Artificial Intelligence Laboratory 4: Reinforcement Learning
DT8042 HT22, Halmstad University

Mohammad Hammoud, mohham19@hh.se, civilingenjör i datateknik
Markus Bürgschwendtner, marbur22@student.hh.se, Embedded and Intelligent Systems

January 2023

Introduction

Q1: What is the learning objective of this lab? Please phrase it in your own words.

The idea of this lab is to use reinforcement learning to optimally solve a Markov Decision process. Being confronted with a stochastic grid world as an environment, the students have to implement and use non-deterministic search algorithms to find the optimal paths. By implementing policy iteration, we can maximize the value function. We can get the value of an action in a specific state by implementing TD-learning and using Q-Learning.

Task 1

Q2: List and describe below what concepts and variables were involved in creating the environment.

Grid

The environment is a maze that is represented as a stochastic grid of states. The parameters of that grid are stored in a text file.

- Width and Height
- Noise
  - represents the probability of the agent taking an unintended action instead of the intended action.
  - used to calculate the transition probabilities between states.
- Immediate rewards
  - reward that the agent receives for reaching any of the non-terminal states
  - Values are stored in the Rewards list
- Terminal states
  - states that mark the end of an episode
  - When the agent reaches a terminal state, it receives a final reward and the episode ends
  - Locations are stored in the Terminals list

- Walls
  - Walls are cells in the grid that cannot be entered or passed through by the agent.
  - Locations are stored in the Walls list

Actions

The agent can take one of four actions in this environment: left, right, up, or down. These actions are stored in the Directions list as strings. For the algorithms, these actions are represented by the indices of the respective action in the Directions list. This enables looping over all possible actions easily.

State transition function

As the environment is interpreted as a Markov decision process, the dynamics have to be stored in a state transition function. This function is stored in the transition matrix (T). For each state s it holds N probabilities that action 'a' leads to the successor state s'. Having a total of N states and 4 possible actions, this creates a transition matrix with the shape of [N][4][N].

Q3: Illustrate the given and the two environments you created.

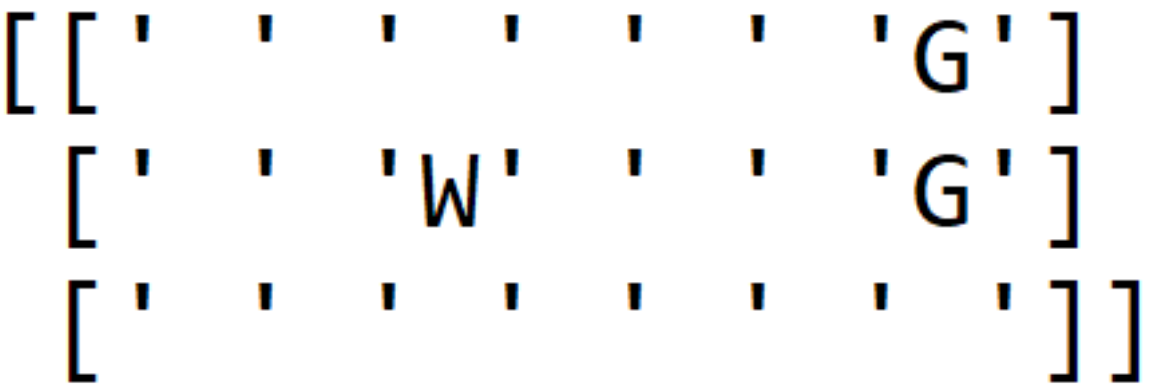

Figure 1: Illustration of the unsolved grid 0

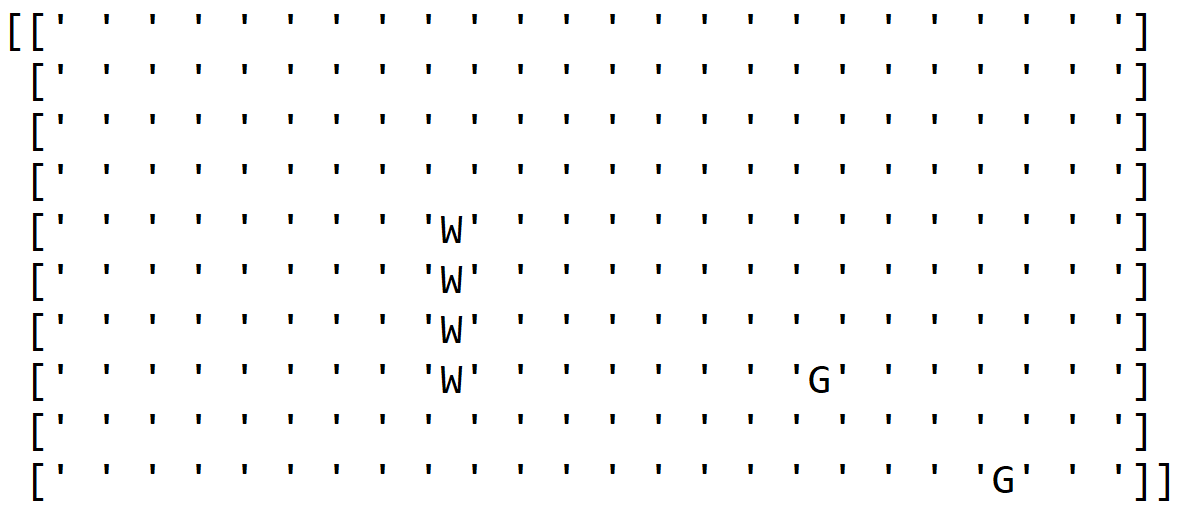

Figure 2: Illustration of the unsolved grid 1

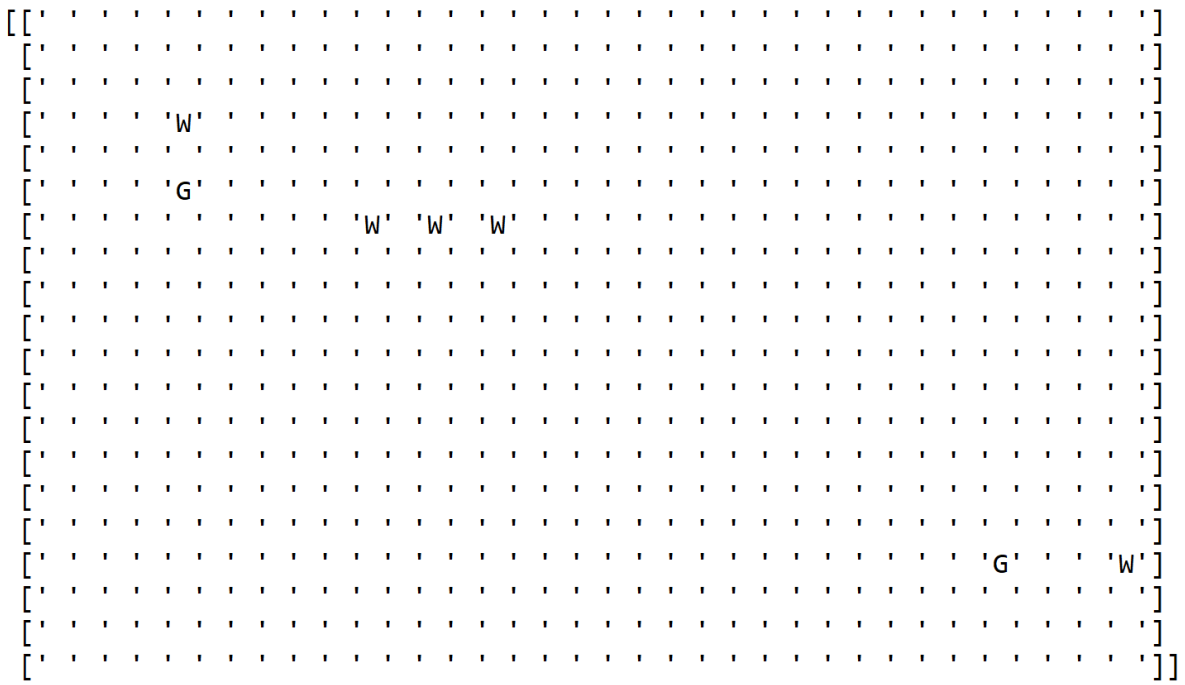

Figure 3: Illustration of the unsolved grid 2

Task 2

Q4: Please include/write the pseudo-code of your Policy Iteration and TD-learning implementation.

pseudo-code of Policy Iteration implementation:

1. Initialize the value function V and policy pi for all states s. Set the flag "policy stable" to False.

2. While policy is not stable:
   a. Evaluate the current policy by iteratively updating the value function:
   i. Set Vold to V.
   ii. For each state s:
   1. If s is a terminal state or a wall, skip to the next state.
   2. Set V[s] to the expected reward for taking action pi[s] in state s and following the policy thereafter, using the Bellman equation: V[s] = Rewards[s] + gamma * sum over s' of T[s, pi[s], s'] * V[s'].
   iii. Calculate the maximum difference between the old and new value function: delta = max over all s of abs(V[s] - Vold[s]).
   iv. If delta is smaller than a predefined threshold, break out of the value iteration loop.
   b. Improve the current policy by setting the action for each state s to the action that maximizes the expected reward, using the updated value function: pi[s] = argmax over a of Rewards[s] + gamma * sum over s' of T[s, a, s'] * V[s'].
   c. Set policystable to True.
   d. For each state s: if the action chosen in the previous step is different from the current action in the policy, set policystable to False.

3. Return the final value function V and policy pi.

pseudo-code of TD-learning implementation:

1. Initialize the value function V for all states and the policy for all states to be random or arbitrary.

2. For each episode:

3. Initialize the current state s to a random or arbitrary state

4. While the current state is not a terminal state:
   Choose an action a based on the current policy for state s.
   Take action a and observe the next state s' and reward r.
   Update the value function for state s using the TD update rule: V(s) ¡- V(s) + alpha * (r + gamma * V(s') - V(s))
   Update the policy for state s using the greedy rule: policy(s) ¡- argmax[a] (r + gamma * V(s'))
   Set the current state s to the next state s'.

5. End while loop.

6. End for loop.

Task 3

Q5: Compare value and policy iterations in terms of convergence time. Relate it to computational complexity, current implementation, and number of iterations needed.

The following table compares the convergence times of value- and policy iteration. For each environment, three experiments were conducted to get reliable time measurements. The execution times shown in the table are the mean values of these three experiments for each grid. It can be seen, that policy iteration is about twice as fast as value iteration. For the larger grid, this difference seems to be smaller. This could still be a coincidence since three experiments do not deliver a statistically reliable result. The difference in speed can be explained by the computational complexity. Value iterations complexity is O(kviSAS'). With Value iteration, approximations will be refined toward optimal values. The policy could converge long before the values which is why it can be slower. For policy iteration, on the other hand, utilities are calculated for fixed policies until the values converge. As we are not calculating utilities for all actions but only for the fixed policy, this step is much faster. Also, Here lies

a significant difference in our implementation as well. The threshold for convergence is much higher in our policy iteration algorithm than in the value iteration, because with the fixed policy the smaller threshold was never reached. After the utility converged, the policy will be adopted until it is stable. This way, under certain conditions, policy iteration can converge much faster.

Environment	Value Iteration [s]	Policy Iteration [s]
Grid 1	0.0108	0.0052
Grid 2	0.4699	0.2973
Grid 3	0.7164	0.6281

Table 1: Convergence times: VI / PI

Q6: How are optimal policies changed with immediate reward values? Show some examples (similar to Figure 17.2b in AIMA).

It is expected that changing the immediate rewards to higher penalties for each non terminal state will result in higher risks to reach the target terminal state. Instead of going around the wall or in the wall to avoid 'slipping' in the negative terminal state. The algorithms choose to go directly and risk reaching the negative terminal state due to the noise probability.

Environment	Immediate Reward
Grid 1	-0.04
Grid 2	-0.01

Table 2: Convergence times: VI / PI

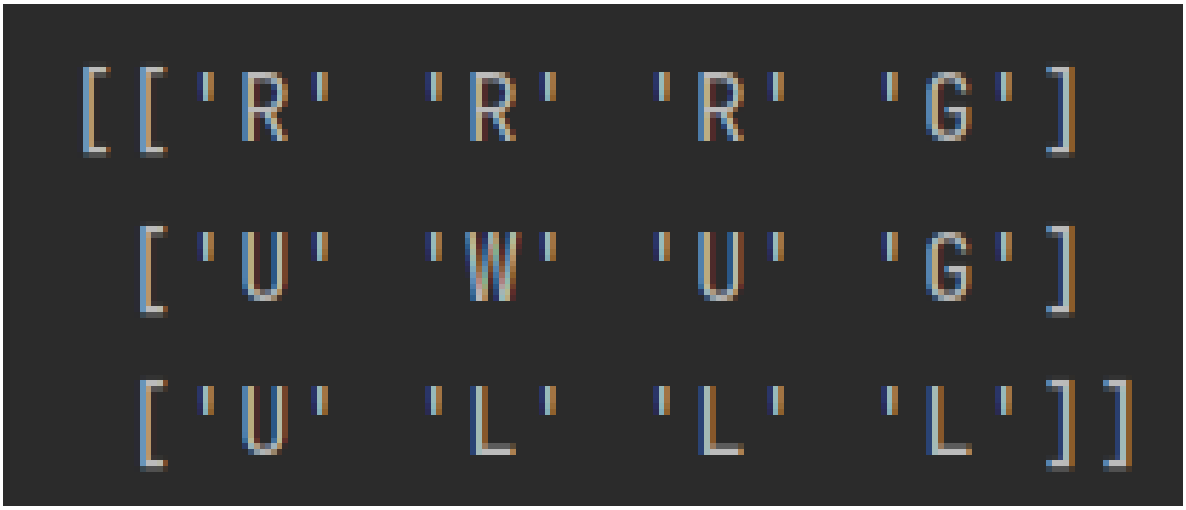

Figure 4: Q6 VI grid0

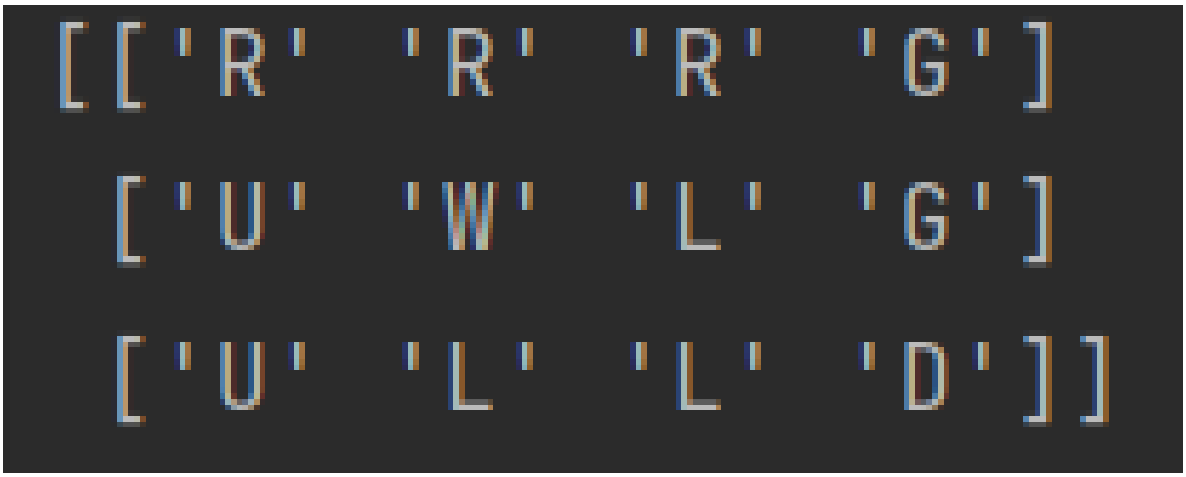

Figure 5: Q6 VI grid1

Q7: Compare TD-Learning and Q-Learning results with each other. Also with value and policy iterations. Remember that value and policy iteration solve Markov Decision Processes where we know the model (T and Rewards). TD-Learning and Q-Learning are (passive and active, respectively) Reinforcement Learning methods that don't have the model but use data from simulations. Data are simulated from the model we know, but the model is not used in TD or Q-Learning.

TD-Learning Vs. Q-Learning
TD-Learning follows a fixed policy while Q-Learning actively selects actions based on its current estimate of the optimal action-value function. This means that Q-Learning may take longer to find the optimal solution, but it has the ability to explore more of the state space and potentially discover better policies. TD-Learning may converge faster, but its performance is limited by the quality of the initial policy it follows.

Value and Policy Iteration
In terms of performance, Value and Policy iteration are guaranteed to find the optimal solution to

the MDP as long as they are run for a sufficient number of iterations. TD-Learning and Q-Learning, on the other hand, can only approximate the optimal solution and may not converge to it. However, the advantage of TD-Learning and Q-Learning is that they do not require knowledge of the T and Rewards of the MDP, which makes them more flexible and applicable to real-world problems where such knowledge may be incomplete or unavailable.

Grids	Episodes	Time
Grid 1	1500	4.468 seconds.
Grid 2	1500	7.05 seconds.
Grid 3	1500	8.926 seconds.

Table 3: Q-Learning

Grids	Episodes	Time
Grid 1	1500	1.318 seconds.
Grid 2	1500	3.195 seconds.
Grid 3	1500	4.003 seconds.

Table 4: TD-Learning

Q8: What are the effects of epsilon and alpha values and how are they modified?

The epsilon value determines the probability of choosing a random action instead of the action with the highest estimated value. A higher epsilon value results in more exploration, and a lower epsilon value results in more exploitation. It is typically decreased over time to allow the agent to increasingly rely on its learned values and exploit its knowledge of the environment.

The alpha value determines the extent to which the agent updates its value estimates based on new information. A higher alpha value results in faster learning, and a lower alpha value results in slower learning. The alpha value is often decreased over time to allow the agent to incorporate new information while also preserving the knowledge it has previously learned. ...

Q9: What is the effect of a number of episodes?

The number of episodes refers to the number of times the agent interacts with the environment in order to learn and improve its policy. Increasing the number of episodes can lead to better performance of the reinforcement learning algorithm, as it allows the agent to gather more data and learn from a larger number of experiences. However, increasing the number of episodes also increases the computational time required to train the agent, and there may be diminishing returns in terms of performance after a certain point.

It is important to balance the trade-off between the performance of the algorithm and the computational time required to train it by choosing an appropriate number of episodes. This will depend on the specific problem and the resources available. In general, more episodes may be needed for more complex environments or tasks, while fewer episodes may be sufficient for simpler environments or tasks.

Task 4 (extra credits)

Q10: Describe how the environment is created in this task. Present and discuss your experiment findings.

The task for extra credits was not implemented.

Conclusion

The objective of this lab was to use reinforcement learning to solve a Markov Decision Process in a grid world. We implemented and used non-deterministic search algorithms, including policy iteration, TD-learning, and Q-learning, to find the optimal paths and maximize the value function in the MDP. The environment was represented as a grid with various parameters, including noise, immediate rewards, terminal states, and walls, which were stored in lists. The agent could take four actions, stored in the Directions list, and the dynamics of the environment were stored in a transition matrix.
The results of these algorithms were compared in terms of convergence time and generated policies.