Assignment 9

TDT4171 — Artificial Intelligence Methods

February 2023

Information

- **Delivery deadline: March 27, 2023 by 23:59**. No late delivery will be graded! Deadline extensions will only be considered for extraordinary situations such as family or health-related circumstances. These circumstances must be documented, e.g., with a doctor's note ("legeerklæring"). Having a lot of work in other classes is not a legitimate excuse for late delivery.
- Cribbing("koking") from other students is not accepted, and if detected, will lead to immediate failure of the course. The consequence will apply to both the source and the one cribbing.
- Students can **not** work in groups. Each student can only submit a solution individually.
- Required reading for this assignment: Chapter 16. Making Complex Decisions (the parts in the curriculum found on Blackboard, "Sources and syllabus" → "Preliminary syllabus") of Artificial Intelligence: A Modern Approach, Global Edition, 4th edition, Russell & Norvig.
- For help and questions related to the assignment, **ask the student assistants during the guidance hours**. The timetable for guidance hours can be found under "Course work" → "Information for Guidance Hours" on Blackboard. For other inquires, an email can be sent to tdt4171@idi.ntnu.no.
- Deliver your solution on Blackboard. Please upload your assignment as one PDF report and one source file containing the code (i.e., one .py file) as shown in Figure 1.

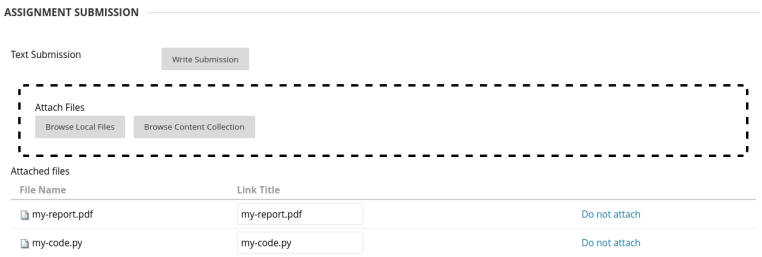

Figure 1: Delivery Example

Value Iteration

Imagine that there is a frozen lake between your home and the school, like the one shown in Table 1. As we can see from the table, states are represented as integers from 0 to 15. To get to the school, you need to walk on the ice by using four different actions: left, down, right, and up. The actions are encoded as integers from 0 to 3, left = 0, down = 1, right = 2, and up = 3.

However, there is a problem! There are holes on the frozen lake, so you will have to be careful while walking on it to avoid falling into one of the holes. Especially since there is 0.1 chance of going to either side and 0.8 in the direction you want when moving forward. Moreover, if you fall into one of the holes, you get -1.0 in reward and get stuck. Because, according to the transition function, the probability of transitioning to the same state when you fall into a hole is 1, regardless of which action you take. Thus, you'll be unable to escape. If you try to take an illegal action, such as going up when you're home, then there is 0.9 chance of staying home or 0.1 going to the side, in this case, right. There is 0 probability of going in the opposite direction of the action taken. Finally, assuming that there are only states on one side when you move forward, then there is 0.1 chance of staying in the same state. All these probabilities can be inspected using the provided function `get_trans_prob`, detailed later in the assignment text.

Your goal is to get from home to school by stepping on the ice without falling into the holes. Also, you would like to do it in the shortest amount of time since you get -0.1 in reward for each time step. You get 1 in reward if you succeed in getting to the school.

Home state = 0 reward = −0.1	Ice state = 1 reward = −0.1	Ice state = 2 reward = −0.1	Ice state = 3 reward = −0.1
Ice state = 4 reward = −0.1	Hole state = 5 reward = −1.0	Ice state = 6 reward = −0.1	Hole state = 7 reward = −1.0
Ice state = 8 reward = −0.1	Ice state = 9 reward = −0.1	Ice state = 10 reward = −0.1	Hole state = 11 reward = −1.0
Hole state = 12 reward = −1.0	Ice state = 13 reward = −0.1	Ice state = 14 reward = −0.1	School state = 15 reward = 1.0

Table 1: The frozen lake environment.

For this assignment, you will implement the value iteration algorithm (shown in Figure 16.6 in the textbook) to find the utility of states. Afterward, extract the corresponding greedy policy from these utilities to get you from home to school. We have provided a skeleton code for this assignment along with the necessary functions and constants. The information about the skeleton code is written as docstrings and comments in the provided skeleton code. The skeleton code provides the following functions:

1. `get_next_states(state: int, action: int) -> list[int]:` fetches the possible next states given the state and action pair. The next states follow from the transition probabilities. For instance, calling `get_next_states(state=0, action=1)` gives us the next possible states if we are at home (state = 0) and takes the action down (action = 1). The function returns, `[0, 4, 1]` which means that we either don't move (since there is nonzero probability to not transition when taking an action), move down (state = 4) or move to the right (state = 1).

If we call, `get_next_states(state=0, action=3)` which is if we are at home (state = 0) and try to go up (action = 3, not possible). The function will return, `[1, 0]` which means that we will either stay home (state = 0) or transition to the right (state = 1).

Finally, if we call `get_next_states(state=5, action=1)`, we get, `[5]` as it's not possible to get out of a hole.

2. `get_trans_prob(state: int, action: int, next_state: int) -> float:` fetches the transition probability for the next state given the state and action pair according to pre-specified transition probabilities. For example, calling `get_trans_prob(state=0, action=1, next_state=0)` returns 0.1. This means that if we are at home (state = 0) and take the action down (action = 1) there is 0.1 chance that we will not transition to a new state and stay at home for the next time step too.

   If we call `get_trans_prob(state=4, action=3, next_state=8)`, we get 0 probability, since there is no probability of transition down to state 8 if we try to go up (using action = 3). The same happens when we call `get_trans_prob(state=4, action=2, next_state=6)`, which returns 0 since state 4 and state 6 are not neighbors.

3. `get_reward(state: int) -> float:` fetches the reward given the state. If we call `get_reward(state=1)` we get -0.1 in reward. The call `get_reward(state=5)` provides -1 in reward. Finally, `get_reward(state=15)` returns 1 in reward. Rewards for all states can be seen in Table 1.

4. `get_action_as_str(action: int) -> str:` fetches the string representation of an action. For instance, the action left is encoded as 0. Thus, `get_action_as_str(action=0)` will return left.

The skeleton code also provides constants needed for the value iteration algorithm. **Do not change these values**, except `DETERMINISTIC` when debugging.

1. `N_STATES`: the number of states in the Markov decision process (MDP).
2. `N_ACTIONS`: the number of actions in the MDP.
3. `GAMMA`: the discount factor. This corresponds to γ in the Figure 16.6 of the textbook.
4. `EPSILON`: the maximum error allowed in the utility of any state. This corresponds to ϵ in the Figure 16.6 of the textbook.
5. `DETERMINISTIC`: whether system dynamics are stochastic or not. You can assign `DETERMINISTIC = True` when you debug your code so that you don't have to deal with stochasticity. That is, you will transition to the state you want without uncertainty. However, your final result needs to use `DETERMINISTIC = False`.

Tasks

1. Find the utilities for this described environment by using the value iteration algorithm and provided functions and constants. Afterward, display utilities similar to how it is done in Figure 16.3 of the textbook. You can do it either by printing the utilities as a table to the console, or visualize it with a library such as Matplotlib by following this guide (Creating annotated heatmaps) or use Seaborn's heatmap function.

2. Based on the utilities you found, find the corresponding greedy policy and visualize it. You can do it either by printing the (state, action) pairs to the console, or visualize it with a library such as Matplotlib.

3. Add your visualizations to your PDF report.

Note

The code must be runnable without any modifications after delivery. Moreover, the code must be readable and contain explaining comments. This is especially important if the code does not work. Finally, do not archive (e.g., zip files) your source file when delivering on *Blackboard*.