Reinforcement Learning Lab 1

Oleguer Canal
oleguer@kth.se
19950727-2871

Federico Taschin
taschin@kth.se
960614-T290

November 2019

Problem 1

A) Markov Decision Process

To formulate the problem as a Markov Decision Process we need to specify the followings:

- **States**: The set of states is given by the cartesian product of the player positions and the minotaur position. A state is therefore a pair (P, M), where $P = (r_p, c_p)$, the position of the player expressed as row, column, and $M = (r_m, c_m)$, the position of the minotaur expressed as row, column.
- **Actions**: The player can move up, down, left, right, or stand still. We model these actions in terms of increase in row or column. Actions are therefore $(1, 0), (0, 1), (-1, 0), (0, -1), (0, 0)$.
- **Transition Probabilities**: Since the player movement is deterministic, the probability $P^a_{ss'}$, probability of experiencing the transition $s \to s'$ by doing action a depends only on the random movement of the minotaur. For each non-valid state s', such as being outside the board or the player being in a wall, $P^a_{ss'} = 0$. In the same way, $P^a_{ss'} = 0$ if, being in s, action a cannot bring us to s'. The remaining probabilities are only given by the minotaur position $M = (r_m, c_m)$. For a state $s = (P_i, M_i)$ and a valid action a, the probability of a new state $s' = (P_j, M_j)$ is $1/N_M$, with N_{M_i} being the number of possible moves the minotaur can do in position M_i.
- **Rewards**: We'll refer to the immediate reward of being in state s with R_s. In this problem, rewards depend only on the state and not on the action we perform on that state. A definition of the rewards can be found in the next sections, as they vary depending on the problem conditions.

B) Finding the optimal policy

With a finite horizon, we can exploit dynamic programming to compute directly the optimal state values $V^*(s)$ of the Bellman equation

$$V_t^*(s_t) = \max_a \left\{ r_t(s_t, a) + \sum_{j \in S} p_t(j|s_t, a) V_{t+1}^*(j) \right\} \tag{1}$$

We set the rewards as follows

- $+1$ for reaching the goal
- 0 otherwise

This way, the agent will collect $+1$ only if it reaches the goal without being eaten by the Minotaur. We backpropagate the values as follows:

- For $t = T$ down to 1
- For all s_{t-1}

$$V_{t-1}(s_{t-1}) = \max_a \left\{ r_{t-1}(s_{t-1}, a) + \lambda \sum_{j \in S} p_{t-1}(j|s_{t-1}, a) V_t(j) \right\} \tag{2}$$

This way, since the only reward is $+1$ at the goal, the probability of reaching it is backpropagated to $V_0(initial\_state)$, where $initial\_state = (agent = (0,0), minotaur = (6,5))$. The probability of reaching the goal is therefore the value of the initial state at time $t = 0$. The optimal policy is then given by

$$\pi_t(s_t) = \arg\max_a \left\{ r_t(s_t, a) + \lambda \sum_{j \in S} p_t(j|s_t, a) V_{t+1}^*(j) \right\} \tag{3}$$

Figure 1 shows the optimal policy for a given position of the Minotaur. As we can see, the policy makes the agent go towards the goal, but avoids the Minotaur. In the last steps it chose to stand still since the minotaur is at least 4 steps away, and therefore waiting one turn or going straight to the goal has no difference, since we don't penalize the steps. It will however decide to move in the following turns. Figure 2 shows the theoretical probability and actual probability -computed by simulating $n = 1000$ games- of reaching the exit of the maze for increasing deadlines. A game sample animation can be seen here.

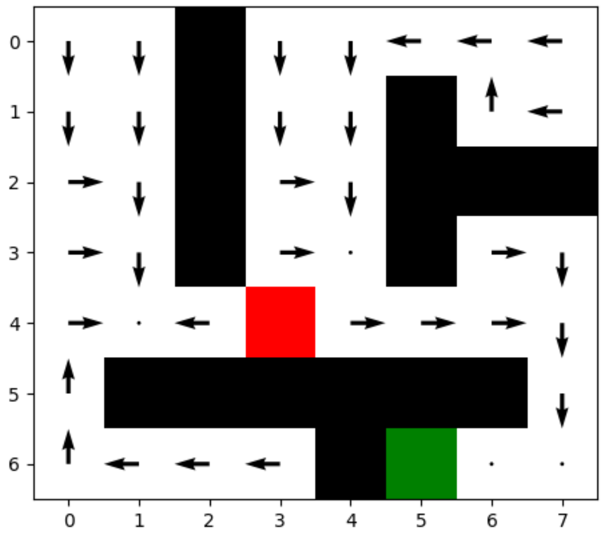

Figure 1: Policy for fixed position of Minotaur at time $t = 0$ with deadline $T = 20$. Minotaur is red, goal is green. A point represents the action "Stand still"

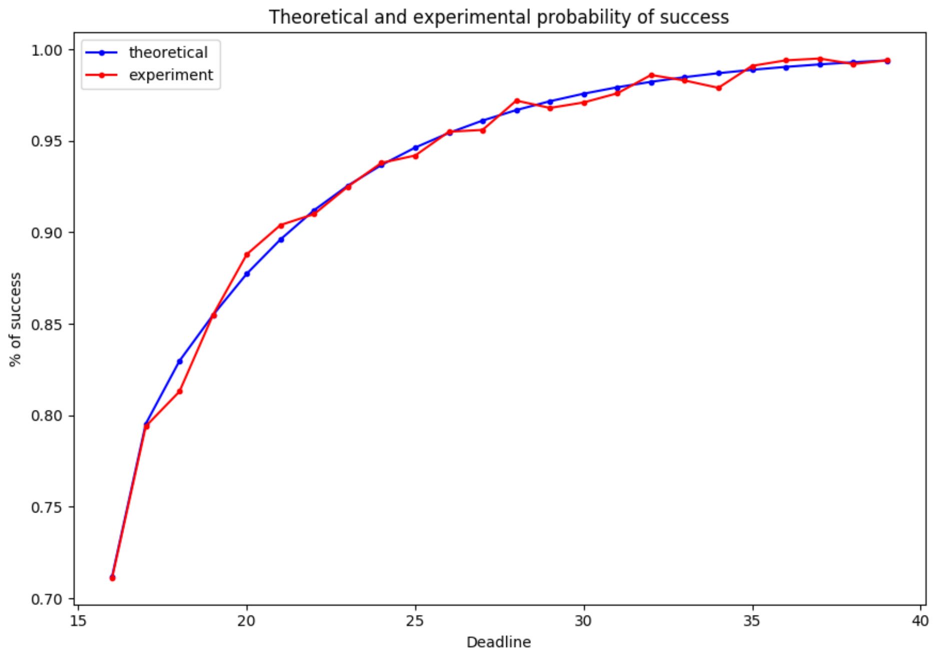

Figure 2: Theoretical and experimental probability of escaping the maze. Each experimental point is the fraction of success over $n = 1000$ games.

Minotaur not allowed to stand still If the Minotaur is not allowed to stand still it will never be able to eat the agent if their initial Manhattan distance (Eq. 4) is odd and agent also never stays still.

$$d_M(\mathbf{x}, \mathbf{y}) = |x_r - y_r| + |x_c - y_c| \tag{4}$$

To eat the agent, the Minotaur must be at distance 0 from the agent after a certain number of turns. However, we prove that the distance remains odd no matter the number of turns.

1. Base case: The Manhattan distance after 0 turns is odd.
2. We assume the distance after n turns is odd.
3. Then, the Manhattan distance after $n+1$ turns is odd. To prove that, we observe that a single step always changes the distance by ±1. Then, if both the agent and the Minotaur make a step, the distance can increase by 2, decrease by 2, or remain unchanged. Therefore, the distance is odd at turn $n+1$.

Given that the initial distance is odd and both the player and the Minotaur make a step at each turn, their distance at any turn will be odd and therefore non zero, and the Minotaur will never eat the agent. The agent will then just follow the shortest path to reach the exit with probability 1.

Minotaur starting at even distance, not allowed to stand still We changed the starting position of the Minotaur to $(6, 6)$ to avoid the particular case described above. However, it comes naturally that the player can still win with probability 1. In fact, the player just has to stand still the first turn, so that the Minotaur will make a move and have again an odd distance. Then the player can just follow the shortest path. Figure 3 shows this clearly: for a deadline of 15, the agent can't wait the first turn and will have to follow the shortest path accepting a certain probability of being eaten. For deadline >15, the agent can wait the first turn and win with probability of 1.

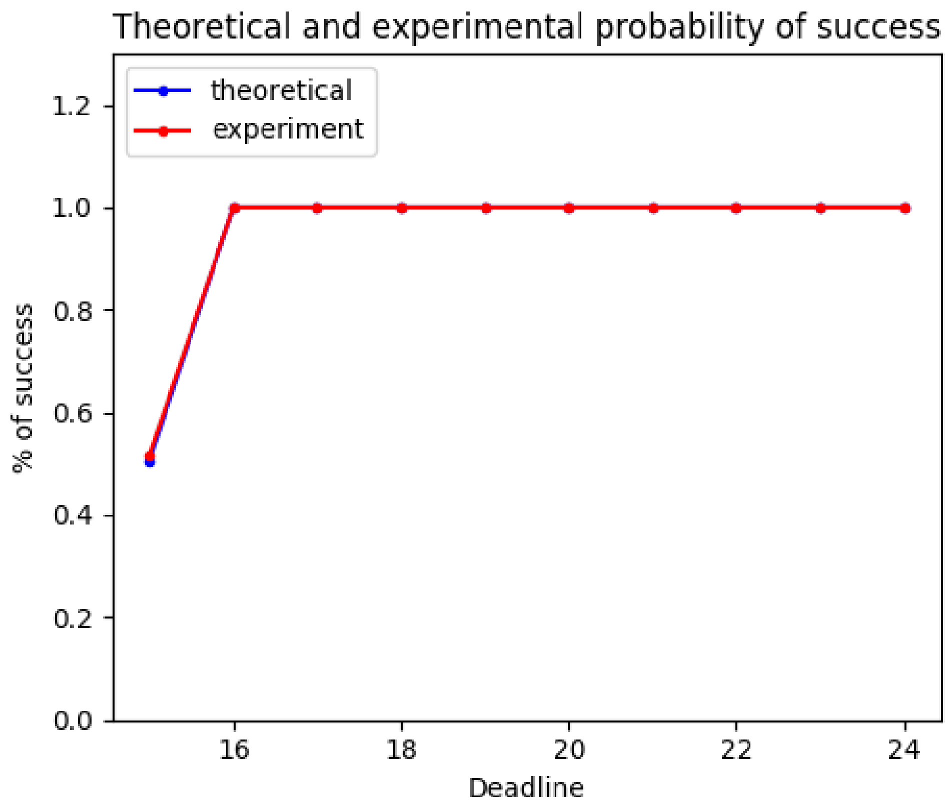

Figure 3: Probability of success for Minotaur not allowed to stand still, starting at (6,6)

C) Geometrically distributed life

We now assume that our life is geometrically distributed with mean 30. This means that at each step we have $P(die) = \frac{1}{30}$ and $P(live) = \frac{29}{30}$. This is an infinite horizon problem since we don't have a fixed deadline, but rather a probabilistic one. Therefore, we will solve it using Value Iteration with a discount factor $\lambda = \frac{29}{30}$, without modifying the rewards. This means that the averaged reward of the next state will be weighted by the probability of actually getting it. Each step is therefore penalized since reaching the exit slowly will lead to additional multiplications by λ in the goal reward of 1, and therefore a lower total reward. The optimal value function is given by Bellman equation

$$V^*(s) = \max_a \left\{ r(s,a) + \lambda \sum_{j \in S} p(j|,a) V^*(j) \right\} \tag{5}$$

We perform Value Iteration by initializing $V^{(0)}(s) = r(s)$, $\forall s \in S$, since our rewards are action independent. Then, we update the values as follows:

1. While not converged
2. For all s

$$V^{(t+1)}(s) = \max_a \left\{ r(s,a) + \lambda \sum_{j \in S} p(j|,a) V^t(j) \right\} \tag{6}$$

3. If $||V^{(t+1)} - V^t|| > \delta$ then converged

where $\delta = \frac{\epsilon(1-\lambda)}{\lambda}$. The optimal policy is then given by

$$\pi(s) = \arg\max_a \left\{ r(s,a) + \lambda \sum_{j \in S} p(j|,a) V(j) \right\} \tag{7}$$

using the values V computed by Value Iteration.

Simulation We then simulate 10000 games for increasing game length, following a policy that considers the geometric probability of dying as described in the paragraph above. Figure 4 shows the result. We observe that despite having a discount factor, the agent still does not risk too much and plays a conservative policy, and the probability of winning approaches 1 when it is allowed to play for more time.

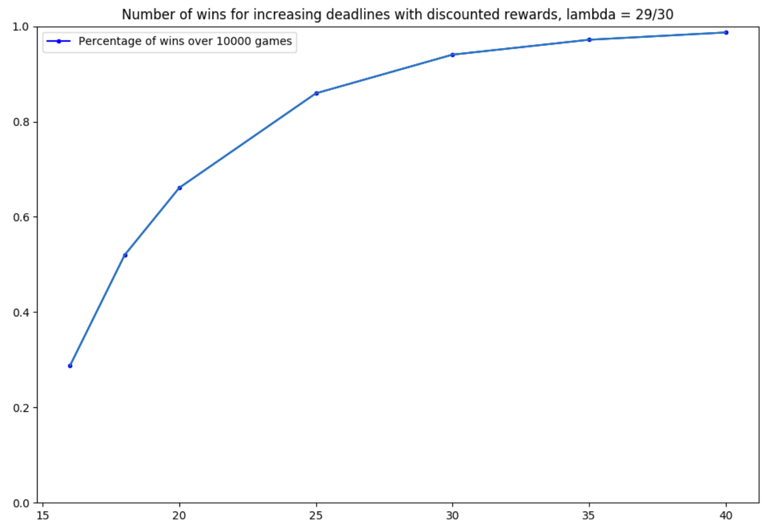

Figure 4: Experimental probability of winning from 10000 games for increasing game length, $\lambda = \frac{29}{30}$

Problem 2

A) Markov Decision Process

We formulate the problem as an MDP. As before, we define States, Actions, Transition Probabilities, and Rewards.

- **States**: The set of states is given by the cartesian product of all the robber positions and all police positions. A state is therefore a pair (R, P), where $R = (r_R, c_R)$, the position of the robber expressed as row, column, and $P = (r_P, c_P)$, the position of the police expressed as row, column.

- **Actions**: The robber can move up, down, left, right, and stand still. Actions are therefore $(1, 0), (0, 1), (-1, 0), (0, -1), (0, 0)$.

- **Transition Probabilities**: Since the robber movement is deterministic, the probability $P^a_{ss'}$, probability of experiencing the transition $s \rightarrow s'$ by doing action a depends only on the random movement of the police. For each action a that would lead the robber outside the board we set $P^a_{ss'} = 0 \; \forall \; s, s' \in S$. In the same way, $P^a_{ss'} = 0$ if, being in s, action a cannot bring us to s'. The remaining probabilities are only given by the police position $P = (r_P, c_P)$. For each state $s = (R_i, P_i)$, the transition probability to a valid state $s' = (R_j, P_j)$ by doing action a is $\frac{1}{N_{P_i}}$ where N_{P_i} is the number of possible movements the police can do in position P_i following the rules in the description of the problem. Moreover, for any

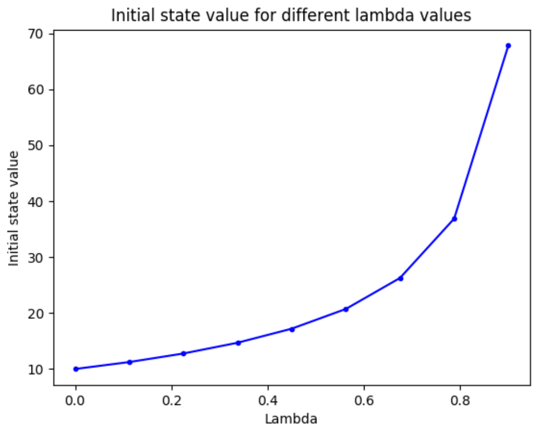

Figure 5: Plot of the state value function at initial state for different lambda values. The bigger the lambda, the more future states are considered, thus the larger the value. At $\lambda = 0$ we only consider the current state, thus the reward is 10 and with $\lambda = 1$ the reward function would be infinite as the problem is infinite, so the function presents and asymptote as it approaches 1.

state in which $R = P$ -the robber is caught by the police-, $P^a_{ss'} = 0$ for each $s' \neq ((0,0),(1,2))$ -the initial state- and $P^a_{ss'} = 1$ for $s' = ((0,0),(1,2))$

- **Rewards**: Rewards are action-independent since they depend only on the state. Therefore, the immediate reward of a state $s = (R, P)$ is 10 if $R \in \{B_i\}_{i=1..4}$ where B is the set of bank positions, -50 if $R = P$, and 0 otherwise.

B) Solving with Value Iteration

We use Value Iteration as described in 7. Figure 5 shows the value of the initial state as function of the discount factor λ. We observe that for $\lambda = 0$ the reward is +10, the immediate reward of being in a bank. The more we consider the future rewards by increasing λ, the more valuable this state becomes.

Figure 6 shows the optimal learned policy for $\lambda = 0$. We can see how it only takes into account one step ahead reward. That is, the reward of taking an action without considering future developments other than the most immediate state. Alternatively, figure 7 shows the policy for $\lambda = 0.8$, where the value of an action is computed by the geometric summation of all rewards of future states.

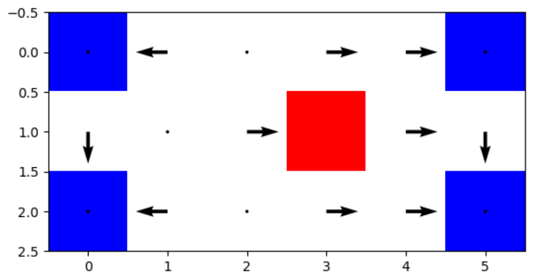

Figure 6: Representation of the policy for $\lambda = 0$ for a fixed police (red) location. Notice how the policy is defined only in states where there is a reward at next time step, as we do not take into account future steps. Full animation here.

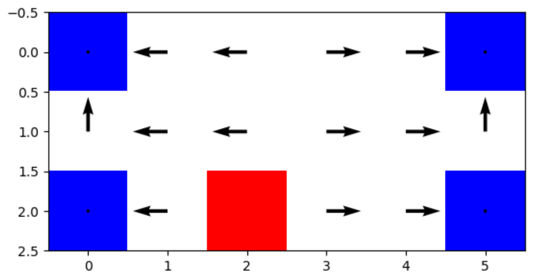

Figure 7: Representation of the policy for $\lambda = 0.8$ for a fixed police (red) location. Notice how now all states have a best action. Full animation here.

Problem 3

This problem can be modeled as a Markov Decision Process where the agent isn't aware of the transitions and rewards. It will learn the policy directly by sampling the environment using two different strategies: first an off-policy algorithm (Q-Learning), sampling from a random uniform policy and second, an on-policy (SARSA) algorithm using a epsilon-greedy policy. To do so, we defined the environment with the following parameters:

- **States**: The set of states is again given by the Cartesian product of the robber positions and the police position. A state is therefore a pair (R, P), where $R = (r_R, c_R)$, the position of the player expressed as row, column, and $P = (r_P, c_P)$, the position of the police expressed as row, column.
- **Actions**: The robber can again move up, down, left, right, or stand still. We model these actions in terms of increase in row or column. Actions are therefore $(1, 0), (0, 1), (-1, 0), (0, -1), (0, 0)$.
- **Hidden Transition Probabilities**: Since the player movement is deterministic, the probability $P^a_{ss'}$, probability of experiencing the transition $s \rightarrow s'$ by doing action a depends only on the random movement of the police. For each non-valid state s', such as being outside the board $P^a_{ss'} = 0$. In the same way, $P^a_{ss'} = 0$ if, being in s, action a cannot bring us to s'. The remaining probabilities are only given by the police position $P = (r_P, c_P)$. For a state $s = (R_i, P_i)$ and a valid action a, the probability of a new state $s' = (R_j, P_j)$ is $1/N_{P_i}$, with N_{P_i} being the number of possible moves the police can do in position P_i.
- **Hidden Rewards**: We'll refer to the immediate reward of being in state s with R_s. Rewards depend only on the state and not on the action we perform on that state. As the statement indicates, we give +1 reward for each turn the player spends in the bank and -10 each time the player and the police meet.

A) Q-Learning

We are asked to apply Q-learning to solve this problem. To do so we initialize, the quality Q function at 0 and iteratively update by sampling (s_t, a_t, r_t, s_{t+1}) from the given **random uniform policy** μ (off-policy) as:

$$\begin{aligned} &Q^{(t+1)}(s, a) = Q^{(t)}(s, a) \\ &+1_{(s_t,a_t)=(s,a)}\alpha_{n^{(t)}(s_t,a_t)} \left[r_t + \lambda \max_{b \in \mathcal{A}} Q^{(t)}\left(s_{t+1}, b\right) - Q^{(t)}\left(s_t, a_t\right)\right] \end{aligned} \tag{8}$$

Where:

$$\alpha_{n^{(t)}(s_t,a_t)} := \frac{1}{n^{(t)}(s, a)^{\frac{2}{3}}} \tag{9}$$

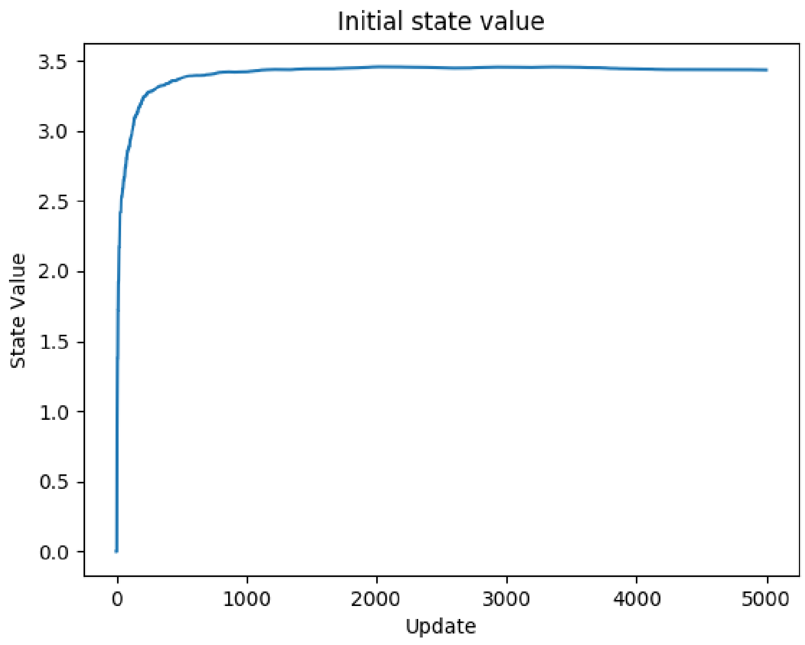

Figure 8: Evolution of initial state value function over the first 5000 times it was updated during training

$$n^{(t)}(s,a) := \sum_{m=1}^{t} 1\left[(s,a) = (s_m, a_m)\right]. \tag{10}$$

Then, we pick a greedy policy:

$$\pi(s) = \arg\max_a Q^*(s,a) \tag{11}$$

Figure 8 shows the convergence of Q-Learning algorithm. To create it, we stored the value of the initial state after each time its value was updated by the algorithm. We can clearly see how after around 1000 updates the value becomes essentially constant, meaning that the algorithm has converged fot this state. In addition, we show a representation of the learned optimal policy in figure 9.

Simulation To assess the quality of the leaned policy, we simulate 1000 games of 100 steps each and average the money won at every game. As a base-line we use a random uniform policy. Results show that, after learning the policy we get an average of 71 SEK, while the uniform random policy gets an average reward of -54 SEK. An animation of a sample game can be seen here.

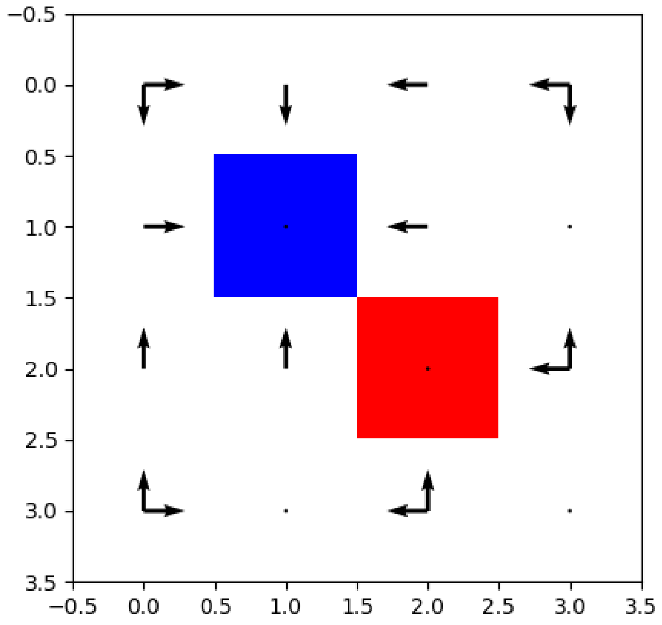

Figure 9: Representation of the policy given the police (red) and bank (blue) positions. A point means that the best action is to ”stand still”. It is interesting to see the optimal learned behaviour around the police so as to avoid meeting at the same spot. Animation of all the states here.

B) SARSA

We are asked to apply the SARSA algorithm to solve this problem. To do so, we initialize the quality Q function at 0 and iteratively update by sampling $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ from the **same policy** π_t we are learning using an ϵ-greedy approach (on-policy) as:

$$\begin{aligned} &Q^{(t+1)}(s,a) = Q^{(t)}(s,a) \\ &+1_{(s_t,a_t)=(s,a)}\alpha_{n^{(t)}(s_t,a_t)} \left[r_t + \lambda Q^{(t)}\left(s_{t+1},a_{t+1}\right) - Q^{(t)}\left(s_t,a_t\right)\right] \end{aligned} \tag{12}$$

Where, again:

$$\alpha_{n^{(t)}(s_t,a_t)} := \frac{1}{n^{(t)}(s,a)^{\frac{2}{3}}} \tag{13}$$

$$n^{(t)}(s,a) := \sum_{m=1}^{t} 1\left[(s,a) = (s_m, a_m)\right) \tag{14}$$

The policy we use in training is

$$\pi_t(s) = \begin{cases} U(Actions) & p = \epsilon \\ \arg\max_a Q_t(s,a) & p = 1-\epsilon \end{cases} \tag{15}$$

and, once trained, we simply use

$$\pi(s) = \arg\max_a Q^*(s,a) \tag{16}$$

Figure 10 shows the algorithm convergence we got for different values of ϵ. It is interesting to see the negative effect that following policies with a large ϵ has. If you try to learn on-policy following a policy with high variability the convergence to values will be lower. Another factor to consider is that a high ϵ means all "not-so-good" states will be visited more because of the higher stochasticity. Therefore, they will be updated more times and will take longer to converge.

Simulation Simulation of this algorithm yields very similar results compared to the ones provided by Q-learning algorithm. Again, while following the learned policy we get an average reward of 71 SEK, while the uniform random policy gets an average reward of -54 SEK.

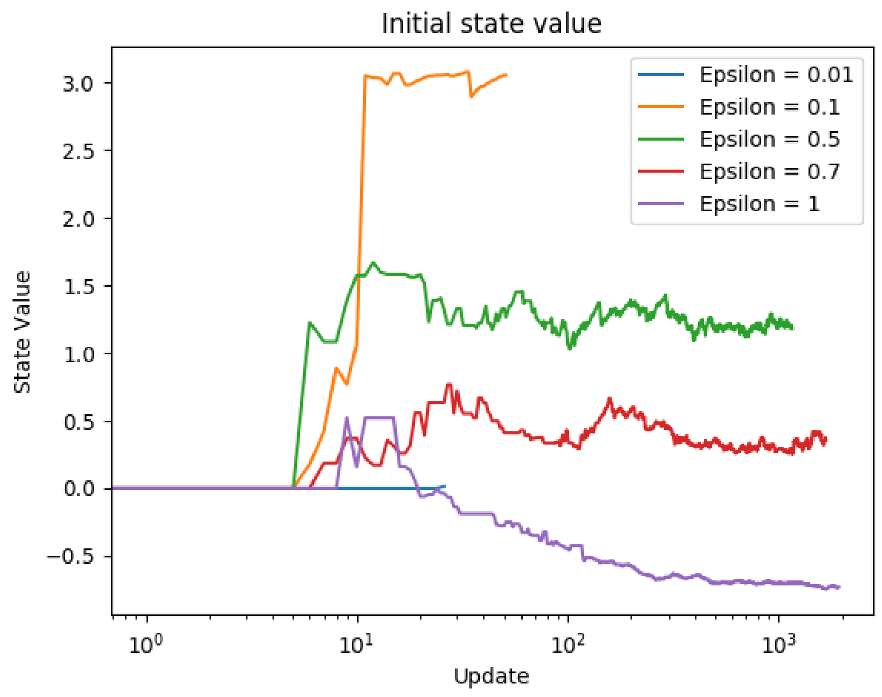

Figure 10: Evolution of first state value while updated by the SARSA algorithm for different values of epsilon. Notice that x axis is presented in a logarithmic scale to ease plot interpretation, as higher epsilon means that first state is updated more times (higher chance of randomly visiting it). Larger epsilon also implies a more random policy, and therefore a lower state value.