Dynamic Programming & Reinforcement Learning 2020 **(Due: 3 jan 2020)**

Submission Assignment #4

Instructor: Ger Koole *Names:* Mick IJzer, *Netids:* 2552611

1 Introduction

The problem of finding the shortest path has been a topic of interest for many years. In most solutions the environment and its transitions probabilities are known. When one has full information about the environment the shortest path can be found in a number of ways; for example using Dijkstra's algorithm, solving a system of equations, or value iteration. However, when one does not have full information these methods won't work. In those situations simple heuristics or more complex reinforcement learning approaches such as SARSA or Q-learning can be used. In this assignment the policies found by having full information, by using a simple heuristic, and by using Q-learning will be evaluated and compared.

To be able to compare these approaches an environment is needed. For this assignment finding the fastest route to a certain destination in a city with many crossings will be used as inspiration. More specifically a gridworld of 50 rows and 50 columns is created. At each state one can choose to go north, east, south, or west. For every direction there is a chance of congestation, which means that the agent will be stuck in the same state for one more time step. The probability of moving to (or staying in) state s' in the next time step given the current state s and chosen action a are given by $p(s', r|s, a)$. All probabilities for successful transitions (i.e. no congestation) are independent and identically distributed for all states **s** and all actions **a** and take values of $[0.1, 0.2, ..., 0.9, 1]$ each with a probability of $\frac{1}{10}$. Thus the probability of congestation is the complement of the probability of a successful transition. When the agent attempts to go east when already in the eastern most state the agent will remain in that state for one more time step. Moving to all states gives a reward of -1, except for the goal state for which the reward is $+1$. The goal state is located at $(1, 10)$ (i.e. at the south-west quadrant of the environment).

2 Methodology

Value Iteration The first method used is value iteration. In value iteration the value of each state is computed by choosing the maximum expected value of all possible actions. This is repeated many times until the state-values converge. Equation 2.1 shows how the value for some state s is calculated. The values for each state were initialized to 0. For this assignment the difference between the old $V(s)$ and the new $\widehat{V}(s)$ had to become smaller than $1e^{-6}$ to stop the value iteration. If the difference between the old and new values is this small it can be concluded that the values have converged. The rewards in equation 2.1 are based on the game rules described in section 1.

$$\widehat{V}(s) = \max_{a \in A} \sum_{s', r} p(s', r|s, a) * [r + V(s')] \tag{2.1}$$

System of Equations The equations that have to be solved are given in equation 2.2. Using a non-linear solver from Scipy (i.e. excitingmixing) the system of equations was solved. The resulting state-values and policy are nearly identical to those found in value iteration. Only the policy in state (45, 25) was different. However, due to the similarity between both policies the results of the policy found by value iteration will be viewed as equal to those found by solving the system of equations. Therefore, no explicit comparison will be made between the other policies and the policy found by solving the system of equations.

$$V_*(s) = \max_{a \in A} \sum_{s', r} p(s', r|s, a) * [r + V_*(s')] \tag{2.2}$$

Simple Heuristic The simple heuristic that was chosen uses the current position of the agent and the goal state. If the agent is currently to the east of the goal state it will always move left. The same goes for being to the west, north and south. This simple heuristic will always reach the goal state in an environment where the probability of congestation is less than 1. However, in the chosen environment this is not the case. Therefore, the simple heuristic is not guaranteed to find the goal state.

Q-Learning The last method that was used is Q-Learning. Q-Learning assigns a value to each state-action combination, which is updated after each action the agent takes, according to equation 2.3. The exploitation/exploration algorithm that was used in combination with Q-Learning was the ϵ-greedy strategy with $\epsilon = 0.05$. Furthermore, the learning rate α and discount factor γ were set to 0.1 and 0.99 respectively. All Q-values are initialized to 0 and the rewards given to each action are based on the game rules described in section 1. However, Q-Learning does not have access to the full information of the environment. So it might have a hard time finding the correct policy. Therefore, the same Q-Learning algorithm was used with a modified reward function. The modified reward function is given in equation 2.4, where $d(s, s_{goal})$ indicates the euclidean distance between the current state and the goal state. Basically, the penalty will become smaller if the agent moves towards the goal state.

$$Q(s,a) = Q(s,a) + \alpha[r_{s'} + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)] \tag{2.3}$$

$$r_s = -\frac{d(s, s_{goal})}{\max_{s' \in S} d(s', s_{goal})} \tag{2.4}$$

3 Results and Discussion

To make it easier for the Q-Learning algorithms 25 different starting points were chosen, these were then ordered from closest to the goal state to furthest from the goal state. Then from each starting state the agent started its run 1.500 times. Each run could last a maximum of 250 steps, or until the agent reached the goal state. These 25 starting states were also used to evaluate the policies found in all four (value iteration, heuristic, regular Q-Learning, and modified Q-Learning) approaches. Due to stochastic environment the evaluation was performed 10 times from each starting state.

The policies found by value iteration, regular Q-Learning, and modified Q-Learning are shown in figures 1, 2, and 3 respectively. Note that the goal state in the figures is $(0, 9)$. Furthermore, these figures only show the policies in the lower left quadrant of the entire environment. The complete policies are shown in figures 5, 6, and 7. The complete policy and state-values found by solving the system of equations is shown in figure 8, which is nearly identical to figure 5. In all figures the grayscale heatmap reflects the (maximum) normalized value of each state and the arrows display the policy. The color of the arrows reflects the probability of congestation, with red being a high probability and green being a low probability. In general the policy around the goal state is quite similar for each approach.

As explained earlier, the found policies were evaluated by measuring the amount of steps needed to reach the goal state from each starting state. For computational purposes a maximum (500 steps) was set to the amount of steps that the agent could take during the evaluation. Figure 4 shows the results. What can be seen is that the average steps for each policy increases if the euclidean distance to the goal increases. Furthermore, the policy found by value iteration (and solving the system of equations) was always able to find the goal state (blue triangles). Since this policy was found using full information of the environment, it can be seen as the optimal policy. As expected the heuristic policy (red circles) did sometimes find the goal state, but most of the time it was unable to find the goal state within 500 steps. When comparing the two Q-Learning approaches it can be concluded that the modified approach (magenta pentagrams) outperformed the regular approach (green squares) since the modified approach was more consistent in finding the goal state. The modified approach was unable to find the goal state from just one starting state. However, whenever either Q-Learning algorithm did find the goal state the amount of steps needed was similar to the optimal policy found by value iteration and solving the system of equations.

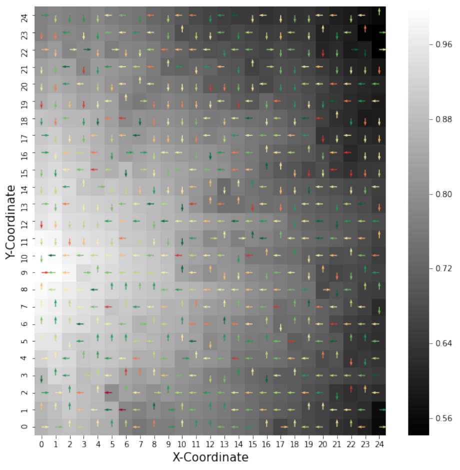

Figure 1: Values and Policy after Value Iteration

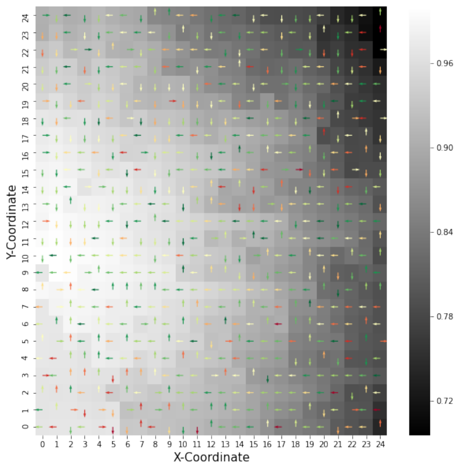

Figure 3: Values and Policy after modified Q-learning

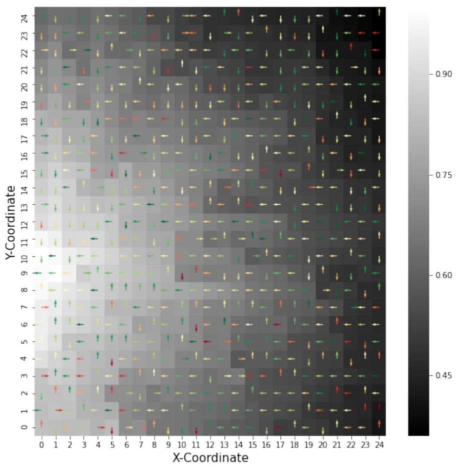

Figure 2: Values and Policy after regular Q-learning

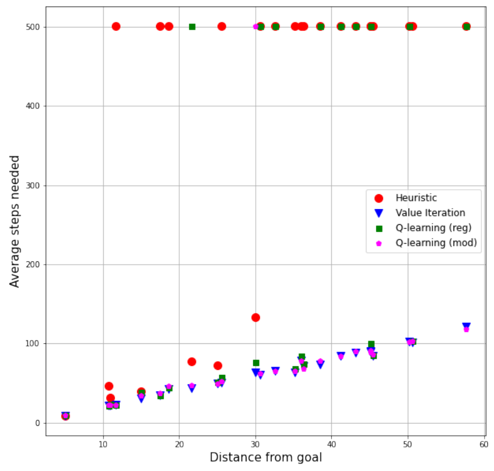

Figure 4: Performance of the different Policies

Appendix

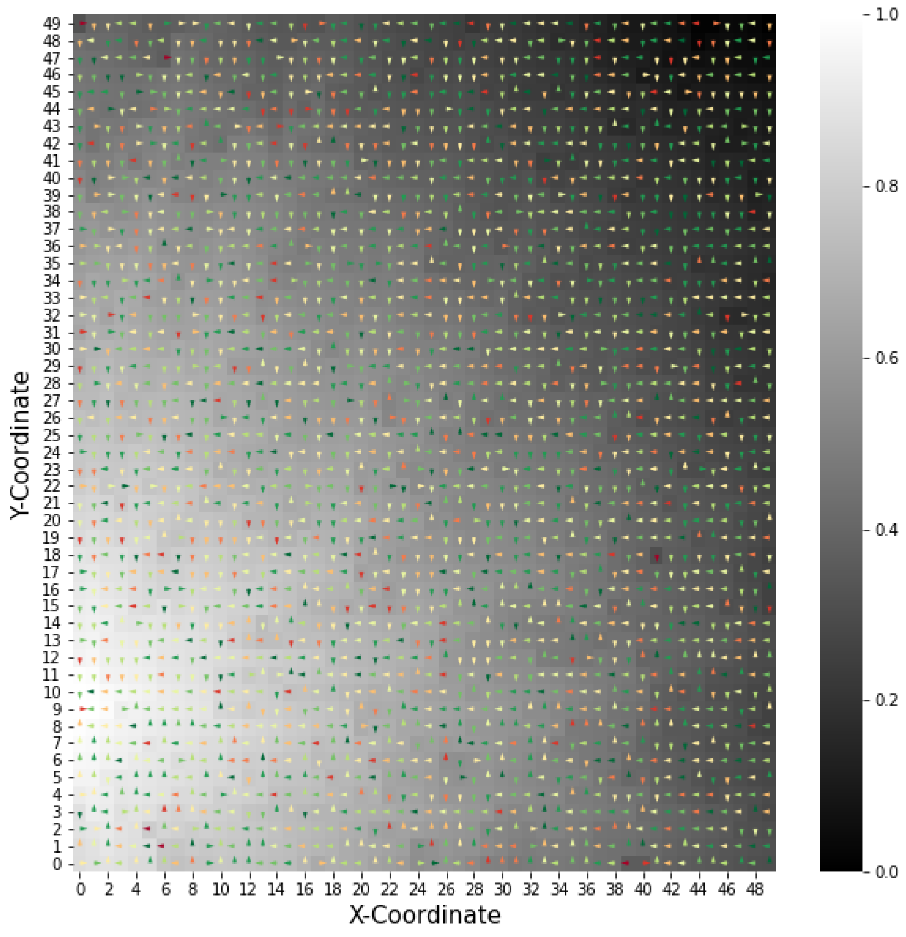

Figure 5: Values and Policy after Value Iteration

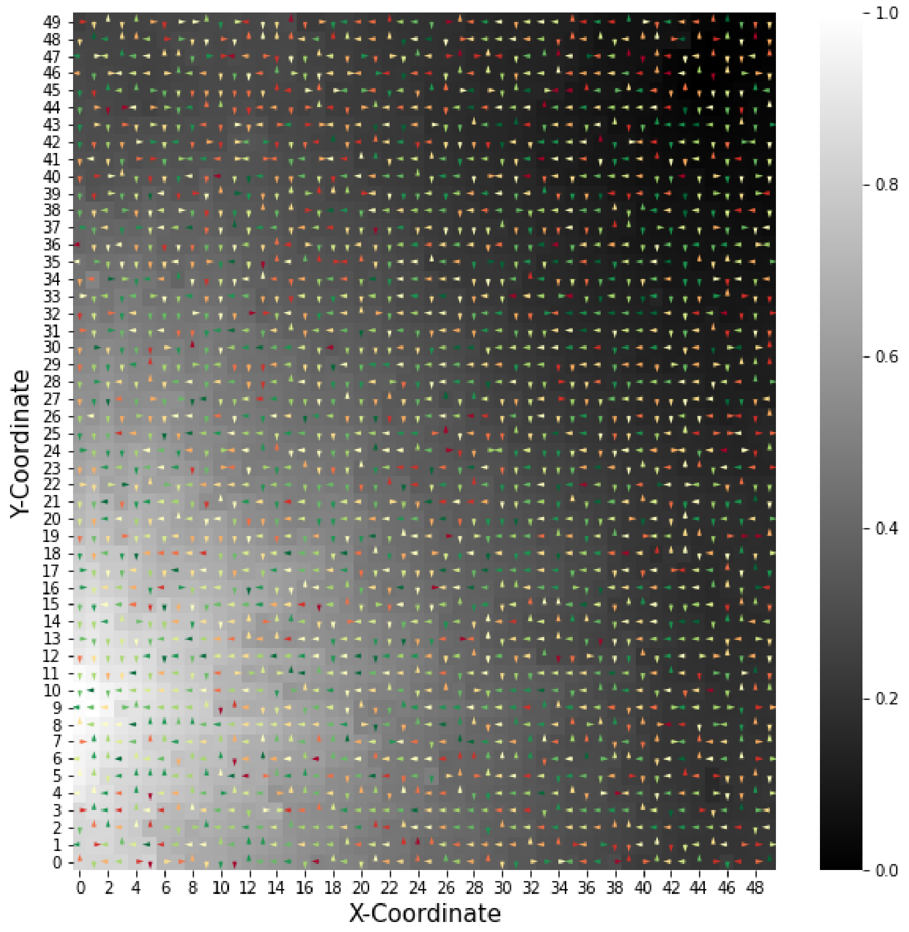

Figure 6: Values and Policy after regular Q-learning

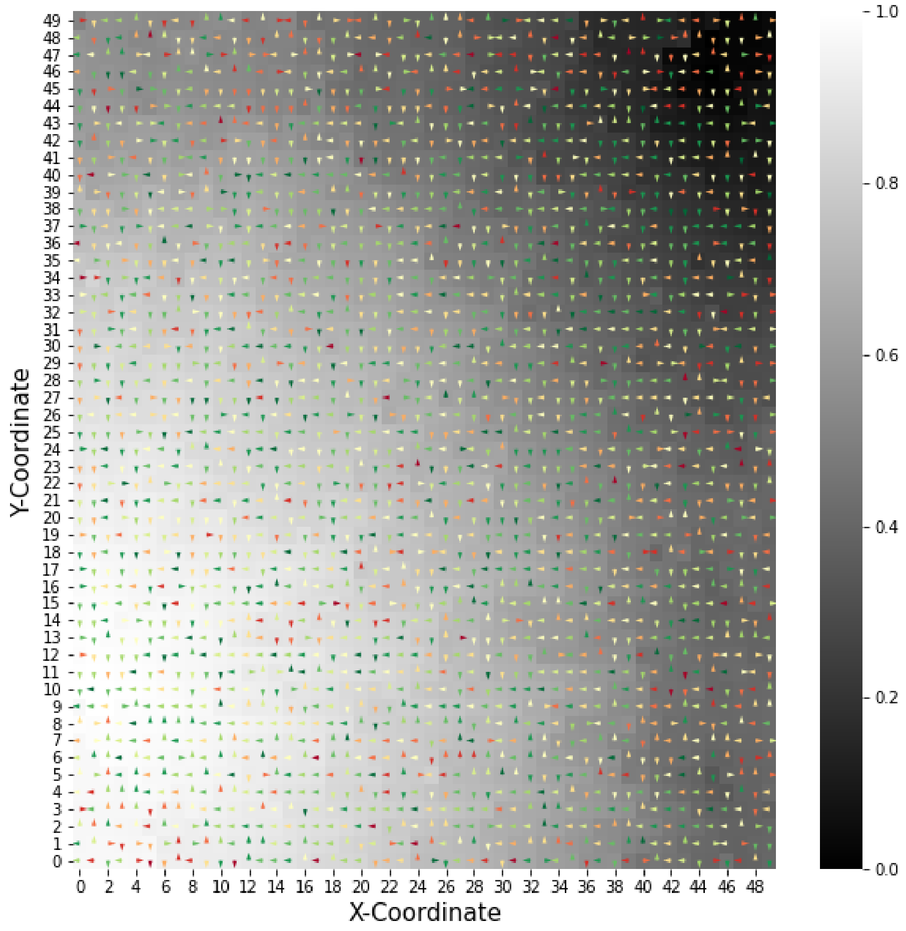

Figure 7: Values and Policy after modified Q-learning

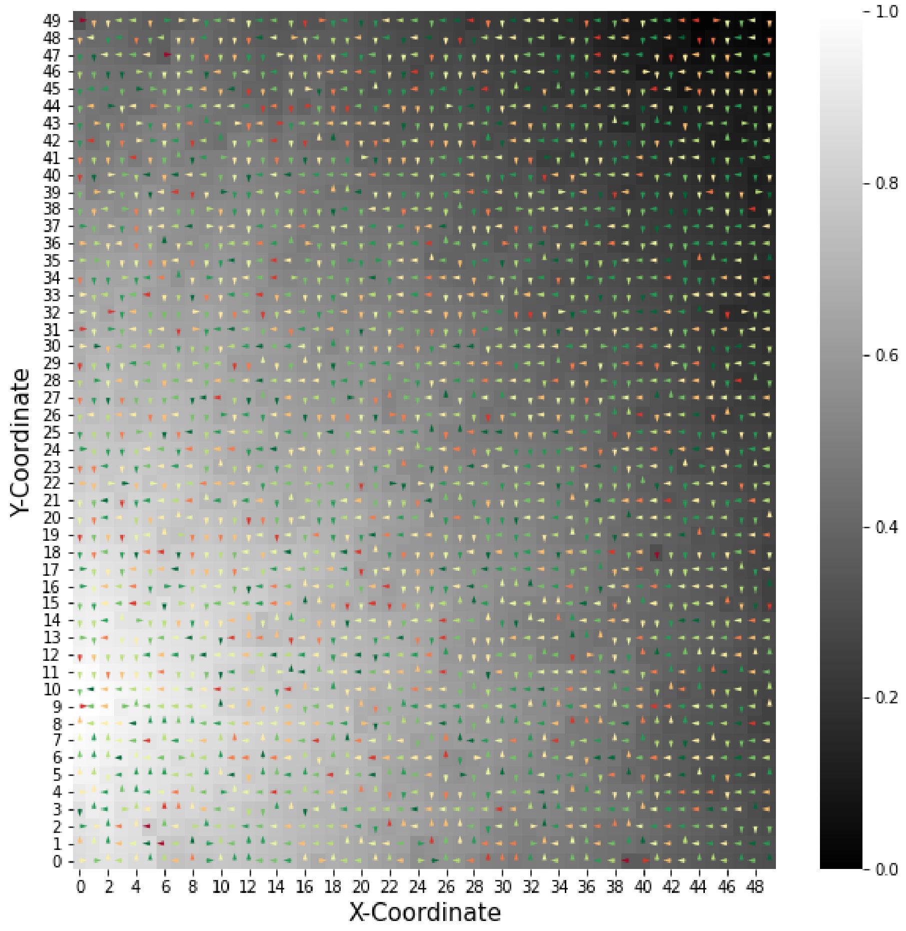

Figure 8: Values and Policy after solving the System of Equations