Markov Decision Processes

Paolo Turrini

Department of Computing, Imperial College London

Introduction to Artificial Intelligence

The lectures

- The agent and the world (**Knowledge Representation**)
  - Actions and knowledge
  - Inference
- Good decisions (**Risk and Decisions**)
  - Chance
  - Gains
- Good decisions in time (**Markov Decision Processes**)
  - Chance and gains in time
  - Patience
  - Finding the best strategy
- Learning from experience (**Reinforcement Learning**)
  - Finding a reasonable strategy

Markov Decision Processes (iii)

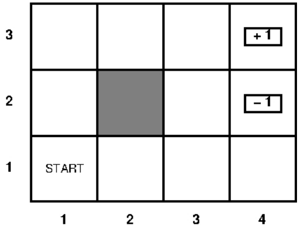
‘One thing is to know you have a good strategy, another thing is to know which one it is’

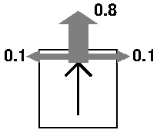
Outline

The book

Stuart Russell and Peter Norvig
Artificial Intelligence: a modern approach
Chapter 17

Recall...

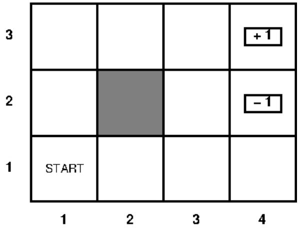


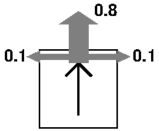


A Markov Decision Process is a sequential decision problem with:

- a fully observable environment
- stochastic actions
- a Markovian transition model
- discounted rewards

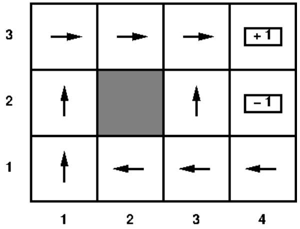
Recall...

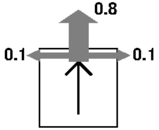




A policy is a specification of moves at each decision point.

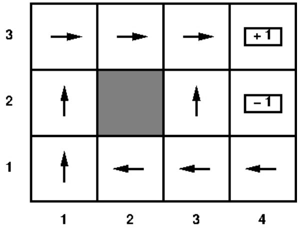
Recall...

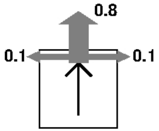




A policy is a specification of moves at each decision point.

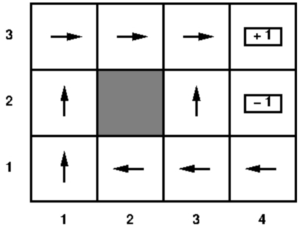
Recall...

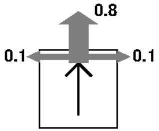




Important: A policy, in general, specifies a move for each sequence of visited states.
So it could well be that it recommends different moves at different times we are in the same state.
A policy is state-based if it recommends the same moves whenever it visits the same state.

Recall...



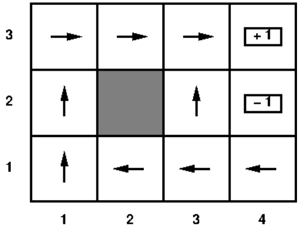


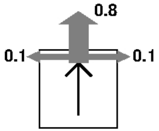
The expected utility (or value) of policy π, from state s is:

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

E is the probability distribution over the sequences induced by: the policy π, the state s (where we start), and the transition model (where we can get to).

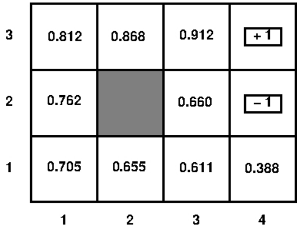
Recall...

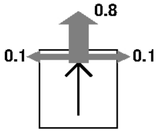




Fact: The optimal policy, the one with highest expected utility, is a state-based policy.
State-based policies are functions $\pi : S \to A$,
where S is the set of states and A is the set of available actions.
We focus on them.

Recall...

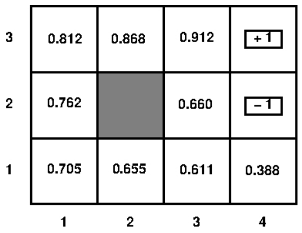


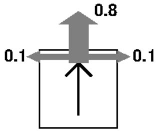


The value of a state
is the expected utility of the optimal policy from that state.

What you see above are the values of a state for $r = -0.04$ and $\gamma = 1$.

Recall...

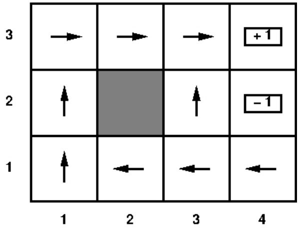


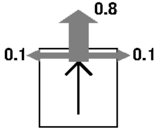


We can construct the optimal policy just maximising the expected utility of the immediate successors.

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

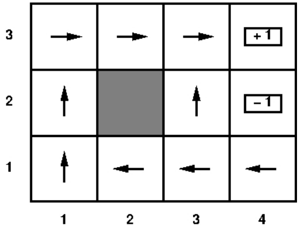
Recall...

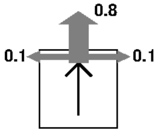




This is the optimal policy for $\gamma = 1$ and $r = -0.04$.

Recall...





Ok all good, but...

how can we find the values without having to calculate the infinitely many sequences that we can generate, also given that infinitely many of them are infinite in length?

The Bellman equation

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 Bellman equation

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**

The Bellman equation

Bellman equation (1957):

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

The Bellman equation

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”

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

Value Iteration Algorithm

- **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

Value Iteration Algorithm

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

Value Iteration Algorithm

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

Value Iteration Algorithm

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

Value Iteration Algorithm

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

Value Iteration Algorithm

```
1. Initialise $\delta_s := \epsilon\frac{(1-\gamma)}{\gamma}$, $v := 0$, storing information to be updated
2. while $\delta_s \geq \epsilon\frac{(1-\gamma)}{\gamma}$, for some $s$, do
       • $v'(s) := r(s) + \gamma \max_a \sum_{s'} P(s' \mid (s,a))v(s')$
       • If $|v'(s) - v(s)| > \delta_s$ then $\delta_s := |v'(s) - v(s)|$, $v := v'$
       • Else return $v$
```

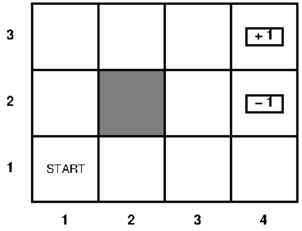
A fundamental fact

Theorem

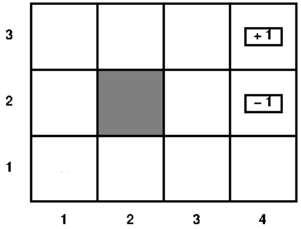
VIA:

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

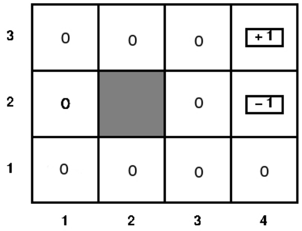
VIA in action



VIA in action

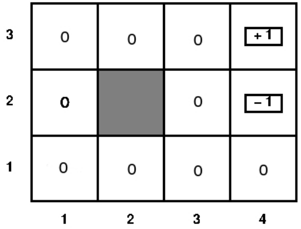


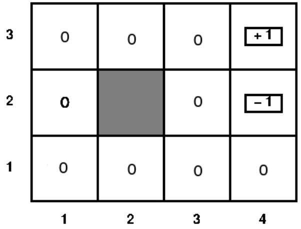
VIA in action



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

VIA in action

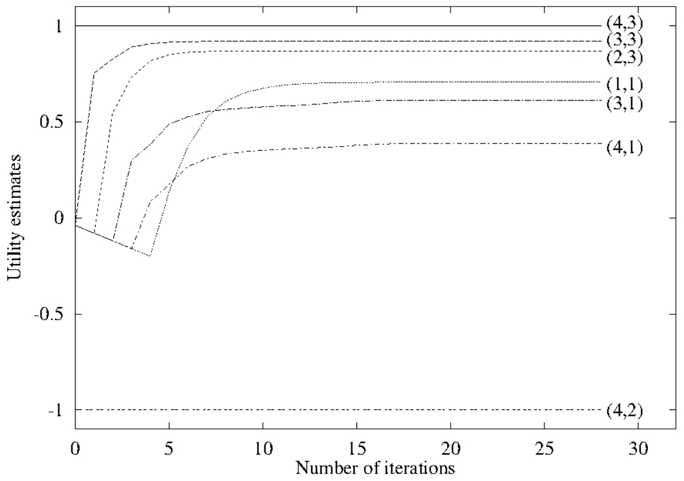




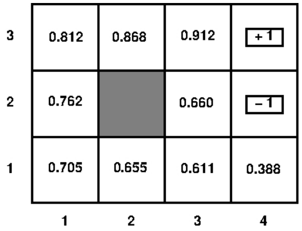
Simultaneously apply the Bellmann update to all states

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

Value Iteration Algorithm

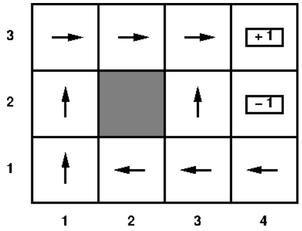


The state values

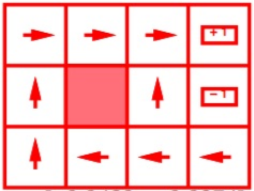


3	0.812	0.868	0.912	$\boxed{+1}$
2	0.762	▨	0.660	$\boxed{-1}$
1	0.705	0.655	0.611	0.388
	1	2	3	4

The optimal policy

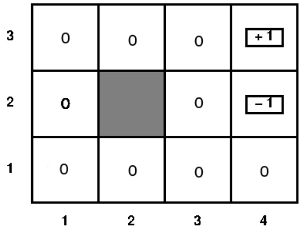


VIA in action



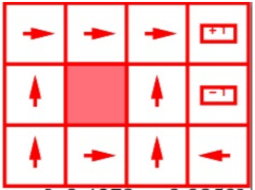
r = [−0.0480 : −0.0274]

VIA in action



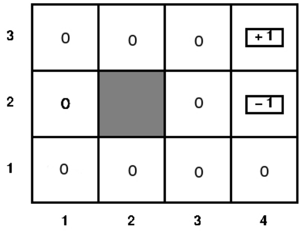
Initialise the values, for $\gamma = 1, r = -0.4$

VIA in action



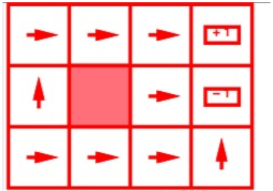
r = [–0.4278 : –0.0850]

VIA in action



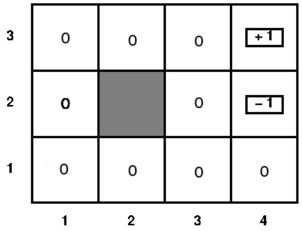
Initialise the values, for $\gamma = 1, r = -4$

VIA in action



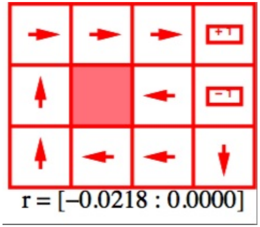
r = [− ∞ : −1.6284]

VIA in action

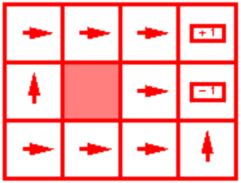


Initialise the values, for $\gamma = 1, r = 0$

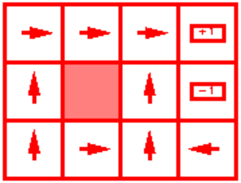
VIA in action



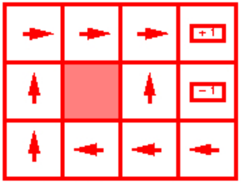
VIA in action



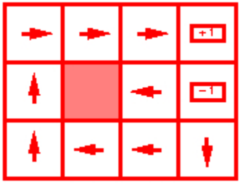
r = [−∞ : −1.6284]



r = [−0.4278 : −0.0850]



r = [−0.0480 : −0.0274]

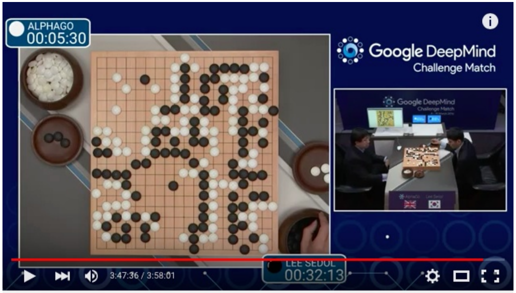


r = [−0.0218 : 0.0000]

Today's lecture

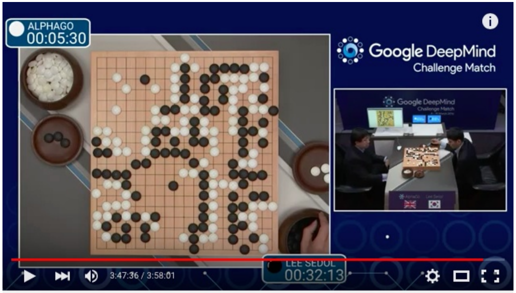
- Stochastic actions can lead to unpredictable outcomes
- But we can still find optimal "strategies", exploiting what happens in case we deviate from the original plan
- If we know what game we are playing and we play long enough...

Coming next



What if we don't know what game we are playing?

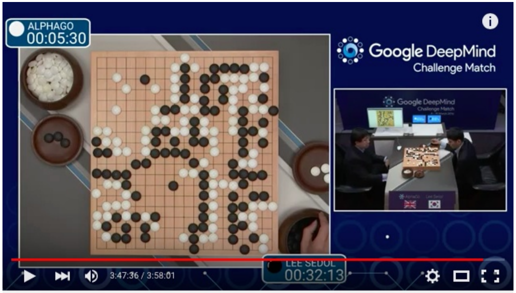
Coming next



What if we don't know what game we are playing?

Play anyway and see what happens!

Coming next



What if we don't know what game we are playing?

Play anyway and see what happens!
and play as much as possible!