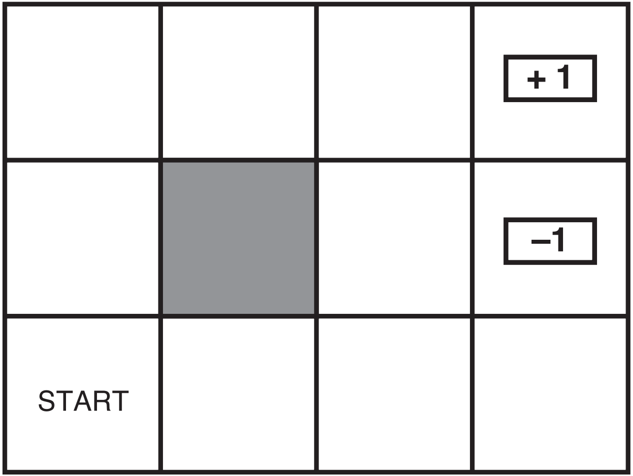
Sequential decisions under uncertainty
Markov Decision Processes (MDP)

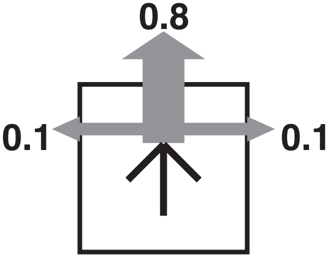
Tomáš Svoboda

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

March 29, 2023

Unreliable actions in *observable* grid world

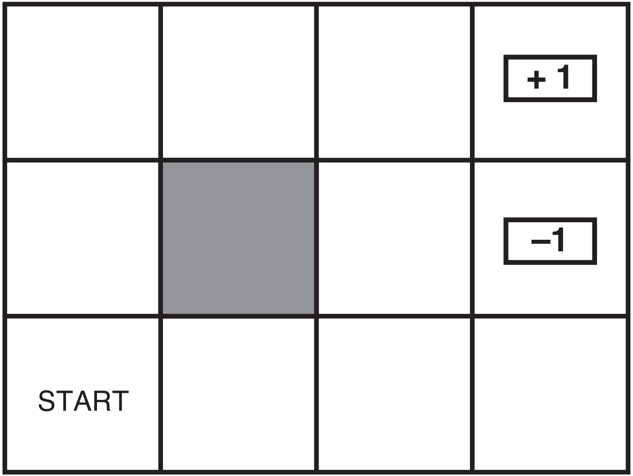


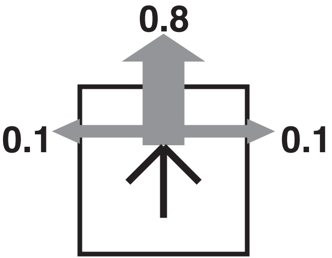


States $s \in \mathcal{S}$, actions $a \in \mathcal{A}$

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

Unreliable actions in *observable* grid world





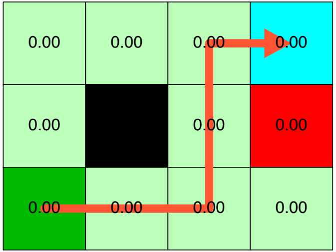
States $s \in \mathcal{S}$, actions $a \in \mathcal{A}$

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

Unreliable (results of) actions

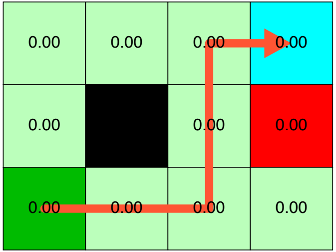
Plan? Policy

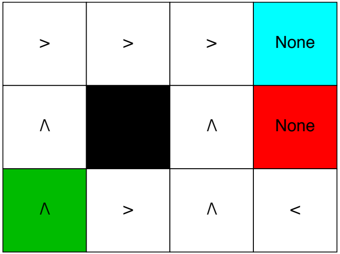
- In deterministic world: Plan – sequence of actions from Start to Goal.
- MDPs, we need a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$.
- An action for each possible state. Why *each*?
- What is the *best* policy?



Plan? Policy

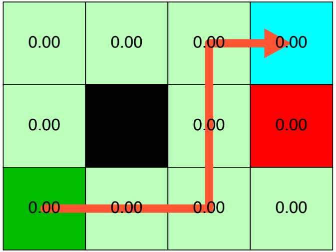
- In deterministic world: Plan – sequence of actions from Start to Goal.
- MDPs, we need a policy $\pi : \mathcal{S} \to \mathcal{A}$.
- An action for each possible state. Why *each*?
- What is the *best* policy?

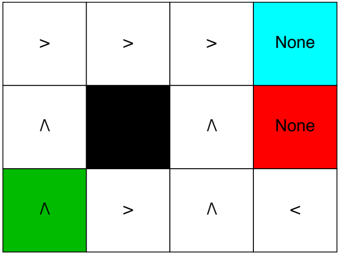




Plan? Policy

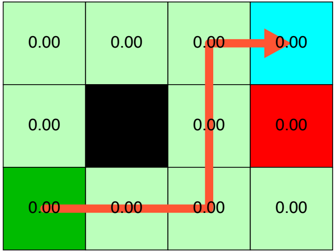
- In deterministic world: Plan – sequence of actions from Start to Goal.
- MDPs, we need a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$.
- An action for each possible state. Why *each*?
- What is the *best* policy?

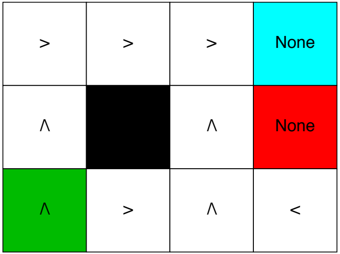




Plan? Policy

- 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?





Rewards

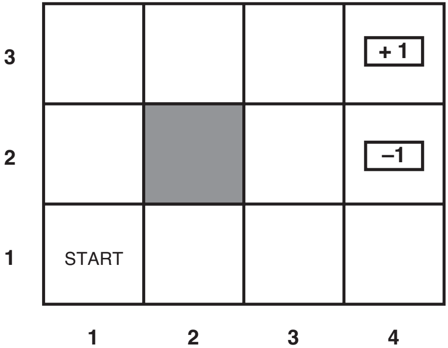
-0.04	-0.04	-0.04	1.00
-0.04		-0.04	-1.00
-0.04	-0.04	-0.04	-0.04

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

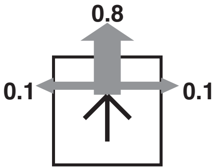
Reward function $r(s)$ (or $r(s,a)$, $r(s,a,s')$)

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

Markov Decision Processes (MDPs)



(a)



(b)

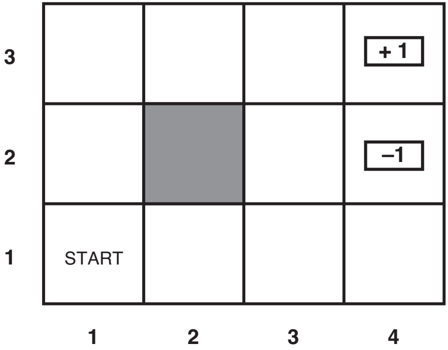
States $s \in S$, actions $a \in \mathcal{A}$

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

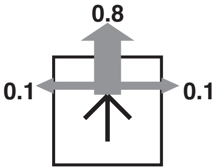
Reward function $r(s)$ (or $r(s, a)$, $r(s, a, s')$)

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

Markov Decision Processes (MDPs)



(a)



(b)

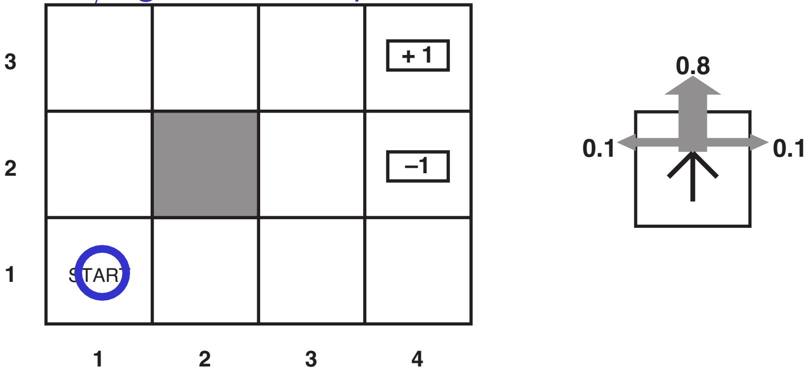
States $s \in \mathcal{S}$, actions $a \in \mathcal{A}$

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

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

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

Robot/Agent walk – Episode

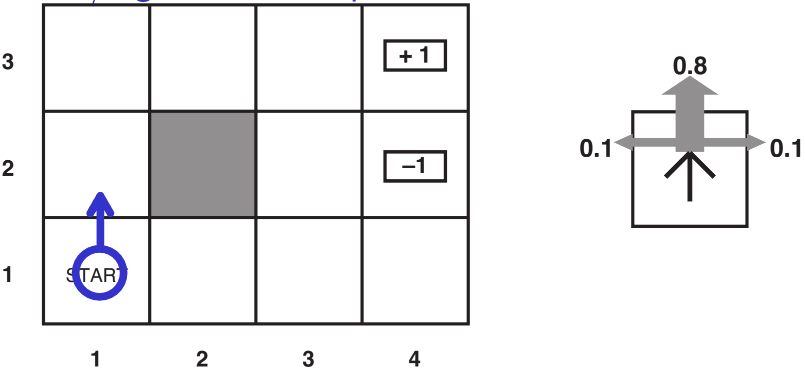


(a) (b)

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

Episode : one walk from S_0 to terminal.

Robot/Agent walk – Episode

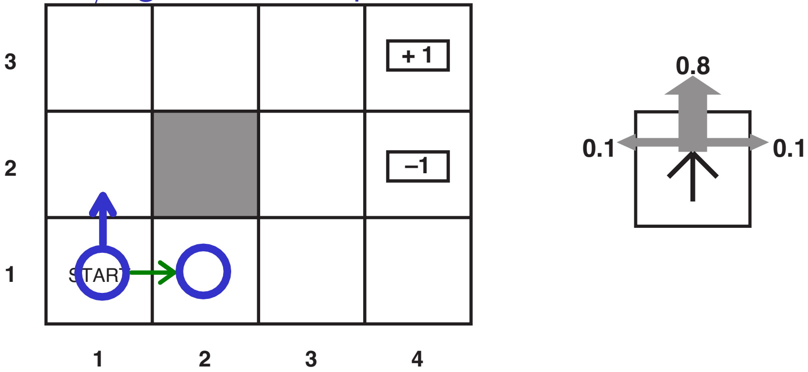


(a) (b)

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

Episode : one walk from S_0 to terminal.

Robot/Agent walk – Episode

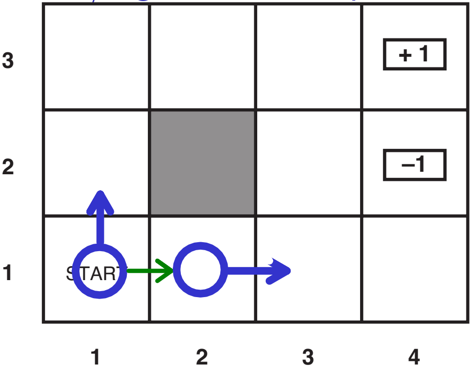


(a) (b)

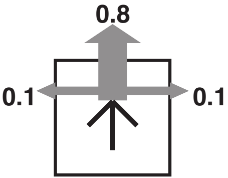
$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2 \dots$$

Episode : one walk from S_0 to terminal.

Robot/Agent walk – Episode



(a)

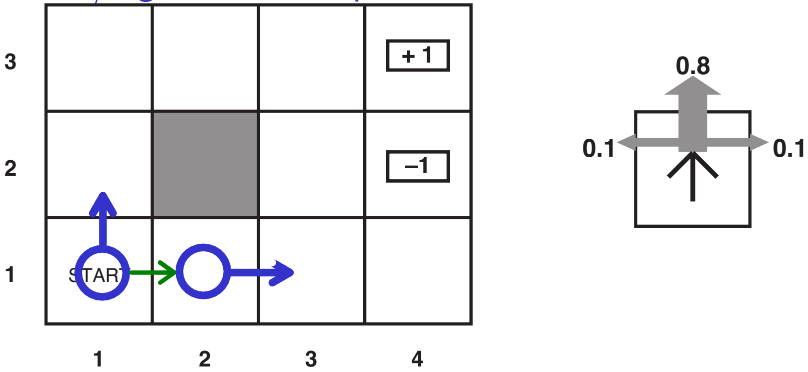


(b)

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

Episode : one walk from S_0 to terminal.

Robot/Agent walk – Episode



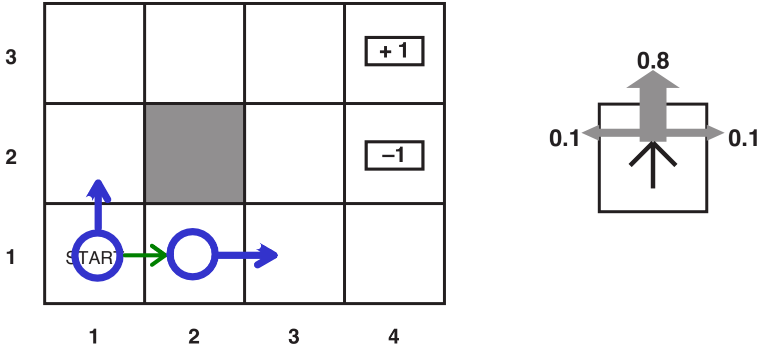
(a) (b)

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

Episode : one walk from S_0 to terminal.

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)

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.

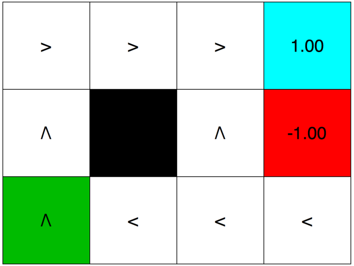
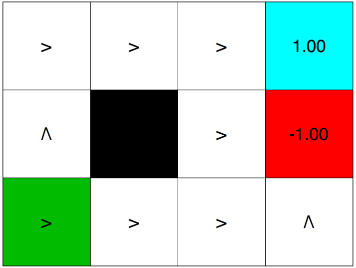
Desired robot/agent behavior specified through rewards

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

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.

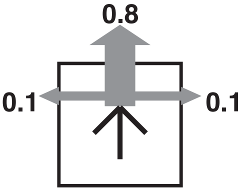
		
A	B	C
$r(s) \in \{-2, 1, -1\}$ a	$r(s) \in \{-0.04, 1, -1\}$ b	$r(s) \in \{-0.01, 1, -1\}$ c

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



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).

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).

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).

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:

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

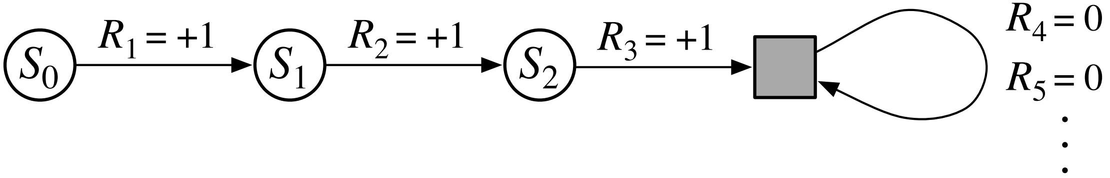
If stationary preferences :
Utility (h-history)
$U_h([S_0, S_1, S_2, \dots,]) = R_1 + R_2 + R_3 + \cdots$

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

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)

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



Horizon too far, infinite – Discount rewards

Problem: Infinite lifetime $\Rightarrow$ additive utilities are infinite.

- Finite horizon: termination at a fixed time $\Rightarrow$ 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

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

Horizon too far, infinite – Discount rewards

Problem: Infinite lifetime $\Rightarrow$ additive utilities are infinite.

- Finite horizon: termination at a fixed time $\Rightarrow$ 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

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

Horizon too far, infinite – Discount rewards

Problem: Infinite lifetime $\Rightarrow$ additive utilities are infinite.

- Finite horizon: termination at a fixed time $\Rightarrow$ 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

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

Horizon too far, infinite – Discount rewards

Problem: Infinite lifetime $\Rightarrow$ additive utilities are infinite.

- Finite horizon: termination at a fixed time $\Rightarrow$ 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

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

Horizon too far, infinite – Discount rewards

Problem: Infinite lifetime $\Rightarrow$ additive utilities are infinite.

- Finite horizon: termination at a fixed time $\Rightarrow$ 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

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

Horizon too far, infinite – Discount rewards

Problem: Infinite lifetime $\Rightarrow$ additive utilities are infinite.

- Finite horizon: termination at a fixed time $\Rightarrow$ 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

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

Horizon too far, infinite – Discount rewards

Problem: Infinite lifetime $\Rightarrow$ additive utilities are infinite.

- Finite horizon: termination at a fixed time $\Rightarrow$ 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

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

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.

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.

Expected Return of a policy π

- Executing policy π $\rightarrow$ 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,

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

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,

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

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,

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

Value functions

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)

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

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

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

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	

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	0
1	0.76		0.66	-1.00	1
2	0.71	0.66	0.61	0.39	2
	0	1	2	3	

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

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

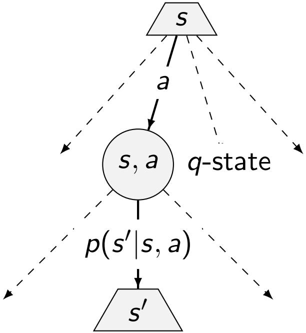
$v^*(s)$ = expected (discounted) sum of rewards (until termination) assuming *optimal* actions.

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

	0	1	2	3	
0	0.95	0.96	0.98	1.00	0
1	0.94		0.89	-1.00	1
2	0.92	0.91	0.90	0.80	2
	0	1	2	3	

	0	1	2	3	
0	>	>	>	1.00	0
1	∧		<	-1.00	1
2	∧	<	<	∨	2
	0	1	2	3	

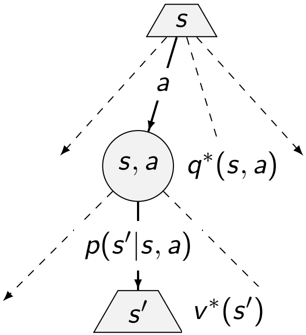
MDP search tree



MDP search tree

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

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



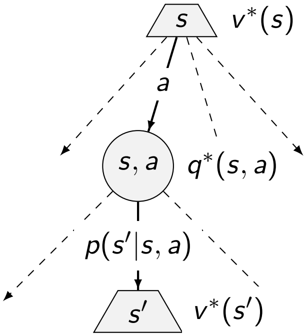
MDP search tree

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

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

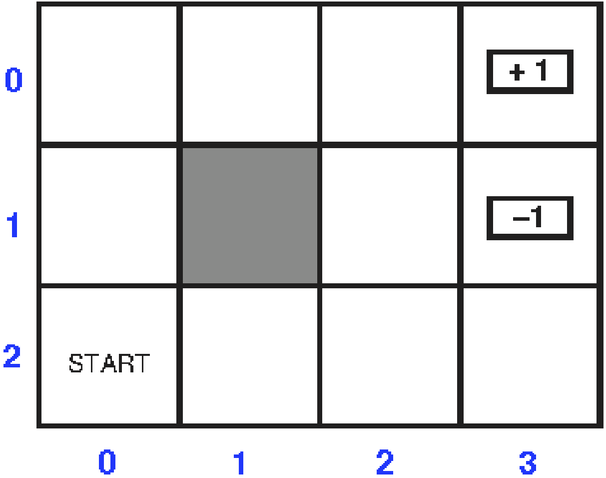
The value of a state s:

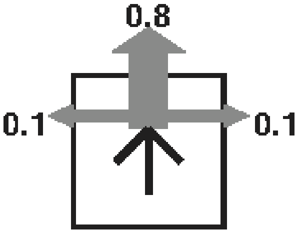
$$v^*(s) = \max_a q^*(s,a)$$



Bellman (optimality) equation

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





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.

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.

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.

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.

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: $O(AS)$

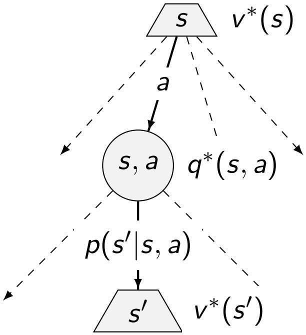
B: $O(S^2)$

C: $O(AS^2)$

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

Value iteration (dynamic programming) vs. direct search

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



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	

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:

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

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

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:

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

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

Convergence cont'd

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

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

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

Rewards are bounded, at the beginning then Value error is

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

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

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

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

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

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

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 $\|V_{k+1} - V_k\| < \epsilon(1-\gamma)/\gamma$

Value iteration algorithm

function VALUE-ITERATION(env,ϵ) **returns:** state values V

  input: env - MDP problem, ϵ

  $V' \leftarrow 0$ in all states

  repeat $\triangleright$ iterate values until convergence

    $V \leftarrow V'$ $\triangleright$ keep the last known values

    $\delta \leftarrow 0$ $\triangleright$ reset the max difference

    for each state s **in** S **do**

      $V'[s] \leftarrow R(s) + \gamma \max\limits_{a \in A(s)} \sum_{s'} P(s'|s,a)V(s')$

      if $|V'[s] - V[s]| > \delta$ **then** $\delta \leftarrow |V'[s] - V[s]|$

    end for

  until $\delta < \epsilon(1-\gamma)/\gamma$

end function

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**

      $V'[s] \leftarrow R(s) + \gamma \max\limits_{a \in A(s)} \sum_{s'} P(s'|s,a)V(s')$

      if $|V'[s] - V[s]| > \delta$ **then** $\delta \leftarrow |V'[s] - V[s]|$

    end for

  until $\delta < \epsilon(1-\gamma)/\gamma$

end function

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**

      $V'[s] \leftarrow R(s) + \gamma \max\limits_{a \in A(s)} \sum_{s'} P(s'|s,a)V(s')$

      if $|V'[s] - V[s]| > \delta$ **then** $\delta \leftarrow |V'[s] - V[s]|$

    end for

  until $\delta < \epsilon(1-\gamma)/\gamma$

end function

Value iteration algorithm

function VALUE-ITERATION(env,ϵ) **returns:** state values V

    input: env - MDP problem, ϵ

    $V' \leftarrow 0$ in all states

    repeat $\triangleright$ iterate values until convergence

        $V \leftarrow V'$ $\triangleright$ keep the last known values

        $\delta \leftarrow 0$ $\triangleright$ reset the max difference

        for each state s **in** S **do**

            $V'[s] \leftarrow R(s) + \gamma \max\limits_{a \in A(s)} \sum_{s'} P(s'|s,a)V(s')$

            if $|V'[s] - V[s]| > \delta$ **then** $\delta \leftarrow |V'[s] - V[s]|$

        end for

    until $\delta < \epsilon(1-\gamma)/\gamma$

end function

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** $\delta \leftarrow |V'[s] - V[s]|$
    end for
  until $\delta < \epsilon(1-\gamma)/\gamma$
end function

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** $\delta \leftarrow |V'[s] - V[s]|$
    end for
  until $\delta < \epsilon(1-\gamma)/\gamma$
end function

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.