

Reinforcement Learning

Exercise 6 - Solution

Jonathan Schnitzler - st166934

Eric Choquet - st160996

June 17, 2024

1 Planning and Learning

a) Why did Dyna-Q+ perform better in both test phase Lets start with the **second phase**. For the phase of the suddenly appearing shortcut in the beginning of the wall, Dyna-Q+ is able to exploit the gained benefit faster, since it generates a higher reward since the state was not visited for a long time. On the other hand Dyna-Q remains on its fixed strategy which proofed to perform better for a longer period of time(i.e. the hole on the left of the wall) and therefore remains slow.

Lets consider the **first phase** and the tricky question: Why doesn't it cost Dyna-Q+ to explore its environment? This can be explained by living in a grid universe and one can not exploit walking diagonal. Therefore, each path to the hole in the wall is of the same length. Dyna-Q+ performs initially better, since it tends to explore the terrain and finds the hole in the wall earlier.

b) Tabular Dyna-Q algorithm Adaptations in order to include stochastic environments could be achieved by implementing a stochastic process in the model itself. This can be done in multiple ways, I will present two approaches here. Either, probability is directly sampled by occurrence

$$Model(S, A) \leftarrow R_i, S'_i \quad \text{for } i = 1, \dots, N \quad (1)$$

$$Model(S, A) := \begin{cases} R_1, & x < p_1 \\ \vdots & \\ R_j, & x < \sum_{i=1}^j p_i \\ \vdots & \\ R_n, & x < 1 \end{cases} \quad (2)$$

where $p_j = \frac{\#R_j}{N}$ and x is a random number drawn from a uniform distribution over the interval $[0, 1]$. Alternatively one can use a Kernel-interpolation, e.g.

with a gaussian Kernel from each Reward (and State if they are also continuous otherwise either do first method or floor to integer representation).

Does this still perform well on changing environments? - If not how could it? The problem is that in the planning phase the Q value for the cost of the state action, could be reduced to such an extent, that it doesn't visit the state again. This can again be solved by using a term in the reward to make state which haven't been visited for a long time more attractive via Dyna-Q+.

2 Monte Carlo Tree Search on the Taxi environment

a) Performance of the MCTS algorithm In the accompanying script `ex06_plan.py` a monte carlo tree approach is implemented. As a test, the Taxi environment is used. It consists of a yellow cab and four zones where the cab can pick up and drop off passengers. The cab can move in four directions (up, down, left, right) and can pick up and drop off passengers. The goal is to pick up a passenger at a certain location and drop him off at another location. The reward is given by the distance driven and the number of steps taken. The algorithm is tested with different values for the number of iterations `maxiter` and repeated three times for stability. The results are shown in figure 1.

```
finished run 1 with reward: -605.0
finished run 2 with reward: -632.0
finished run 3 with reward: -515.0
mean reward: -515.0
finished run 1 with reward: -848.0
finished run 2 with reward: -812.0
finished run 3 with reward: -821.0
mean reward: -821.0
finished run 1 with reward: -839.0
finished run 2 with reward: -320.0
finished run 3 with reward: -767.0
mean reward: -767.0
finished run 1 with reward: -830.0
finished run 2 with reward: -686.0
finished run 3 with reward: -704.0
mean reward: -704.0
finished run 1 with reward: -731.0
finished run 2 with reward: -839.0
finished run 3 with reward: -812.0
mean reward: -812.0
finished run 1 with reward: -794.0
finished run 2 with reward: -767.0
finished run 3 with reward: -839.0
mean reward: -839.0
[-515.0, -821.0, -767.0, -704.0, -812.0, -839.0]
```

Figure 1: Output for Trees with `maxiter` = [10, 20, 50, 100, 200, 500]

Another way to display this is via an table

maxiter	1st Run	2nd run	3rd run	average
10	-605	-632	-615	-617
20	-848	-812	-821	-827
50	-839	-320	-767	-642
100	-830	-666	-704	-733
200	-731	-839	-812	-794
500	-794	-767	-839	-800

Table 1: Tabular Version of terminal output

b) Depth evaluation of trees To our surprise the tree doesn't seem to grow with the number of iterations. It stays roughly stable at a depth of 7. This is shown in figure 2. This is a bit surprising, since one would expect the tree to grow with the number of iterations.

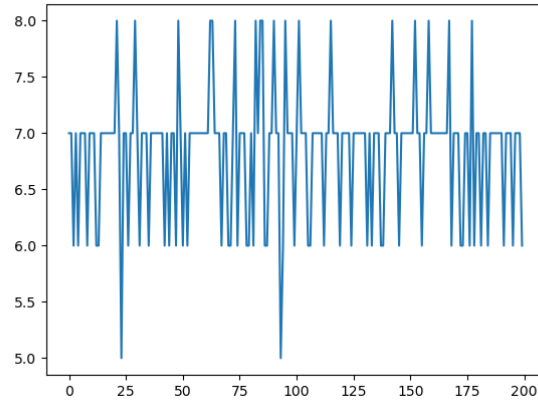


Figure 2: depth evaluation for Trees with `maxiter` = 500