Value and Policy Iteration, Monte Carlo, Temporal Difference

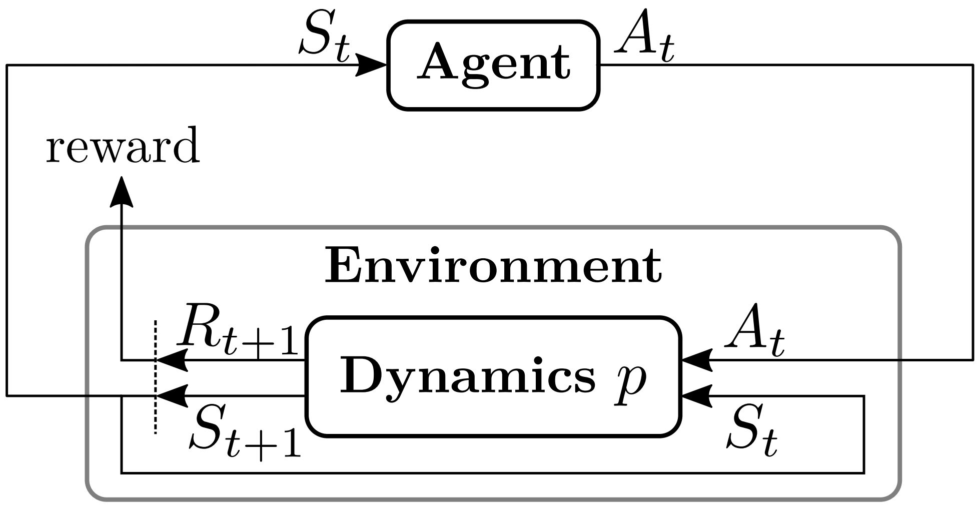
Milan Straka

February 26, 2024

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

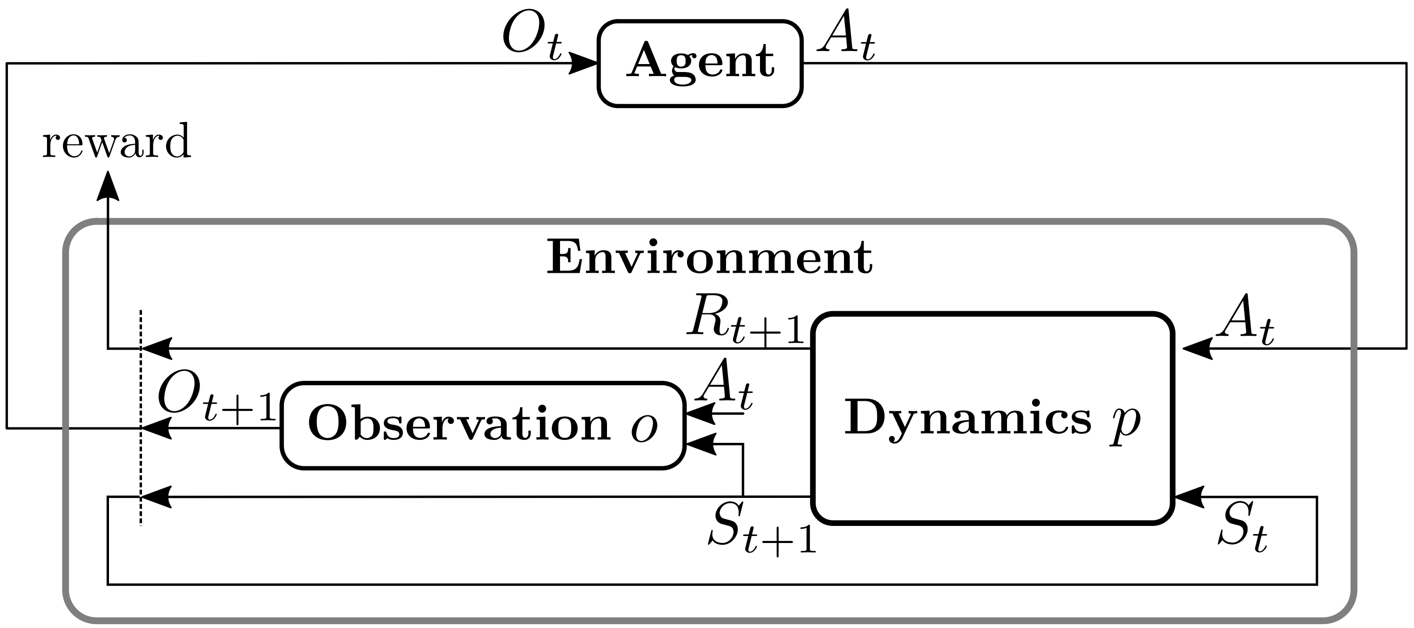


A **Markov decision process** (MDP) is a quadruple $(\mathcal{S}, \mathcal{A}, p, \gamma)$, where:

- $\mathcal{S}$ 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 $G_t \stackrel{\text{def}}{=} \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$. The goal is to optimize $\mathbb{E}[G_0]$.

Partially Observable MDPs



Partially observable Markov decision process extends the Markov decision process to a sextuple $(\mathcal{S}, \mathcal{A}, p, \gamma, \mathcal{O}, o)$, where in addition to an MDP,

- $\mathcal{O}$ is a set of observations,
- $o(O_{t+1} | S_{t+1}, A_t)$ is an observation model, where observation O_t is used as agent input instead of the state S_t.

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-horizon 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

$$\begin{aligned} v_\pi(s) &\overset{\text{def}}{=} \mathbb{E}_\pi\left[G_t \middle| S_t = s\right] = \mathbb{E}_\pi\left[\sum\nolimits_{k=0}^\infty \gamma^k R_{t+k+1} \middle| S_t = s\right] \\ &= \mathbb{E}_{A_t\sim\pi(s)} \mathbb{E}_{S_{t+1},R_{t+1}\sim p(s,A_t)} \left[R_{t+1} + \gamma \mathbb{E}_{A_{t+1}\sim\pi(S_{t+1})} \mathbb{E}_{S_{t+2},R_{t+2}\sim p(S_{t+1},A_{t+1})} \left[R_{t+2} + \ldots\right]\right] \end{aligned}$$

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

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

The value function and action-value function can be of course expressed using one another:

$$v_\pi(s) = \mathbb{E}_{a\sim\pi}\big[q_\pi(s, a)\big], \qquad q_\pi(s, a) = \mathbb{E}_{s',r\sim p}\big[r + \gamma v_\pi(s')\big].$$

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

Existence

In finite-horizon tasks or if $\gamma < 1$, there always exists a unique optimal state-value function, a unique optimal action-value function, and a (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 $\mathcal{S}$, finite number of actions $\mathcal{A}$, and known MDP dynamics p. Note that without loss of generality, we can assume that every episode takes exactly H steps (by introducing a suitable absorbing state, if necessary).

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

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

It must hold for an optimal value function in a MDP, because future decisions do 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 sign to an assignment:

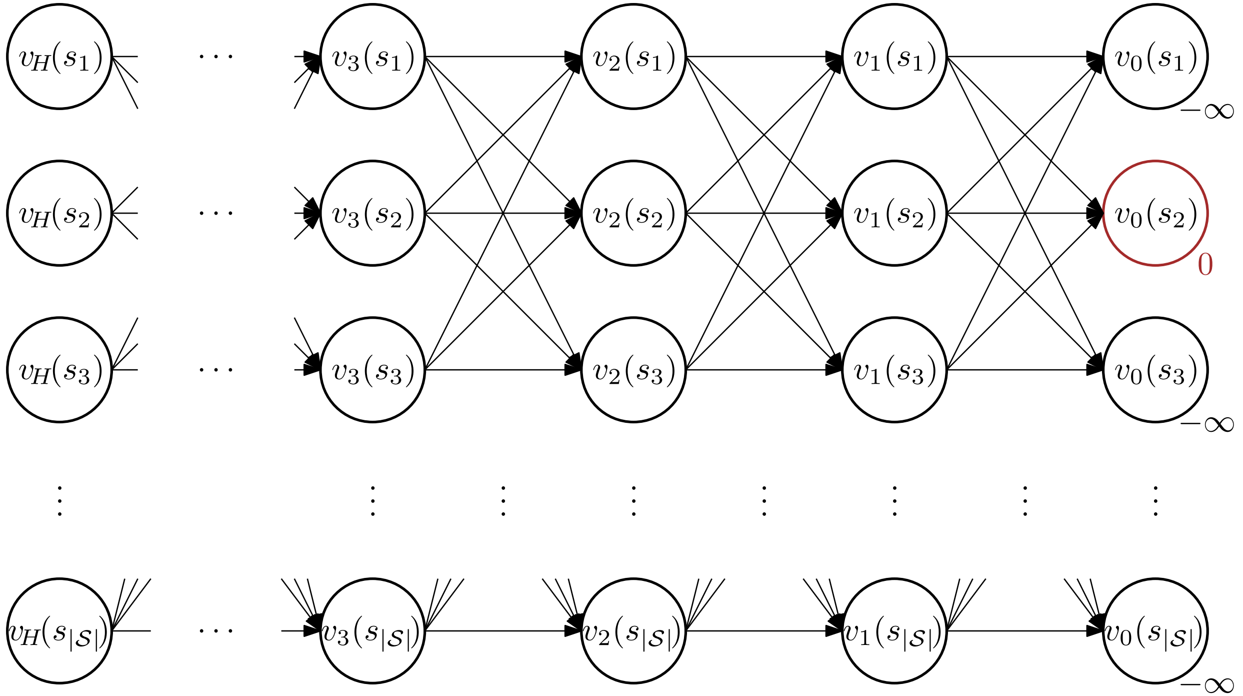
$$v_0(s) \leftarrow \begin{cases} 0 & \text{for the terminal state } s \\ -\infty & \text{otherwise} \end{cases}$$
$$v_{k+1}(s) \leftarrow \max_a \mathbb{E}\big[R_{t+1} + \gamma v_k(S_{t+1}) \big| S_t = s, A_t = a\big].$$

In a finite-horizon task with 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 last k steps of an episode.
- If every episode ends in at most H steps, then v_{H+1} must be equal to v_H.

Relations to Graph Algorithms

In current settings, searching for the optimal value function of a deterministic MDP problem (i.e., when there are always just a single next state and a single reward) is in fact the same as searching for the longest (maximum weighted) path in a suitable graph:



where the value of an edge going from $v_{t+1}(s_i)$ to $v_t(s_j)$ is either the highest reward some transition $p(s_i, a) \rightarrow (s_j, r)$ produces, or $-\infty$ if no action from the state s_i leads to s_j.

Bellman-Ford-Moore Algorithm

Consider the dynamic programming algorithm of the repeated Bellman equation application:

$$v_0(s) \leftarrow \begin{cases} 0 & \text{for the terminal state } s \\ -\infty & \text{otherwise} \end{cases}$$

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

The Bellman-Ford-Moore shortest-path algorithm can be considered its special-case:

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

for iteration 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
```

Uniqueness of Bellman Equation Solution

Not only does the optimal value function fulfill the Bellman equation in the current settings, the converse 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 some solution of Bellman equation is not an optimal value function. Therefore, there exist states with different than optimal values.

Among those states, we choose such a state that all trajectories from it contains only states with optimal values. We can find it by starting in an arbitrary state with different than optimal value, and then repeatedly switching into a reachable state with different than optimal value function.

For such a state, however, if its value is not optimal, then the Bellman equation cannot hold in this state, which is a contradiction.

Bellman Backup Operator

Our goal is now to handle also infinite-horizon tasks, using discount factor of $\gamma < 1$. However, we still assume finite number of states and actions.

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

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

Considering the supremum norm $||x||_\infty \stackrel{\text{def}}{=} \sup_s |x(s)|$, we will show that Bellman backup operator is a *contraction* (even for infinite number of states), i.e.,

$$\sup_s \big|Bv_1(s) - Bv_2(s)\big| = \big|\big|Bv_1 - Bv_2\big|\big|_\infty \leq \gamma ||v_1 - v_2||_\infty.$$

Applying the Banach fixed-point theorem on the normed vector space $\mathbb{R}^{|\mathcal{S}|}$ with the supremum norm then yields that there exists a *unique value function* v_* such that $Bv_* = v_*$.

Such a 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

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

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 $\pi \approx \pi_*$

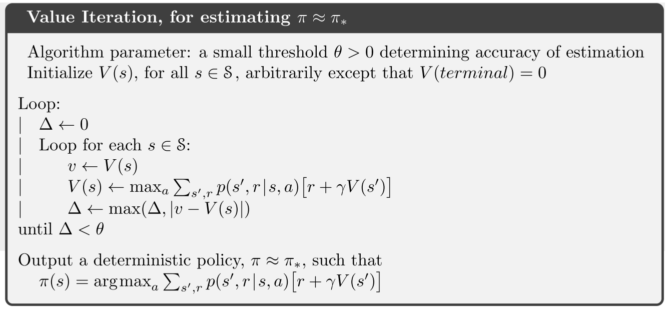
```
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}$, arbitrarily except that $V(terminal) = 0$

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

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*).

- for $s \in S$ in some fixed order:
  - $v(s) \leftarrow \max_a \mathbb{E}\big[R_{t+1} + \gamma v(S_{t+1}) \big| S_t = s, A_t = a\big]$



```
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 $V(terminal) = 0$

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

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

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

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 last k steps of an episode" to include not only all trajectories of k steps, but also any number of longer trajectories.

If you are interested, try proving that the above Gauss-Seidel iteration is also a contraction.

Bellman Backup Operator as a Contraction

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

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

Speed of Convergence

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

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

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

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

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

Value Iteration Example

Consider a simple betting game, where a gambler repeatedly bets on the outcome of a coin flip (with a given win probability), either losing their stake or winning the same amount of coins 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 the 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

For a coin flip win probability 40%, the value iteration proceeds as follows.

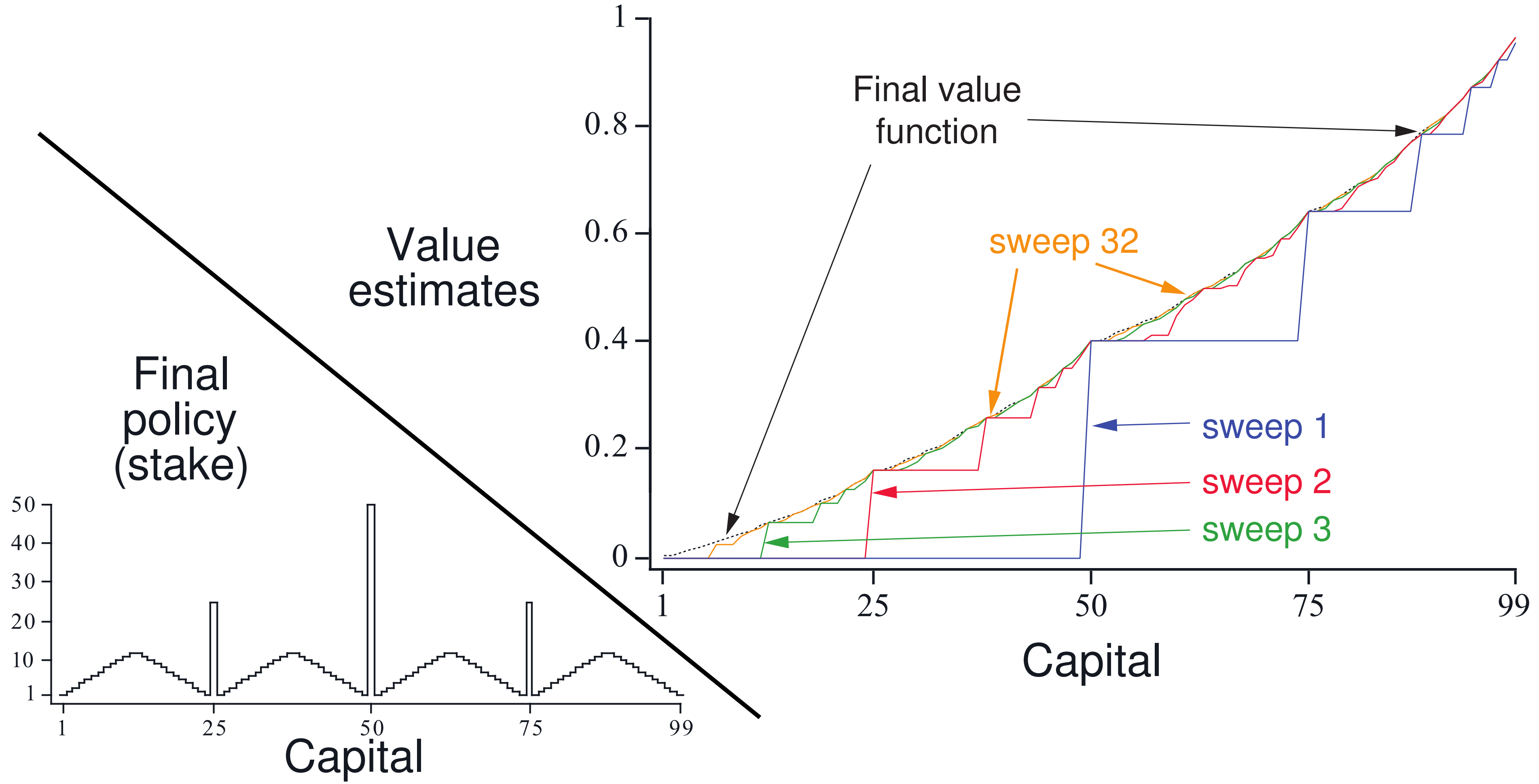


Figure 4.3 of "Reinforcement Learning: An Introduction, Second Edition".

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

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

- iterative application of the Bellman equation to any v converges to v_π (the proof is easier than for the optimality equation, because v_π is defined using an expectation and expectations are linear, so we get the first half of the proof "for free").

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

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

For such π', by construction it obviously holds that

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

Policy Improvement Theorem

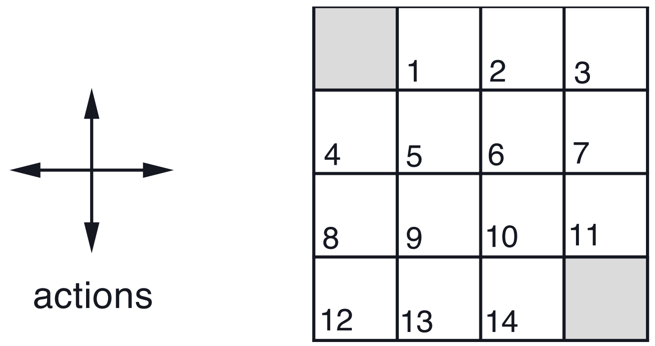
Let π and π' be any pair of deterministic policies, such that $q_\pi(s, \pi'(s)) \geq v_\pi(s)$.

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:

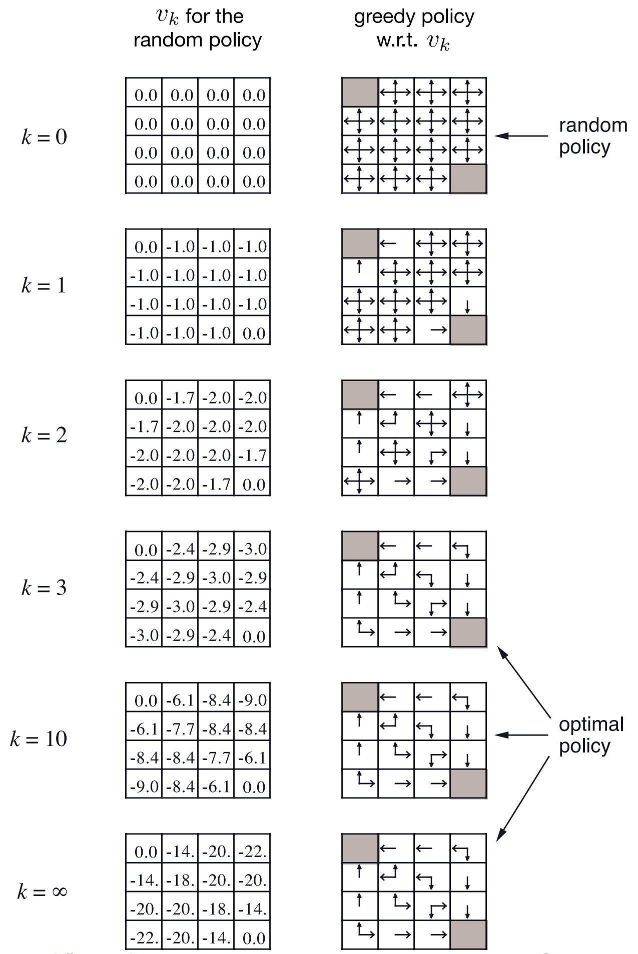
$$\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1}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

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 $s \in \mathcal{S}$:
           $v \leftarrow V(s)$
           $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \big[r + \gamma V(s')\big]$
           $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   $policy\text{-}stable \leftarrow true$
   For each $s \in \mathcal{S}$:
       $old\text{-}action \leftarrow \pi(s)$
       $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a) \big[r + \gamma V(s')\big]$
       If $old\text{-}action \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
   If $policy\text{-}stable$, 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:

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

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

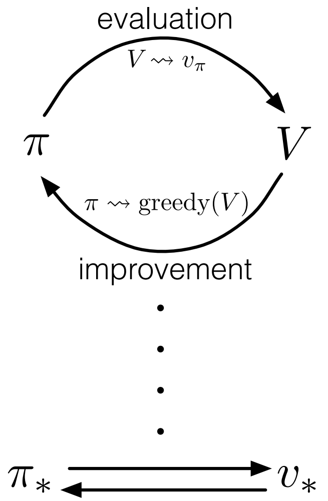
Substituting the former into the latter, we get

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

Generalized Policy Iteration

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

Generalized Policy Evaluation is a general concept 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".

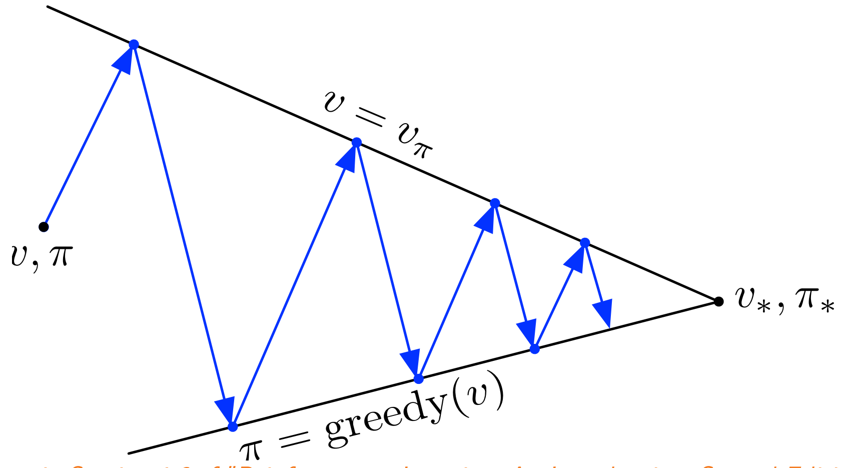


Figure in Section 4.6 of "Reinforcement Learning: An Introduction, Second Edition".

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

Monte Carlo Methods

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 hope for 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. 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 $Returns(S_t, A_t)$
        $Q(S_t, A_t) \leftarrow \operatorname{average}(Returns(S_t, A_t))$
        $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$
```

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

Monte Carlo and ε-soft Policies

The problem with exploring starts is that in many situations, we either cannot start in an arbitrary state, or it is impractical.

A policy is called ε-soft, if

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

and we call it ε-greedy, if one action has a 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 $t = T-1, T-2, \ldots, 0$:
  - $G \leftarrow \gamma G + R_{t+1}$
  - $C(S_t, A_t) \leftarrow C(S_t, A_t) + 1$
  - $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{C(S_t, A_t)}(G - Q(S_t, A_t))$

Action-values and Afterstates

The reason we estimate *action-value* function q is that the policy is defined as

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

and the latter form might be impossible to evaluate if we do not have the model of the environment.

However, if the environment is known, it is often better to estimate returns only for states, because there can be substantially less states than state-action pairs.

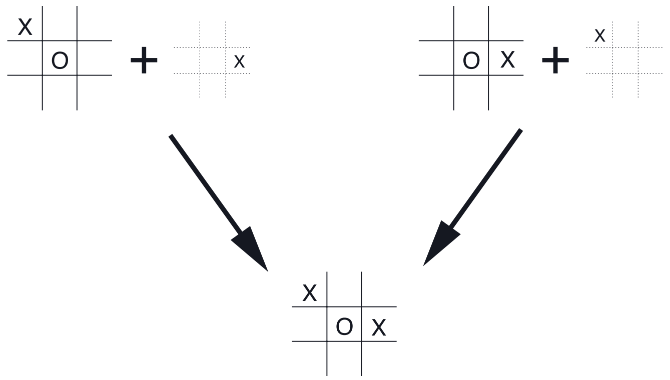


Figure from section 6.8 of "Reinforcement Learning: An Introduction, Second Edition".

TD Methods

Temporal-difference methods estimate action-value returns using one iteration of Bellman equation instead of complete episode return.

Compared to Monte Carlo method with constant learning rate α, which performs

$$v(S_t) \leftarrow v(S_t) + \alpha\big(G_t - v(S_t)\big),$$

the simplest temporal-difference method computes the following:

$$v(S_t) \leftarrow v(S_t) + \alpha\big(R_{t+1} + [\neg\text{done}] \cdot \gamma v(S_{t+1}) - v(S_t)\big),$$

where $[\neg\text{done}]$ has a value of 1 if the episode continues in the state S_{t+1}, and 0 otherwise.

We say TD methods are **bootstraping**, because they base their update on an existing (action-)value function estimate.

TD Methods

State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

Example 6.1 of "Reinforcement Learning: An Introduction, Second Edition".

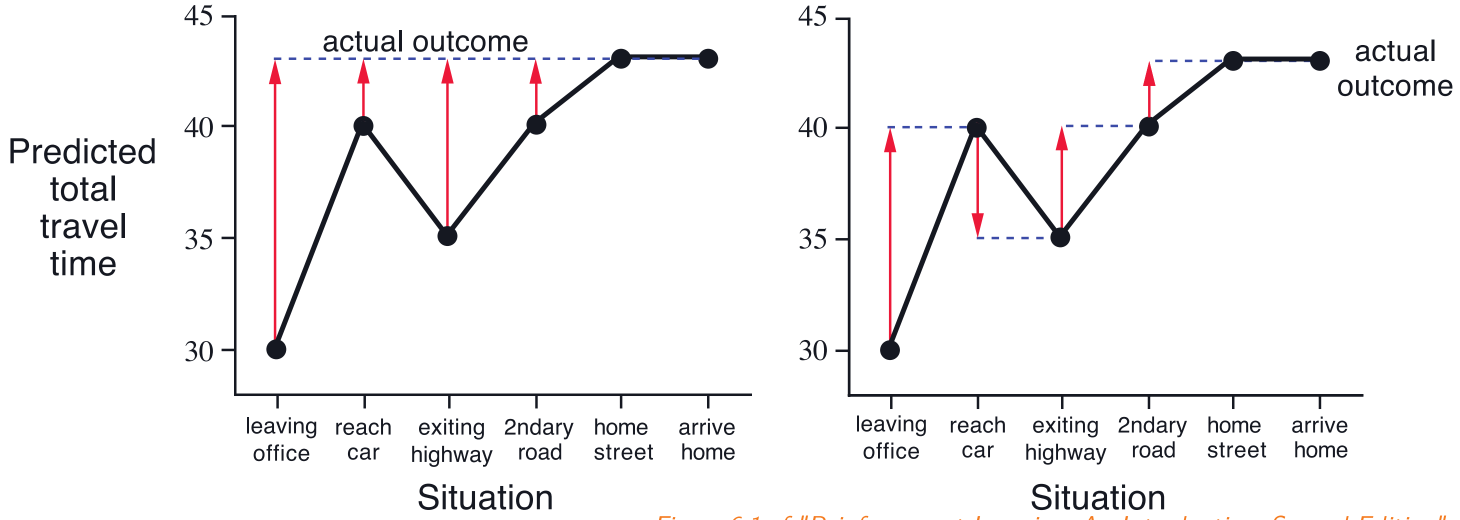


Figure 6.1 of "Reinforcement Learning: An Introduction, Second Edition".

TD Methods

An obvious advantage of TD methods compared to Monte Carlo is that they are naturally implemented in *online*, *fully incremental* fashion, while the Monte Carlo methods must wait until an episode ends, because only then the return is known.

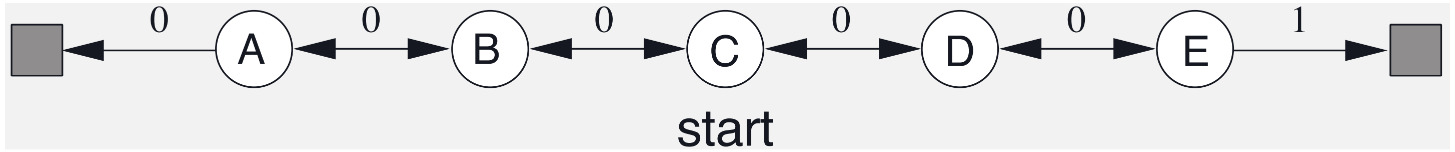
The possibility of immediate learning is useful for:

- continuous environments,
- environments with extremely large episodes,
- environments ending after some nontrivial goal is reached, requiring some coordinated strategy from the agent (i.e., it is improbable that random actions will reach it).

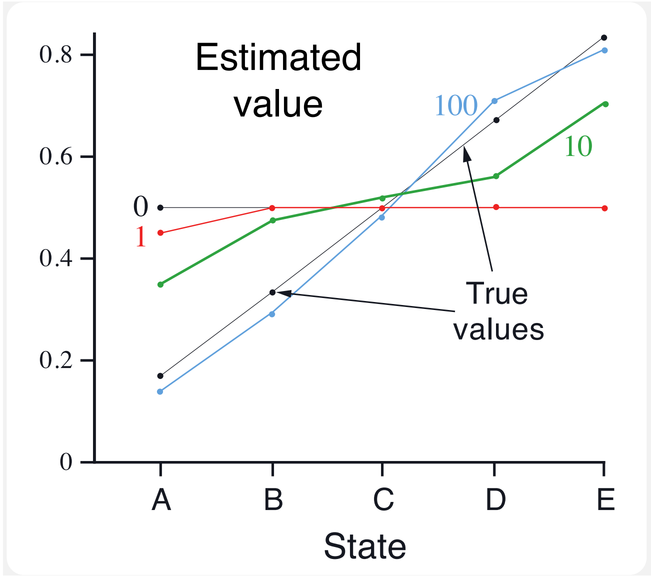
TD and MC Comparison

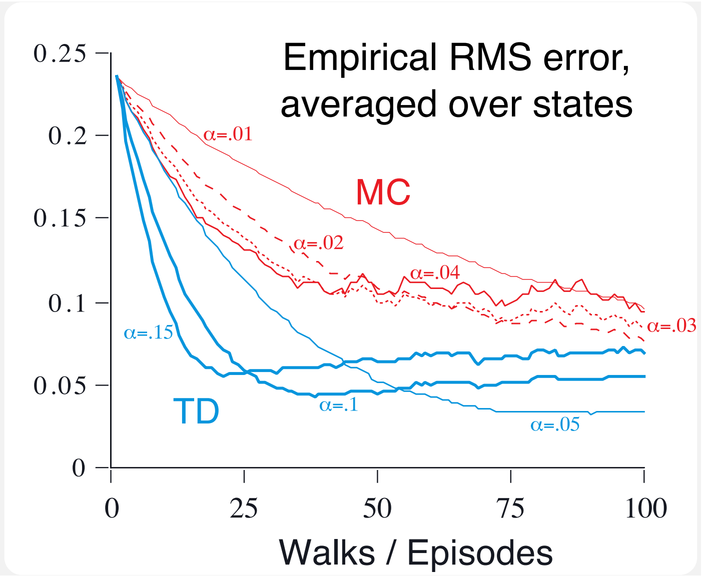
As with Monte Carlo methods, for a fixed policy π (i.e., the policy evaluation part of the algorithms), TD methods converge to v_π.

On stochastic tasks, TD methods usually converge to v_π faster than constant-α MC methods.



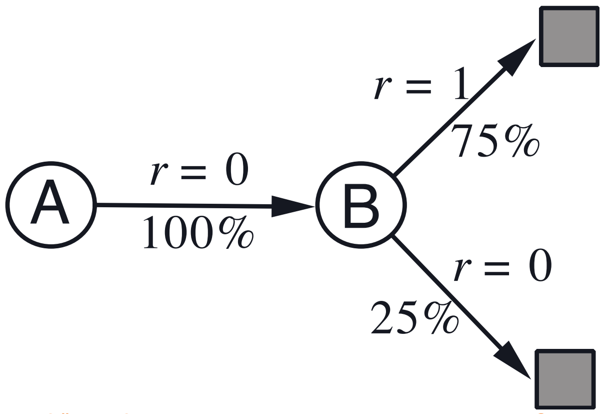
Example 6.2 of "Reinforcement Learning: An Introduction, Second Edition".





Example 6.2 of "Reinforcement Learning: An Introduction, Second Edition".

Optimality of MC and TD Methods



Example 6.4 of "Reinforcement Learning: An Introduction, Second Edition".

A, 0, B, 0	B, 1
B, 1	B, 1
B, 1	B, 1
B, 1	B, 0

Example 6.4 of "Reinforcement Learning: An Introduction, Second Edition".

For state B, 6 out of 8 times return from B was 1 and 0 otherwise. Therefore, $v(B) = 3/4$.

- [TD] For state A, in all cases it transferred to B. Therefore, $v(A)$ could be $3/4$.
- [MC] For state A, in all cases it generated return 0. Therefore, $v(A)$ could be 0.

MC minimizes mean squared error on the returns from the training data, while TD finds the estimates that would be exactly correct for a maximum-likelihood estimate of the Markov process model (the estimated transition probability from s to t is the fraction of observed transitions from s that went to t, and the corresponding reward is the average of the rewards observed on those transitions).

Sarsa

A straightforward application to the temporal-difference policy evaluation is Sarsa algorithm, which after generating $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$ computes

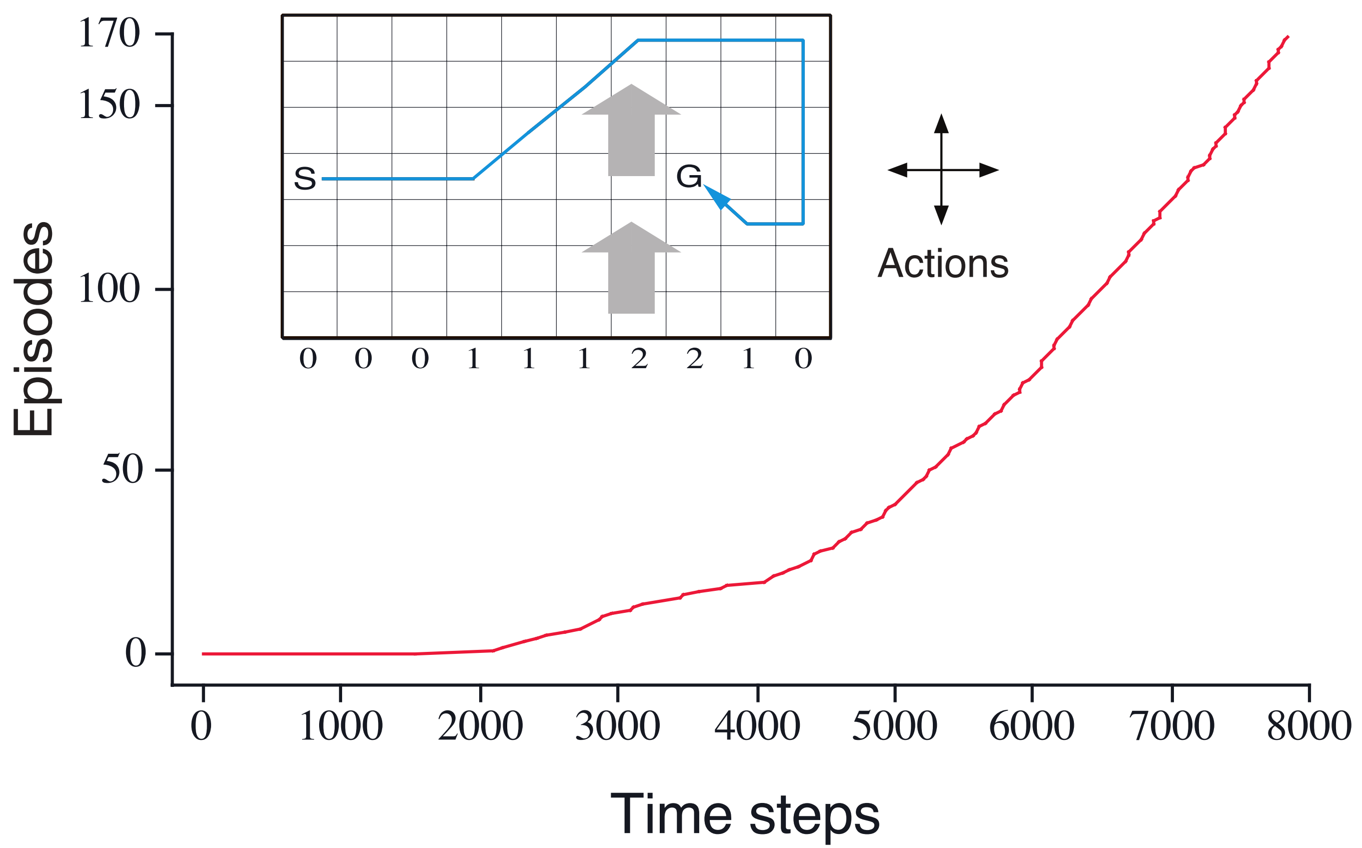
$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha\Big(R_{t+1} + [\neg\text{done}] \cdot \gamma q(S_{t+1}, A_{t+1}) - q(S_t, A_t)\Big).$$

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Loop for each step of episode:
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until $S$ is terminal
```

Modification of Algorithm 6.4 of "Reinforcement Learning: An Introduction, Second Edition" (replacing S+ by S).

Sarsa



Example 6.5 of "Reinforcement Learning: An Introduction, Second Edition".

MC methods cannot be easily used, because an episode might not terminate if the current policy causes the agent to stay in the same state.

Q-learning

Q-learning was an important early breakthrough in reinforcement learning (Watkins, 1989).

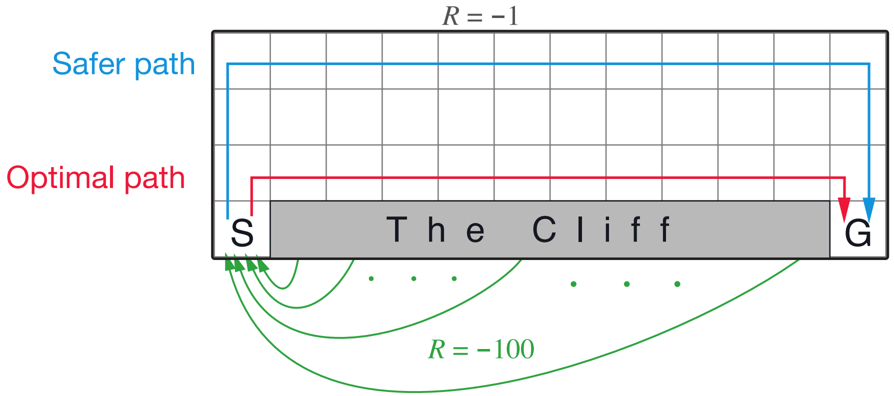
$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha\Big(R_{t+1} + [\neg\mathrm{done}] \cdot \gamma \max_a q(S_{t+1}, a) - q(S_t, A_t)\Big).$$

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

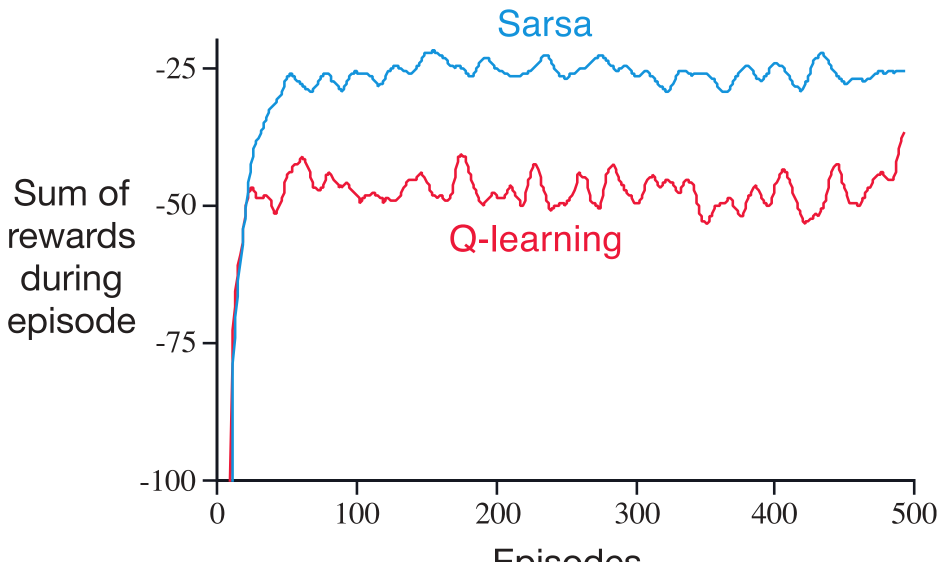
```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

Modification of Algorithm 6.5 of "Reinforcement Learning: An Introduction, Second Edition" (replacing S+ by S).

Q-learning versus Sarsa



Example 6.6 of "Reinforcement Learning: An Introduction, Second Edition".



Example 6.6 of "Reinforcement Learning: An Introduction, Second Edition".