Sequential decisions under uncertainty
Markov Decision Processes (MDP)

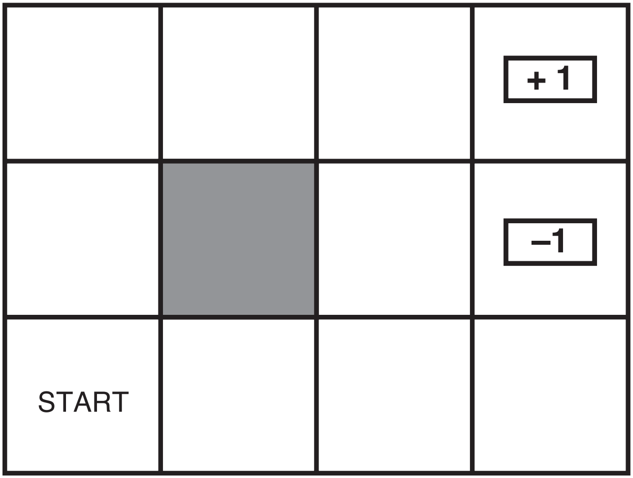
Tomáš Svoboda, Petr Pošík

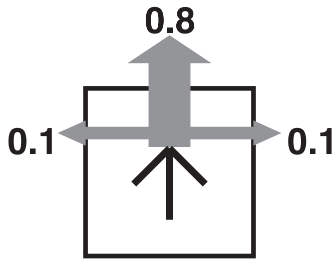
Vision for Robots and Autonomous Systems, Center for Machine Perception
Department of Cybernetics
Faculty of Electrical Engineering, Czech Technical University in Prague

March 20, 2024

Notes

Unreliable actions in *observable* grid world





States $s \in \mathcal{S}$, actions $a \in \mathcal{A}$
(Transition) Model $T(s, a, s') \equiv p(s'|s, a)$ = probability that a in s leads to s'

Notes

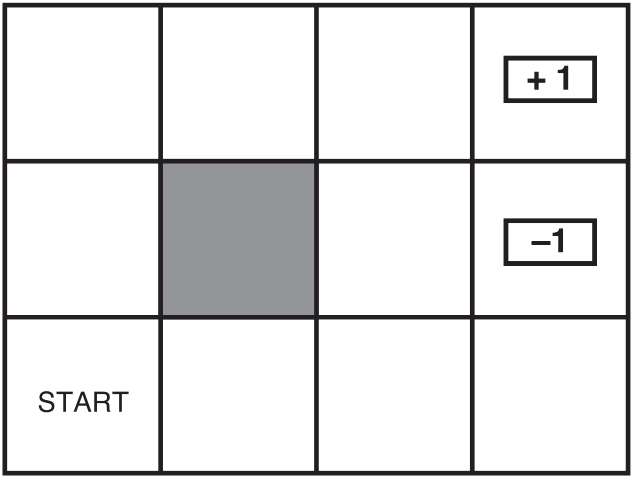
Beginning of semester – search – *deterministic* and (fully) *observable* environment
Now:

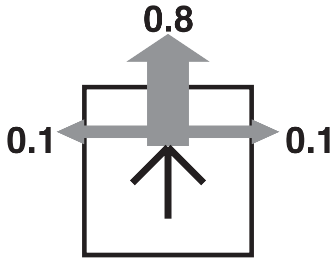
- Observable – we keep for now – agent knows where it is.
- Deterministic – We introduce "imperfect" agent that does not always obey the command – *stochastic action outcomes*.

There is a treasure (desired goal/end state) but there is also some danger (unwanted goal/end state).
The danger state: think about a mountainous area with safer but longer and shorter but more dangerous paths – a dangerous node may represent a chasm.

Notation note: caligraphic letters like $\mathcal{S}, \mathcal{A}$ will denote the set(s) of all states/actions.

Unreliable actions in *observable* grid world





States $s \in \mathcal{S}$, actions $a \in \mathcal{A}$

(Transition) Model $T(s, a, s') \equiv p(s'|s, a)$ = probability that a in s leads to s'

Notes

Beginning of semester – search – *deterministic* and (fully) *observable* environment
Now:

- Observable – we keep for now – agent knows where it is.
- Deterministic – We introduce "imperfect" agent that does not always obey the command – *stochastic action outcomes*.

There is a treasure (desired goal/end state) but there is also some danger (unwanted goal/end state).
The danger state: think about a mountainous area with safer but longer and shorter but more dangerous paths – a dangerous node may represent a chasm.

Notation note: caligraphic letters like $\mathcal{S}, \mathcal{A}$ will denote the set(s) of all states/actions.

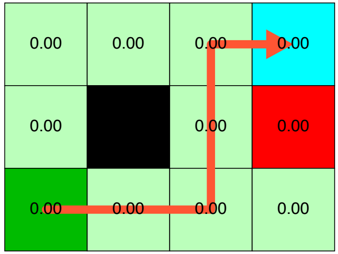
Unreliable (results of) actions

Notes

Actions: go over a glacier bridge or around?

Plan? Policy

- In deterministic world: Plan – sequence of actions from Start to Goal.
- MDPs, we need a policy $\pi : \mathcal{S} \to \mathcal{A}$.
- An action for each possible state. Why *each*?
- What is the *best* policy?



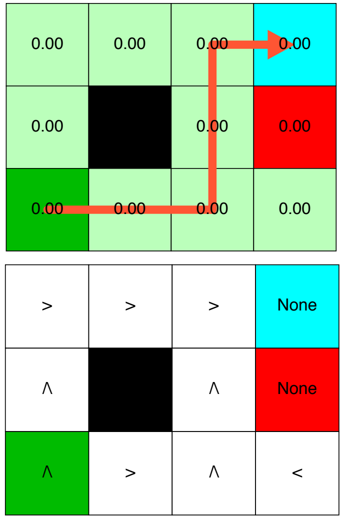
Notes

Ignore the 0.00 numbers in the cells.
Unlike in deterministic environment (also search problems), with stochastic action outcomes, we can end up in any state. Thus, in any state, the robot/agent has to know what to do.

What is the best policy? We will come to that in a minute, ...

Plan? Policy

- In deterministic world: Plan – sequence of actions from Start to Goal.
- MDPs, we need a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$.
- An action for each possible state. Why *each*?
- What is the *best* policy?



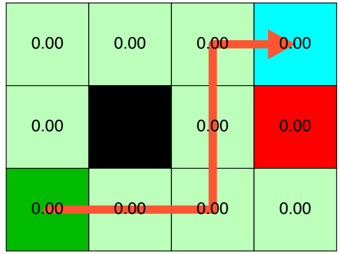
Notes

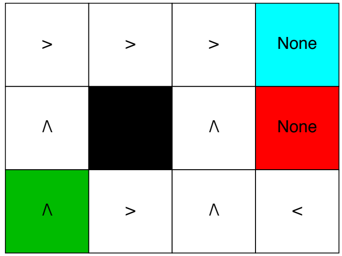
Ignore the 0.00 numbers in the cells.
Unlike in deterministic environment (also search problems), with stochastic action outcomes, we can end up in any state. Thus, in any state, the robot/agent has to know what to do.

What is the best policy? We will come to that in a minute, ...

Plan? Policy

- In deterministic world: Plan – sequence of actions from Start to Goal.
- MDPs, we need a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$.
- An action for each possible state. Why *each*?
- What is the *best* policy?





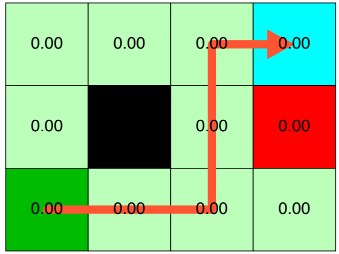
Notes

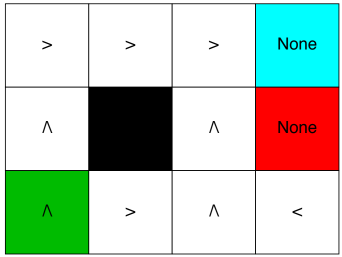
Ignore the 0.00 numbers in the cells.
Unlike in deterministic environment (also search problems), with stochastic action outcomes, we can end up in any state. Thus, in any state, the robot/agent has to know what to do.

What is the best policy? We will come to that in a minute, ...

Plan? Policy

- In deterministic world: Plan – sequence of actions from Start to Goal.
- MDPs, we need a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$.
- An action for each possible state. Why *each*?
- What is the *best* policy?





Notes

Ignore the 0.00 numbers in the cells.
Unlike in deterministic environment (also search problems), with stochastic action outcomes, we can end up in any state. Thus, in any state, the robot/agent has to know what to do.

What is the best policy? We will come to that in a minute, ...

Rewards

-0.04	-0.04	-0.04	1.00
-0.04		-0.04	-1.00
-0.04	-0.04	-0.04	-0.04

Reward : Robot/Agent takes an action a and it is **immediately** rewarded.

Reward function $r(s)$ (or $r(s,a)$, $r(s,a,s')$)

$$= \begin{cases} -0.04 & \text{(small penalty) for nonterminal states} \\ \pm 1 & \text{for terminal states} \end{cases}$$

Notes

What do the rewards express? Reward to an agent to be/dwell in that state? Obviously we want the robot to go to the goal and do not stay too long in the maze. The negative reward of –0.04 gives the agent an incentive to reach the goal state quickly, so our environment is a *stochastic generalization of the search problems*.

Thinking about Reward: Robot/Agent takes an action a and it is immediately rewarded for this. The reward may depend on

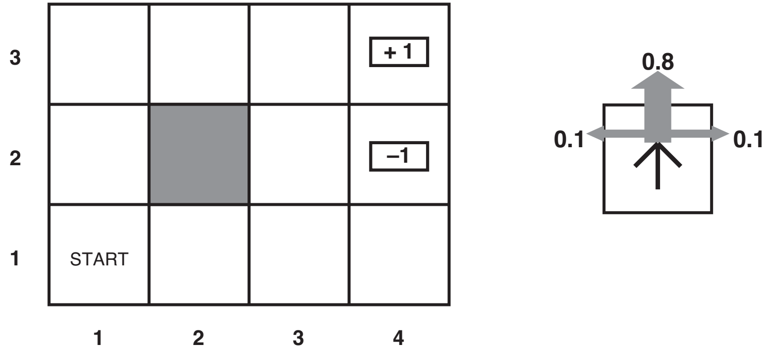
- current state s,
- the action taken a
- the next state s' - result of the action, *and* robot receives reward r for all this.

Rewards for terminal states can be understood as follows: there is only one action: $a = \text{exit}$. We will come to this soon.

The **reward function** is a property of (is related to) the problem.

Notation remark: lowercase letters will be used for functions like $p, r, v, f, \ldots$

Markov Decision Processes (MDPs)



(a) (b)

States $s \in \mathcal{S}$, actions $a \in \mathcal{A}$

Model $T(s, a, s') \equiv p(s'|s, a)$ = probability that a in s leads to s'

Reward function $r(s)$ (or $r(s, a)$, $r(s, a, s')$)

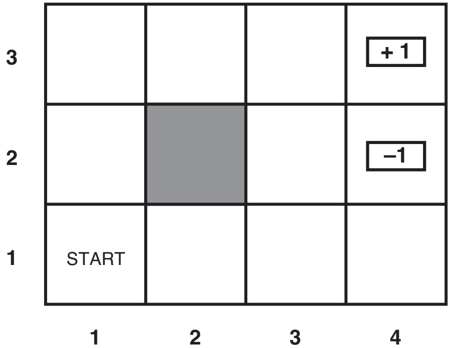
$$= \begin{cases} -0.04 & \text{(small penalty) for nonterminal states} \\ \pm 1 & \text{for terminal states} \end{cases}$$

Notes

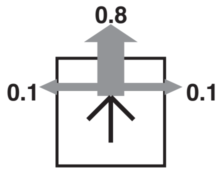
States: x, y or r, c coordinates of the position

Actions: UP, LEFT, RIGHT, DOWN or N, W, E, S

Markov Decision Processes (MDPs)



(a)



(b)

States $s \in \mathcal{S}$, actions $a \in \mathcal{A}$

Model $T(s, a, s') \equiv p(s'|s, a)$ = probability that a in s leads to s'

Reward function $r(s)$ (or $r(s, a)$, $r(s, a, s')$)

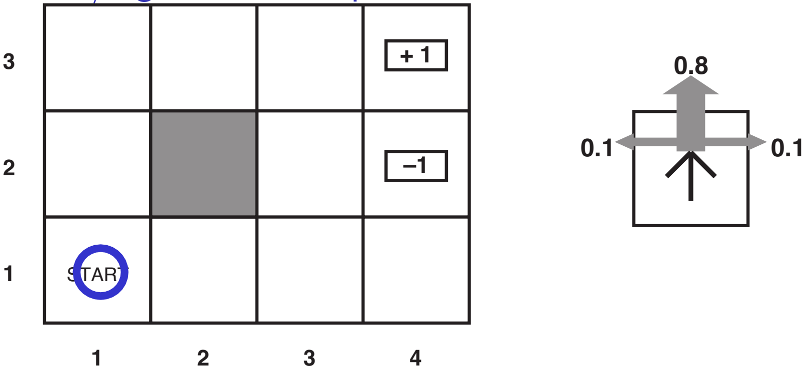
$$= \begin{cases} -0.04 & \text{(small penalty) for nonterminal states} \\ \pm 1 & \text{for terminal states} \end{cases}$$

Notes

States: x, y or r, c coordinates of the position

Actions: UP, LEFT, RIGHT, DOWN or N, W, E, S

Robot/Agent walk – Episode



(a) (b)

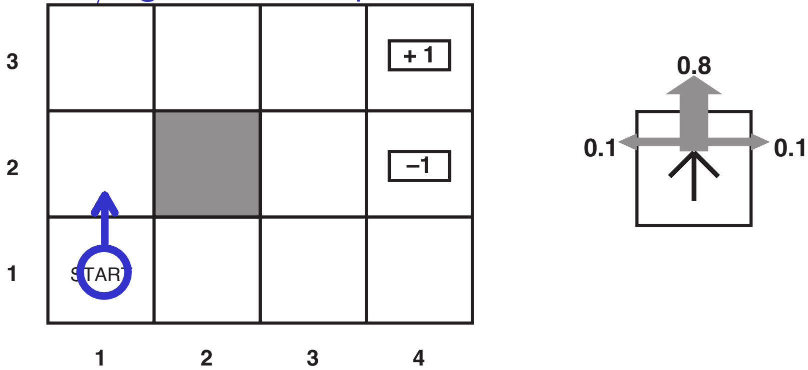
$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2 \ldots$$

Episode : one walk from S_0 to terminal.

Notes

At the START, agents decides UP/NORTH but ends in a state right to START.

Robot/Agent walk – Episode



(a) (b)

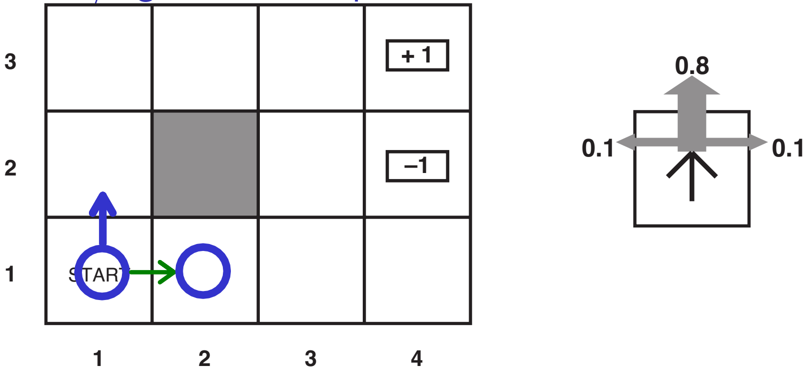
$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2 \ldots$$

Episode : one walk from S_0 to terminal.

Notes

At the START, agents decides UP/NORTH but ends in a state right to START.

Robot/Agent walk – Episode



(a) (b)

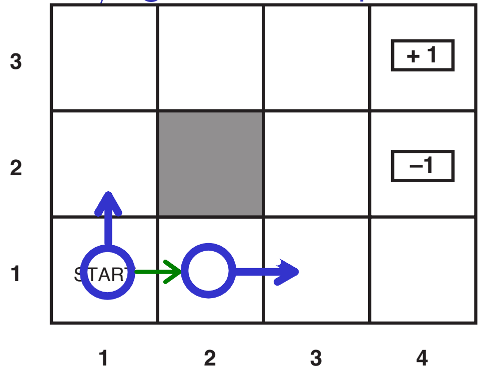
$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2 \ldots$$

Episode : one walk from S_0 to terminal.

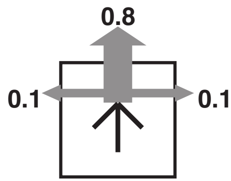
Notes

At the START, agents decides UP/NORTH but ends in a state right to START.

Robot/Agent walk – Episode



(a)



(b)

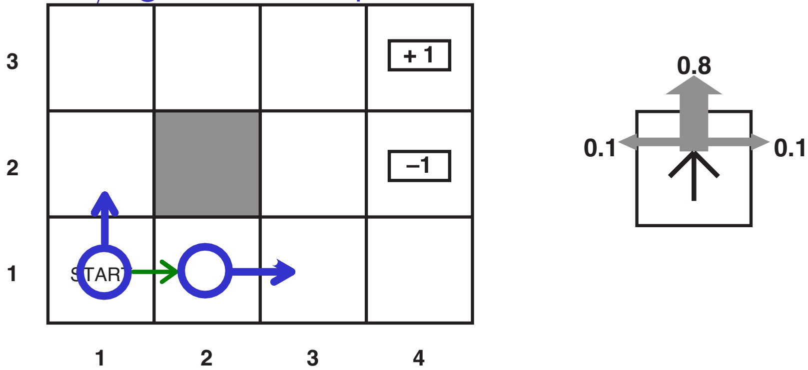
$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2 \ldots$$

Episode : one walk from S_0 to terminal.

Notes

At the START, agents decides UP/NORTH but ends in a state right to START.

Robot/Agent walk – Episode



(a) (b)

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2 \dots$$

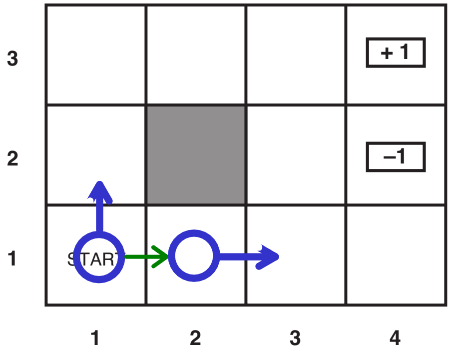
Episode : one walk from S_0 to terminal.

Notes

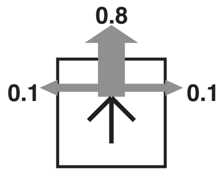
At the START, agents decides UP/NORTH but ends in a state right to START.

Markovian property

- Given the present state, the future and the past are independent.
- MDP: Markov means action depends only on the current state.
- In search: successor function (transition model) depends on the current state only.



(a)



(b)

Notes

- Properties are somewhat obvious, reasonable.
- However, you may break it if wrongly formalized.
- Always check before you go (do the calculations).
- It is a property of the state not the decision process.

Desired robot/agent behavior specified through rewards

- Before: shortest/cheapest path
- Solution found by search.
- Environment/problem is defined through the reward function.
- Optimal policy is to be computed/learned.

Notes

Desired robot/agent behavior specified through rewards

- Before: shortest/cheapest path
- Solution found by search.
- Environment/problem is defined through the reward function.
- Optimal policy is to be computed/learned.

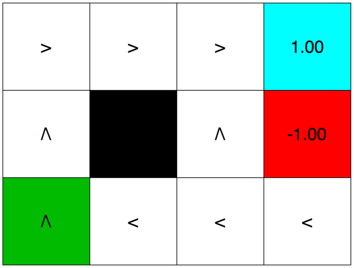
Notes

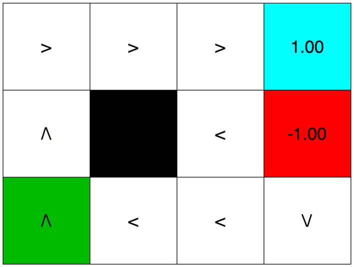
Desired robot/agent behavior specified through rewards

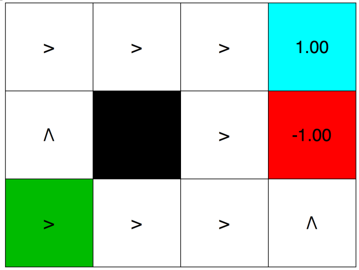
- Before: shortest/cheapest path
- Solution found by search.
- Environment/problem is defined through the reward function.
- Optimal policy is to be computed/learned.

We come back to this in more detail when discussing RL.

Notes







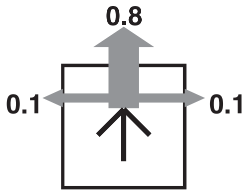
A	B	C
$r(s) \in \{-2, 1, -1\}$ a	$r(s) \in \{-0.04, 1, -1\}$ b	$r(s) \in \{-0.01, 1, -1\}$ c

A: A-a, B-b, C-c

B: A-b, B-a, C-c

C: A-b, B-c, C-a

D: A-c, B-a, C-b



Notes

Notation: $reward(state) \in \{$living reward/penalty, reward in blue state, reward in red state$\}$

- $r(s) \in \{-0.04, 1, -1\}$
- $r(s) \in \{-2, 1, -1\}$ – environment very hostile (think about burning floor) – heading for nearest exit even if it's with negative reward
- $r(s) \in \{-0.01, 1, -1\}$ – environment very mildly unpleasant – conservative policy (banging head against the wall to avoid negative terminal state at all cost)

Quiz assignment: Match the environments (A, B, C) and the policies (arrows in every state) with the corresponding reward functions (a,b,c).
(Use common sense.)

Utilities of sequences; what is a better walk

- State reward at time/step t, R_t.
- State at time t, S_t. State sequence $[S_0, S_1, S_2, \ldots,]$

Typically, consider stationary preferences on reward sequences:

$$[R, R_1, R_2, R_3, \ldots] \succ [R, R_1', R_2', R_3', \ldots] \Leftrightarrow [R_1, R_2, R_3, \ldots] \succ [R_1', R_2', R_3', \ldots]$$

If stationary preferences :
Utility (h-history)
$U_h([S_0, S_1, S_2, \ldots,]) = R_1 + R_2 + R_3 + \cdots$

If the horizon is finite - limited number of steps - preferences are nonstationary (depends on how many steps left).

Notes

We consider discrete time t. S_t, R_t notation emphasises the time sequence - not a sequence of particular states. The reward is for an action (transition)

Finite vs non-finite horizon. Think about the simple 3×4 grid from the last slides and having limited budget of 3,4,5 steps.

Utilities of sequences; what is a better walk

- State reward at time/step t, R_t.
- State at time t, S_t. State sequence $[S_0, S_1, S_2, \ldots,]$

Typically, consider stationary preferences on reward sequences:

$$[R, R_1, R_2, R_3, \ldots] \succ [R, R_1', R_2', R_3', \ldots] \Leftrightarrow [R_1, R_2, R_3, \ldots] \succ [R_1', R_2', R_3', \ldots]$$

If stationary preferences :
Utility (h-history)
$U_h([S_0, S_1, S_2, \ldots,]) = R_1 + R_2 + R_3 + \cdots$

If the horizon is finite - limited number of steps - preferences are nonstationary (depends on how many steps left).

Notes

We consider discrete time t. S_t, R_t notation emphasises the time sequence - not a sequence of particular states. The reward is for an action (transition)

Finite vs non-finite horizon. Think about the simple 3×4 grid from the last slides and having limited budget of 3,4,5 steps.

Utilities of sequences; what is a better walk

- State reward at time/step t, R_t.
- State at time t, S_t. State sequence $[S_0, S_1, S_2, \ldots,]$

Typically, consider stationary preferences on reward sequences:

$$[R, R_1, R_2, R_3, \ldots] \succ [R, R'_1, R'_2, R'_3, \ldots] \Leftrightarrow [R_1, R_2, R_3, \ldots] \succ [R'_1, R'_2, R'_3, \ldots]$$

If stationary preferences :
Utility (h-history)
$U_h([S_0, S_1, S_2, \ldots,]) = R_1 + R_2 + R_3 + \cdots$

If the horizon is finite - limited number of steps - preferences are nonstationary (depends on how many steps left).

Notes

We consider discrete time t. S_t, R_t notation emphasises the time sequence - not a sequence of particular states. The reward is for an action (transition)

Finite vs non-finite horizon. Think about the simple 3×4 grid from the last slides and having limited budget of 3,4,5 steps.

Utilities of sequences; what is a better walk

- State reward at time/step t, R_t.
- State at time t, S_t. State sequence $[S_0, S_1, S_2, \dots,]$

Typically, consider stationary preferences on reward sequences:

$$[R, R_1, R_2, R_3, \dots] \succ [R, R'_1, R'_2, R'_3, \dots] \Leftrightarrow [R_1, R_2, R_3, \dots] \succ [R'_1, R'_2, R'_3, \dots]$$

If stationary preferences :
Utility (h-history)
$U_h([S_0, S_1, S_2, \dots,]) = R_1 + R_2 + R_3 + \cdots$

If the horizon is finite - limited number of steps - preferences are nonstationary (depends on how many steps left).

Notes

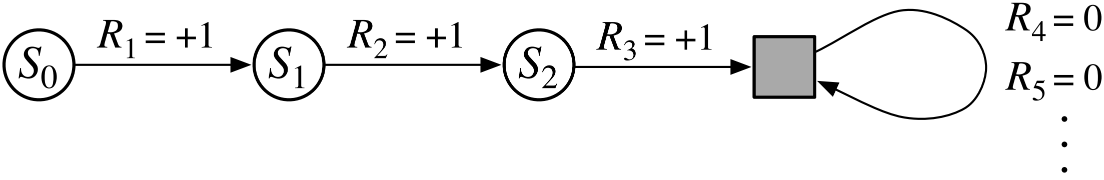
We consider discrete time t. S_t, R_t notation emphasises the time sequence - not a sequence of particular states. The reward is for an action (transition)

Finite vs non-finite horizon. Think about the simple 3×4 grid from the last slides and having limited budget of 3,4,5 steps.

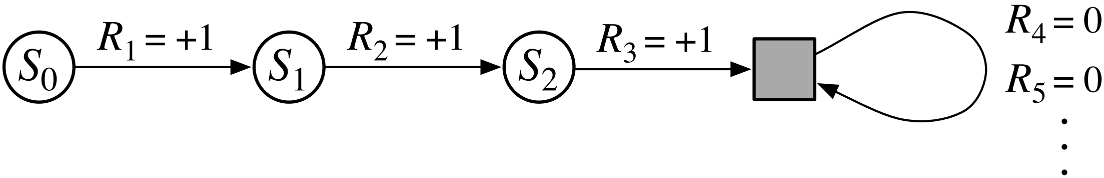
Finite walk – Episode – and its Return (by introducing Terminal state)

- Executing policy - sequence of states and **rewards**.
- Episode starts at t, ends at T (ending in a terminal state).
- Return (Utility) of the episode (policy execution)

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$



Notes



Solid square – *absorbing state* – end of an episode.
(transitions only to itself and generates only rewards of zero)
Allows to unify two formulations of return (G_t) as a finite and infinite sum of rewards.

Horizon too far, infinite – Discount rewards

Problem: Infinite lifetime ⇒ additive utilities are infinite.

- Finite horizon: termination at a fixed time ⇒ nonstationary policy, $\pi(s)$ depends on the time left.
- Absorbing (terminal) state. (sooner or later walk ends here)
- Discounted return , $\gamma < 1, R_t \leq R_{\max}$

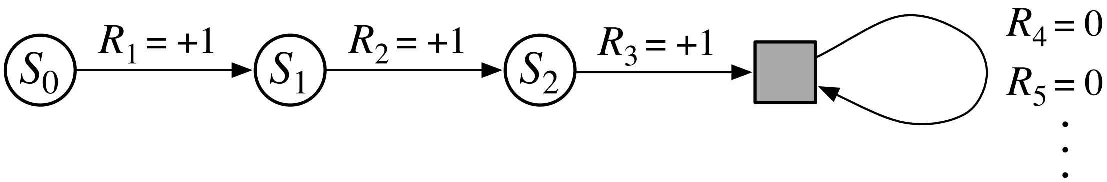
$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \leq \frac{R_{\max}}{1-\gamma}$$

Returns are successive steps related to each other

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Notes

Discounting is quite natural choice. Think about your preferences/rewards. Go to pub with friends tonight, studying (for the far future reward of getting A in the course)?



Returns are successive steps related to each other

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Solid square – *absorbing state* – end of an episode.
(transitions only to itself and generates only rewards of zero)

Allows to unify two formulations of return (G_t) as a finite and infinite sum of rewards.

Horizon too far, infinite – Discount rewards

Problem: Infinite lifetime ⇒ additive utilities are infinite.

- Finite horizon: termination at a fixed time ⇒ nonstationary policy, $\pi(s)$ depends on the time left.
- Absorbing (terminal) state. (sooner or later walk ends here)
- Discounted return , $\gamma < 1, R_t \leq R_{\max}$

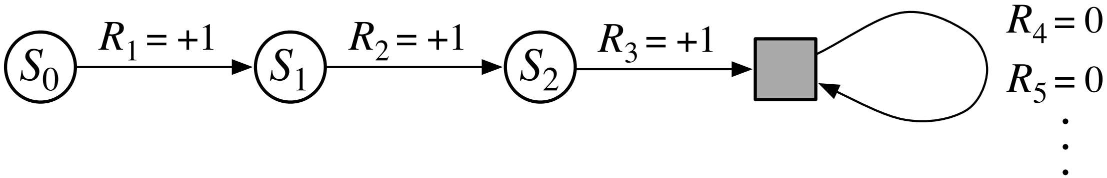
$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \leq \frac{R_{\max}}{1-\gamma}$$

Returns are successive steps related to each other

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Notes

Discounting is quite natural choice. Think about your preferences/rewards. Go to pub with friends tonight, studying (for the far future reward of getting A in the course)?



Returns are successive steps related to each other

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Solid square – *absorbing state* – end of an episode.
(transitions only to itself and generates only rewards of zero)

Allows to unify two formulations of return (G_t) as a finite and infinite sum of rewards.

Horizon too far, infinite – Discount rewards

Problem: Infinite lifetime ⇒ additive utilities are infinite.

- Finite horizon: termination at a fixed time ⇒ nonstationary policy, $\pi(s)$ depends on the time left.
- Absorbing (terminal) state. (sooner or later walk ends here)
- Discounted return , $\gamma < 1, R_t \leq R_{\max}$

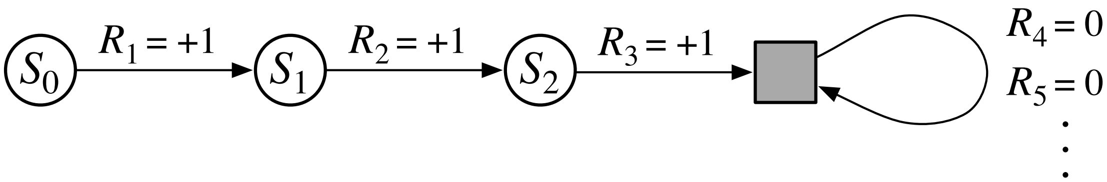
$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \leq \frac{R_{\max}}{1-\gamma}$$

Returns are successive steps related to each other

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Notes

Discounting is quite natural choice. Think about your preferences/rewards. Go to pub with friends tonight, studying (for the far future reward of getting A in the course)?



Returns are successive steps related to each other

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Solid square – *absorbing state* – end of an episode.
(transitions only to itself and generates only rewards of zero)

Allows to unify two formulations of return (G_t) as a finite and infinite sum of rewards.

Horizon too far, infinite – Discount rewards

Problem: Infinite lifetime ⇒ additive utilities are infinite.

- Finite horizon: termination at a fixed time ⇒ nonstationary policy, $\pi(s)$ depends on the time left.
- Absorbing (terminal) state. (sooner or later walk ends here)
- Discounted return , $\gamma < 1, R_t \leq R_{\max}$

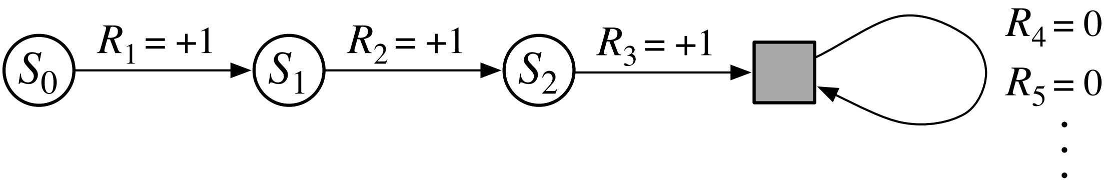
$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \leq \frac{R_{\max}}{1-\gamma}$$

Returns are successive steps related to each other

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Notes

Discounting is quite natural choice. Think about your preferences/rewards. Go to pub with friends tonight, studying (for the far future reward of getting A in the course)?



Returns are successive steps related to each other

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Solid square – *absorbing state* – end of an episode.
(transitions only to itself and generates only rewards of zero)

Allows to unify two formulations of return (G_t) as a finite and infinite sum of rewards.

Horizon too far, infinite – Discount rewards

Problem: Infinite lifetime ⇒ additive utilities are infinite.

- Finite horizon: termination at a fixed time ⇒ nonstationary policy, $\pi(s)$ depends on the time left.
- Absorbing (terminal) state. (sooner or later walk ends here)
- Discounted return , $\gamma < 1, R_t \leq R_{\text{max}}$

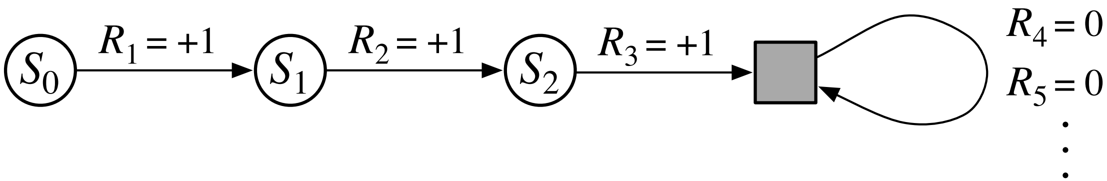
$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \leq \frac{R_{\text{max}}}{1-\gamma}$$

Returns are successive steps related to each other

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Notes

Discounting is quite natural choice. Think about your preferences/rewards. Go to pub with friends tonight, studying (for the far future reward of getting A in the course)?



Returns are successive steps related to each other

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Solid square – *absorbing state* – end of an episode.
(transitions only to itself and generates only rewards of zero)

Allows to unify two formulations of return (G_t) as a finite and infinite sum of rewards.

Horizon too far, infinite – Discount rewards

Problem: Infinite lifetime ⇒ additive utilities are infinite.

- Finite horizon: termination at a fixed time ⇒ nonstationary policy, $\pi(s)$ depends on the time left.
- Absorbing (terminal) state. (sooner or later walk ends here)
- Discounted return , $\gamma < 1, R_t \leq R_{\max}$

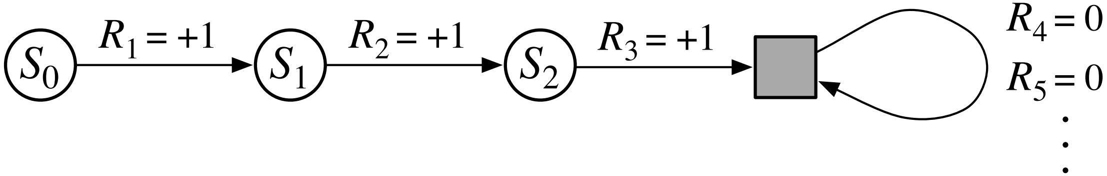
$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \leq \frac{R_{\max}}{1-\gamma}$$

Returns are successive steps related to each other

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma (R_{t+2} + \gamma^1 R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Notes

Discounting is quite natural choice. Think about your preferences/rewards. Go to pub with friends tonight, studying (for the far future reward of getting A in the course)?



Returns are successive steps related to each other

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma (R_{t+2} + \gamma^1 R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Solid square – *absorbing state* – end of an episode.
(transitions only to itself and generates only rewards of zero)

Allows to unify two formulations of return (G_t) as a finite and infinite sum of rewards.

Horizon too far, infinite – Discount rewards

Problem: Infinite lifetime ⇒ additive utilities are infinite.

- Finite horizon: termination at a fixed time ⇒ nonstationary policy, $\pi(s)$ depends on the time left.
- Absorbing (terminal) state. (sooner or later walk ends here)
- Discounted return , $\gamma < 1, R_t \leq R_{\max}$

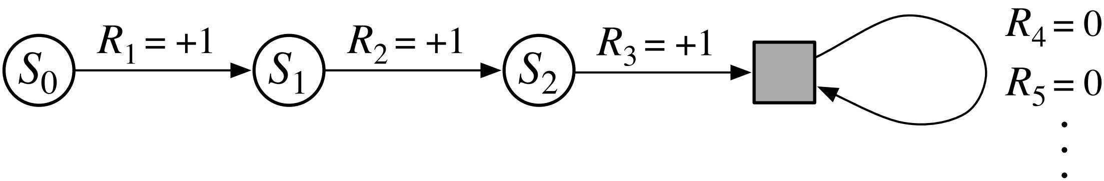
$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \leq \frac{R_{\max}}{1-\gamma}$$

Returns are successive steps related to each other

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Notes

Discounting is quite natural choice. Think about your preferences/rewards. Go to pub with friends tonight, studying (for the far future reward of getting A in the course)?



Returns are successive steps related to each other

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma^1 R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Solid square – *absorbing state* – end of an episode.
(transitions only to itself and generates only rewards of zero)

Allows to unify two formulations of return (G_t) as a finite and infinite sum of rewards.

MDPs recap

Markov decision processes (MDPs):

- Set of states $\mathcal{S}$
- Set of actions $\mathcal{A}$
- Transitions $p(s'|s,a)$ or $T(s,a,s')$
- Reward function $r(s,a,s')$; and discount γ
- Alternative to last two: $p(s',r|s,a)$.

MDP quantities:

- (deterministic) Policy $\pi(s)$ – choice of action for each state
- Return (Utility) of an episode (sequence) – sum of (discounted) rewards.

Notes

Think about what is given and what we want to compute.

MDPs recap

Markov decision processes (MDPs):

- Set of states $\mathcal{S}$
- Set of actions $\mathcal{A}$
- Transitions $p(s'|s,a)$ or $T(s,a,s')$
- Reward function $r(s,a,s')$; and discount γ
- Alternative to last two: $p(s',r|s,a)$.

MDP quantities:

- (deterministic) Policy $\pi(s)$ – choice of action for each state
- Return (Utility) of an episode (sequence) – sum of (discounted) rewards.

Notes

Think about what is given and what we want to compute.

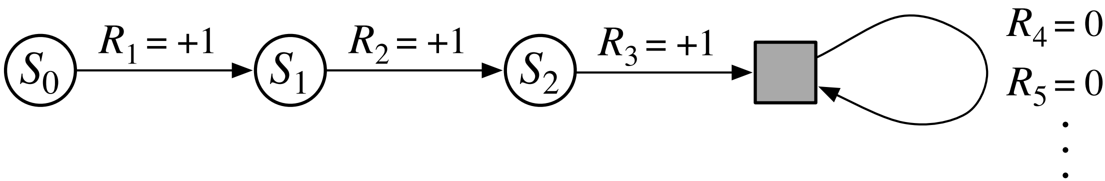
Expected Return of a policy π

- Executing policy $\pi \rightarrow$ sequence of states (and rewards).
- Utility of a state sequence.
- But actions are unreliable – environment is stochastic.
- Expected return of a policy π.

Starting at time t, i.e. S_t,

$$U^{\pi}(S_t) \doteq \mathrm{E}^{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right]$$

Notes



Contrast *return* of a particlar episode vs. *value* – expected utility of a state sequence in general – *expected return*. Expected value can be also computed by running (executing) the policy many times and then computing average of the returns – Monte Carlo simulation methods.

It is worth to mention that value function and action-value function are both tightly connected to a particular policy π.

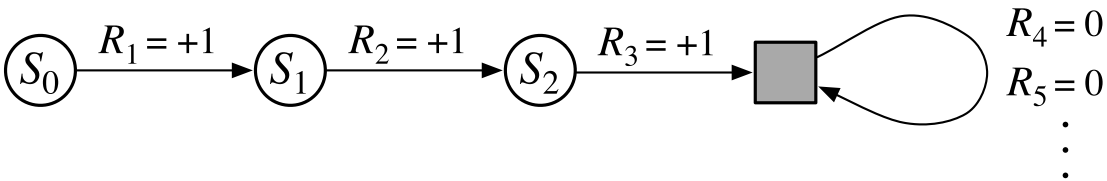
Expected Return of a policy π

- Executing policy $\pi \rightarrow$ sequence of states (and rewards).
- Utility of a state sequence.
- But actions are unreliable - environment is stochastic.
- Expected return of a policy π.

Starting at time t, i.e. S_t,

$$U^\pi(S_t) \doteq \mathrm{E}^\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right]$$

Notes



Contrast *return* of a particlar episode vs. *value* – expected utility of a state sequence in general – *expected return*. Expected value can be also computed by running (executing) the policy many times and then computing average of the returns – Monte Carlo simulation methods.

It is worth to mention that value function and action-value function are both tightly connected to a particular policy π.

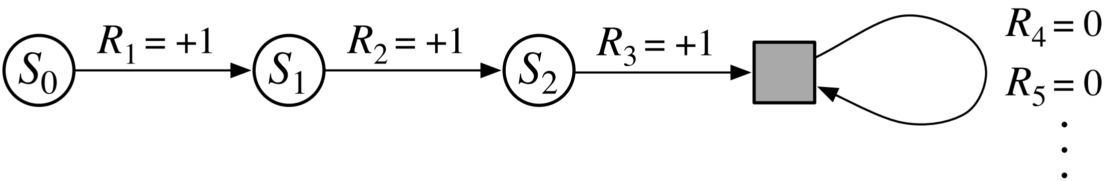
Expected Return of a policy π

- Executing policy $\pi \rightarrow$ sequence of states (and rewards).
- Utility of a state sequence.
- But actions are unreliable - environment is stochastic.
- Expected return of a policy π.

Starting at time t, i.e. S_t,

$$U^\pi(S_t) \doteq \mathsf{E}^\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \right]$$

Notes



Contrast *return* of a particlar episode vs. *value* – expected utility of a state sequence in general – *expected return*. Expected value can be also computed by running (executing) the policy many times and then computing average of the returns – Monte Carlo simulation methods.

It is worth to mention that value function and action-value function are both tightly connected to a particular policy π.

Value functions given policy π

Expected return from that state (state, action)

Value function

$$v^\pi(s) \doteq \mathrm{E}^\pi\left[G_t \mid S_t = s\right] = \mathrm{E}^\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

Action-value function (q-function)

$$q^\pi(s, a) \doteq \mathrm{E}^\pi\left[G_t \mid S_t = s, A_t = a\right] = \mathrm{E}^\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

Notes

Optimal policy π^*, and optimal value $v^*(s)$

$v^*(s) =$ expected (discounted) sum of rewards (until termination) assuming *optimal* actions.

Notes

Showing cases for

- $r(s) = \{-0.04, 1, -1\}$, $\gamma = 0.999999$, $\epsilon = 0.03$
- $r(s) = \{-0.01, 1, -1\}$, $\gamma = 0.999999$, $\epsilon = 0.03$

What is the difference in the optimal policy? Try to explain why it happened.
We still do not know *how* to compute the optimality, ... right?

Optimal policy π^*, and optimal value $v^*(s)$

$v^*(s)$ = expected (discounted) sum of rewards (until termination) assuming *optimal* actions.

Example 1, Robot *deterministic*: $r(s) = \{-0.04, 1, -1\}$, $\gamma = 0.999999$, $\epsilon = 0.03$

	0	1	2	3	
0	0.88	0.92	0.96	1.00	0
1	0.84		0.92	-1.00	1
2	0.80	0.84	0.88	0.84	2
	0	1	2	3	

	0	1	2	3	
0	>	>	>	None	0
1	/\		/\	None	1
2	/\	>	/\	<	2
	0	1	2	3	

Notes

Showing cases for

- $r(s) = \{-0.04, 1, -1\}$, $\gamma = 0.999999$, $\epsilon = 0.03$
- $r(s) = \{-0.01, 1, -1\}$, $\gamma = 0.999999$, $\epsilon = 0.03$

What is the difference in the optimal policy? Try to explain why it happened.
We still do not know *how* to compute the optimality, ... right?

Optimal policy π^*, and optimal value $v^*(s)$

$v^*(s)$ = expected (discounted) sum of rewards (until termination) assuming *optimal* actions.

Example 2, Robot *non-deterministic*: $r(s) = \{-0.04, 1, -1\}$, $\gamma = 0.999999$, $\epsilon = 0.03$

	0	1	2	3
0	0.81	0.87	0.92	1.00
1	0.76		0.66	-1.00
2	0.71	0.66	0.61	0.39

	0	1	2	3
0	>	>	>	None
1	/\		/\	None
2	/\	<	<	<

Notes

Showing cases for

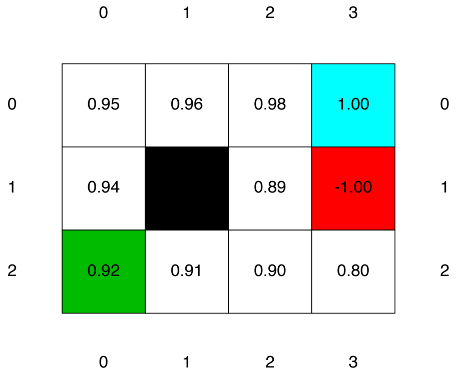
- $r(s) = \{-0.04, 1, -1\}$, $\gamma = 0.999999$, $\epsilon = 0.03$
- $r(s) = \{-0.01, 1, -1\}$, $\gamma = 0.999999$, $\epsilon = 0.03$

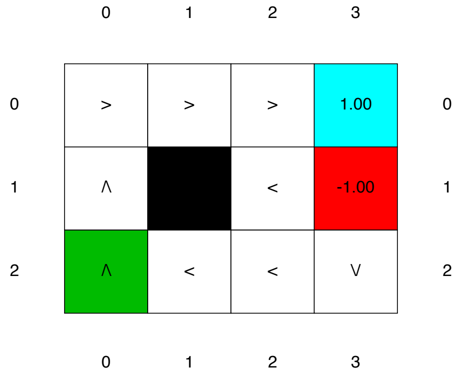
What is the difference in the optimal policy? Try to explain why it happened.
We still do not know *how* to compute the optimality, ... right?

Optimal policy π^*, and optimal value $v^*(s)$

$v^*(s)$ = expected (discounted) sum of rewards (until termination) assuming *optimal* actions.

Example 3, Robot *non-deterministic*: $r(s) = \{-0.01, 1, -1\}$, $\gamma = 0.999999$, $\epsilon = 0.03$





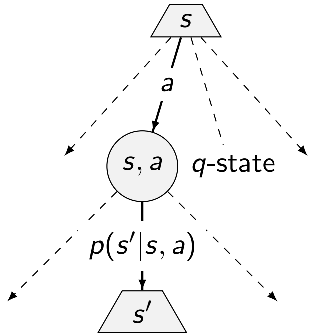
Notes

Showing cases for

- $r(s) = \{-0.04, 1, -1\}$, $\gamma = 0.999999$, $\epsilon = 0.03$
- $r(s) = \{-0.01, 1, -1\}$, $\gamma = 0.999999$, $\epsilon = 0.03$

What is the difference in the optimal policy? Try to explain why it happened.
We still do not know *how* to compute the optimality, ... right?

MDP search tree



Notes

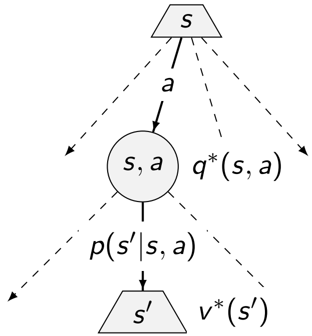
$$
\begin{aligned}
v^{\pi}(s) &= \mathrm{E}^{\pi}\left[G_t \mid S_t = s\right] \\
&= \mathrm{E}^{\pi}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \sum_{s'} p(s'|a,s)\Big[r(s,a,s') + \gamma \mathrm{E}^{\pi}\left[G_{t+1} \mid S_{t+1} = s'\right]\Big]
\end{aligned}
$$

Recall Expectimax algorithm from the last lecture.
How to compute $V(s)$? Well, we could solve the expectimax search, but it grows quickly. We can see $R(s)$ as the price for leaving the state s just anyhow.

MDP search tree

The value of a q-state (s, a):

$$q^*(s,a) = \sum_{s'} p(s'|a,s)\left[r(s,a,s') + \gamma\, v^*(s')\right]$$



Notes

$$\begin{aligned} v^\pi(s) &= \mathrm{E}^\pi\left[G_t \mid S_t = s\right] \\ &= \mathrm{E}^\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\ &= \sum_{s'} p(s'|a,s)\Big[r(s,a,s') + \gamma \mathrm{E}^\pi\left[G_{t+1} \mid S_{t+1} = s'\right]\Big] \end{aligned}$$

Recall Expectimax algorithm from the last lecture.
How to compute $V(s)$? Well, we could solve the expectimax search, but it grows quickly. We can see $R(s)$ as the price for leaving the state s just anyhow.

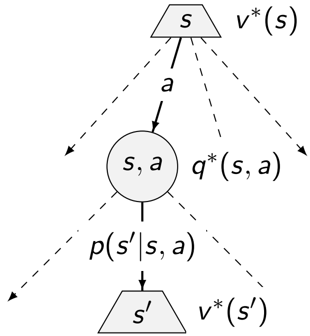
MDP search tree

The value of a q-state (s, a):

$$q^*(s,a) = \sum_{s'} p(s'|a,s)\left[r(s,a,s') + \gamma\, v^*(s')\right]$$

The value of a state s:

$$v^*(s) = \max_a q^*(s,a)$$



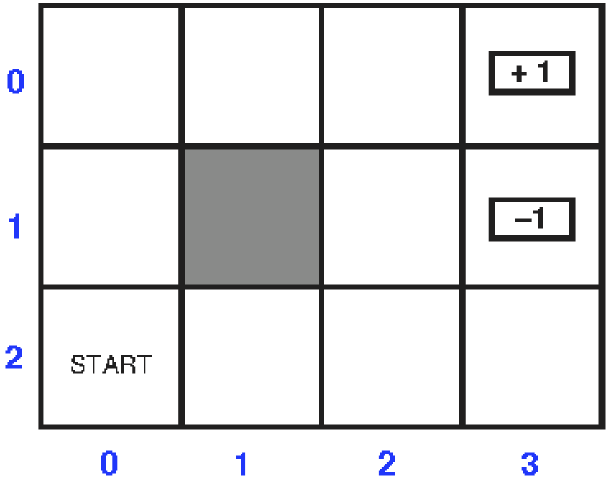
Notes

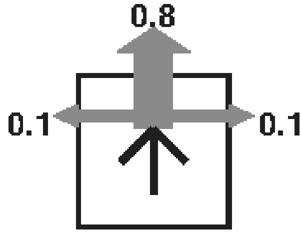
$$\begin{aligned} v^\pi(s) &= \mathrm{E}^\pi\left[G_t \mid S_t = s\right] \\ &= \mathrm{E}^\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\ &= \sum_{s'} p(s'|a,s)\Big[r(s,a,s') + \gamma \mathrm{E}^\pi\left[G_{t+1} \mid S_{t+1} = s'\right]\Big] \end{aligned}$$

Recall Expectimax algorithm from the last lecture.
How to compute $V(s)$? Well, we could solve the expectimax search, but it grows quickly. We can see $R(s)$ as the price for leaving the state s just anyhow.

Bellman (optimality) equation

$$v^*(s) = \max_{a \in A(s)} \sum_{s'} p(s'|a,s) \left[r(s,a,s') + \gamma v^*(s') \right]$$





Notes

v computation on the table - one row for each action. We got n equations for n unknown - n states. But max is a non-linear operator!

Value iteration – turn Bellman equation into Bellman update

$$v^*(s) = \max_{a \in A(s)} \sum_{s'} p(s'|a,s)\left[r(s,a,s') + \gamma v^*(s')\right]$$

- Start with arbitrary $V_0(s)$ (except for terminals)
- Compute Bellman update (one ply of expectimax from each state)

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} p(s'|s,a) V_k(s')$$

- Repeat until convergence

The idea: Bellman update makes local consistency with the Bellmann equation. Everywhere locally consistent $\Rightarrow$ globally optimal.

Value iteration algorithm is an example of Dynamic Programming method.

Notes

What is the complexity of each iteration?

Value iteration – turn Bellman equation into Bellman update

$$v^*(s) = \max_{a \in A(s)} \sum_{s'} p(s'|a,s)\left[r(s,a,s') + \gamma v^*(s')\right]$$

- Start with arbitrary $V_0(s)$ (except for terminals)
- Compute Bellman update (one ply of expectimax from each state)

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} p(s'|s,a) V_k(s')$$

- Repeat until convergence

The idea: Bellman update makes local consistency with the Bellmann equation. Everywhere locally consistent $\Rightarrow$ globally optimal.

Value iteration algorithm is an example of Dynamic Programming method.

Notes

What is the complexity of each iteration?

Value iteration – turn Bellman equation into Bellman update

$$v^*(s) = \max_{a \in A(s)} \sum_{s'} p(s'|a,s) \left[r(s,a,s') + \gamma v^*(s') \right]$$

- Start with arbitrary $V_0(s)$ (except for terminals)
- Compute Bellman update (one ply of expectimax from each state)

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} p(s'|s,a) V_k(s')$$

- Repeat until convergence

The idea: Bellman update makes local consistency with the Bellmann equation. Everywhere locally consistent $\Rightarrow$ globally optimal.

Value iteration algorithm is an example of Dynamic Programming method.

Notes

What is the complexity of each iteration?

Value iteration – turn Bellman equation into Bellman update

$$v^*(s) = \max_{a \in A(s)} \sum_{s'} p(s'|a,s)\left[r(s,a,s') + \gamma v^*(s')\right]$$

- Start with arbitrary $V_0(s)$ (except for terminals)
- Compute Bellman update (one ply of expectimax from each state)

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} p(s'|s,a) V_k(s')$$

- Repeat until convergence

The idea: Bellman update makes local consistency with the Bellmann equation. Everywhere locally consistent $\Rightarrow$ globally optimal.

Value iteration algorithm is an example of Dynamic Programming method.

Notes

What is the complexity of each iteration?

Value iteration - Complexity of one estimation sweep

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} p(s'|s,a) V_k(s')$$

A: $O(AS)$

B: $O(S^2)$

C: $O(AS^2)$

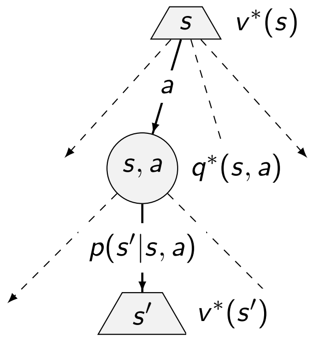
D: $O(A^2 S^2)$

Notes

- The sweep goes through all the states S.
- From each state, we need evaluate all actions A.
- Each action may, in principle, land in any other state S.

Value iteration (dynamic programming) vs. direct search

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} p(s'|s,a) V_k(s')$$



Notes

Value iteration demo

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) V_k(s')$$

	0	1	2	3	
0	0.81	0.87	0.92	1.00	0
1	0.76		0.66	-1.00	1
2	0.71	0.66	0.61	0.39	2
	0	1	2	3	

Notes

Run `mdp_agents.py` and try to compute next state value in advance. Remind the $R(s) = -0.04$ and $\gamma = 1$ in order to simplify computation. Then discuss the course of the Values.

Convergence

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) V_k(s')$$

$$\gamma < 1$$

$$-R_{\max} \leq R(s) \leq R_{\max}$$

Max norm:

$$||V|| = \max_{s} |V(s)|$$

$$U([s_0, s_1, s_2, \ldots, s_\infty]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \frac{R_{\max}}{1-\gamma}$$

Notes

Keep in mind that V is a vector of all state values. If the problem has 12 states (3×4 grid) then it is a 12-dim vector.

Convergence

$$V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) V_k(s')$$

$$\gamma < 1$$

$$-R_{\max} \leq R(s) \leq R_{\max}$$

Max norm:

$$\|V\| = \max_s |V(s)|$$

$$U([s_0, s_1, s_2, \ldots, s_\infty]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \frac{R_{\max}}{1-\gamma}$$

Notes

Keep in mind that V is a vector of all state values. If the problem has 12 states (3×4 grid) then it is a 12-dim vector.

Convergence cont'd

$V_{k+1} \leftarrow BV_k \ldots B$ as the Bellman update $V_{k+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} p(s'|s,a) V_k(s')$

$\|BV_k - BV'_k\| \le \gamma \|V_k - V'_k\|$

$\|BV_k - V_{\text{true}}\| \le \gamma \|V_k - V_{\text{true}}\|$

Rewards are bounded, at the beginning then Value error is

$\|V_0 - V_{true}\| \le \frac{2R_{\max}}{1-\gamma}$

We run N iterations and reduce the error by factor γ in each and want to stop the error is below ϵ:

$\gamma^N 2R_{\max}/(1-\gamma) \le \epsilon$ Taking logs, we find: $N \ge \frac{\log(2R_{\max}/\epsilon(1-\gamma))}{\log(1/\gamma)}$

To stop the iteration we want to find a bound relating the error to the size of *one* Bellman update for any given iteration.

We stop if

$$\|V_{k+1} - V_k\| \le \frac{\epsilon(1-\gamma)}{\gamma}$$

then also: $\|V_{k+1} - V_{\text{true}}\| \le \epsilon$ Proof on the next slide

Notes

Try to prove that for any a:

$$\|\max f(a) - \max g(a)\| \le \max \|f(a) - g(a)\|$$

Then it holds that

$$\|BV_k - BV'_k\| \le \gamma \|V_k - V'_k\|$$

Note: The Bellman update is a *contraction* by a factor of γ on the space of utility vectors. ([1], 17.2.3)

Convergence cont'd

$\|V_{k+1} - V_{\text{true}}\| \leq \epsilon$ is the same as $\|V_{k+1} - V_{\infty}\| \leq \epsilon$
Assume $\|V_{k+1} - V_k\| = \text{err}$
In each of the following iteration steps we reduce the error by the factor γ (because $\|BV_k - V_{\text{true}}\| \leq \gamma \|V_k - V_{\text{true}}\|$). Till ∞, the total sum of reduced errors is:

$$\text{total} = \gamma\text{err} + \gamma^2\text{err} + \gamma^3\text{err} + \gamma^4\text{err} + \cdots = \frac{\gamma\text{err}}{(1-\gamma)}$$

We want to have total $< \epsilon$.

$$\frac{\gamma\text{err}}{(1-\gamma)} < \epsilon$$

From it follows that

$$\text{err} < \frac{\epsilon(1-\gamma)}{\gamma}$$

Hence we can stop if $\|V_{k+1} - V_k\| < \epsilon(1-\gamma)/\gamma$

Notes

Value iteration algorithm

function VALUE-ITERATION(env,ϵ) **returns:** state values V
  input: env - MDP problem, ϵ
  $V' \leftarrow 0$ in all states
  repeat ▷ iterate values until convergence
    $V \leftarrow V'$ ▷ keep the last known values
    $\delta \leftarrow 0$ ▷ reset the max difference
    for each state s **in** S **do**
      $V'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)V(s')$
      if $|V'[s] - V[s]| > \delta$ **then** $\delta \leftarrow |V'[s] - V[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$

Notes

Value iteration algorithm

function VALUE-ITERATION(env,ϵ) **returns:** state values V
  input: env - MDP problem, ϵ
  $V' \leftarrow 0$ in all states
  repeat ▷ iterate values until convergence
    $V \leftarrow V'$ ▷ keep the last known values
    $\delta \leftarrow 0$ ▷ reset the max difference
    for each state s **in** S **do**
      $V'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)V(s')$
      if $|V'[s] - V[s]| > \delta$ **then** $\delta \leftarrow |V'[s] - V[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$

Notes

Value iteration algorithm

function VALUE-ITERATION(env,ϵ) **returns:** state values V
    input: env - MDP problem, ϵ
    $V' \leftarrow 0$ in all states
    repeat ▷ iterate values until convergence
        $V \leftarrow V'$ ▷ keep the last known values
        $\delta \leftarrow 0$ ▷ reset the max difference
        for each state s **in** S **do**
            $V'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)V(s')$
            if $|V'[s] - V[s]| > \delta$ **then** $\delta \leftarrow |V'[s] - V[s]|$
    until $\delta < \epsilon(1-\gamma)/\gamma$

Notes

Value iteration algorithm

function VALUE-ITERATION(env,ϵ) **returns:** state values V
  input: env - MDP problem, ϵ
  $V' \leftarrow 0$ in all states
  repeat ▷ iterate values until convergence
    $V \leftarrow V'$ ▷ keep the last known values
    $\delta \leftarrow 0$ ▷ reset the max difference
    for each state s **in** S **do**
      $V'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)V(s')$
      if $|V'[s] - V[s]| > \delta$ **then** $\delta \leftarrow |V'[s] - V[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$

Notes

Value iteration algorithm

function VALUE-ITERATION(env,ϵ) **returns:** state values V
  input: env - MDP problem, ϵ
  $V' \leftarrow 0$ in all states
  repeat ▷ iterate values until convergence
    $V \leftarrow V'$ ▷ keep the last known values
    $\delta \leftarrow 0$ ▷ reset the max difference
    for each state s **in** S **do**
      $V'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)V(s')$
      if $|V'[s] - V[s]| > \delta$ **then** $\delta \leftarrow |V'[s] - V[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$

Notes

Sync vs. async Value iteration

function VALUE-ITERATION(env,ϵ) **returns:** state values V
  input: env - MDP problem, ϵ
  $V' \leftarrow 0$ in all states
  repeat ▷ iterate values until convergence
    $V \leftarrow V'$ ▷ keep the last known values
    $\delta \leftarrow 0$ ▷ reset the max difference
    for each state s **in** S **do**
      $V'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)V(s')$
      if $|V'[s] - V[s]| > \delta$ **then** $\delta \leftarrow |V'[s] - V[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$

Notes

Synchronous update: To update $V_t(s)$, $V_{t-1}(s)$ is used for all states $s1, ..., sn$.
Asynchronous update: Proceeds state by state. Imagine states $s1, s2, s3$ are neighbors in the state space (connected by some action).

1. Update $V_t(s1)$ using $V_{t-1}(s2)$ and $V_{t-1}(s3)$.
2. Update $V_{t+1}(s2)$ using $V_t(s1)$ and $V_t(s3)$, whereby $V_t(s3) = V_{t-1}(s3)$, but $V_t(s1) \neq V_{t-1}(s1)$.

Note: Asynchronous update can be more than that. One can choose to pick the states for value update based on their relevance – some heuristics. This can practically speed up convergence. At the same time, asymptotic convergence remains guaranteed under certain conditions (basically that all states get to get updated at least "every now and then"). (see [2], 4.5 Asynchronous Dynamic Programming)

References

Some figures from [1] (chapter 17) but notation slightly changed in order to adapt notation from [2] (chapters 3, 4) which will help us in the Reinforcement Learning part of the course. Note that the book [2] is available on-line.

[1] Stuart Russell and Peter Norvig.
Artificial Intelligence: A Modern Approach.
Prentice Hall, 3rd edition, 2010.
http://aima.cs.berkeley.edu/.

[2] Richard S. Sutton and Andrew G. Barto.
Reinforcement Learning; an Introduction.
MIT Press, 2nd edition, 2018.
http://www.incompleteideas.net/book/the-book-2nd.html.

Notes