Voluntary Programming Assignment: Reinforcement Learning

The task in this assignment is to select and implement a reinforcement learning algorithm and apply it to the standard testbed “Mountain Car”. The goal is to draw a heat-map of the two dimensional value function. The exact RL learning algorithm you use to compute it is up to you to choose (and might influence your experience).

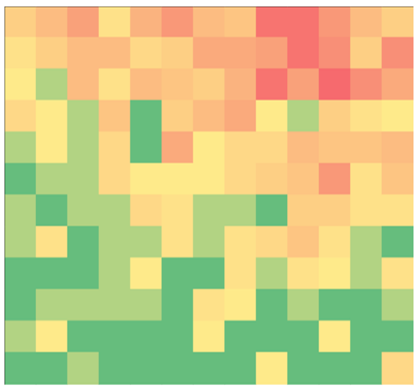
Mountain Car is a simplification of simulated car on a hill that is too steep to drive up without any initial speed. The task was an early favourite benchmark in reinforcement learning because the optimal solution requires that the car first drives away from the goal, misleading algorithms such as A*, at least initially. The state of the car is two dimensional, consisting of its position and speed. The position ranges from -1.2 to 0.6, although episodes end as soon as the car reaches a position greater than 0.5. The speed of the car ranges from -0.07 to 0.07. The actions that the car can execute are full throttle ahead, full throttle reversed or no throttle at all. Full throttle increases or decreases the speed by 0.001. The hill is sine-shaped and gravity pulls with a 0.0025 force. There is no friction caused by tires or air. The exact computation can be found in the example code supplied. The rather depressing reward is defined as -1 for each time step until the episode ends. Hence, the goal is to leave as soon as possible.

There are two options readily available for you to interface with this environment. The first and recommended choice is through OpenAI Gym (https://gym.openai.com). This is a Python interface for reinforcement learning environments, including the old benchmarks such as Mountain Car, but also more interesting ones such as the Arcade Learning Environment. You can find how to install gym on your Mac OS or Linux system at https://gym.openai.com/docs. On Canvas, I provide some example Python code that generates interaction episodes using random action selection.

The second option is to interface with the Java version of the environment implemented by yours truly. The source of this code is also available on Canvas. It should not be treated as a good example of Java coding, but it seemed to pass most stress tests over the last few years. It does seem to run faster than the Gym version if you turn off the graphical display on both.

The third option is to re-implement the environment yourself in your language of choice: Matlab, R, Prolog, ... Without graphics, this is really easy.

Then you need to decide on a reinforcement learning algorithm. Although the actions are discrete, the state space is in principle continuous. This means you can either decide to work with this as is and use some kind of generalisation technique, or discretise the state space. You’re free to make your own choice for an algorithm, but keep in mind that you need to visualise the state-value function at the end. Also, it might be worth considering that you have access to the inner workings of the environment because you have access to the code implementing it.



If you find that there is little information in your visualisation, think about which parameters you could change to make it more illustrative of what is going on.

Handing In (= not required)

First and foremost, this assignment is designed to help you porcess the material from the first two lectures and make lecture 3 more enjoyable for you. However, if you want, you can send in your heatmap through email to `kurt.driessens@maastrichtuniversity.nl` and we can discuss your experience and result. It might be good to mention which algorithm you use and explain how you dealt with the continuous state space and how you constructed the visualisation of the state-value function in your email.