Markov Decision Process, Optimal Solutions, Monte Carlo Methods

Milan Straka

October 14, 2019

LANGTECH

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Markov Decision Process

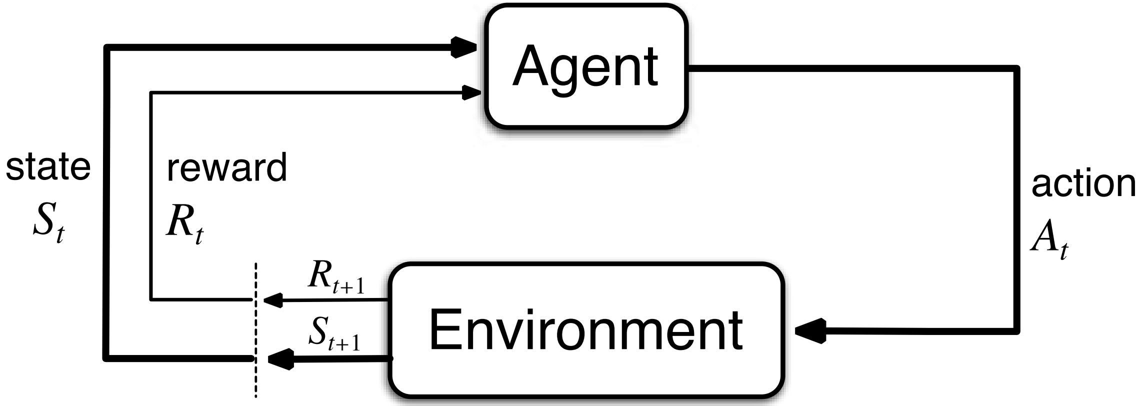


Figure 3.1 of "Reinforcement Learning: An Introduction, Second Edition".

A *Markov decision process* (MDP) is a quadruple $(\mathcal{S}, \mathcal{A}, p, \gamma)$, where:

- $\mathcal{S}$ is a set of states,
- $\mathcal{A}$ is a set of actions,
- $p(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$ is a probability that action $a \in \mathcal{A}$ will lead from state $s \in \mathcal{S}$ to $s' \in \mathcal{S}$, producing a *reward* $r \in \mathbb{R}$,
- $\gamma \in [0, 1]$ is a *discount factor*.

Let a *return* G_t be $G_t \overset{\text{def}}{=} \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$. The goal is to optimize $\mathbb{E}[G_0]$.

Multi-armed Bandits as MDP

To formulate n-armed bandits problem as MDP, we do not need states. Therefore, we could formulate it as:

- one-element set of states, $\mathcal{S} = \{S\}$;
- an action for every arm, $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$;
- assuming every arm produces rewards with a distribution of $\mathcal{N}(\mu_i, \sigma_i^2)$, the MDP dynamics function p is defined as

$$p(S, r|S, a_i) = \mathcal{N}(r|\mu_i, \sigma_i^2).$$

One possibility to introduce states in multi-armed bandits problem is to have separate reward distribution for every state. Such generalization is usually called *Contextualized Bandits* problem. Assuming that state transitions are independent on rewards and given by a distribution $\mathit{next}(s)$, the MDP dynamics function for contextualized bandits problem is given by

$$p(s', r|s, a_i) = \mathcal{N}(r|\mu_{i,s}, \sigma_{i,s}^2) \cdot \mathit{next}(s'|s).$$

Episodic and Continuing Tasks

If the agent-environment interaction naturally breaks into independent subsequences, usually called *episodes*, we talk about **episodic tasks**. Each episode then ends in a special *terminal state*, followed by a reset to a starting state (either always the same, or sampled from a distribution of starting states).

In episodic tasks, it is often the case that every episode ends in at most H steps. These *finite-horizont tasks* then can use discount factor $\gamma = 1$, because the return $G \stackrel{\text{def}}{=} \sum_{t=0}^{H} \gamma^t R_{t+1}$ is well defined.

If the agent-environment interaction goes on and on without a limit, we instead talk about **continuing tasks**. In this case, the discount factor γ needs to be sharply smaller than 1.

(State-)Value and Action-Value Functions

A *policy* π computes a distribution of actions in a given state, i.e., $\pi(a|s)$ corresponds to a probability of performing an action a in state s.

To evaluate a quality of a policy, we define *value function* $v_\pi(s)$, or *state-value function*, as

$$v_\pi(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi\left[G_t \middle| S_t = s\right] = \mathbb{E}_\pi\left[\sum\nolimits_{k=0}^\infty \gamma^k R_{t+k+1} \middle| S_t = s\right].$$

An *action-value function* for a policy π is defined analogously as

$$q_\pi(s, a) \stackrel{\text{def}}{=} \mathbb{E}_\pi\left[G_t \middle| S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum\nolimits_{k=0}^\infty \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right].$$

Evidently,

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[q_\pi(s, a)], \\ q_\pi(s, a) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a]. \end{aligned}$$

Optimal Value Functions

Optimal state-value function is defined as

$$v_*(s) \overset{\text{def}}{=} \max_\pi v_\pi(s),$$

analogously

$$q_*(s, a) \overset{\text{def}}{=} \max_\pi q_\pi(s, a).$$

Any policy π_* with $v_{\pi_*} = v_*$ is called an *optimal policy*. Such policy can be defined as $\pi_*(s) \overset{\text{def}}{=} \arg\max_a q_*(s, a) = \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$. When multiple actions maximize $q_*(s, a)$, the optimal policy can stochastically choose any of them.

Existence

In finite-horizont tasks or if $\gamma < 1$, there always exists a unique optimal state-value function, unique optimal action-value function, and (not necessarily unique) optimal policy.

Dynamic Programming

Dynamic programming is an approach devised by Richard Bellman in 1950s.

To apply it to MDP, we now consider finite-horizon problems with finite number of states $\mathcal{S}$ and actions $\mathcal{A}$, and known MDP dynamics p.

The following recursion is usually called the *Bellman equation*:

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

It must hold for an optimal value function in a MDP, because future decisions does not depend on the current one. Therefore, the optimal policy can be expressed as one action followed by optimal policy from the resulting state.

Dynamic Programming

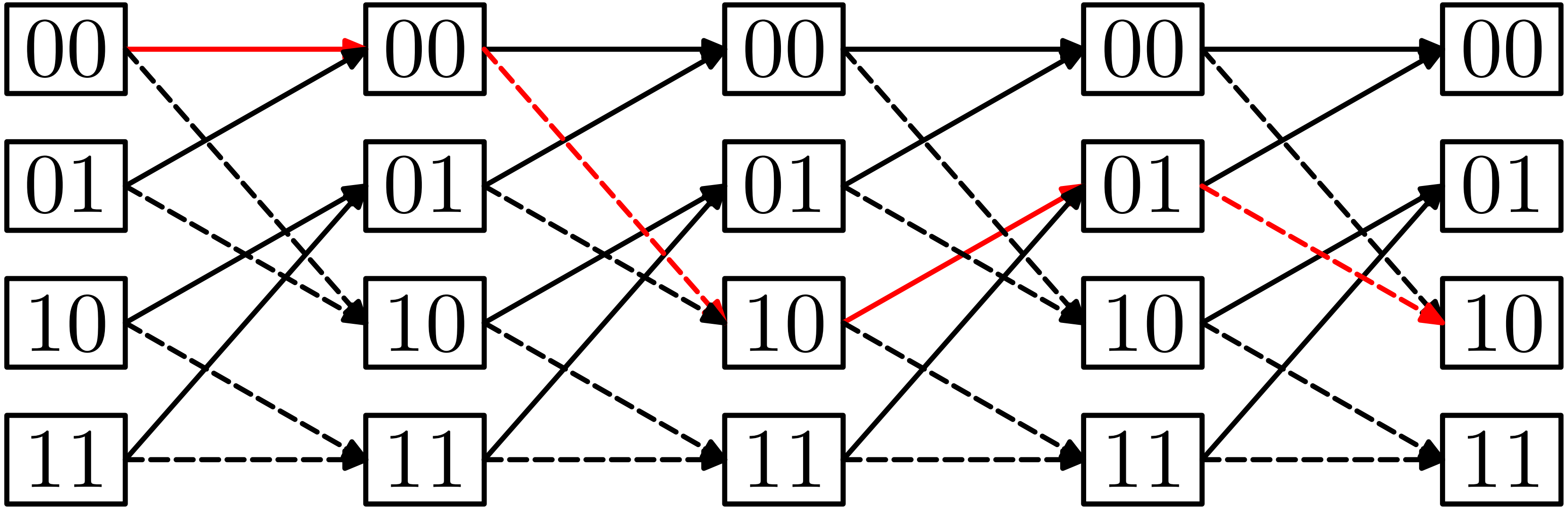
To turn the Bellman equation into an algorithm, we change the equal signs to assignments:

$$\begin{aligned}
v_0(s) &\leftarrow \begin{cases} 0 & \text{for terminal state } s \\ -\infty & \text{otherwise} \end{cases} \\
v_{k+1}(s) &\leftarrow \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a\right].
\end{aligned}$$

In a finite-horizon task with at most H steps, the optimal value function is reached after H iterations of the above assignment – we can show by induction that $v_k(s)$ is the maximum return reachable from state s in k steps.

Relations to Graph Algorithms

Searching for optimal value functions of deterministic problems is in fact search for the shortest path in a suitable graph.



https://upload.wikimedia.org/wikipedia/commons/a/a0/Convolutional_code_trellis_diagram.svg

Bellman-Ford-Moore Algorithm

$$v_{k+1}(s) \leftarrow \max_a \mathbb{E}\left[R_{t+1} + \gamma v_k(S_{t+1}) \middle| S_t = s, A_t = a\right].$$

Bellman-Ford-Moore algorithm:

```
# input: graph `g`, initial vertex `s`
for v in g.vertices: d[v] = 0 if v == s else +∞

for i in range(len(g.vertices) - 1):
  for e in g.edges:
    if d[e.source] + e.length < d[e.target]:
      d[e.target] = d[e.source] + e.length
```

Bellman Equation Solutions

If we fix value of terminal states to 0, the Bellman equation has a unique solution. Therefore, not only does the optimal value function satisfy the Bellman equation, but the converse statement is also true: If a value function satisfies the Bellman equation, it is optimal.

To sketch the proof of the statement, consider for a contradiction that the value function is not optimal. Then there exists a state s which has different than optimal value.

Consider now a trajectory following some optimal policy. Such a trajectory eventually reaches a terminal state.

Lastly focus on the last state on the trajectory with different than optimal value – the Bellman Equation cannot be fulfilled in this state.

Bellman Backup Operator

Our goal is now to handle also infinite horizon tasks, using discount factor of $\gamma < 1$.

For any value function $v \in \mathbb{R}^{|\mathcal{S}|}$ we define *Bellman backup operator* $B : \mathbb{R}^{|\mathcal{S}|} \rightarrow \mathbb{R}^{|\mathcal{S}|}$ as

$$Bv(s) \stackrel{\text{def}}{=} \max_a \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \middle| S_t = s, A_t = a\right].$$

It is not difficult to show that Bellman backup operator is a *contraction*:

$$\max_s \left|Bv_1(s) - Bv_2(s)\right| \leq \gamma \max_s \left|v_1(s) - v_2(s)\right|.$$

Considering a normed vector space $\mathbb{R}^{|\mathcal{S}|}$ with sup-norm $||\cdot||_\infty$, from Banach fixed-point theorem it follows there exist a *unique value function* v_* such that

$$Bv_* = v_*.$$

Such unique v_* is the *optimal value function*, because it satistifes the Bellman equation.

Bellman Backup Operator

Furthermore, iterative application of B on arbitrary v converges to v_*, because

$$||Bv - v_*||_\infty = ||Bv - Bv_*||_\infty \leq \gamma ||v - v_*||,$$

and therefore $B^n v \rightarrow v_*$.

Value Iteration Algorithm

We can turn the iterative application of Bellman backup operator into an algorithm.

$$Bv(s) \overset{\text{def}}{=} \max_a \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \middle| S_t = s, A_t = a\right]$$

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 $V(\textit{terminal}) = 0$

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

Output a deterministic policy, $\pi \approx \pi_*$, such that
    $\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r | s, a)\big[r + \gamma V(s')\big]$
```

Modification of Algorithm 4.4 of "Reinforcement Learning: An Introduction, Second Edition" (replacing S+ by S).

Value Iteration Algorithm

Although we have described the so-called *synchronous* implementation requiring two arrays for v and Bv, usual implementations are *asynchronous* and modify the value function in place (if a fixed ordering is used, usually such value iteration is called *Gauss-Seidel*).

Even with such asynchronous update value iteration can be proven to converge, and usually performs better in practise.

For example, the Bellman-Ford-Moore algorithm also updates the distances in-place. In the case of dynamic programming, we can extend the invariant from "$v_k(s)$ is the maximum return reachable from state s in k steps" to include not only all trajectories of k steps, but also some number of longer trajectories.

Bellman Backup Operator as a Contraction

To show that Bellman backup operator is a contraction, we proceed as follows:

$$\begin{aligned}
||Bv_1 - Bv_2||_\infty &= ||\max_a \mathbb{E}\left[R_{t+1} + \gamma v_1(S_{t+1})\right] - \max_a \mathbb{E}\left[R_{t+1} + \gamma v_2(S_{t+1})\right]||_\infty \\
&\le \max_a \left(||\mathbb{E}\left[R_{t+1} + \gamma v_1(S_{t+1})\right] - \mathbb{E}\left[R_{t+1} + \gamma v_2(S_{t+1})\right]||_\infty\right) \\
&= \max_a \left(\left|\left|\sum\nolimits_{s',r} p\left(s', r|s, a\right) \gamma (v_1(s') - v_2(s'))\right|\right|_\infty\right) \\
&= \gamma \max_a \left(\left|\left|\sum\nolimits_{s'} p\left(s'|s, a\right) (v_1(s') - v_2(s'))\right|\right|_\infty\right) \\
&\le \gamma||v_1 - v_2||_\infty,
\end{aligned}$$

where the second line follows from $|\max_x f(x) - \max_x g(x)| \le \max_x |f(x) - g(x)|$ and the last line from the fact that from any given s and a, the $\sum_{s'} p(s'|s,a)$ sums to 1.

Speed of Convergence

Assuming maximum reward is $R_\textrm{max}$, we have that

$$v_*(s) \leq \sum_{t=0}^\infty \gamma^t R_\textrm{max} = \frac{R_\textrm{max}}{1-\gamma}.$$

Starting with $v(s) \leftarrow 0$, we have

$$||B^k v - v_*||_\infty \leq \gamma^k ||v - v_*||_\infty \leq \gamma^k \frac{R_\textrm{max}}{1-\gamma}.$$

Compare to finite horizon case, where $B^T v = v_*$.

Value Iteration Example

Consider a simple betting game, where a gambler bets on the outcomes of a sequence of coin flips, either losing their stake or winning the same amount of coints that was bet. The gambler wins if they obtain 100 coins, and lose if they run our of money.

We can formulate the problem as an undiscounted episodic MDP. The states are the coins owned by the gambler, $\{1, \ldots, 99\}$, and actions are stakes $\{1, \ldots, \min(s, 100 - s)\}$. The reward is $+1$ when reaching 100 and 0 otherwise.

The state-value function then gives probability of winning from each state, and policy prescribes a stake with a given capital.

Value Iteration Example

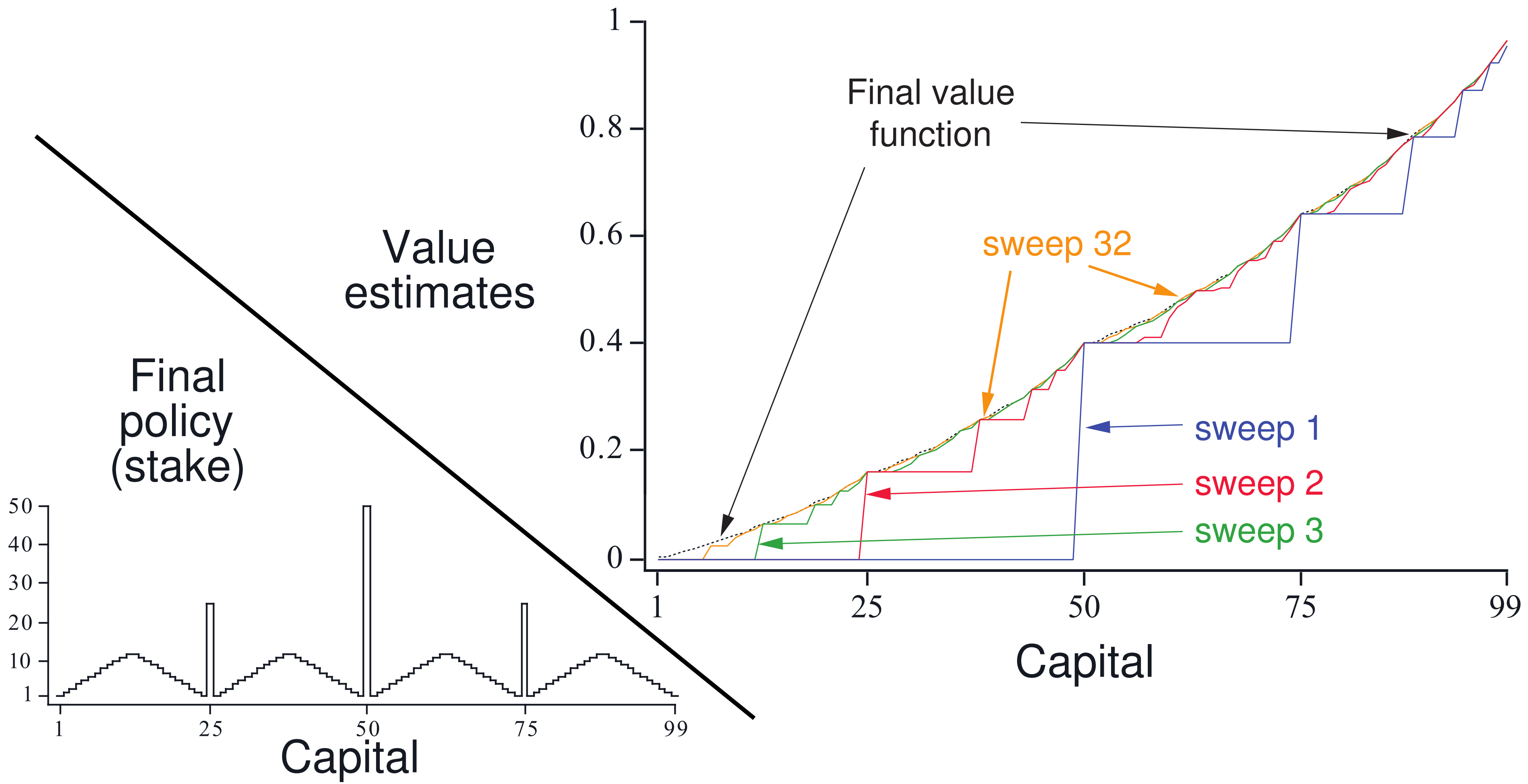


Figure 4.3 of "Reinforcement Learning: An Introduction, Second Edition".

Policy Iteration Algorithm

We now propose another approach of computing optimal policy. The approach, called *policy iteration*, consists of repeatedly performing policy *evaluation* and policy *improvement*.

Policy Evaluation

Given a policy π, policy evaluation computes v_π.

Recall that

$$\begin{aligned} v_\pi(s) &\stackrel{\text{def}}{=} \mathbb{E}_\pi\left[G_t \middle| S_t = s\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \middle| S_t = s\right] \\ &= \sum\nolimits_a \pi(a|s) \sum\nolimits_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right]. \end{aligned}$$

If the dynamics of the MDP p is known, the above is a system of linear equations, and therefore, v_π can be computed exactly.

Policy Evaluation

The equation

$$v_\pi(s) = \sum\nolimits_a \pi(a|s) \sum\nolimits_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s')\right]$$

is called *Bellman equation for v_π* and analogously to Bellman optimality equation, it can be proven that

- under the same assumptions as before ($\gamma < 1$ or termination), v_π exists and is unique;
- v_π is a fixed point of the Bellman equation

$$v_{k+1}(s) = \sum\nolimits_a \pi(a|s) \sum\nolimits_{s',r} p(s', r|s, a) \left[r + \gamma v_k(s')\right];$$

- iterative application of the Bellman equation to any v converges to v_π.

Policy Evaluation

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

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

Modification of Algorithm 4.1 of "Reinforcement Learning: An Introduction, Second Edition" (replacing S+ by S).

Policy Improvement

Given π and computed v_π, we would like to *improve* the policy. A straightforward way to do so is to define a policy using a *greedy* action

$$\begin{aligned} \pi'(s) &\overset{\text{def}}{=} \arg\max_a q_\pi(s, a) \\ &= \arg\max_a \sum\nolimits_{s', r} p(s', r|s, a) \left[r + \gamma v_\pi(s')\right]. \end{aligned}$$

For such π', we can easily show that

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

Policy Improvement Theorem

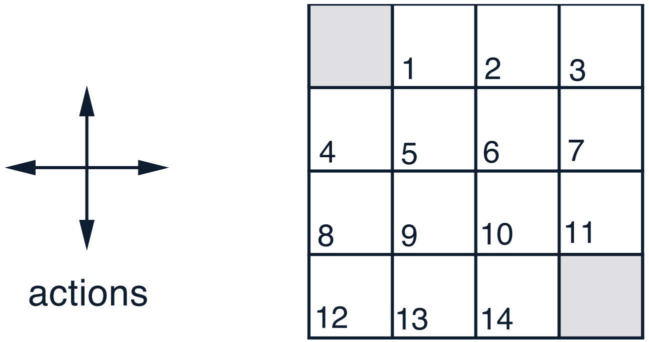
Let π and π' be any pair of deterministic policies, such that $q_\pi(s, \pi'(s)) \geq v_\pi(s)$.

Then for all states s, $v_{\pi'}(s) \geq v_\pi(s)$.

The proof is straightforward, we repeatedly expand q_π and use the assumption of the policy improvement theorem:

$$\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] | S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) | S_t = s] \\
&\ldots \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s] = v_{\pi'}(s)
\end{aligned}$$

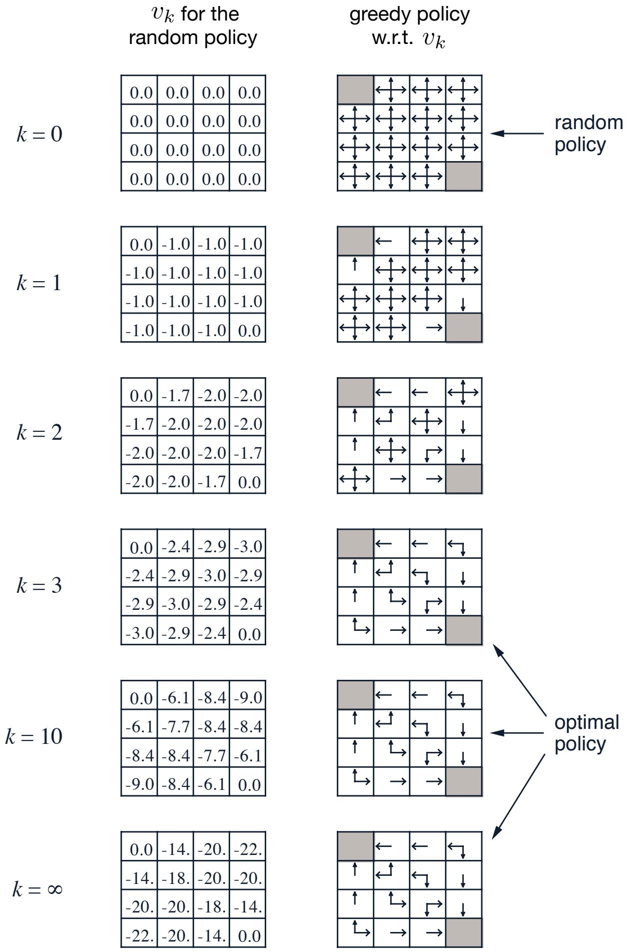
Policy Improvement Example



$$R_t = -1$$

on all transitions

Example 4.1 of "Reinforcement Learning: An Introduction, Second Edition".



$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Figure 4.1 of "Reinforcement Learning: An Introduction, Second Edition".

Policy Iteration Algorithm

Policy iteration consists of repeatedly performing policy evaluation and policy improvement:

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} v_{\pi_2} \xrightarrow{I} \ldots \xrightarrow{I} \pi_* \xrightarrow{E} v_{\pi_*}.$$

The result is a sequence of monotonically improving policies π_i. Note that when $\pi' = \pi$, also $v_{\pi'} = v_\pi$, which means Bellman optimality equation is fulfilled and both v_π and π are optimal.

Considering that there is only a finite number of policies, the optimal policy and optimal value function can be computed in finite time (contrary to value iteration, where the convergence is only asymptotic).

Note that when evaluating policy π_{k+1}, we usually start with v_{π_k}, which is assumed to be a good approximation to $v_{\pi_{k+1}}$.

Policy Iteration Algorithm

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 $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
       $\Delta \leftarrow 0$
       Loop for each $s \in \mathcal{S}$:
           $v \leftarrow V(s)$
           $V(s) \leftarrow \sum_{s',r} p(s', r \,|\, s, \pi(s)) \big[r + \gamma V(s')\big]$
           $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   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* $\leftarrow \pi(s)$
       $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \,|\, s, a) \big[r + \gamma V(s')\big]$
       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
```

Algorithm 4.3 of "Reinforcement Learning: An Introduction, Second Edition".

Value Iteration as Policy Iteration

Note that value iteration is in fact a policy iteration, where policy evaluation is performed only for one step:

$$\pi'(s) = \arg\max_a \sum\nolimits_{s',r} p(s',r|s,a)\left[r + \gamma v(s')\right] \qquad \textit{(policy improvement)}$$

$$v'(s) = \sum\nolimits_a \pi'(a|s) \sum\nolimits_{s',r} p(s',r|s,a)\left[r + \gamma v(s')\right] \qquad \textit{(one step of policy evaluation)}$$

Substituting the former into the latter, we get

$$v'(s) = \max_a \sum\nolimits_{s',r} p(s',r|s,a)\left[r + \gamma v(s)\right] = Bv(s).$$

Generalized Policy Iteration

Therefore, it seems that to achieve convergence, it is not necessary to perform policy evaluation exactly.

Generalized Policy Evaluation is a general idea of interleaving policy evaluation and policy improvement at various granularity.

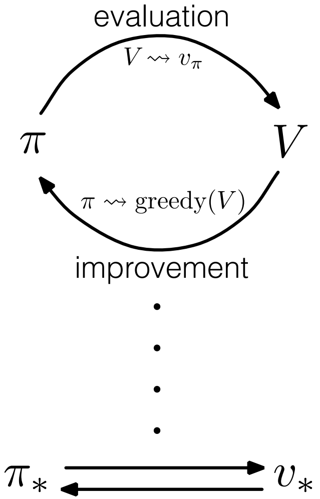

evaluation
$V \rightsquigarrow v_\pi$
π
V
$\pi \rightsquigarrow \mathrm{greedy}(V)$
improvement
π_*
v_*

Figure in Section 4.6 of "Reinforcement Learning: An Introduction, Second Edition".

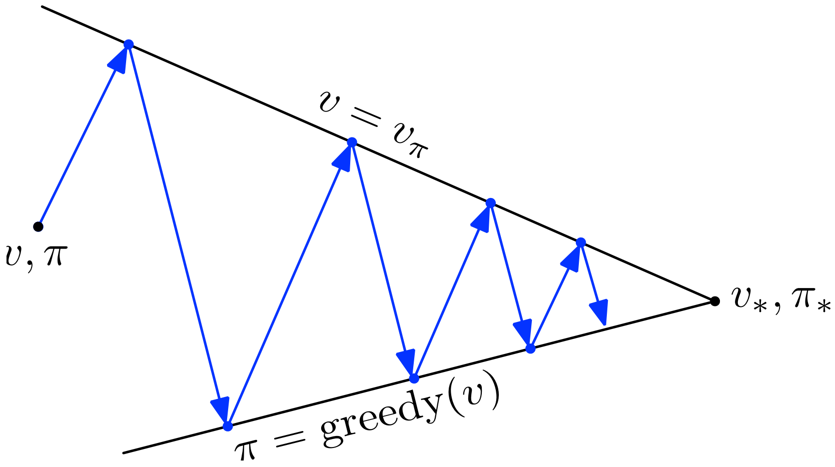


Figure in Section 4.6 of "Reinforcement Learning: An Introduction, Second Edition".

If both processes stabilize, we know we have obtained optimal policy.

Monte Carlo Methods

We now present the first algorithm for computing optimal policies without assuming a knowledge of the environment dynamics.

However, we still assume there are finitely many states $\mathcal{S}$ and we will store estimates for each of them.

Monte Carlo methods are based on estimating returns from complete episodes. Furthermore, if the model (of the environment) is not known, we need to estimate returns for the action-value function q instead of v.

We can formulate Monte Carlo methods in the generalized policy improvement framework.

Keeping estimated returns for the action-value function, we perform policy evaluation by sampling one episode according to current policy. We then update the action-value function by averaging over the observed returns, including the currently sampled episode.

Monte Carlo Methods

To guarantee convergence, we need to visit each state infinitely many times. One of the simplest way to achieve that is to assume *exploring starts*, where we randomly select the first state and first action, each pair with nonzero probability.

Furthermore, if a state-action pair appears multiple times in one episode, the sampled returns are not independent. The literature distinguishes two cases:

- *first visit*: only the first occurence of a state-action pair in an episode is considered
- *every visit*: all occurences of a state-action pair are considered.

Even though first-visit is easier to analyze, it can be proven that for both approaches, policy evaluation converges. Contrary to the Reinforcement Learning: An Introduction book, which presents first-visit algorithms, we use every-visit.

Monte Carlo with Exploring Starts

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

```
Initialize:
    $\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
    $Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
    $Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
    Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability $> 0$
    Generate an episode from $S_0, A_0$, following $\pi$: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Append $G$ to $Returns(S_t, A_t)$
        $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
        $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$
```

Modification of algorithm 5.3 of "Reinforcement Learning: An Introduction, Second Edition" from first-visit to every-visit.

Monte Carlo and ε-soft Policies

A policy is called ε-soft, if

$$\pi(a|s) \geq \frac{\varepsilon}{|\mathcal{A}(s)|}.$$

For ε-soft policy, Monte Carlo policy evaluation also converges, without the need of exploring starts.

We call a policy ε-greedy, if one action has maximum probability of $1 - \varepsilon + \frac{\varepsilon}{|A(s)|}$.

The policy improvement theorem can be proved also for the class of ε-soft policies, and using ε-greedy policy in policy improvement step, policy iteration has the same convergence properties. (We can embed the ε-soft behaviour “inside” the environment and prove equivalence.)

Monte Carlo for ε-soft Policies

On-policy every-visit Monte Carlo for ε-soft Policies

Algorithm parameter: small $\varepsilon > 0$

Initialize $Q(s, a) \in \mathbb{R}$ arbitrarily (usually to 0), for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $C(s, a) \in \mathbb{Z}$ to 0, for all $s \in \mathcal{S}, a \in \mathcal{A}$

Repeat forever (for each episode):

- Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, by generating actions as follows:
  - With probability ε, generate a random uniform action
  - Otherwise, set $A_t \stackrel{\text{def}}{=} \arg\max_a Q(S_t, a)$
- $G \leftarrow 0$
- For each $t = T-1, T-2, \ldots, 0$:
  - $G \leftarrow \gamma G + R_{t+1}$
  - $C(S_t, A_t) \leftarrow C(S_t, A_t) + 1$
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{C(S_t, A_t)}(G - Q(S_t, A_t))$