be shown that it is better to change π so that it selects a when in state s. Formally, if

$$\pi'(s) = \arg\max_a Q^\pi(s,a)$$

then $V^{\pi'}(s) \geq V^\pi(s)$ $\forall s$. This result

Ⓐ is known as the

Furthermore, if equality holds $\forall s$, then

$$V^{\pi'}(s) = \max_a Q^{\pi}(s,a)$$

$$= \max_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^{\pi}(s') \right]$$

$$= \max_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^{\pi'}(s') \right]$$

which is the Bellman optimality equation for V^*.

Recall that the solution to this system is guaranteed to exist and be unique.

Therefore, if π is suboptimal, π' is guaranteed to improve upon π for at least one state.

Policy Iteration

Suppose we apply policy evaluation/improvement iteratively

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \longrightarrow \cdots$$

Note: each E is itself an iterative step.

For a finite MDP, policy iteration is guaranteed to converge to π^* in a finite (and often small) number of steps.

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 λ is the expected number. Suppose λ 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 4.4 shows the sequence of policies found by policy iteration starting from the policy that never moves any cars. ■

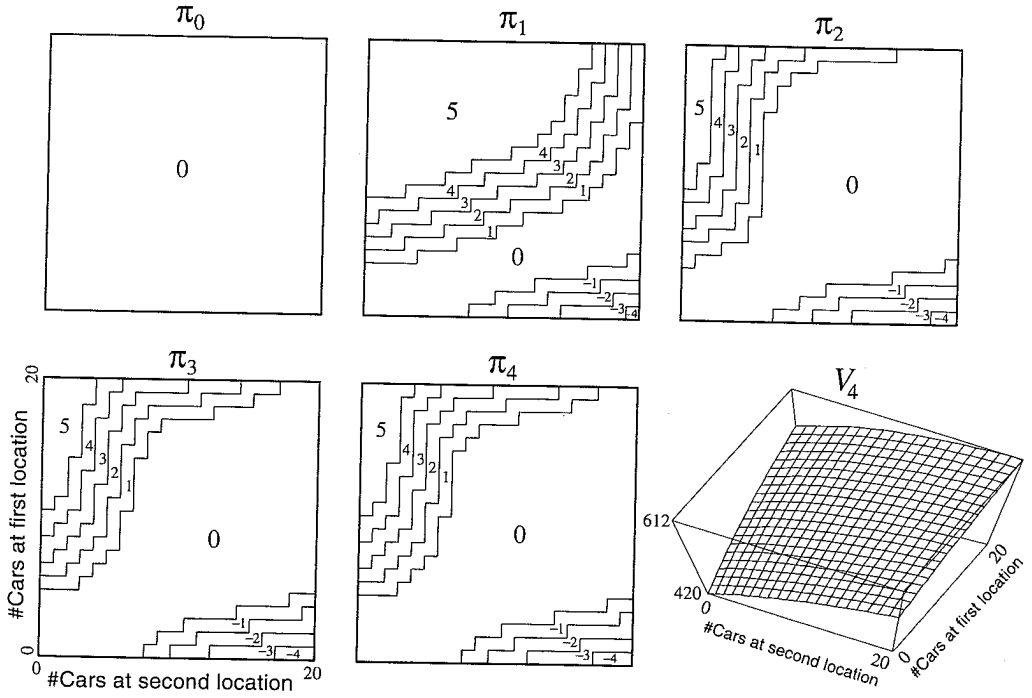


Figure 4.4 The sequence of policies found by policy iteration on Jack's car rental problem, and the final state-value function. The first five diagrams show, for each number of cars at each location at the end of the day, the number of cars to be moved from the first location to the second (negative 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.

Value Iteration

Value iteration is like policy iteration, except only one iteration of policy evaluation is applied at each E step.

Then value iteration produces a sequence $V_1, V_2, \ldots$ satisfying

(B) $$V_{k+1}(s) =$$

In essence, we are using the Bellman optimality equation for V^* as an update rule.

Like policy iteration, value iteration converges to the policy/value function.

More generally, convergence holds if you apply any finite number of iterations of policy evaluation at each step. Different choices will lead to different rates of convergence.

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, $s \in \{1, 2, \ldots, 99\}$ and the actions are stakes, $a \in \{1, 2, \ldots, \min(s, 100 - s)\}$. The reward is zero on all transitions except those on which the gambler reaches his goal, when it is $+1$. A 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 $p = 0.4$. ■

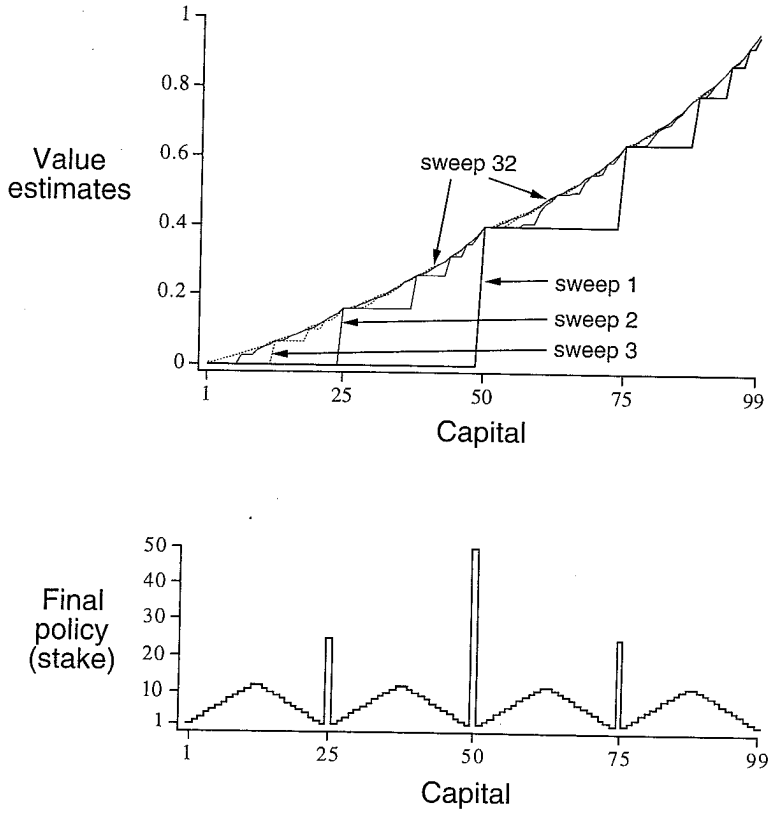


Figure 4.6 The solution to the gambler's problem for $p = 0.4$. The upper graph shows the value functions found by successive sweeps of value iteration. The lower graph shows the final policy.

Asynchronous Updates

So far we have assumed that every iteration of policy evaluation or policy improvement conducts a full "sweep" through all the states.

If the state space is too large, this may be infeasible.

In asynchronous dynamic programming, some states are updated more often than others. Still, the algorithms usually converge provided each state is updated infinitely often. The advantage is that we can improve our policy without making a full sweep, by focusing more on the "important" states.

Key

A. policy improvement theorem

B. $$V_{k+1}(s) = \max_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V_k(s') \right]$$