Artificial Intelligence for Robotics
INSA Lyon
2023
Olivier Simonin & Tristan Roussillon & Aurélien Delage

Course 1: Implementing basic solving algorithms for robot path planning

Exercise 1: Formalisation

Example 1 (Grid World) *An agent is in the bottom left cell of a grid $n \times p$. There are some uncrossable walls in the grid, one goal cell and a negative-rewarded well. The agents can move up,right,down and left. But! Things can go wrong — sometimes the effects of the actions are not what we want:*

- *the agent moves to the direction she intended to, with probability 0.7;*
- *the agent moves to one of the three other possible directions with probability 0.1;*
- *if the agent tries to go in a wall or if she slips (condition above, case that happens with probability 0.1), the agent stays where she is.*

The task is to navigate from the start cell in the bottom left to maximise the expected reward. What would the best sequence of actions be for this problem?

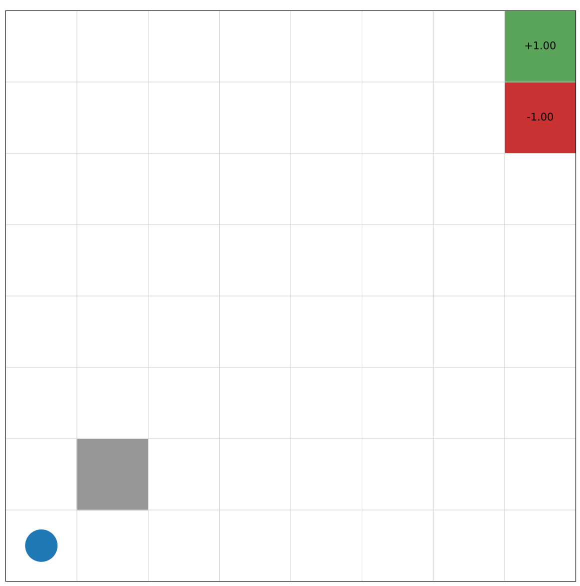

Figure 1: Illustration of the grid world environment

1. Propose a `MDP` formalism for this problem, when $n = p = 3$.
2. An implementation of the Grid World environment is given the github repository. Start implementing your own custom policy, *e.g*:

```python
from gridworld import *

mdp = GridWorld()

print("states:",mdp.get_states())
print("terminal states:",mdp.get_goal_states())
print("actions:",mdp.get_actions())
print(mdp.get_transitions(mdp.get_initial_state(),mdp.UP))

def policy_custom(state):
    return mdp.UP

while(1):
    state=mdp.get_initial_state()
    new_state,_ = mdp.execute(state,policy_custom(state))
    mdp.initial_state=new_state
    mdp.visualise()
```

*Remark: the environment defines a "terminal state" (*i.e *the goal state).*

Exercise 2: Solving algorithms

Recall that the dynamic programming algorithm implements the following scheme:

Algorithm 1: Dynamic Programming

Input: $\epsilon > 0$;
Initialize $V, \tilde{V} : S \to \mathbb{R}$
do
  $\tilde{V} \leftarrow V$;
  for $s \in S$ **do**
    $V(s) \leftarrow \max_a r(s,a) + \gamma \sum_{s'} P(s,a,s')\tilde{V}(s')$;
  end
while $\|V - \tilde{V}\|_\infty \geq \epsilon$;
return V

1. Implement the dynamic programming algorithm to learn optimal paths
2. `TD/Q-learning`
   (a) Q-learning generates episodes. When does the episode stop?
   (b) Implement Q-learning. Do you observe something?
      i. tips: use the "defaultdict" library in Python;
      ```python
      from collections import defaultdict
      ```
      ii. tips: a pair (s, a) can be represented by a tuple in Python and given as a key of a defautdict.
3. (bonus) The environment provides visualisation tools for value function and policy. For dynamic programming, create classes "*value_function*" and "*policy*", which should respectively implement "*get_value*(*states*)" and "*select_action*(*states*)". Then, call the visualisation tool of the environment with:
   ```python
   mdp.visualise_value_function(value_function)
   mdp.visualise_policy(policy)
   ```
4. (bonus) Visualise the Q-learning & `MC` q-value function. To do so, create classes "*q_function*". Then, call the visualisation tool for the q-function:
   ```python
   mdp.visualise_q_function(q_function)
   ```

Exercise 3: Analysis

Now that the algorithms are running - and hopefully learning something -, an important step is to analyze (i) their learning process and (ii) the resulting policy after reaching the time-limit budget for the learning process.

1. Implement adequate tools for analyzing the learning process.
2. Compare the policies obtained for all three algorithms after different time budgets.