MVA: Reinforcement Learning (2020/2021) **Homework 1**

Dynamic Programming

Lecturers: *A. Lazaric, M. Pirotta* *(October 27, 2020)*

Solution by Clément Bonnet

1 Question

1.1

Finding the shortest path to state 14 corresponds to a deterministic policy. The reward r_s has to be negative to ensure the shortest path goal while not being too low to prevent the agent from visiting red terminal state 1. Precisely, r_s must respect: $-10 < r_s < 0$.
Let us define $\boxed{r_s = -1}$. Since the transitions are deterministic, the optimal policy is indeed the shortest path to state 14. For such rewards, the value function of the optimal policy for each state is the following.

5	-10	7	6
6	7	8	5
7	8	9	4
8	9	10	3

1.2

In a general MDP, a policy π induces a value function V_1^π for a reward signal $r_1(s,a)$. Let us apply an affine transformation to the reward r_1 of the form: $r_2(s,a) = \alpha r_1(s,a) + \beta$ with $(\alpha, \beta) \in \mathbb{R}^2$. For the same policy π, the new value function V_2^π is thus the following:

$$\begin{aligned} V_2^\pi(s) &= \mathbb{E}\Big[\sum_{t=0}^{+\infty} \gamma^t r_2(s_t, d_t(h_t)) | s_0 = s; \pi\Big] \\ &= \mathbb{E}\Big[\sum_{t=0}^{+\infty} \gamma^t (\alpha r_1(s_t, d_t(h_t)) + \beta) | s_0 = s; \pi\Big] \\ &= \alpha \mathbb{E}\Big[\sum_{t=0}^{+\infty} \gamma^t r_1(s_t, d_t(h_t)) | s_0 = s; \pi\Big] + \sum_{t=0}^{+\infty} \gamma^t \beta \end{aligned}$$

Therefore, after an affine transformation of the reward signal, the new value function becomes:

$$\boxed{V_2^\pi(s) = \alpha V_1^\pi(s) + \frac{\beta}{1-\gamma}}$$

Let $\pi_1^\star$ be the optimal policy corresponding to reward r_1 and $\pi_2^\star$ the optimal policy after the affine

transformation.

$$\pi_1^\star(s) \in \arg\max_\pi V_1^\pi(s)$$

$$\pi_2^\star(s) \in \arg\max_\pi V_2^\pi(s) \iff \pi_2^\star(s) \in \arg\max_\pi \alpha V_1^\pi(s) + \frac{\beta}{1-\gamma} \iff \boxed{\pi_2^\star(s) \in \arg\max_\pi [\text{sign}(\alpha) V_1^\pi(s)]}$$

If $\alpha > 0$, the optimal policy is not changed. However, if $\alpha \leq 0$, it can be modified in the general case. In conclusion, the optimal policy is invariant to positive affine transformation of the reward function. Yet, it is not invariant to negative affine transformation.

1.3

In the setting of question 1.1, the Markov Decision Process is episodic and undiscounted. Let us modify the reward function with an additive term $c = 5$ on r_s. This gives us a reward $r_s := r_s + c$. If we were to choose $-10 < r_s < 5$, the optimal policy wouldn't change.
Yet, we chose $r_s = -1$. After adding the positive term c, one obtains $r_s = 4 > 0$. Thus an agent can obtain an infinite amount of reward by going back and forth between two non-terminal states whose rewards are r_s. The optimal policy is not the shortest path to state 14 anymore, which would only produce a finite amount of reward.
Therefore, the optimal policy is not preserved and the new value function is the following:

$+\infty$	-10	$+\infty$	$+\infty$
$+\infty$	$+\infty$	$+\infty$	$+\infty$
$+\infty$	$+\infty$	$+\infty$	$+\infty$
$+\infty$	$+\infty$	10	$+\infty$

2 Question

Starting from the definition of the infinite norm,

$$\begin{aligned}
\|V^\star - V^{\pi_Q}\|_\infty &= \max_s |V^\star(s) - V^{\pi_Q}(s)| \\
&\leq \max_s \Big[|Q^\star(s, \pi^\star(s)) - Q(s, \pi_Q(s))| + |Q(s, \pi_Q(s)) - \mathcal{T}^{\pi_Q} V^\star(s)| + |\mathcal{T}^{\pi_Q} V^\star(s) - V^{\pi_Q}(s)| \Big] \\
&\leq \max_s |\max_a Q^\star(s,a) - \max_a Q(s,a)| + \max_s |Q(s, \pi_Q(s)) - Q^\star(s, \pi_Q(s))| \\
&\quad + \max_s |\mathcal{T}^{\pi_Q} V^\star(s) - \mathcal{T}^{\pi_Q} V^{\pi_Q}(s)| \\
&\leq \max_s \max_a |Q^\star(s,a) - Q(s,a)| + \max_s \max_a |Q^\star(s,a) - Q(s,a)| + \|\mathcal{T}^{\pi_Q} V^\star - \mathcal{T}^{\pi_Q} V^{\pi_Q}\|_\infty \\
&\leq 2\|Q^\star - Q\|_\infty + \gamma \|V^\star - V^{\pi_Q}\|_\infty
\end{aligned}$$

Therefore, one obtains the following inequality on $\|V^\star - V^{\pi_Q}\|_\infty$:

$$\|V^\star - V^{\pi_Q}\|_\infty \leq \frac{2\|Q^\star - Q\|_\infty}{1-\gamma}$$

If one majorizes $V^\star(s) - V^{\pi_Q}(s)$ by $\|V^\star - V^{\pi_Q}\|_\infty$, the wanted inequality is directly derived.

$$V^\star(s) - V^{\pi_Q}(s) \leq \frac{2\|Q^\star - Q\|_\infty}{1-\gamma}$$

$$\boxed{V^{\pi_Q}(s) \geq V^\star(s) - \frac{2\|Q^\star - Q\|_\infty}{1-\gamma}}$$

3 Question

$$g^{\pi'} = \sum_s \mu^{\pi'}(s) \sum_a \pi'(a|s) r(s,a)$$

Using the Bellman equation in the average reward setting, one can express the $r(s,a)$ as a function of Q^π.

$$\forall (s,a), \quad r(s,a) = Q^\pi(s,a) + g^\pi - \sum_{s'} p(s'|s,a) \sum_{a'} \pi(a'|s') Q^\pi(s',a')$$

Replacing $r(s,a)$ in the previous equation of $g^{\pi'}$, one obtains a relation between $g^{\pi'}$ and g^π.

$$\begin{aligned}
g^{\pi'} &= \sum_s \mu^{\pi'}(s) \sum_a \pi'(a|s) \big[Q^\pi(s,a) + g^\pi - \sum_{s'} p(s'|s,a) \sum_{a'} \pi(a'|s') Q^\pi(s',a') \big] \\
g^{\pi'} &= \sum_s \mu^{\pi'}(s) \sum_a \pi'(a|s) Q^\pi(s,a) \\
&\quad + g^\pi \sum_s \mu^{\pi'}(s) \sum_a \pi'(a|s) \\
&\quad - \sum_s \mu^{\pi'}(s) \sum_a \pi'(a|s) \sum_{s'} p(s'|s,a) \sum_{a'} \pi(a'|s') Q^\pi(s',a')
\end{aligned}$$

Thus, one can isolate $g^{\pi'} - g^\pi$ since $\sum_s \mu^{\pi'}(s) \sum_a \pi'(a|s) = 1$,

$$\begin{aligned}
g^{\pi'} - g^\pi &= \sum_s \mu^{\pi'}(s) \sum_a \pi'(a|s) Q^\pi(s,a) \\
&\quad - \sum_s \mu^{\pi'}(s) \sum_a \pi'(a|s) \sum_{s'} p(s'|s,a) \sum_{a'} \pi(a'|s') Q^\pi(s',a') \\
&= \sum_s \mu^{\pi'}(s) \sum_a \pi'(a|s) Q^\pi(s,a) \\
&\quad - \sum_{s'} \sum_{a'} \pi(a'|s') Q^\pi(s',a') \sum_s \mu^{\pi'}(s) \sum_a \pi'(a|s) p(s'|s,a)
\end{aligned}$$

Since $\mu^{\pi'}$ is a stationary distribution, $\forall s'$,

$$\begin{aligned}
&\mu^{\pi'}(s') \\
=& \sum_s \mu^{\pi'}(s) p^{\pi'}(s'|s) \\
=& \sum_s \mu^{\pi'}(s) \sum_a \pi'(a|s) p(s'|s,a)
\end{aligned}$$

This gives us:

$$\begin{aligned}
g^{\pi'} - g^\pi &= \sum_s \mu^{\pi'}(s) \sum_a \pi'(a|s) Q^\pi(s,a) \\
&\quad - \sum_{s'} \mu^{\pi'}(s') \sum_{a'} \pi(a'|s') Q^\pi(s',a')
\end{aligned}$$

$$\boxed{g^{\pi'} - g^\pi = \sum_s \mu^{\pi'}(s) \sum_a (\pi'(a|s) - \pi(a|s)) Q^\pi(s,a)}$$

4 Question

Let us consider a 6-story building with 2 elevators. The elevator controllers operate in continuous state spaces and continuous time. However, what matter for decision making are discrete events such as an elevator reaching a floor or a passenger pressing a button. One can discretize in time the dispatching system in which events happen at discrete times, yet the amount of time between two events remains a real value. Since two consecutive actions may take different amounts of time to complete, discounted rewards cannot be computed with a constant discount factor anymore. Moreover, the passenger arrivals are modelled as a random variable in continuous time.
Decisions are to be taken when an elevator is reaching a floor or when it is stopped at a floor. If it is arriving at a floor, it can either stop at the next floor or continue to the next floor. If it just stopped at a floor, it must either stay, move up or down. There has to be some constraints on the decisions to make. For instance, an elevator cannot pass a floor if a passenger pressed the button to get off there. It also cannot change direction until it has serviced all the floors indicated by pressed buttons in its current direction. One could also add other specific constraints such as not stopping at a floor to pick passengers up if the other elevator is already stopped there.
The reinforcement learning agent can be the global controller that can take actions regarding both elevators. Below is a detailed description of the MDP elements:

- *State space*: A conservative estimate gives around $256.2^{10}.2^{12} \approx 1.10^9$ different states, which are made of:
  - The rounded **location** and **direction** of both elevators (6 locations and 3 directions which include not moving, yet cannot move up at the top and down when at the bottom): $(6.3-2)^2 = 256$ possibilities.
  - Except top and bottom, **each floor has two buttons** (up and down) that can be pressed by passengers, that make up 10 buttons: 2^{10} possibilities.
  - **Each elevator has 6 buttons**: 2^{12} possible outcomes.

  As a remark, the state space is quite huge and dynamic programming methods may start to appear limited in such a large space. To minimize a function of the waiting time, the dispatching system must also know the amount of time passengers have been waiting. For each floor button, a counter can be implemented and stored in the state description. This would make 10 counters. If their values are approximated with 8 bits, they add up $2^{80} = 1.10^{24}$ new states for the counter. The total state space is multiplied by this amount and therefore becomes not only very large but completely out of the range of dynamic programming.

- *Action space*: The environment asks the agent for an action when an elevator is reaching a floor or when it has stopped and a button has been pressed. The possible actions are "**stop at the floor**", "**continue to the next floor**", "**move up**", "**move down**", "**do not move**". Actions are taken for one elevator at a time.

- *Dynamics and problem*: The environment is stochastic since passengers can press buttons at any moment. Their arrivals can be modelled with a geometric or Poisson law. The environment is only partially observable since for instance, one cannot know how many passengers are waiting on a floor when a button is pressed.

- *Reward*: If one wants to optimize the waiting time of passengers, one can use an increasing convex function such as the squared waiting time in order to better penalize long waiting times. The reward can thus be minus the sum of squared waiting times of passengers waiting to be served. In order to take into account energy minimization, one can add a negative reward function of the electricity consumption of each elevator. This last function can also be the squared electricity consumption between two actions. It must be a function that is bounded by the maximum power of the elevator.

5 Question

5.1

Code for `policy_iteration` is available in attached `vipi.py` file.

5.2

Code for `value_iteration` is available in attached `vipi.py` file.

5.3

One can see in figure 1 that stochasticity slows down convergence of the value iteration algorithm in the sense that it takes more iterations to converge to the optimal value function.
Stochasticity also affects the resulting optimal policy. In figure 2, one can observe that the higher the stochasticity, the further from the bottom row the optimal policy tends to get. Indeed, the bottom row triggers highly negative rewards and must be avoided. If one is too close, one might end up in one of these bad states when the environment is stochastic. Thus, the further away from these states, the better. For a very high stochasticity with $proc\_succ = 0.4$, the optimal policy even becomes quite surprising and tends to push the agent in either of top corners.

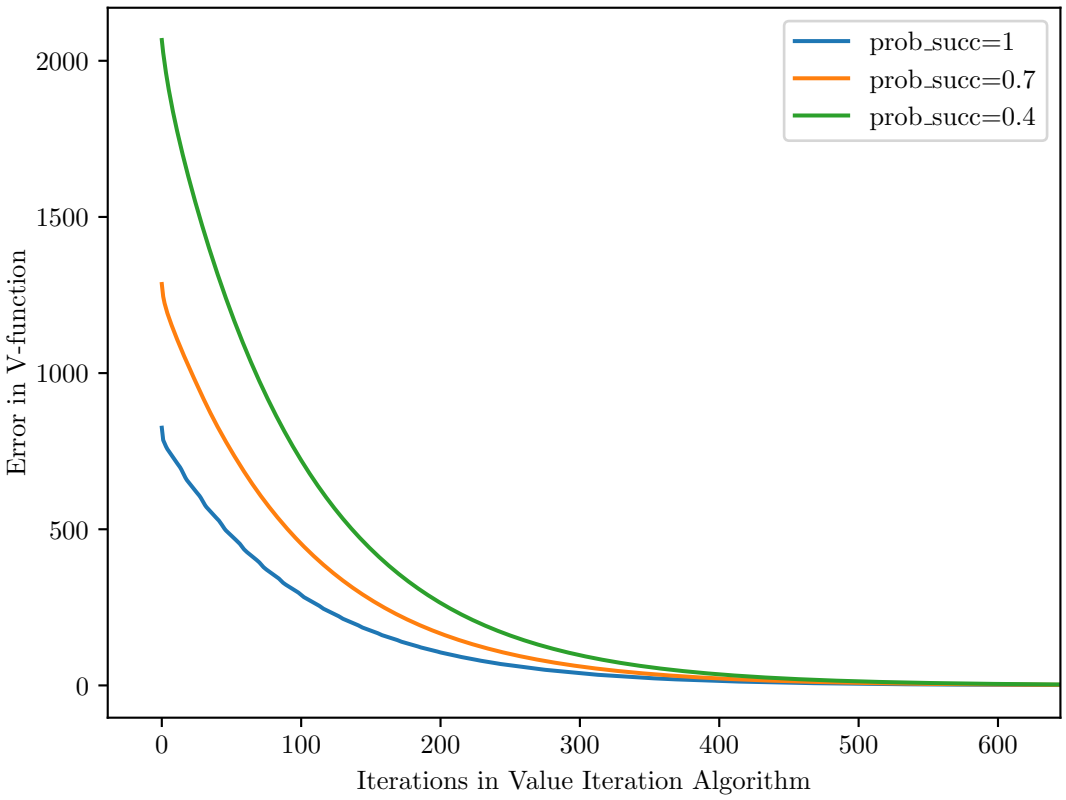

Figure 1: Comparison of convergence between the deterministic and the stochastic version of the environment.

(a) Deterministic environment, $proba\_succ = 1$

(b) Stochastic environment, $proba\_succ = 0.7$

(c) Stochastic environment, $proba\_succ = 0.4$

Figure 2: Comparison of policies for the deterministic and the stochastic version of the environment.

5.4

Policy iteration has nested loops where it runs an inner loop of policy evaluation until convergence. It requires few iterations of the outer loop until convergence, although each of these policy evaluations is time-consuming.
Value iteration has only a single loop where it constantly updates the current V-function by acting greedily without evaluating a while policy. It does not wait for the whole policy evaluation. Therefore it requires many more iterations of this single outer loop but each of them is much quicker than for policy evaluation.
Moreover, let us analyze their complexity with respect to the size of the action set. Policy iteration preforms a maximization over the action set at each iteration of the outer loop, after a policy evaluation. Thus, it computes this maximization only a few times. Whereas value iteration performs the maximization over the action set at every iteration of the single loop is has. Since it requires many more iterations to converge, maximization over the action set can become a bottleneck for value iteration when the size of the action set is big.
In our case, the action set (size 4) is much smaller than the state space (size 48). As a result, value iteration appears to be much quicker to converge. Running time of both algorithms is compared in figure 3 and table 1, where value iteration is about thirty times as fast as policy iteration after being optimized using vectorization. It must be noted that policy iteration could not take fully advantage of vectorization. Yet, even without vectorization, value iteration remains about twice as quick as policy iteration.
In conclusion, policy iteration can be faster for very large action spaces, whereas value iteration is quicker for smaller action spaces. Also, the latter generally converges faster.

(a) Using Python "for" loops

(b) Using vectorization for value iteration

Figure 3: Running time of value iteration and policy iteration in the deterministic version of the environment with the same tolerance toward convergence, best value over 20 runs.

Algorithm	Non vectorized	Vectorized
Policy iteration	959 ms	946 ms
Value iteration	594 ms	30 ms

Table 1: Running times of policy iteration and value iteration, best value over 20 runs.