National University of Singapore

CS4246/CS5446 AI Planning and Decision Making

Homework 1

Submission Deadline: **Feb 28, 2020 @ 23:59**

Please write the following at the top of your files (for both written assignment and programming assignment)

- Name
- Collaborators (write none if no collaborators)
- Source, if you obtained the solution through research, e.g. through the web.

On Collaboration

- While you may collaborate, you *must write up the solution yourself.*
- It is okay for the solution ideas to come from discussions. However, it is considered as plagiarism if the solution write-up is highly similar to your collaborator's write-up or to other sources.

Submission

- Your solution for both the written part and programming assignment should be submitted on LumiNUS.
  - For the written part, please type out your answers and submit a PDF file.
  - Name the file `[YourMatricNumber]_Homework1-Written.pdf`
  - For the programming part, you need to submit a zip file. Please follow the instructions at the end of the document clearly.
- For the programming assignment, code should be uploaded to aiVLE evaluation server **in addition to LumiNUS.**
- ***Late submission:*** You will incur a late penalty of 20% of your score for the late submissions.
  - If you submit the written and programming assignments at two different times, we will consider the later one as your submission time.
  - No submission will be accepted after March 2, 2020 @ 23:59.

---

1. **Classical Planning**

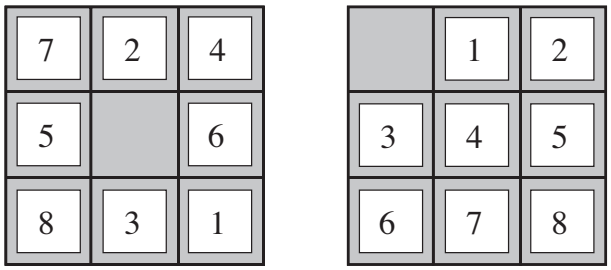
   (a) Consider the 8 puzzle with the Slide schema.

   Consider

   (i) ignoring $Blank(s_2)$ in the precondition as a heuristic and

   (ii) ignoring $Blank(s_2) \wedge Adjacent(s_1, s_2)$ in the precondition as a heuristic.

   Which of (i) or (ii) will result in fewer nodes being explored when used with the A^* algorithm?



Start State Goal State

$Action(Slide(t, s_1, s_2)$,
PRECOND: $On(t, s_1) \wedge Tile(t) \wedge Blank(s_2) \wedge Adjacent(s_1, s_2)$
EFFECT: $On(t, s_2) \wedge Blank(s_1) \wedge \neg On(t, s_1) \wedge \neg Blank(s_2))$

$Init(At(C_1,\ SFO) \wedge At(C_2,\ JFK) \wedge At(P_1,\ SFO) \wedge At(P_2,\ JFK)$
$\wedge Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2)$
$\wedge Airport(JFK) \wedge Airport(SFO))$
$Goal(At(C_1,\ JFK) \wedge At(C_2,\ SFO))$
$Action(Load(c,\ p,\ a)$,
PRECOND: $At(c,\ a) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $\neg\ At(c,\ a) \wedge In(c,\ p))$
$Action(Unload(c,\ p,\ a)$,
PRECOND: $In(c,\ p) \wedge At(p,\ a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $At(c,\ a) \wedge \neg\ In(c,\ p))$
$Action(Fly(p,\ from,\ to)$,
PRECOND: $At(p,\ from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg\ At(p,\ from) \wedge At(p,\ to))$

(b) Consider the Air Cargo problem.
Describe how to modify the problem so that each plane can only carry one cargo.

(c) In the Air Cargo problem, write the successor state axiom for the fluent $At(P_1, SFO)$.

2. **Decision Theory**

(a) Bob is risk adverse but rational. His utilities for A, B, and C are $U(A) = 0$, $U(B) = 100$ and $U(C) = 40$. He is given a choice between C and a lottery $[0.4, A; 0.6, B]$. Which would he choose and why?

(b) Alice's utility function for money is $U(x) = x^2$. Argue that Alice is risk seeking. (Hint: $U(x)$ is a strictly convex function. Jensen's inequality may be useful here.)

(c) Cathy prefers A to B but prefers lottery $C = [0.2, A; 0.8, B]$ to lottery $D = [0.3, A; 0.7, B]$. Argue that there is no utility function that satisfies Cathy's preferences.

3. **Programming Assignment**

   In this task, we will learn to generate the PDDL description files which will be used to solve 2 different planning problems. Before proceeding further, follow the instructions below to complete the setup required for this programming assignment.

   Installation Instructions[1]

   - Setup docker on your machine, instructions for which can be found here.
   - Pull the docker that we have already setup for you with all the required dependencies. You can follow the instructions here to do so.
   - After pulling the docker image, test your your installation by running
     `docker run -it --rm -v $PWD:/workspace cs4246/base python test_installation.py`
     for Linux/Mac or
     `docker run -it --rm -v \%(cd)\%:/workspace cs4246/base`
     `python test_installation.py` for Windows (file given along with this assignment) on the docker container.

   We will be using the gym_grid_environment (https://github.com/cs4246/gym-grid-driving) to simulate the solution obtained on feeding the PDDL files generated to a planner. These dependencies have been installed in the docker image "cs4246/base" which you have downloaded.

   Read through the introduction and example from https://github.com/pellierd/pddl4j/wiki/A-tutorial-to-start-with-PDDL to understand the PDDL description format. You can look at sample problem and domain files for further understanding. If you want to, you can also feed any problem and domain PDDL files to a planner by running the following command on the docker container (using the `docker run ...` command):

   ```
   /fast_downward/fast-downward.py  domain.pddl problem.pddl  --search "lazy_greedy([ff()], preferred=[ff()])"
   ```

   Note that in this PDDL format, specifying negative literals in preconditions is allowed.

   For you to get used to the PDDL format, we have prepared a sample python script to generate PDDL files for the Air Cargo problem. You can run the file `pddl_cargo_example.py` (using the `docker run ...` command)] to see the PDDL files generated `cargodomain.pddl`, `cargoproblem.pddl` and also relevant portions of the code that generates it. Specifically, it will be helpful to look at functions :

   (a) `generateDomainPDDLFile()`,

   (b) `generateProblemPDDLFile()`

   (c) `generateInitString()`

[1]The files to be installed are very large. Do your installation in while in SoC if it is too slow to do at home.

(d) `generateGoalString()`

To complete this assignment, you need to know some details about the environment : how a state is represented, the actions the agent can take, and few other small intricacies. We have prepared an IPython Notebook file on Google Colab here to walk you through all such details. It would be helpful for you to go through it before working on the tasks. While this may seem like a lot to understand for one homework assignment, we will be using the same environment for the other homeworks and for the project!

You are now ready to begin solving the two tasks which are listed below.

(a) **Parking Task :**
You are employed as a valet at a parking lot. You have to drive the car given to you and park it at a parking spot assigned to the car (identified as goal state in the environment). But, you also have to plan your way from your current position to the spot while wasting the least fuel possible (your tip depends on it!).

By virtue of being a Computing student, you decide to put your *planning* skills to use. You first retrieve an old python script written for a similar task by the TAs of CS4246, parts of which have been lost. The script used to generate the PDDL files and fed them to the fast-downward solver to generate a solution for this planning problem. It also runs the plan on the simulator to see if it does the job.

The script `python __init__.py` is missing 3 code snippets (marked with "FILL ME" in the file) .

i. Code which generates the action schemas of the three actions available namely, UP, DOWN and FORWARD. These action schemas should reflect the 3 actions available in the environment simulator. The agent's speed range in the environment is restricted to [-1, -1] (negative speed moves towards the left, see the environment for more details).
ii. Code which generates the initial state condition.
iii. Code which generates the goal description.

You can test your code with the test configurations given in the script by running `python __init__.py` (present in the .zip file) on the docker (using the `docker run ...` command). **It might be helpful to look at the generated PDDL files for debugging.**

(b) **Crossing the road :**
A customer forgets directions to the parking lot, and ends up on the other side of the road. The customer is not skilled enough to cross a busy multi-lane road with **moving cars**, where the speed of cars can differ across lanes, but cars in the same lane move with the same speed. To reach the entrance of the parking lot (identified as goal state in the environment), he calls you up and asks you to drive the car from his spot (the initial state) to the destination. Unlike the previous task, the agent's speed range in the environment is restricted to [-3, -1].

You decide to modify the script you wrote in Task 1 to handle this situation. You can test your code with the test configurations given in `__init__py` (Replace the existing test cases with the one for the Crossing Task) .

Hint: Instead of modeling the parking lot/road as a 2 dimensional grid, it might be useful to include time as an additional dimension.

Submission Instructions

Please follow the instructions listed here. You have to make 2 separate submissions as mentioned below :

i. For Task (a) : You have been provided with a .zip file in the correct submission format. Complete the functions marked with "FILL ME" in `__init__.py`. Your submission zip file should have the exact same structure as the zip file you received.

ii. For Task (b) : Modify the `__init__.py` as required and make a separate submission.

Note :

i. Please remember to set the variable SUBMISSION in `__init__.py` to False when testing locally, and to True before making a submission.

ii. Please do not print anything to the console, as it might interfere with the grading script.