INFO8003 - Optimal decision making for complex problems

Reinforcement Learning in a Discrete Domain (Part 1)

Prof. Ernst Damien
TA: Samy Aittahar - Bardhyl Miftari

Debor Antoine	Antoine.Debor@student.uliege.be
Navez Pierre	Pierre.Navez@student.uliege.be

Academic year 2020-2021

1 Implementation of the domain

The corresponding Python code can be found in the file `section1.py`.

A class named `Domain` has been created. This class contains four attributes and four methods, allowing to get information about the dynamics, the reward as well as the state and action spaces. The four attributes are the reward matrix (as given in the statement), the type of the domain (deterministic or stochastic), the value of the discount factor and the stochastic threshold. The domain instance represented by the reward matrix has been written in a `csv` file which is loaded when the domain is instantiated.

When launching the program, one can add arguments to specify the value of several parameters. In particular, in order to change the type of the domain in which the agent will take actions, one just needs to add `--stochastic` in the command line of the terminal. Otherwise, the deterministic domain is selected by default.

The rule-based policy that has been chosen consists in always going to the right. In other words, the action selected by the intelligent agent in the action space will always be (0, 1).

Starting from the state $x_0 = (0, 3)$ and following the policy described here above, the 10 steps trajectory followed by the agent in the deterministic domain is depicted in FIGURE 1.1, each step being written as (x_t, u_t, r_t, x_{t+1}), with u_t the action performed at time t and r_t the corresponding reward. The trajectory in the stochastic domain is presented in FIGURE 1.2.

The trajectory is computed thanks to the function `trajectory` of the code. This function takes as parameters the type of domain, the initial state and the number of steps the agent will effectuate. Then, using the methods of the `Domain` class, along with the defined policy, one can easily simulate the journey of the agent in the environment.

```
Deterministic domain chosen (default)

Initial state of the agent : (3, 0)

Instance of the domain:
[[ -3   1  -5   0  19]
 [  6   3   8   9  10]
 [  5  -8   4   1  -8]
 [  6  -9   4  19  -5]
 [-20 -17  -4  -3   9]]

Followed trajectory for 10 steps (Deterministic domain):
(x_0, u_0, r_0, x_1) : ((3, 0), (0, 1), -9, (3, 1))
(x_1, u_1, r_1, x_2) : ((3, 1), (0, 1), 4, (3, 2))
(x_2, u_2, r_2, x_3) : ((3, 2), (0, 1), 19, (3, 3))
(x_3, u_3, r_3, x_4) : ((3, 3), (0, 1), -5, (3, 4))
(x_4, u_4, r_4, x_5) : ((3, 4), (0, 1), -5, (3, 4))
(x_5, u_5, r_5, x_6) : ((3, 4), (0, 1), -5, (3, 4))
(x_6, u_6, r_6, x_7) : ((3, 4), (0, 1), -5, (3, 4))
(x_7, u_7, r_7, x_8) : ((3, 4), (0, 1), -5, (3, 4))
(x_8, u_8, r_8, x_9) : ((3, 4), (0, 1), -5, (3, 4))
(x_9, u_9, r_9, x_10) : ((3, 4), (0, 1), -5, (3, 4))
(x_10, u_10, r_10, x_11) : ((3, 4), (0, 1), -5, (3, 4))
```

FIGURE 1.1: 10 steps trajectory in the deterministic domain

```
Stochastic domain chosen

Initial state of the agent : (3, 0)

Instance of the domain:
[[ -3   1  -5   0  19]
 [  6   3   8   9  10]
 [  5  -8   4   1  -8]
 [  6  -9   4  19  -5]
 [-20 -17  -4  -3   9]]

Followed trajectory for 10 steps (Stochastic domain):
(x_0, u_0, r_0, x_1) : ((3, 0), (0, 1), -3, (3, 1))
(x_1, u_1, r_1, x_2) : ((3, 1), (0, 1), 4, (0, 0))
(x_2, u_2, r_2, x_3) : ((0, 0), (0, 1), 1, (0, 1))
(x_3, u_3, r_3, x_4) : ((0, 1), (0, 1), -3, (0, 0))
(x_4, u_4, r_4, x_5) : ((0, 0), (0, 1), 1, (0, 1))
(x_5, u_5, r_5, x_6) : ((0, 1), (0, 1), -5, (0, 0))
(x_6, u_6, r_6, x_7) : ((0, 0), (0, 1), 1, (0, 0))
(x_7, u_7, r_7, x_8) : ((0, 0), (0, 1), -3, (0, 1))
(x_8, u_8, r_8, x_9) : ((0, 1), (0, 1), -3, (0, 2))
(x_9, u_9, r_9, x_10) : ((0, 2), (0, 1), 0, (0, 0))
(x_10, u_10, r_10, x_11) : ((0, 0), (0, 1), -3, (0, 1))
```

FIGURE 1.2: 10 steps trajectory in the stochastic domain

2 Expected return of a policy

The corresponding Python code can be found in the file `section2.py`.

Using the stationary policy stating that the agent will always go to the right, one can approximate the expected return of this policy when starting from the initial state $x_0 = x$ by the following recurrence equation

$$J_N^\mu(x) = E\{r(x, \mu(x), w) + \gamma J_{N-1}^\mu(f(x, \mu(x), w))\}, \quad \forall N \geq 1 \tag{2.1}$$

with $J_0^\mu = 0$ and $\forall x$. This equation is based on the reward signal r associated with state x and the stationary policy $\mu(x)$. The decay factor γ is also taken into account at each iteration step.

For the deterministic case, one can get rid of the mathematical expectation over the disturbance w. In the case where the domain is stochastic, however, the dynamics of the domain differ from the deterministic case and one needs to take w into account, as for $w > 0.5$ a "jump" to state (0,0) is performed.

From the dynamic programming theory, one knows that for a infinite number N of iterations, the approximation of equation 2.1 tends to the true one in infinite norm[1]. Additionally, the theory gives a bound on the approximation error; this bound is expressed as a function of the discount factor γ, the number of iterations N, and the maximum reward signal B_r. Mathematically, this bound writes

$$\|J_N^\mu - J^\mu\|_\infty \leq \frac{\gamma^N}{1-\gamma} B_r. \tag{2.2}$$

[1]Alternatively, one can say that the infinite norm error tends towards 0.

N	Bound
10	1718.33
100	695.46
1000	0.082
3000	1.53×10^{-10}
10000	4.27×10^{-41}

TABLE 2.1: Value of the bound for different number of iterations

The number of iterations N should be chosen large enough so as to reach a reasonably small value for this bound, in order to well approximate the actual expected return, but not too large to avoid useless computation time. TABLE 2.1 illustrates the value of this bound depending on the number of iterations.

It has been decided to use $N = 3000$ as the magnitude of the error is sufficiently small to be negligible in that case, while the number of iterations is still at the order of the thousand and leads to a quite low computing time.

As a result, $J^{\mu}_{3000}(x)$, for each initial state x, in the deterministic domain, is displayed in the TABLE 2.2, with the upper left cell corresponding to state $(0, 0)$. One can observe that the expected return of the policy is maximum for initial state (4,0), where the reward signal is the largest and equal to 19. One can also note that, in a given line, all cells present scores close to each other; this is related to the chosen policy, which prevents the agent from changing its line during its trajectory.

1839.62	1857.19	1881	1900	1900
990.04	997.01	999	1000	1000
-779.3	-779.09	-791	-800	-800
-471.57	-467.24	-476	-500	-500
849.37	875.12	888	900	900

TABLE 2.2: Expected return of the stationary policy over all the state space in the deterministic domain

The resulting expected return in the stochastic domain is completely different, as it can be seen in TABLE 2.3.

-72.02	-71.46	-64.25	-54.75	-54.75
-67.78	-64.91	-64.16	-63.66	-63.66
-77.41	-73.25	-76.99	-81.49	-81.49
-75.35	-68.08	-66.52	-78.52	-78.52
-82.34	-74.12	-70.65	-64.65	-64.65

TABLE 2.3: Expected return of the stationary policy over all the state space in the stochastic domain

In the stochastic case, the reward signal has a value of -3 when a "jump" back to state (0 0) occurs, *i.e.* when w, which is uniformly distributed on $[0, 1]$, is greater than 0.5. This can explain the low (negative) scores observed in this case.

3 Optimal policy

The corresponding Python code can be found in the file `section3.py`.

In order to derive the optimal policy, one first needs to estimate the parameters of the Markov decision process (MDP) associated with the domain. An MDP is based on some transition probabilities $p(x'|x,u)$ and a reward function $r(x,u)$. The transition probabilities express the probability of reaching the state x' from the state x taking the action u. The reward function is simply the reward that the agent will acquire by taking the action u from the state x.

In the deterministic domain, the transition probability associated to a future state x' and an action u performed from a state x is either 1 or 0, as there is only one state x' reachable from the state x through the action u. Therefore, one can again get rid of the mathematical expectation over w for this case. On the contrary, in the stochastic domain, there is an uncertainty on the reached state due to the stochastic dynamics of the domain, and the conditional probabilities are derived taking into account the probability of "jumping" to the state (0,0).

Similarly, the reward is either perfectly known in the deterministic domain, in which case one can again get rid of the mathematical expectation over w, or it is a random variable if the domain is stochastic.

Then, the Q-function is computed using the recurrence equation

$$Q_N(x,u) = E\{r(x,u,w) + \gamma \max_{u' \in U} Q_{N-1}(f(x,u,w),u')\}, \quad \forall N \geq 1 \tag{3.1}$$

with $Q_0(x,u) = 0$. The state-action value functions $Q_N(x,u)$ are computed for each initial state x and each possible action u by the computation of N iterations of the recurrence equation.

It can be shown that when N goes to infinity, $Q_N(x,u)$ tends to the Q-function in infinite norm[2], defined as the unique solution of the Bellman equation which allows to find the optimal control policy $\mu^*(x)$, as a control policy $\mu^*(x)$ is optimal if and only if $\mu^*(x) \in \arg\max_{u \in U} Q(x,u)$. The associated value function can be found as well, using $J_{\mu^*} = \max_{u \in U} Q(x,u)$.

A theoretical upper bound on the suboptimality of μ_N^* with respect to μ^* allows to choose a value of N large enough to provide a good approximation, but not excessively large in order to avoid useless computation time. Equation (3.2) defines this bound, and it is computed in Table 3.1 for different values of N.

$$\|J^{\mu_N^*} - J^{\mu^*}\|_\infty \leq \frac{2\gamma^N}{(1-\gamma)^2} B_r. \tag{3.2}$$

N	Bound
10	343665.18
100	193092.29
1000	16.41
3000	3.05×10^{-8}
10000	8.55×10^{-39}

Table 3.1: Value of the bound for different number of iterations

Again, as in the previous section, a value $N = 3000$ is chosen, as it leads to a good accuracy while implying acceptable computing time.

[2]Alternatively, one can say that the infinite norm error tends towards 0.

For the deterministic case, the results are presented in Table 3.2 and Table 3.3.

1842.03	1857.19	1881	1900	1900
1854.58	1870.28	1881.09	1891	1900
1842.03	1855.58	1870.28	1881.09	1881
1828.61	1849.01	1863.65	1863.28	1864.09
1816.32	1826.52	1849.01	1863.65	1842.01

Table 3.2: Expected return of the optimal policy in the deterministic domain for all states

↓	→	→	→	→
→	→	→	→	↑
↑	→	↑	↑	↑
↑	→	→	↑	↑
↑	→	↑	↑	←

Table 3.3: Optimal policy in the deterministic domain for all states

One can observe that, whatever its initial state, the optimal policy brings the agent towards the upper right state, that is the one corresponding to the highest reward, and makes it stay there afterwards. It is also obvious that the expected returns are way better than for a naive rule-based policy as used in the previous sections.

For the stochastic case, the results are presented in Table 3.4 and Table 3.5.

159.45	159.64	163.05	172.13	172.13
159.64	163.05	164.9	167.63	172.13
159.45	160.14	163.05	167.21	167.63
159.26	162.2	167.21	162.2	167.21
159.26	155.71	162.2	167.21	162.23

Table 3.4: Expected return of the optimal policy in the stochastic domain for all states

↓	↓	↓	→	→
→	→	→	→	↑
↑	→	↑	↓	↑
←	→	→	←	←
↑	→	↑	↑	↓

Table 3.5: Optimal policy in the stochastic domain for all states

One can observe that the optimal policy is different from the deterministic one, and no longer necessarily brings the agent to the upper right state. This could be explained by the fact that, since at any step the agent can "jump" to the state (0,0) (which has a negative associated reward), the optimal behaviour consists in trying to maximize the short-term[3] return rather than planning a long-term strategy to reach the best state in terms of reward. It is also obvious that the expected returns are way better than for a naive rule-based policy as used in the previous sections. One however

[3]Related to the next few steps.

observes that, quite logically, these expected returns are smaller than for the deterministic case, as there is a 0.5 probability to jump to the state (0,0) at any step, and therefore to get a -3 reward.

To determine the smallest value of N, N_{min}, such that a greater value does not change the policy inferred from the last Q-function of the sequence, one repeats the previously described process for finding the optimal policy for growing values of N, starting from $N = 1$. Once N is such that the derived optimal policy is not different anymore from the one derived for $N - 1$, one has found $N_{min} = N - 1$.

The results of this experience can be found in TABLE 3.6. One has observed that for $N > 7$, the same policy is obtained as for $N = 7$; hence, for both types of domain, $N_{min} = 7$. Indeed, one can compare TABLE 3.3 with TABLE 3.6m, as well as TABLE 3.5 regarding TABLE 3.6n and see that there is no difference of policy for each state as soon as $N = 7$. For each type of domain, the derived policy is thus the optimal one for $N \geq 7$.

↓	↓	↓	→	→
←	→	→	→	↑
↓	←	↑	↓	↑
←	←	→	←	←
↑	→	↑	↑	↓

(a) N=1, deterministic domain

↓	↓	↓	→	→
←	→	→	→	↑
↓	←	↑	↓	↑
←	←	→	←	←
↑	→	↑	↑	↓

(b) N=1, stochastic domain

↓	↓	→	→	→
←	→	→	→	↑
↓	→	↓	↓	↑
←	→	→	←	←
↑	→	↑	↑	↓

(c) N=2, deterministic domain

↓	↓	↓	→	→
←	→	→	→	↑
↓	←	↓	↓	↑
←	→	→	←	←
↑	→	↑	↑	↓

(d) N=2, stochastic domain

↓	↓	→	→	→
→	→	→	→	↑
↓	→	↓	↓	↑
←	→	→	←	←
↑	→	↑	↑	↓

(e) N=3, deterministic domain

↓	↓	↓	→	→
←	→	→	→	↑
↓	→	↑	↓	↑
←	→	→	←	←
↑	→	↑	↑	↓

(f) N=3, stochastic domain

↓	→	→	→	→
→	→	→	→	↑
↑	→	↓	↑	↑
←	→	→	←	←
↑	→	↑	↑	←

(g) N=4, deterministic domain

↓	↓	↓	→	→
←	→	→	→	↑
↓	→	↑	↓	↑
←	→	→	←	←
↑	→	↑	↑	↓

(h) N=4, stochastic domain

↓	→	→	→	→
→	→	→	→	↑
↑	→	↑	↑	↑
→	→	→	↑	←
↑	→	↑	↑	↓

(i) N=5, deterministic domain

↓	↓	↓	→	→
←	→	→	→	↑
↓	→	↑	↓	↑
←	→	→	←	←
↑	→	↑	↑	↓

(j) N=5, stochastic domain

↓	→	→	→	→
→	→	→	→	↑
↑	→	↑	↑	↑
←	→	→	↑	↑
↑	→	↑	↑	←

(k) N=6, deterministic domain

↓	↓	↓	→	→
→	→	→	→	↑
↓	→	↑	↓	↑
←	→	→	←	←
↑	→	↑	↑	↓

(l) N=6, stochastic domain

↓	→	→	→	→
→	→	→	→	↑
↑	→	↑	↑	↑
↑	→	→	↑	↑
↑	→	↑	↑	←

(m) N=7, deterministic domain

↓	↓	↓	→	→
→	→	→	→	↑
↑	→	↑	↓	↑
←	→	→	←	←
↑	→	↑	↑	↓

(n) N=7, stochastic domain

TABLE 3.6: On the left, the evolution of the optimal policy in the deterministic domain for N=1 to 7. On the right, the evolution of the optimal policy in the stochastic domain for N=1 to 7