# Reinforcement Learning, Assignment 1

**Alex Hermansson**
921118-4933
grobgeld@kth.se

**Gianluigi Silvestri**
940106-4614
giasil@kth.se

## Abstract

Something about the assignment

## 1 Problem 1: The Maze and the Random Minotaur

### a) Formulate the Problem as an MDP

- **State-Space**: $\mathcal{S} = (\{1,2,3,4,5,6\} \times \{1,2,3,4,5\})^2 \cup \{Done\}$. The position of the player is denoted with $\boldsymbol{p} = (p_x, p_y)$ and the position of the minotaur is denoted with $\boldsymbol{m} = (m_x, m_y)$. Therefore, for a certain time $t$, we either have a state $s_t = (\boldsymbol{p}, \boldsymbol{m})$, or a state $s_t = Done$.
- **Actions**: $\mathcal{U}$ - Up, $\mathcal{D}$ - Down, $\mathcal{L}$ - Left, $\mathcal{R}$ - Right, $\mathcal{W}$ - Wait. We assume that in any state all the actions are always allowed, and if the agent moves towards a wall it stays in the same position. This applies both to the walls inside the maze, and to the boundaries of it.
- **Time-Horizon and Objective**: The time-horizon is finite, and the objective is to optimize the expected rewards:
$$\mathbb{E}\left[\sum_{t=0}^{T-1} r(s_t, a_t) + r(s_T)\right]$$
.
- **Rewards**:
Define the goal position in the maze as $\boldsymbol{G}$. The only non-zero non-terminal reward is then given by
$$r_t(s = (\boldsymbol{p} = \boldsymbol{G}, \boldsymbol{m} \neq \boldsymbol{p}), a = \cdot) = 1,$$
The only non-zero terminal reward, for time $T$, is given by
$$r_T(s = (\boldsymbol{p} = \boldsymbol{G}, \boldsymbol{m} \neq \boldsymbol{p})) = 1,$$
- **Transition probabilities**: There are two cases in which we move to the state *Done* with probability one for all the actions, that is when the player and the minotaur are in the same position (the player dies) and when the player is on the exit while the minotaur is not (the player escapes the maze safely). The state *Done* is an absorbing state, meaning that it will remain so with probability one and all the other transitions have probability zero.
$$p_t(s' = Done \mid s = (\boldsymbol{p} = \boldsymbol{G}, \boldsymbol{m} \neq \boldsymbol{p}), a = \cdot) = 1$$
$$p_t(s' = Done \mid s = (\boldsymbol{p}, \boldsymbol{m} = \boldsymbol{p}), a = \cdot) = 1$$
$$p_t(s' = Done \mid s = Done) = 1$$

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

For the other states, since the minotaur moves independently from the player, we can write the transition probabilities as:

$$p_t(s' = (\boldsymbol{p'}, \boldsymbol{m'}) \mid s = (\boldsymbol{p}, \boldsymbol{m}), a = \cdot) = p_t(\boldsymbol{p'} \mid \boldsymbol{p}, a = \cdot) p_t(\boldsymbol{m'} \mid \boldsymbol{m})$$

Following our model of the problem, the player moves with probability one to the next position if the action is allowed (i.e. if the action is $\mathcal{R}$ and there is no wall on the right, then the player will end up in the square on the right), or if the player takes a not allowed action or the action $\mathcal{W}$, it will stay in the same position with probability one (so we can imagine that, in case of not allowed action, the player will bump into a wall and end up in the same position). To compute the transition probability to the next state, we need to multiply the probability of the next position of the player ( either one or zero) with the probability of the next position of the minotaur, that is uniform with respect to its allowed random actions (i.e. $\frac{1}{4}$ if the minotaur has four possible actions, $\frac{1}{3}$ or $\frac{1}{2}$ if the minotaur is close to one or two boundaries respectively of the maze, $\frac{1}{5}$ if all the actions are allowed and the minotaur can also wait, and so on).

**b) Solve the problem, and illustrate an optimal policy for T = 15. Plot the maximal probability of exiting the maze as a function of T. Is there a difference if the minotaur is allowed to stand still? If so, why?**

After modelling the problem as an MDP as described above, we solve the problem using Bellman's equations with time horizon $T = 15$. Thus, we first compute the optimal value function $V^*$ for the terminal states at time $t = T$:

$$\forall s_T,\ V_T^*(s_T) = r_T(s_T).$$

Then, for $t = T-1, T-2, ..., 0$, we compute the optimal value function for each state and the optimal policy $\pi_t^*$ as:

$$\forall s_t,\ V_t^*(s_t) = \max_{a \in A_{s_t}} \left[ r_t(s_t, a) + \sum_{j \in S} p_t(j \mid s_t, a) V_{t+1}^*(s_t, a, j) \right]$$

$$\pi_t^*(s_t) = \arg\max_{a \in A_{s_t}} \left[ r_t(s_t, a) + \sum_{j \in S} p_t(j \mid s_t, a) V_{t+1}^*(s_t, a, j) \right]$$

The optimal policy found for the beginning of the game, so $t = 0$, is shown in figure 1.

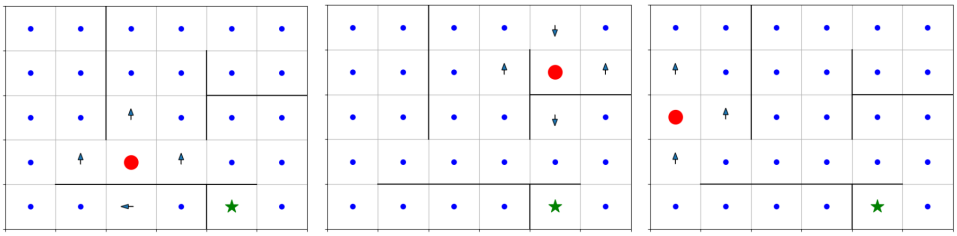

Figure 1: Optimal policy at time step $t = 0$ and different minotaur positions. The red dot is the minotaur, the star is the exit of the maze, and the small arrows indicate the optimal policy for the player, while the small blue dots corresponds to the action *Wait* for the player.

Since the probabilities for the player to exit the maze are one in almost all the positions for time $t = 0$ (so 15 steps left), and since in the way we model the problem there is no advantage in exiting the maze earlier, the player can either wait or avoid the minotaur, and still be able to exit the maze optimally.

If we show the optimal policy with time $t = 8$ (so 7 steps left), we obtain a slightly different behaviour: in the positions too far from the exit, the optimal action will be to wait, since the probability of exiting the maze is zero, and the same is for the positions very close to the exit, where the player can reach

the exit with probability one even if waiting. The different behaviour arises in the positions where the player can reach the exit in 6 or 7 steps, since sometimes moving a bit earlier can avoid potential issues with the minotaur. A few examples are shown in figure 2.

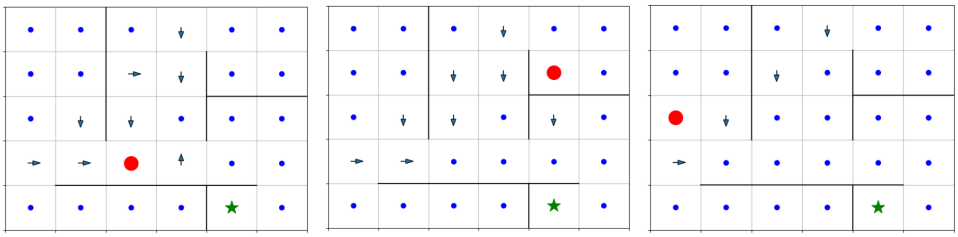

Figure 2: Optimal policy for time $t = 8$ and different minotaur positions.

With our formulation of the MDP, where the only reward is 1 if the player exits the maze, we can use the value function for the state corresponding to the initial configuration of the maze as maximal probability of exiting the maze (that we call $p_{exit}$). The probabilities obtained are $p_{exit} = 0$ for $T < 10$, $p_{exit} \approx 0.443$ for $T = 10$, and $p_{exit} = 1$ for $T > 10$. A plot is shown in figure 3.

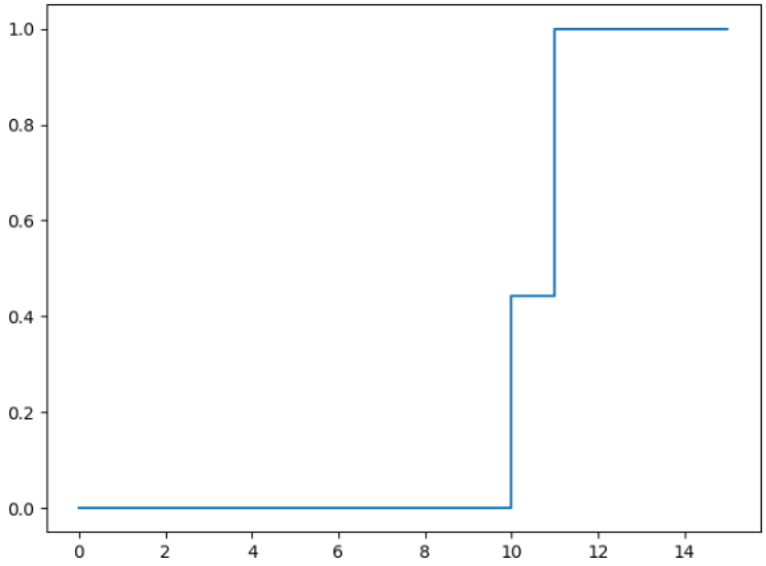


Figure 3: Maximal probability of exiting the maze as a function of T. The x-axis is the time horizon, while the y-axis is the probability.

If the minotaur is allowed to stand still, the difference is that it will generally move slower around the maze, but most importantly the player cannot use the strategy of moving towards the minotaur to avoid it with probability one if they are in adjacent cells. Moreover, the player will never have probability one of exiting the maze, since the minotaur has a small probability to stand on the exit of the maze or in other positions where the player has to go through in orther to reach the exit. We show the probability for the player to exit the maze with respect to time, starting from the initial position (figure 4), where we can see how if the time horizon is smaller than 10, then the probability is still zero, and afterwards it converges slowly to one.

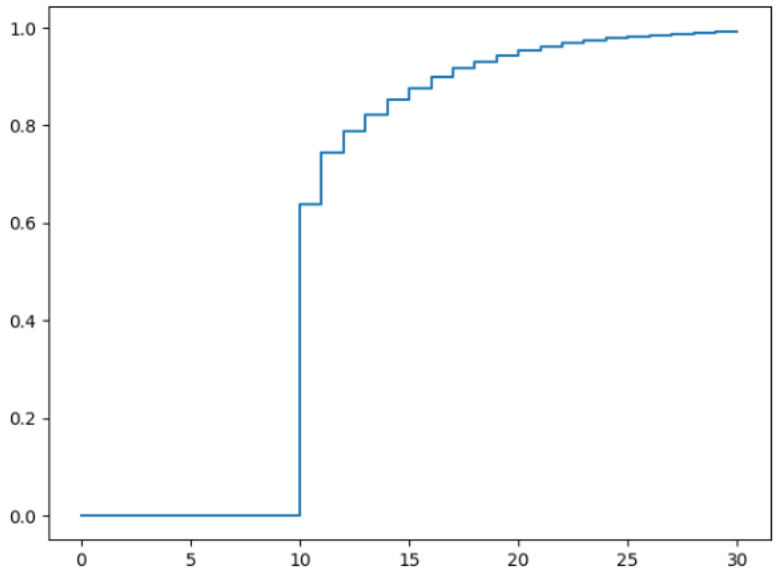


Figure 4: Maximal probability of exiting the maze as a function of T when the minotaur is allowed to stand still. The x-axis is the time horizon, while the y-axis is the probability.

We show the optimal policy for the player when the minotaur is allowed to stand still in figure 5.

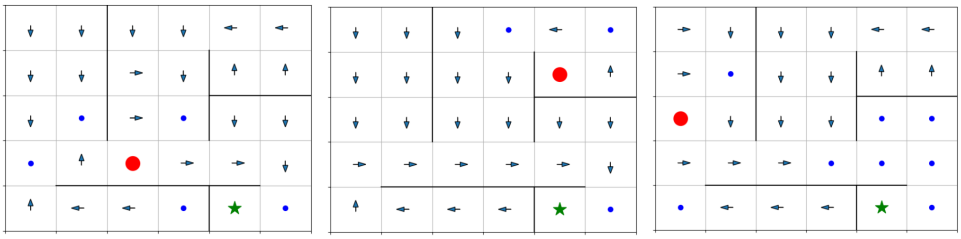

Figure 5: Optimal policy at time step $t = 0$ and different minotaur positions, when the minotaur is allowed to stand still.

**c) Assume now that your life is geometrically distributed with mean 30. Modify the problem so as to derive a policy minimising the expected time to exit the maze. Motivate your new problem formulation (model). Estimate the probability of getting out alive using this policy by simulating 10 000 games.**

If the life of the player is geometrically distributed with mean 30, we can model the problem as an MDP with infinite time horizon and discounted expected reward. Therefore, we can find an optimal policy with the Policy Iteration algorithm, using a discount factor $\lambda = 0.967$, obtained by solving:

$$E[T] = \frac{1}{(1-\lambda)} = 30.$$

The Policy Iteration algorithm converges after 13 iterations, and some example of the optimal policy is shown in figure 6.

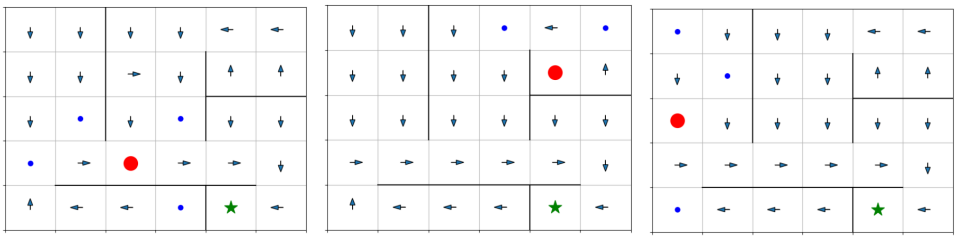

Figure 6: Optimal policy returned by the Policy Iteration algorithm with discount factor $\lambda = 0.967$ and different minotaur positions.

Using the optimal policy found with the Policy Iteration algorithm, we ran 10000 games with the additional transition probability:

$$p_t(s' = Done \mid s \neq Done, a = \cdot) = p_d$$

where $p_d = 1 - \lambda = 0.033$ is the probability of dying randomly. In addition, the other transition probabilities are modified in order to be consistent. Thus, for a player action $a_{s \to s'}$ that should take us from state $s$ to state $s'$ we get that the probability is not $1/n$ anymore, but since we have a small probability of dying, it is $(1 - p_d)/n$, where $n$ is the number of allowed actions for the minotaur.

After running the 10000 games several times, we win on average $\sim 6900$ games, and the probability of getting out the maze alive roughly $69\%$. If we keep the same settings, but we allow the minotaur to stand still, the Policy Iteration algorithm converges in 15 iterations, and we obtain the policy in figure 7. The probability of getting out the maze alive becomes roughly $61.5\%$.

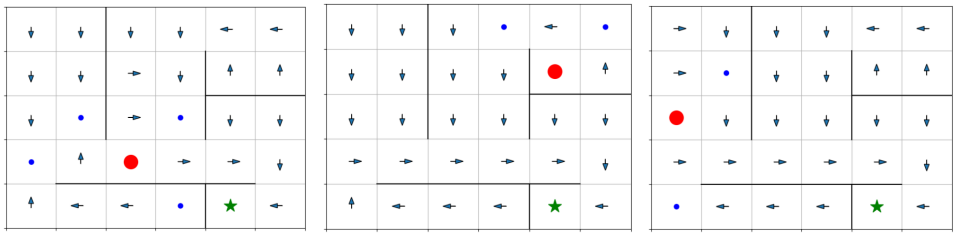

Figure 7: Optimal policy returned by the Policy Iteration algorithm with discount factor $\lambda = 0.967$ and different minotaur positions, when the minotaur is allowed to stand still.

## 2 Problem 2: Robbing Banks

**a) Formulate the problem as an MDP.**

- **State-Space**: $\mathcal{S} = (\{1, 2, 3, 4, 5, 6\} \times \{1, 2, 3\})^2$. The position of the robber is denoted with $\boldsymbol{r} = (r_x, r_y)$ and the position of the police is denoted with $\boldsymbol{p} = (p_x, p_y)$. Therefore, for each time step $t$, our state will be $s_t = (\boldsymbol{r}, \boldsymbol{p})$. In the following, we will denote the position of the four banks as $\boldsymbol{b_1} = (1, 1)$, $\boldsymbol{b_2} = (1, 3)$, $\boldsymbol{b_3} = (6, 3)$, $\boldsymbol{b_4} = (6, 1)$.
- **Actions**: These are the same as in problem 1-a: $\mathcal{U}$ - Up, $\mathcal{D}$ - Down, $\mathcal{L}$ - Left, $\mathcal{R}$ - Right, $\mathcal{W}$ - Wait. We assume that in any state all the actions are always allowed, and if the robber moves towards the border of the map, it stays in the same position.
- **Time-Horizon and Objective**: We model the problem as an infinite-horizon discounted MDP, we start at time $t = 1$ in state $s_1$ and every time the robber gets caught he goes back to $s_1$. In this case we know the MDP, therefore the value function can be computed as:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=1}^{\infty} \lambda^t r_t(s_t, a_t)\right].$$

- **Rewards**: The only non-zero rewards that we introduce in the MDP are:

$$r_t(s = (\boldsymbol{r} = \{\boldsymbol{b_1} \vee \boldsymbol{b_2} \vee \boldsymbol{b_3} \vee \boldsymbol{b_4}\}, \boldsymbol{r} \neq \boldsymbol{p}), a = \cdot) = 0.1$$
$$r_t(s = (\boldsymbol{r} = \boldsymbol{p}), a = \cdot) = -0.5$$

  In words, the robber receives 0.1 reward to be in a bank if the police is not in the same cell of the robber, and $-0.5$ reward if the robber is in the same cell as the police, regardless the action we take.

- **Transition probabilities**: This part is very similar to problem 1-a. In particular, the actions of the robber are independent from the ones of the police. Thus, the probability of ending up in a state $s' = (\boldsymbol{r'}, \boldsymbol{p'})$ is given by the probability $p(\boldsymbol{r'} \mid \boldsymbol{r}, a)p(\boldsymbol{p'} \mid \boldsymbol{p}, \boldsymbol{r})$ (note how, in this case, the probability of the next position of the police is dependent on the current position of the robber). The part $p(\boldsymbol{r'} \mid \boldsymbol{r}, a)$ is deterministic (i.e. if we move right and there is no wall on the right, we will end up with probability one on the cell on the right), and the police will move following the instructions given for the problem. A special case is the transition probability:

$$p_t(s' = (\boldsymbol{r'} = (1,1), \boldsymbol{p'} = (3,2)) \mid s = (\boldsymbol{r} = \boldsymbol{p}), a = \cdot) = 1$$

  This means that, if the robber is in the same position of the police, the next state will be the initial configuration with probability one.

**b) Solve the problem, and display the value function (evaluated at the initial state) as a function of $\lambda$. Illustrate an optimal policy for different values of $\lambda$ – comment on the behaviour.**

We solve the problem using the Value Iteration algorithm, with precision $\epsilon = 0.01$, initial delta $\delta = 10$, and different values of discount factor $\lambda$. The corresponding value functions at the initial configuration $s_i$ are plotted in figure 8, and the exact values are:

- $\lambda = 0.98 \rightarrow V(s_i) = 3.463$
- $\lambda = 0.9 \rightarrow V(s_i) = 0.746$
- $\lambda = 0.7 \rightarrow V(s_i) = 0.286$
- $\lambda = 0.5 \rightarrow V(s_i) = 0.1878$
- $\lambda = 0.2 \rightarrow V(s_i) = 0.1242$
- $\lambda = 0.1 \rightarrow V(s_i) = 0.111$
- $\lambda = 0.01 \rightarrow V(s_i) = 0.101$

The optimal policies with different values of $\lambda$ are shown in figure 9. We can see that, for $\lambda$ close to one, the best policy is to move away from the police, eventually reaching the banks on the opposite sides. In fact, there won't be a big loss if the rewards are collected few steps later, and it is a good strategy to keep the police away in order to stand on a bank for several consecutive time steps. With $\lambda$ decreasing, the robber still wants to avoid the police, but it goes as quick as possible to the closest bank, even when the police is close. Having small values for $\lambda$ means that rewards collected later will have less value, and the agent aims to collect as much reward as possible in a short time. For very small $\lambda$, it is still important to avoid the police and possibly reach a bank, but sometimes the agent does not move, since due to the 'strong' discount factor, no reward will be collected by the time a bank is reached.

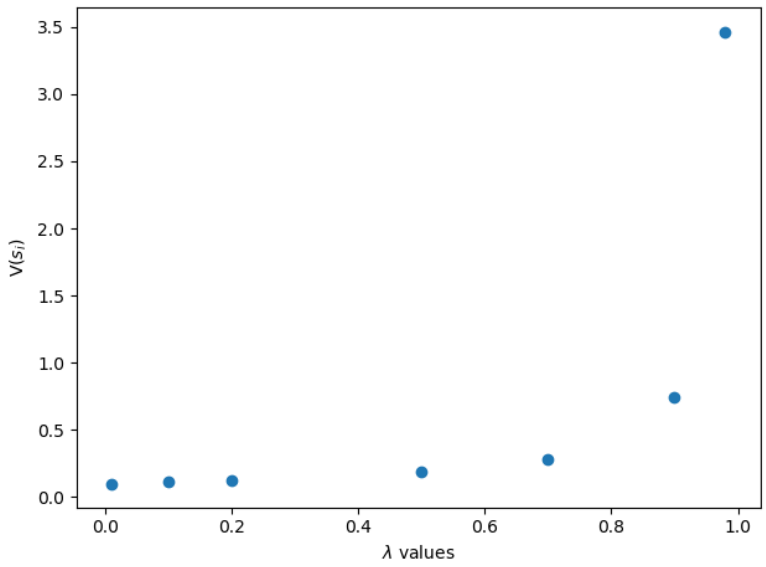


Figure 8: Value function in the initial state for different discount factor $\lambda$.

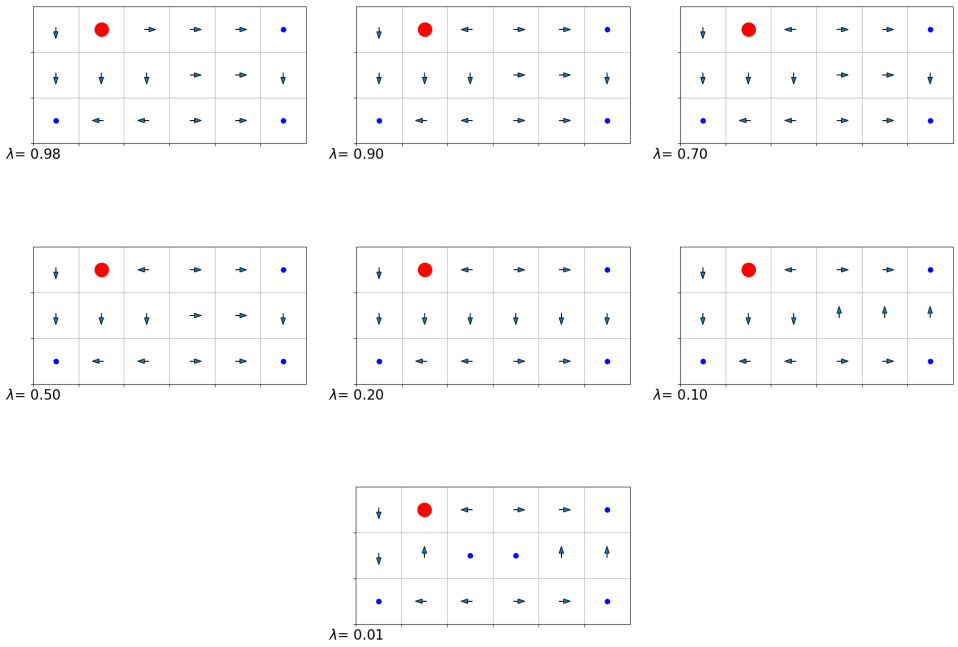


Figure 9: Optimal policies found with the Value Iteration algorithm for different discount factors $\lambda$. The red dot is the position of the police.

## 3 Problem 3: Bank Robbing (Reloaded)

In this problem, we assume that the model is unknown. Therefore, model-free methods are needed to solve the problem. We interpret the time-horizon to be infinite, meaning that even if the police catches the robber, the game continues and the robber gets a negative reward. We use the rewards as given in the problem, but re-scaled so that $|r| \leq 1$ to ensure convergence, i.e. $0.1$ if the robber

is in the bank and the police is not, $-1$ if the robber and police are on the same position, and zero elsewhere. In addition, we assume that the police always moves somewhere in each time step and that the robber and police move simultaneously.

**a) Solve the problem by implementing the Q-learning algorithm exploring actions uniformly at random. Create a plot of the value function over time (in particular, for the initial state), showing the convergence of the algorithm.**

First, we use Q-learning, which is an off-policy method, to solve the problem through random uniform exploration. We initialize a matrix $Q \in \mathbb{R}^{S \times A}$ filled with zeros that is used to estimate the value of state-action pairs. Then, we simulate the game by sampling random actions $a \sim \text{Uniform}(\mathcal{A})$ and acquire experience tuples $(s_t, a_t, r_t, s_{t+1})$ that we use to do online updates of our estimate of the Q-values at each time step $t$.

The update of $Q$ for the experience $(s, a, r, s')$ is

$$Q(s,a) \leftarrow Q(s,a) + \alpha_{n(s,a)} \left[ r + \lambda \max_{a'} Q(s', a') - Q(s,a) \right], \tag{1}$$

which means that we shift the value of $Q(s,a)$ towards $r + \lambda \max'_a Q(s', a')$ with a step size $\alpha_{n(s,a)}$ that is dependent on how many updates we have done for that particular state-action pair. In order to find the value $v^\pi(s)$ of a certain state $s$ under the current greedy policy $\pi$ (which is deterministic w.r.p to the current $Q$), we can use

$$v^\pi(s) = \max_a Q(s,a). \tag{2}$$

Taking $s$ to be the initial state, we show the convergence of the algorithm in Figure 10.

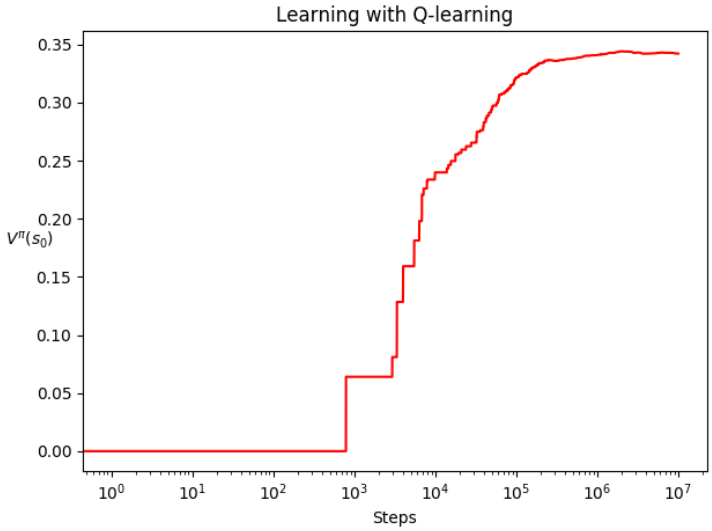


Figure 10: Value of initial state during the learning process.

**b) Solve the problem by implementing the SARSA algorithm using $\epsilon$-greedy exploration (initially $\epsilon = 0.1$). Show the convergence for different values of $\epsilon$.**

The Sarsa algorithm is an on-policy method where we follow an $\epsilon$-greedy policy, meaning that we choose actions according to

$$\pi(a|s) = \begin{cases} \arg\max_a Q(s,a), & \text{with probability } 1-\epsilon \\ a \sim \text{Uniform}(\mathcal{A}), & \text{with probability } \epsilon. \end{cases} \tag{3}$$

Also, updates of the Q-values are now done differently and from the experience $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$. For an experience $(s, a, r, s', a')$ we get the update

$$Q(s,a) \leftarrow Q(s,a) + \alpha_{n(s,a)} \left[ r + \lambda Q(s', a') - Q(s,a) \right], \tag{4}$$

therefore moving the Q-value for the state-action pair $(s, a)$ towards $r + \lambda Q(s', a')$. We can see that the difference from Q-learning is that now the next action is chosen according to our policy and not by the max. This is what makes the algorithm an on-policy method, i.e. we only learn about the policy that we act with.

Again using equation (2), we can get the value function of a certain state. In Figure 11 we show the convergence of the value of being in the initial state with respect to the current greedy policy, derived from $Q$, for different exploration rates $\epsilon \in [0.1, 0.2, 0.3, 0.4]$.

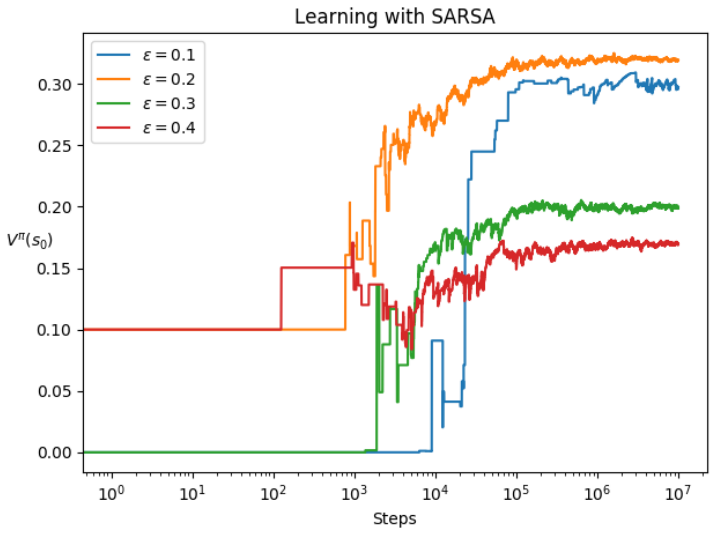


Figure 11: Value of initial state during the learning process.