Agent-Based Systems

Paolo Turrini

www.dcs.warwick.ac.uk/~pturrini   p.turrini@warwick.ac.uk

Markov Decision Processes

Policies are strategies

Plan for Today

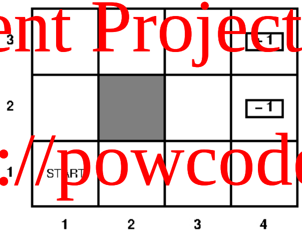
We now go back to a "typical" AI framework: Markov Decision Processes

- Plans and policies
- Optimal policies

These are "one player" games with perfect information.
Except they are not played on trees.

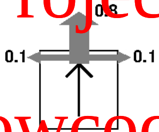
This (and more) in RN's chapters 17-18

Stuart Russell and Peter Norvig
Artificial Intelligence: a modern approach
2014 (3rd edition)



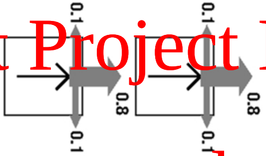
- Start at the starting square
- Move to adjacent squares
- Collision results in no movement
- The game ends when we reach either goal state $+1$ or -1

The agent



The agent chooses between $\{Up, Down, Left, Right\}$
and goes:

- to the intended direction with probability: e.g., 0.8
- to the left of the intended direction with probability: e.g., 0.1
- to the right of the intended direction with probability: e.g., 0.1



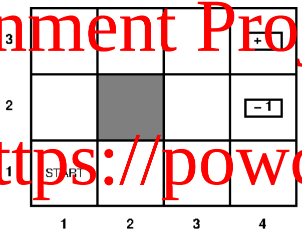
Walking is a repetition of throws:

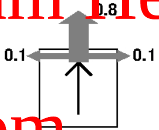
- The probability that I walk right the first time: 0.8
- The probability that I walk right the second time: 0.8
- The probability that I walk right both times... is a product! 0.8^2

The environment is **Markovian**: the probability of reaching a state only depends on the state the agent is in and the action they perform.

It is also **fully observable**, like an extensive game (of imperfect information).

Plans

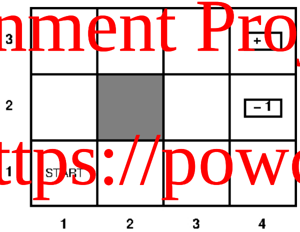


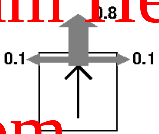


$\{Up, Down, Left, Right\}$ denote the intended directions.

A plan is a finite sequence of **intended** moves, **from the start**.

So $[Up, Down, Up, Right]$ is going to be the plan that, from the starting square, selects the intended moves in the specified order.





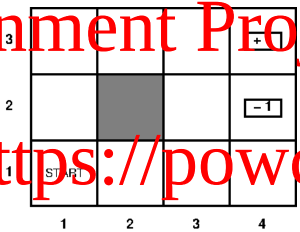
Goal: get to $+1$

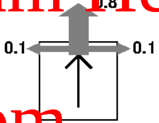
Consider the plan $[Up, Up, Right, Right, Right]$.

- With deterministic agents, it gets us to $+1$ with probability 1.
- But what happens to our stochastic agent instead?

What's the probability that $[Up, Up, Right, Right, Right]$ gets us to $+1$?

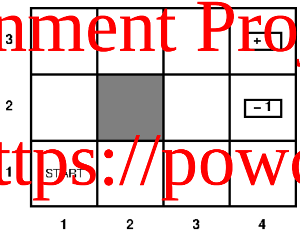
Makings plans

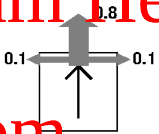




- It's not 0.8^5! This is the probability that we get to $+1$ when the plan works!

Makings plans





- It's not 0.8^5! This is the probability that we get to $+1$ when the plan works!
- The probability the plan does not work but still reaches $+1$ is $0.1^4 \times 0.8 = 0.00008$
- The correct answer is $0.8^5 + 0.1^4 \times 0.8$
- Notice $0.8^5 + 0.1^4 \times 0.8 < \frac{1}{3}$, not great.

Policies

S^+ is set of possible sequences of states (just like the histories of an extensive game!)

A the set of available actions.

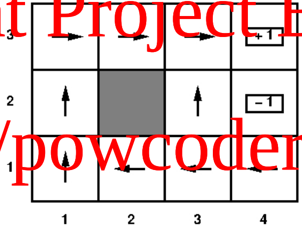
Then a policy is a function:

$$\pi : S^+ \to A$$

In words a policy is a protocol that at each possible decision point prescribes an action.

This **is** a strategy.

A policy



This is a **state-based** policy. It recommends the same action at each state (so if two sequences end up with the same state, this policy is going to recommend the same action)

Now let's complicate things a little bit...

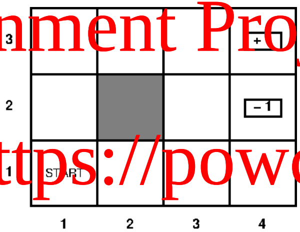
Rewards

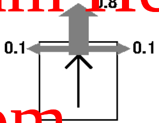
A reward function is a (utility) function of the form

$$r : S \to \mathbb{R}$$

All states, not just the terminal ones, get a reward!

Obviously, if you only care about terminal states, you may want to give zero to every other state. This is a more general model.

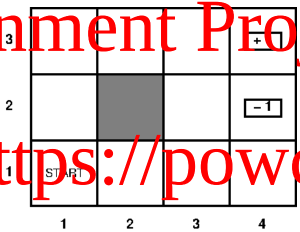


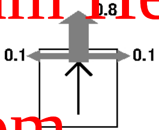


For instance, each non-terminal state:

- has 0 reward, i.e., only the terminal states matter;
- has negative reward, e.g., each move consumes -0.04 of battery;
- has positive reward, e.g., I like wasting battery

Rewards are usually small, negative and uniform at non-terminal states. But the reward function allows for more generality.

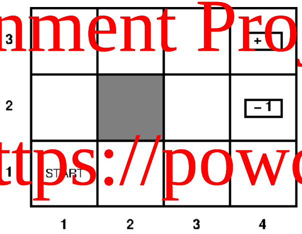


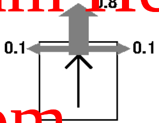


Consider now the following. The reward is:
$+1$ at state $+1$, -1 at state -1, -0.04 in all other states.

What's the expected utility of $[Up, Up, Right, Right, Right]$?

Rewards

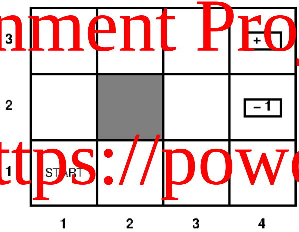


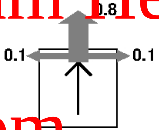


Consider now the following. The reward is:
$+1$ at state $+1$, -1 at state -1, -0.04 in all other states.

What's the expected utility of $[Up, Up, Right, Right, Right]$?

IT DEPENDS





Consider now the following. The reward is:
$+1$ at state $+1$, -1 at state -1, -0.04 in all other states.

What's the expected utility of [*Up*, *Up*, *Right*, *Right*, *Right*]?

IT DEPENDS on how we are going to put rewards together!

Comparing states

Many ways of comparing states:

- summing all the rewards
- giving priority to the immediate rewards
- ...

There is only one general and 'reasonable' way to combine rewards over time.
Discounted utility function: $u([s_0, s_1, s_2, \ldots]) = r(s_0) + \gamma r(s_1) + \gamma^2 r(s_2) + \cdots$

where $\gamma \in [0, 1]$ is the **discounting factor**

Notice: additive utility function $u([s_0, s_1, s_2, \ldots]) = r(s_0) + r(s_1) + r(s_2) + \cdots$ is just a discounted utility function where $\gamma = 1$.

Discounting factor

γ is a measure of the agent patience. How much more they value a gain of five today than a gain of five tomorrow, the day after etc...

- Used everywhere in AI, game theory, cognitive psychology
- A lot of experimental research on it
- Variants: discounting the discounting! I care more about the difference between today and tomorrow than the difference between some distant moment and the day after that!

Discounting

- $\gamma = 1$ today is just another day
- $\gamma = 0$ today is all that matters

Basically γ is my attitude to risk towards the future!

Notice that stochastic actions introduce further gambling into the picture

A problem

Here is a 3×101 world.

50	-1	-1	-1	$\cdots$	-1	-1	-1	-1
s				$\cdots$				
-50	1	1	1	$\cdots$	1	1	1	1

- start at *s*.
- two deterministic actions at *s*: either *Up* or *Down*
- beyond *s* you can only go *Right*.
- the numbers are the rewards you are going to get.

A problem

Here is a 3×101 world.

50	-1	-1	-1	$\cdots$	-1	-1	-1	-1
s				$\cdots$				
-50	1	1	1	$\cdots$	1	1	1	1

- start at *s*.
- two deterministic actions at *s*: either *Up* or *Down*
- beyond *s* you can only go *Right*.
- the numbers are the rewards you are going to get.

Compute the expected utility of each action as a function of γ

Solution

The utility of *Up* is

$$50\gamma - \sum_{t=2}^{101} \gamma^t = 50\gamma - \gamma^2 \frac{1-\gamma^{100}}{1-\gamma}$$

Solution

The utility of *Up* is

$$50\gamma - \sum_{t=2}^{101}\gamma^t = 50\gamma - \gamma^2\frac{1-\gamma^{100}}{1-\gamma}$$

The utility of *Down* is

$$-50\gamma + \sum_{t=2}^{101}\gamma^t = -50\gamma + \gamma^2\frac{1-\gamma^{100}}{1-\gamma}$$

The indifference point is

$$50\gamma - \gamma^2 \frac{1-\gamma^{100}}{1-\gamma} = -50\gamma + \gamma^2 \frac{1-\gamma^{100}}{1-\gamma}$$

The indifference point is

$$50\gamma - \gamma^2 \frac{1-\gamma^{100}}{1-\gamma} = -50\gamma + \gamma^2 \frac{1-\gamma^{100}}{1-\gamma}$$

Solving numerically, we have $\gamma \approx 0.9844$.

Solution

The indifference point is

$$50\gamma - \gamma^2 \frac{1-\gamma^{100}}{1-\gamma} = -50\gamma + \gamma^2 \frac{1-\gamma^{100}}{1-\gamma}$$

Solving numerically, we have $\gamma \approx 0.9844$.

- If γ is strictly larger than this then *Down* is better than *Up*;

Solution

The indifference point is

$$50\gamma - \gamma^2 \frac{1-\gamma^{100}}{1-\gamma} = -50\gamma + \gamma^2 \frac{1-\gamma^{100}}{1-\gamma}$$

Solving numerically, we have $\gamma \approx 0.9844$.

- If γ is strictly larger than this then *Down* is better than *Up*;
- If γ is strictly smaller than this then *Up* is better than *Down*;

Solution

The indifference point is

$$50\gamma - \gamma^2 \frac{1-\gamma^{100}}{1-\gamma} = -50\gamma + \gamma^2 \frac{1-\gamma^{100}}{1-\gamma}$$

Solving numerically, we have $\gamma \approx 0.9844$.

- If γ is strictly larger than this then *Down* is better than *Up*;
- If γ is strictly smaller than this then *Up* is better than *Down*;
- Else, it does not matter.

A Markov Decision Process is a sequential decision problem for a:

A Markov Decision Process is a sequential decision problem for a:

- fully observable environment

A Markov Decision Process is a sequential decision problem for a:

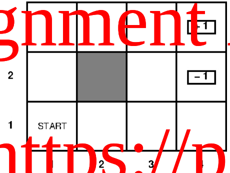
- fully observable environment
- with stochastic actions

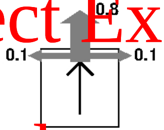
A Markov Decision Process is a sequential decision problem for a:

- fully observable environment
- with stochastic actions
- with a Markovian transition model

A Markov Decision Process is a sequential decision problem for a:

- fully observable environment
- with stochastic actions
- with a Markovian transition model
- and with discounted (possibly additive) rewards





Definition

Let s be a state and a and action

Model $P(s'|s,a) \equiv$ probability that a in s leads to s'

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

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

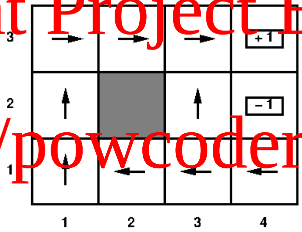
Expected utility of a policy

The expected utility (or value) of policy π, from state s is:

$$v^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t r(S_t)]$$

E is the expected utility of the sequences induced by:

- the policy π (the actions we are actually going to make)
- the initial state s (where we start)
- the transition model (where we can get to)



- In principle we can go on forever!
- We are going to assume we need to keep going unless we hit a terminal state (**infinite horizon assumption**)

With discounting the utility of an infinite sequence is in fact **finite**.
If $\gamma < 1$ and rewards are bounded above by $\mathbf{r}$, we have:

$$u[s_1, s_2, \ldots] = \sum_{t=0}^{\infty} \gamma^t r(s_t) \leqslant \sum_{t=0}^{\infty} \gamma^t \mathbf{r} = \frac{\mathbf{r}}{1-\gamma}$$

An **optimal** policy (from a state) is the policy with the highest expected utility, starting from that state.

$$\pi_s^* = \operatorname*{argmax}_{\pi} v^{\pi}(s)$$

We want to find the **optimal** policy.

A remarkable fact

Theorem

With discounted rewards and infinite horizon

$$\pi^*_s = \pi^*_{s'}, \text{ for each } s' \in S$$

This means that the optimal policy does not depend on the sequences of states, but on the states only.
In other words, the optimal policy is a state-based policy.

A remarkable fact

Theorem

With discounted rewards and infinite horizon

$$\pi_s^* = \pi_{s'}^*, \text{ for each } s' \in S$$

This means that the optimal policy does not depend on the sequences of states, but on the states only.
In other words, the optimal policy is a state-based policy.

Idea: Take π_a^* and π_b^*. If they both reach a state c, because they are both optimal, there is no reason why they should disagree (modulo indifference!). So π_c^* is identical for both (modulo indifference!). But then they behave the same at all states!

Value of states

The value of a state is the value of the optimal policy from that state.

But then (VERY IMPORTANT): Given the values of the states, choosing the best action is just maximisation of expected utility!

maximise the expected utility of the immediate successors

Figure: The values with $\gamma = 1$ and $r(s) = -0.04$

Figure: The optimal policy

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' \mid s, a) v(s')$$

Maximise the expected utility of the subsequent state

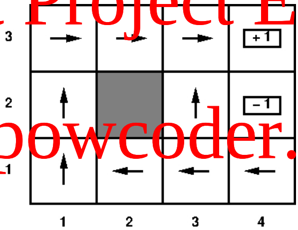


Figure: The optimal policy

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' \mid s, a) v(s')$$

Maximise the expected utility of the subsequent state

The definition of values of states, i.e., the expected utility of the optimal policy from there, leads to a simple relationship among values of neighbouring states:

The definition of values of states, i.e., the expected utility of the optimal policy from there, leads to a simple relationship among values of neighbouring states:

expected sum of rewards $=$
current reward $+ \gamma \times$ **expected sum of rewards after taking best action**

Bellman equation (1957):

$$v(s) = r(s) + \gamma \max_a \sum_{s'} P(s' \mid (s, a))\, v(s')$$

Bellman equation (1957):

$$v(s) = r(s) + \gamma \max_{a} \sum_{s'} P(s' \mid (s, a))\, v(s')$$

We can use it to compute the optimal policy!

Value Iteration Algorithm

1. Start with arbitrary values
2. Repeat for every s simultaneously until "no change"

$$v(s) \leftarrow r(s) + \gamma \max_a \sum_{s'} v(s')P(s' \mid (s, a))$$

- **Input** S, A, γ, r, and $P(s' \mid (s, a))$ for each $s, s' \in S$.
- **Input** $\epsilon > 0$, the error you want to allow

- **Output** v, the value of each state

1. **Initialise** $\delta_s := \epsilon\frac{(1-\gamma)}{\gamma}$ for all s[1], $v := 0$, storing information to be updated

[1] Typically this is uniform across states.

1. **Initialise** $\delta_s := \epsilon\frac{(1-\gamma)}{\gamma}$ for all s[1], $v := 0$, storing information to be updated
2. **while** $\delta_{s'} \geqslant \epsilon\frac{(1-\gamma)}{\gamma}$, for some s', **do**

[1] Typically this is uniform across states.

Value Iteration Algorithm

1. **Initialise** $\delta_s := \epsilon\frac{(1-\gamma)}{\gamma}$ for all s[1], $v := 0$, storing information to be updated
2. **while** $\delta_{s'} \geqslant \epsilon\frac{(1-\gamma)}{\gamma}$, for some s', **do**
   - $v'(s) := r(s) + \gamma \max_a \sum_{s'} P(s' \mid (s, a))v(s')$

[1] Typically this is uniform across states.

1. **Initialise** $\delta_s := \epsilon\frac{(1-\gamma)}{\gamma}$ for all s[1], $v := 0$, storing information to be updated
2. **while** $\delta_{s'} \geqslant \epsilon\frac{(1-\gamma)}{\gamma}$, for some s', **do**
   - $v'(s) := r(s) + \gamma \max_a \sum_{s'} P(s' \mid (s,a))v(s')$
   - $\delta_s := |v'(s) - v(s)|$

[1] Typically this is uniform across states.

Value Iteration Algorithm

1. **Initialise** $\delta_s := \epsilon\frac{(1-\gamma)}{\gamma}$ for all s[1], $v := 0$, storing information to be updated
2. **while** $\delta_{s'} \geqslant \epsilon\frac{(1-\gamma)}{\gamma}$, for some s', **do**
   - $v'(s) := r(s) + \gamma \max_a \sum_{s'} P(s' \mid (s, a))v(s')$
   - $\delta_s := |v'(s) - v(s)|$
   - $v(s) := v'(s)$

[1] Typically this is uniform across states.

1. **Initialise** $\delta_s := \epsilon\frac{(1-\gamma)}{\gamma}$ for all s[1], $v := 0$, storing information to be updated
2. **while** $\delta_{s'} \geqslant \epsilon\frac{(1-\gamma)}{\gamma}$, for some s', **do**
   - $v'(s) := r(s) + \gamma \max_a \sum_{s'} P(s' \mid (s, a))v(s')$
   - $\delta_s := |v'(s) - v(s)|$
   - $v(s) := v'(s)$
3. Return v

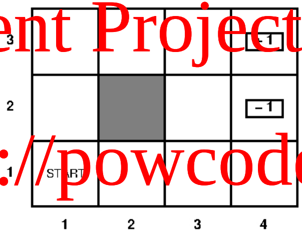
[1] Typically this is uniform across states.

A fundamental fact

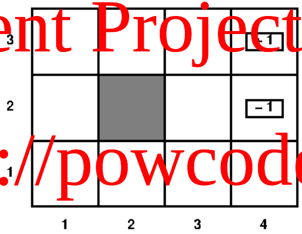
Theorem

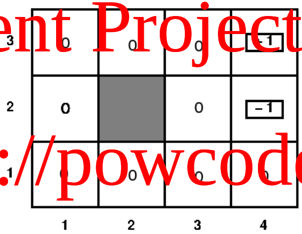
VIA:

- *terminates*
- *returns the optimal policy (for the input values)*


3
2
1
– 1
START
1
2
3
4

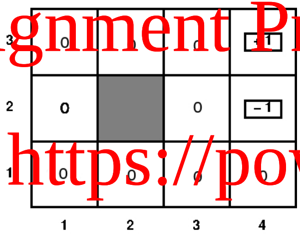
VIA in action

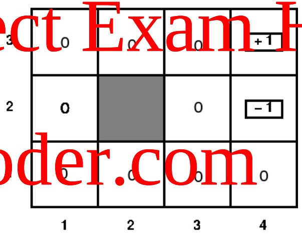




Initialise the values, for $\gamma = 1, r = -0.04$

VIA in action

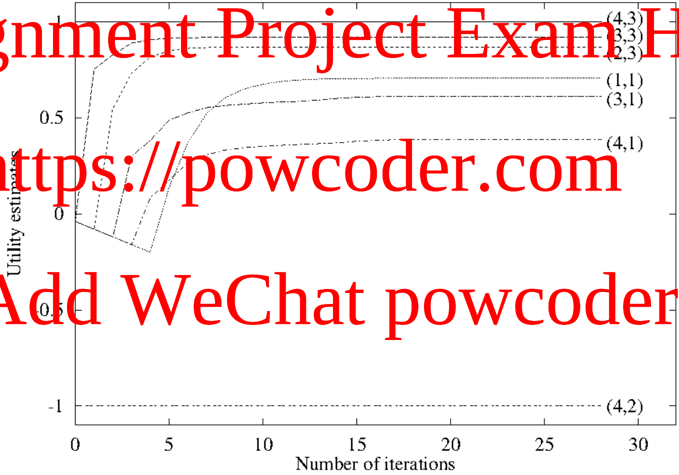


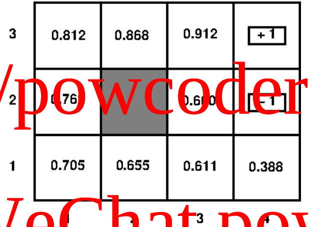


Simultaneously apply the Bellmann update to all states

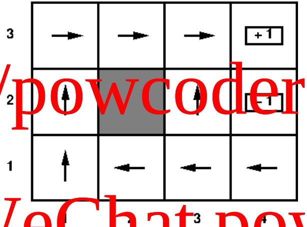
$$v(s) = r(s) + \gamma \max_a \sum_{s'} P(s' \mid (s,a)) v(s')$$

Value Iteration Algorithm

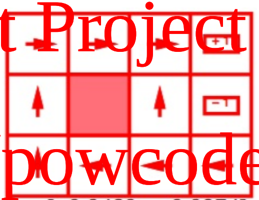



3
0.812
0.868
0.912
+ 1
2
- 1
1
0.705
0.655
0.611
0.388
3

The optimal policy



VIA in action

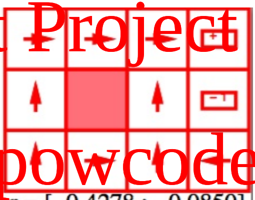


r = [−0.0480 : −0.0274]

VIA in action

3	0	0	0	[illegible]
2	0		0	$\boxed{-1}$
1	0	0	0	0
	1	2	3	4

Initialise the values, for $\gamma = 1, r = -0.4$

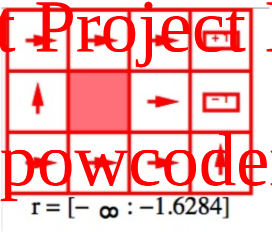


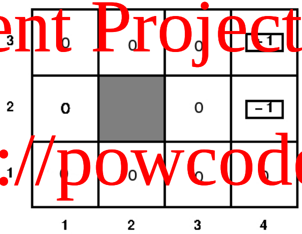
r = [–0.4278 : –0.0850]

3	0	0	0	$\boxed{-1}$
2	0		0	$\boxed{-1}$
1	0	0	0	0
	1	2	3	4

Initialise the values, for $\gamma = 1, r = -4$

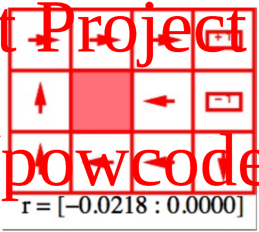
VIA in action



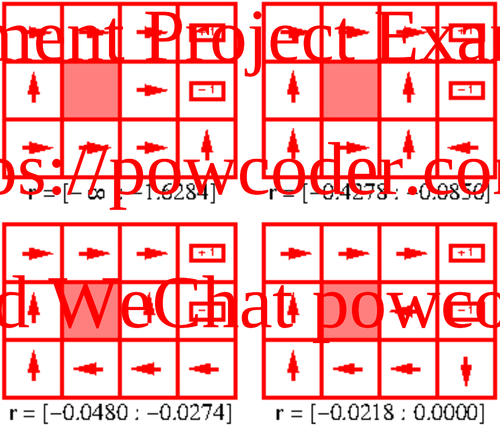


Initialise the values, for $\gamma = 1, r = 0$

VIA in action



VIA in action



Stocktaking

- Stochastic actions can lead to unpredictable outcomes
- But we can still find optimal "strategies" exploiting what happens in case we deviate from the original plan
- If we know what game we are playing and we play long enough...

What next? Learning in MDPs