Planning Algorithms in AI
PS3: Value Iteration and Markov Decision Process

Gonzalo Ferrer
Skoltech

1 December 2025

This problem set has two tasks and is individual work. You are encouraged to talk at the conceptual level with other students, discuss on the equations and even on results, but you may not show/share/copy any non-trivial code.

Submission Instructions

Your assignment must be submitted by 11:59pm on **December 11**. You are to upload your assignment directly to Canvas as **two** attachments:

1. A `.tar.gz` or `.zip` file *containing a directory* named after your name with the structure shown below.

```
yourname_ps3.zip:
yourname_ps3/run.py
yourname_ps3/utils.py
yourname_ps3/vi.py
yourname_ps3/mdp.py
yourname_ps3/plan_vi.mp4
yourname_ps3/plan_mdp.mp4
```

2. A PDF with the answers to the questions below. Printed notebooks or scanned versions of hand-written documents, converted to PDFs, are perfectly acceptable (reduced size). No other formats (e.g., .doc) are acceptable. Your PDF file should adhere to the following naming convention: `yourname_ps3.pdf`.

Homework received after 11:59pm is considered late and will be penalized as per the course policy. The ultimate timestamp authority is the one assigned to your upload by Canvas. No exceptions to this policy will be made.

NOTE: you can submit a notebook instead of `run.py`, making sure it runs correctly with the give order of cells and the corresponding dependencies. We suggest to run the scripts.

Problem Description

The environment is a simple 2d grid world with some obstacles. You can find the environment in the file `data_ps3.npz` and visualize it by simply executing the `run.py` script.

To support you and your algorithm, we provide a set of functions found in the `utils.py` file:

- `action_space`: This list contains all available actions.

- `plot_enviroment`: This function is for visualization.
- `transition_function`: This is an implementation of $s' = f(s, a)$. The action a is an element of `action_space`. It returns the next propagated state plus the results of the propagation: False, if it failed to propagate du to obstacles or end of bounds.
- `state_consistency_check`: This function checks whether or not the state is valid, i.e., is an obstacle or out of bounds.
- `probabilistic_transition_function`: This is an implementation of the probability of transition $Pr\{s' \mid s, a\}$ as a result of the noisy transition function $s' = f(s, a) + \epsilon$ for ϵ a r.v. The output is a list of propagated states and its corresponding probabilities.
- `step`: This is used for visualizations, and it propagates the system according to the action given and the internal state.

Look for more details in help and in the code of these functions.

Task 1: Value Iteration G^*

In this task, you will calculate the optimal *cost-to-go* by using the Value Iteration algorithm explained in class.

A. (5 pts) Enumerate the action space. The coordinates of actions are $a = (\text{row}, \text{column})$.

B. (5 pts) Re-formulate mathematically the optimal cost-to-go $G^*(s)$ in recursive form. Note we are using the later notation discussed in class for the states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$, however, the objective is to minimize G^*.

C. (20 pts) Implement the VI algorithm for infinite length sequences. To show this, you are asked to include a picture of the final G^* (using `imshow()` for instance).

The cost of traversing each node $l(s, a) = 1$ only if propagation is possible (there is not obstacle or out of bounds)

Hint: You can implement a convergence criterion or simply run for a large number of iteration, say 100. Both options will be correct.

D. (5 pts) Formulate how to obtain the optimal policy $a^* = \pi_{VI}(s)$ from G^*.

E. (10 pts) Implement an algorithm to obtain the optimal policy a^* from G^*. This policy can be a table. To test this, start at an initial position and execute the result of your policy and the transition function until you reach the goal. For simplicity, the `step` function is set to behave very close to a deterministic function ($\epsilon = 0.001$). You will upload the video, that should be automatically generated if using the code in `run.py`.

Task 2: Markov Decision Process (MDP)

A. (5 pts) Formulate the optimal value function $v_*(s)$ in recursive form.

B. (5 pts) Formulate how to obtain the greedy deterministic policy $a^* = \pi_{MDP}(s)$ from $v_*(s)$.

C. (20 pts) Implement the MDP optimal value function until convergence of v_* using the same criterion as in 1.C. Include in your report a picture of the values v_*.

Note: Now, instead of cost, we will work with rewards having the objective of maximizing the return. Also, we have to assign some values, in this case, we recommend to start the algorithm with $v(s_{\text{goal}}) = 1$, the reward to collide with an obstacle or out of bounds $r(s' = \text{obstacle}) = -1$ (it also includes out of bounds) and the successful transition $r(s' = \text{free}) = 0$.

D. (10 pts) Implement the optimal policy a^* from v_* using same principles as in 1.E. To test this, start at an initial position and execute the result of your policy and the transition function until you reach the goal. You will upload the video, that should be automatically generated if using the code in `run.py`.

E. (5 pts) Explain the difference between the VI plan in 1.E and the MDP in 2.D.

F. (5 pts) Experiment with different values of epsilon at the `step` function in visualization for both methods. Explain your observations.

G. (5 pts) Experiment with different parameters, such as starting states, goal states, different values of epsilon at training, the convergence parameter `gamma` from `mdp()`. Explain your observations.