Model-based learning

The other RL technic

Guillaume Lozenguez

@imt-lille-douai.fr

IMT Lille Douai
École Mines-Télécom
IMT-Université de Lille

Model-based learning

Main Idea:

- Random trajectories (a lot)
- Until each transition is visited several times.
- Compute an optimal policy.

Potentially:

- Require driving exploration
- Only incomplete exploration can be performed

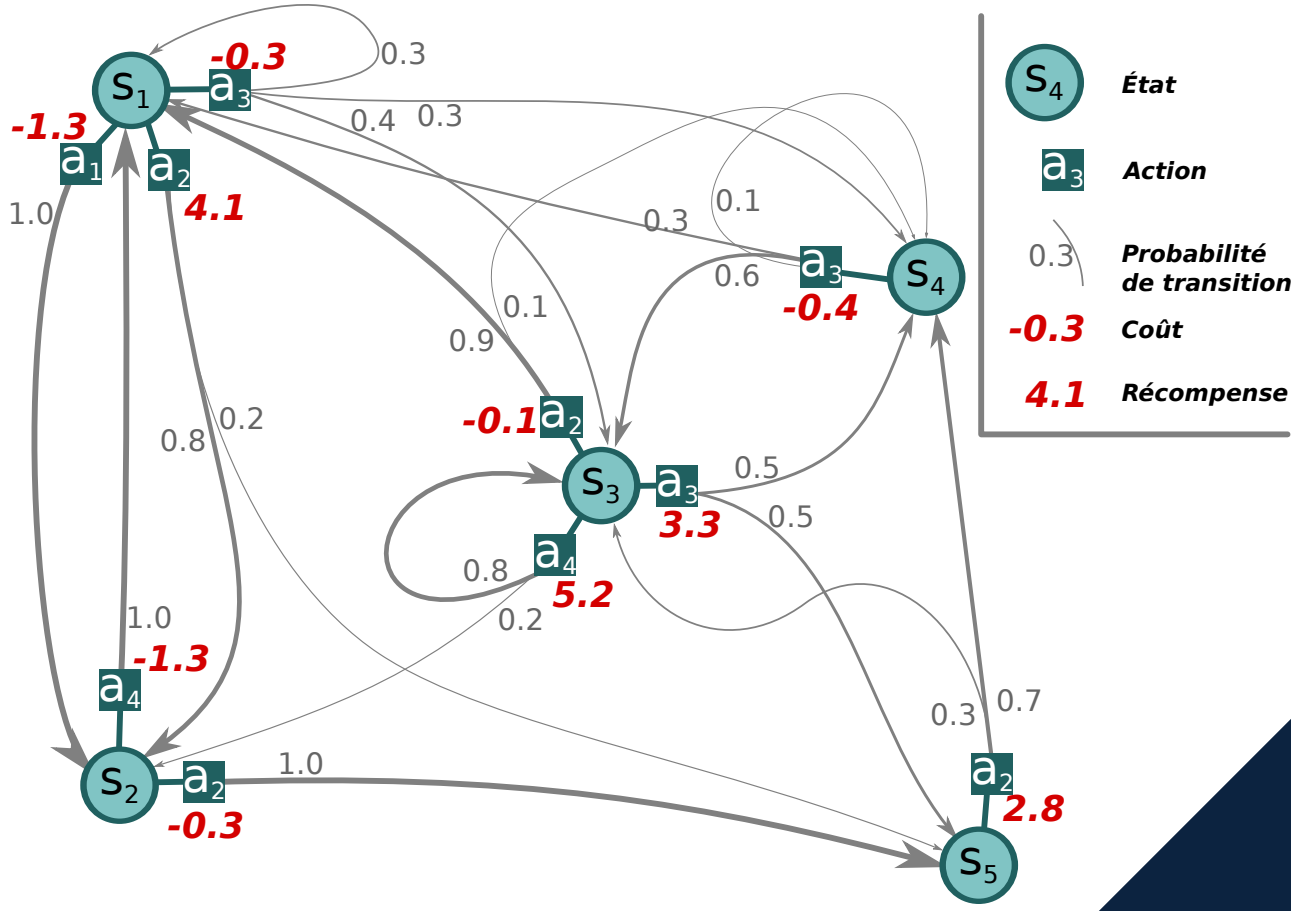
Markov Decision Process

MDP: $\langle S, A, T, R \rangle$:

S : set of system's states
A : set of possible actions
T : S × A × S → [0, 1] : transitions
R : S × A → R : cost/rewards

Optimal policy:

The policy π^* maximizing Bellman



Bellman Equation

State evaluation for a given policy π:

$$V^{\pi}(s) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \times V^{\pi}(s')$$

(with $\pi(s) = a$, s' the potential state at the next time step and γ the discount factor)

As a sum of gains:

- The immediate reward: $R(s,a)$
- The future gains $V^{\pi}(s')$, proportional to the probability to reach them $T(s,a,s')$
- with the parameter $\gamma \in [0,1]$, balancing immediate and future gains

Solving MDP: Value Iteration

Input: an **MDP:** $\langle S, A, T, R \rangle$; precision error: ϵ ; discount factor γ ; initial $V(s)$

1. Repeat until the **maximal delta** < ϵ

   For each state $s \in S$

   - Search the action a^* maximizing the Bellman Equation
   - Update $\pi(s)$ and $V(s)$ by considering action a^*
   - Compute the delta value between the previous and the new $V(s)$

Output: an optimal π^* and associated **V-values** (s': state at the next time step).

Solving MDP: Policy Iteration

Input: an **MDP:** $\langle S, A, T, R \rangle$; precision error: ϵ ; discount factor γ ; initial $V(s)$

1. Repeat until $\pi(s)$ is stable
   - Update $V(s)$ at ϵ error, for each state $s \in S$
   - Update $\pi(s)$, for each state $s \in S$

Output: an optimal π^* and associated **V-values**.

$$V^{\pi}(s) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \times V^{\pi}(s')$$

(s': state at the next time step)

Application to 421

- Python implementation - playerMDP.py :

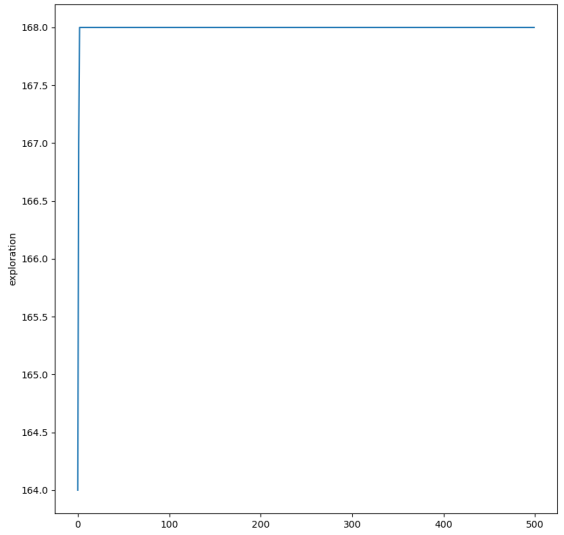
```
solver= MDP()
solver.learnModel( Engine() )
solver.valueIteration()

player= PiPlayer( solver.policy() )
```

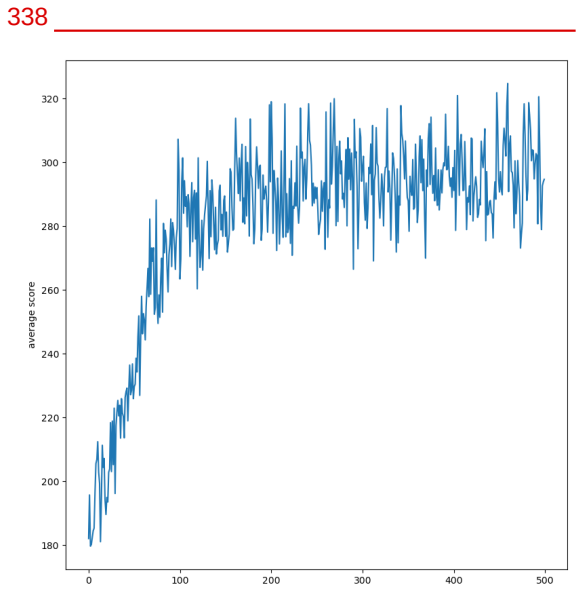
- Learning phase: Estimate **t** and **r**:
  - *10 000* simulations for each couple (s, a)
- Value iteration:
  - *3* iterations (finit game in **3** time steps)
- Average score (100 000 games): **~338**

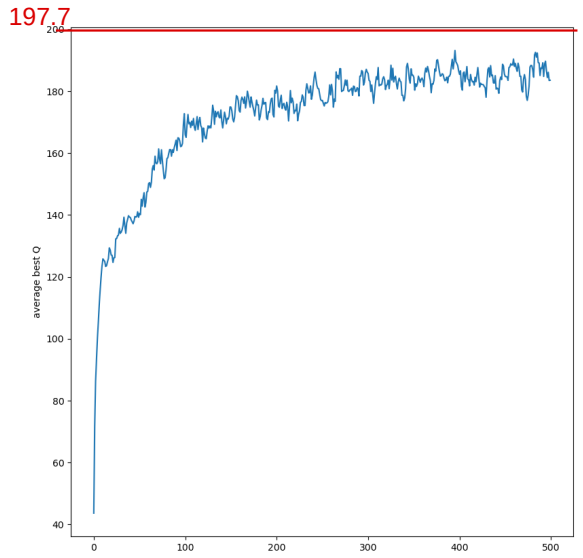
(To notice: decreasing the learning phase impact the average score)

In comparison to QLearning results

- With **500** steps of **500** games:







- α : 0.1 ; ϵ : 0.1 ; γ : 0.99

Lets play to a more complexe game...

Example: Zombie Dice

Eat maximum brains

without dying (3 damages)

- Players are zombies.
- They try to catch humans three at a time.
- Humans are dice with probability to fight back.