LIÈGE université
Sciences Appliquées

Master in applied sciences
Computer science and engineering
Year 2018-2019

LIÈGE université

Introduction to Complex problem solving

Assignement 2

INFO8003-1
Damien Ernst

Olivier Moitroux
s152122

Liège

Février 2019

Domain representation

The **origin** of the grid has been put in the **bottom left** of the board so that, in the provided domain instance of the assignment, the red cell is considered to be at position (0,1). The value of y increases as we go up (0,1) while the value of x increases by going to the right (1,0). The value of the reward `grid`[0, 0] is thus considered to be -20.

-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

If more information relative to the implementation is expected, please refer to more details explanation in the assignment 1.

4 Optimal policy

The routine that compute $p(x'|x,u)$ is called `transitionProb()` while the one used to get the reward is called `rewardFun()`. They compute respectively

$$r(x,u) = E_{w\ P_w(|x,u)}[r(x,u,w)]\ \forall x \in X, u \in U \tag{1}$$

$$p(x'|x,u) = E_{w\ P_w(|x,u)}[I_{\{x'=f(x,u,w)\}}]\ \forall x, x' \in X, u \in U \tag{2}$$

The recurrent equation defining the functions Q_N can be written:

$$Q_N(x,u) = r(x,u) + \gamma \sum_{x' \in X} p(x'|x,u) \max_{u' \in U} Q_{N-1}(x',u'),\ \forall N \geq 1 \tag{3}$$

$$Q_0(x,u) \equiv 0 \tag{4}$$

and can be easily computed with the above routines. The code does not does not store the final state-action value function as it was not explicitly asked[1] but returns rather the expected cumulative reward of each cells in the grid and the corresponding optimal policy. Figure 1 displays the results obtained with N=500 and table 1 and 2 the corresponding optimal policy in both settings.

5 System identification

Estimation of Q and of an optimal policy

The next step was to compute the estimation of the transition probability $p(x'|x,u)$ and the reward $\hat{r}(x,u)$. In order to estimate these values, one has to first generate a trajectory of given

[1] Not difficult at all to get or display it though

```
J:
[[1831.5299 1846.5833 1870.3933 1889.3933 1889.3933]
 [1843.9696 1859.6724 1870.4833 1880.3933 1889.3933]
 [1831.4238 1844.9696 1859.6724 1870.4833 1880.3933]
 [1818.0035 1838.4031 1853.0396 1852.6724 1853.4833]
 [1805.7174 1815.9130 1838.4031 1853.0396 1831.5092]]
```

(a) Deterministic

```
J:
[[296.6330 301.1331 314.2903 329.4903 329.4903]
 [301.1257 309.8380 315.7879 322.2903 329.4903]
 [296.6255 301.9257 309.8380 315.7879 322.2903]
 [292.2614 301.6505 309.4905 304.2380 309.4979]
 [289.6032 289.0412 301.6505 309.4905 296.0579]]
```

(b) Stochastic ($\beta = 0.2$)

Figure 1 – $J^{500}_{\mu^*}$

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

Table 1 – μ^* for N=500 (deterministic)

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

Table 2 – μ^* for N=500 (stochastic, $\beta = 0.2$)

```
J:
[[1831.5299 1846.5833 1870.3933 1889.3933 1889.3933]
 [1843.9696 1859.6724 1870.4833 1880.3933 1889.3933]
 [1831.4238 1844.9696 1859.6724 1870.4833 1880.3933]
 [1818.0035 1838.4031 1853.0396 1852.6724 1853.4833]
 [1805.7174 1815.9130 1838.4031 1853.0396 1831.5092]]
```

(a) Deterministic

```
J:
[[1544.2555 1525.8129 1562.8190 1542.1909 1887.5161]
 [1565.2129 1578.1145 1565.3334 0.0000 1887.5161]
 [1606.6124 1617.5859 1833.9652 1851.6061 1878.5161]
 [1616.7802 1652.2361 1755.3303 1741.7770 1562.8190]
 [1598.8071 1639.3686 1674.5903 1653.8444 1634.3060]]
```

(b) Stochastic ($\beta = 0.2$)

Figure 2 – Estimation $\hat{J}^{500}_{\mu^*}$

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

Table 3 – Deterministic setting

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

Table 4 – Stochastic setting ($\beta = 0.2$)

length. By looking at the speed of convergence (see next section), a length of 500 seemed to be a good idea. The function `buildRandTrajectory` create a plausible trajectory by applying a random policy on a starting cell (that can be chosen randomly).

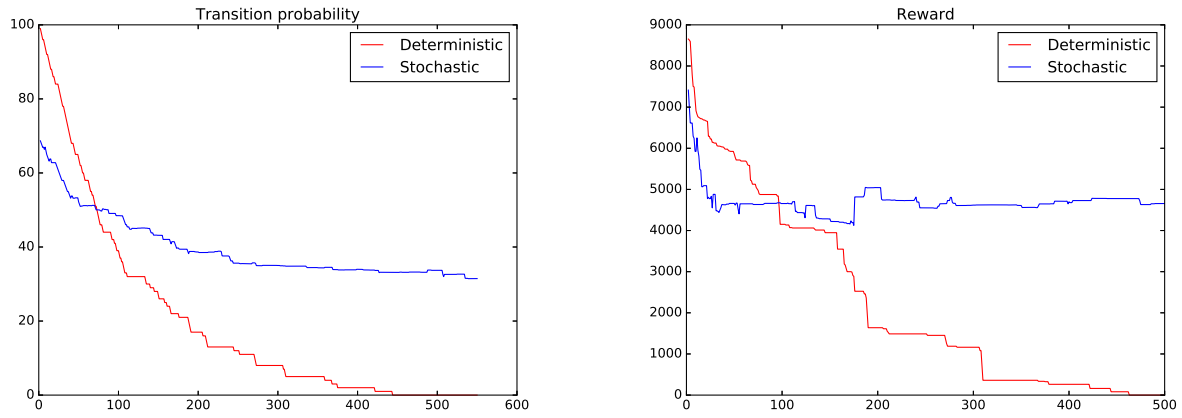
Figure 3 displays the expected cumulative reward obtained in each cell by following an optimal policy and a trajectory of length equal to 500. It appears that the deterministic estimation is perfect while the stochastic one is poor. Looking at the convergence speed in the next section will bring a new light on these results. By extracting the policy from $\hat{Q}$, we observe no difference with the real optimal policy in the deterministic case. However, the stochastic estimation performs not very well.

Analysis of the speed of convergence

For increasing values of the size of the trajectory $h = (x_0, u_0, r_0, \ldots, x_t)$, an estimation of the transition probability $p(x'|x, u)$ and the reward $\hat{r}(x, u)$ have been computed for each possible states. The trajectory was directly generated with its final length and starting from an inital random cell. Afterwards, only a subpart of it was considered during the process to limit the performance hit. Then, I reported the sum of the square error made between these estimations and their real values (cfr. Q4) for each length of the trajectory to plot them.

The result is figure 5 which highlights the fact that the estimation estimates perfectly the transition probability (of each possible combination of states and actions) when the trajectory reached a length of 450. The convergence decrease in exponential way. However, the square error seems to decrease slowyer and to plateau at around 38. Looking at the reward, the same behaviour can be observed for the deterministic case as for the the transition probability. Again, the reward in the stochastic domain presents an offset that is non-negligible.

A proposed explanation of this problem (offset in the convergence) is the fact that, starting from a given cell near [0, 0] in a stochastic setting, some cells that are at the opposite of the reference [0, 0] geographically might have little chances to be reached due to the defined dynamic. This way, the algorithm may not have enough information gathered in the trajectory about these cells to compute accurate estimations. A workaround would be to consider several smaller trajectories that start at a random cell in the grid to increase the gathering of information. As beta is increased, this phenomenon is expected to be more severe.



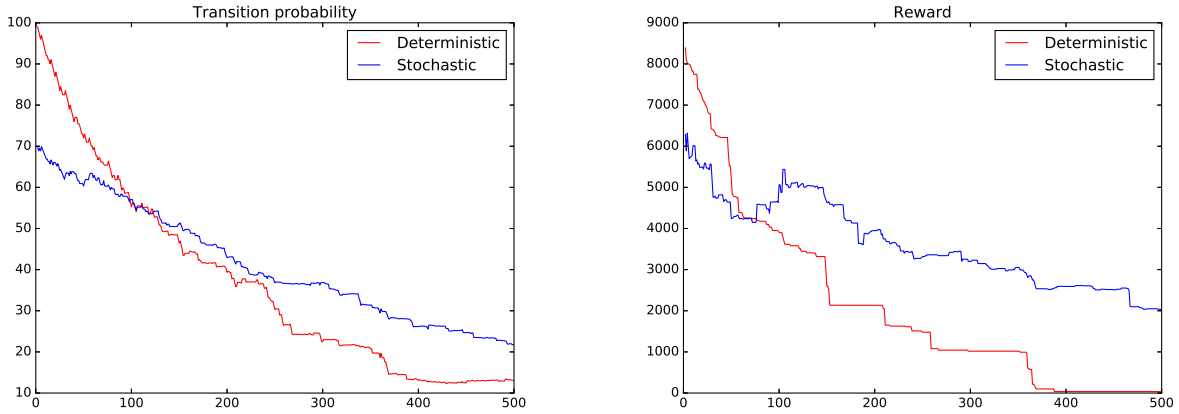
(a) Deterministic

(b) Stochastic ($\beta = 0.2$)

Figure 3 – Evolution of the **square error** of the estimation **over N**, the length of the trajectory.

In order to further investigate the reasons of such an offset for the stochastic settings, I made a simple test that concatenate many small trajectories together to form a global (but unrealistic [2]) trajectory. The hope is to highlight the fact that indeed, the lack of information in hard to reach cells causes the offset. Like before, the estimation is computed by increasing the number of time steps considered in the global trajectory. The results of the test with 100 trajectories of length 5 concatenated are as follows:

[2]Indeed, as these small trajectories have their starting point drawn randomly and that these small trajectories are concatenated.



(a) Deterministic

(b) Stochastic ($\beta = 0.2$)

Figure 4 – Evolution of the **square error** of the estimation **over N**, the length of the global trajectory that is the result of the concatenation of 100 trajectories of length 5.

All in all, it appears that the gain in information greatly reduces the offset in both settings. This gain is due to a deeper exploration of the states via several trajectories starting in different random positions. It should be noted that this last test consider a global trajectory that is not realistic and is just motivated to better analyse the reasons of the behaviour highlighted in figure 3.