Universitat Pompeu Fabra Barcelona

Autonomous Systems – EMAI 2024-25

Lab Project 2: Classical Planning

Deadline: Friday 22 November at 23:55

This assignment must be done in groups of 2 students.

The purpose of this project is to get you acquainted with classical planning, and specifically how to model classical planning problems in PDDL.

You have to **submit just one zip** file with extension .zip. It's name must be "p2_uX_uY.zip", where X and Y are each student's personal U number at UPF. Any other compression format (e.g., rar, tar, bzip2, etc.) will not be marked. The zip file should include one directory for each of the three problem sets below (i.e., `1_navigation`, `2_keys_and_doors`, `3_generalized`), that contain the **PDDL domain**, the **PDDL instances** (in case of classical planning), and the **solution file(s)** on the indicated format.

Academic Misconduct: You may not collaborate with people outside your group, or plagiarise their work. Groups are expected to present the results of their own thinking, problem solving, and coding. Never copy another group's work and never give your written work to others. Never copy your solution, or part of it, from the web or any other resource. Adapting someone else solution does not make it your own work: you are meant to generate the solution to the questions by yourself. You may however reuse code or techniques that are auxiliary to the problem being solved, for small things (e.g., use a NumPy function to perform an operation), as long as you understand well the code being reused and document where it comes from. In this case, make sure to document the source of this code in the comments, and the specific use you are making.

Plagiarism is a very serious issue. Suspected collusion or plagiarism will be dealt accordingly.

Forum postings on assignment: Do not ever post any information on the forum that may disclose how to solve a question or what the solution may be. You can only post assignment related questions for clarification on what is being asked, for example, whether a formal proof is required in a given exercise or to clarify certain notation. Any post discussing possible solutions or strategies may directly be considered plagiarism, see above. **If in doubt, do not post** and ask your question to the tutor instead.

Good luck!

1 Navigation problems

For starters, we will solve simple navigation problems where an agent has to navigate from an initial location to a goal location. The agent has access to the four actions (`up`), (`down`), (`left`) and (`right`).

The solution to a navigation problem is in the form of a **plan**, i.e. an action sequence that brings the agent from the initial location to the goal location. The solution to Problem A has to be saved in a text file called '`A.sol`' on the following format:

```
(up)
(left)
(up)
...
```

The first line identifies the problem by its letter, and is followed by a list of actions in parentheses, one per line. The solution to Problem B should be saved in a text file called '`B.sol`', etc.

To solve a problem in PDDL, you have to write a PDDL definition of the navigation problem.To model the problem using only four actions, it is necessary to use **conditional effects** (cf. Lecture 8). Note that some problems (C-F) have walls (marked with X) that cannot be traversed. If the agent takes an action that would enter a wall or leave the grid, the agent should instead remain in the current location.

The initial configuration of each problem appears in a text file in the supplied material `lab2-files` (e.g. '`A.txt`'). One way to save time is to write a small program (e.g. in Python or C++) that **translates** the initial configuration to a PDDL instance, rather than writing the definition manually for each problem.

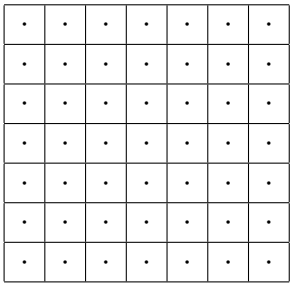
Once you have written the PDDL domain and the PDDL instance of a problem, you can use a classical planner (e.g. `editor.planning.domains`) to automatically produce a plan that solves the problem. Make sure that your actions have the same names as in the example.

Problem A

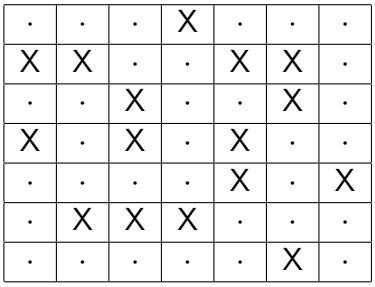
#	·	·	·	·
·	·	·	·	·
·	·	·	·	·
·	·	·	·	·
·	·	·	·	@

To illustrate, the initial location (bottom right corner) is marked with @, and the goal location (top left corner) is marked with #. In the following problems, the initial location (bottom right corner) and goal location (top left corner) are always the same, but not marked.

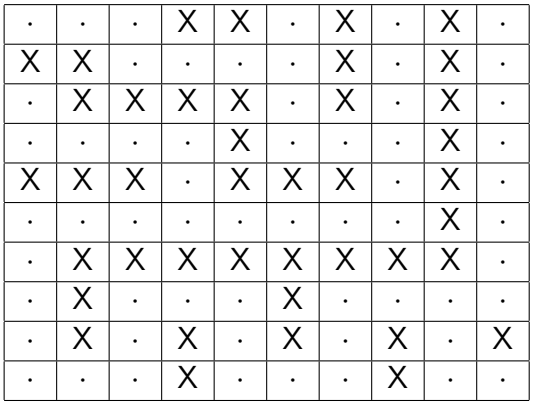
Problem B



Problem C



Problem D



Problem E

·	X	·	·	·	X	·	·	·	X	·	·	·	·	·
·	X	·	X	·	·	·	X	·	·	·	X	X	X	·
·	·	·	X	X	X	X	X	X	X	·	·	·	X	·
X	X	X	X	·	X	·	·	·	X	X	X	X	X	·
·	·	·	·	·	X	·	X	·	·	X	·	·	·	·
·	X	X	X	X	X	·	X	X	·	X	·	X	X	X
·	·	·	·	·	·	·	·	X	·	X	·	·	·	·
X	X	X	X	X	X	X	X	X	·	X	X	X	X	·
·	·	·	·	·	·	·	·	X	·	·	·	·	X	·
·	X	X	X	X	X	X	·	X	X	X	X	·	X	·
·	X	·	·	X	·	X	·	·	·	·	·	·	X	·
·	X	·	X	X	·	X	X	X	X	X	X	X	X	·
·	X	·	·	·	·	X	·	·	·	X	·	·	·	·
·	X	X	·	X	·	X	·	X	·	X	·	X	·	X
·	·	·	·	X	·	·	·	X	·	·	·	X	·	·

Problem F

·	·	·	·	·	X	·	·	·	·	X	·	·	·	·	X	·	·	·	·
X	X	X	X	·	X	X	X	X	·	X	·	X	X	X	X	X	X	X	·
·	·	·	·	·	X	·	·	·	·	X	·	·	·	·	·	X	·	X	·
·	X	X	X	X	X	·	X	X	X	X	X	X	·	X	·	X	·	X	·
·	X	·	·	·	·	·	X	·	·	·	·	·	·	X	·	X	·	X	·
·	·	·	X	X	X	·	X	·	X	X	X	X	X	X	·	X	·	·	·
X	X	X	X	·	·	·	X	·	X	·	·	·	·	·	·	X	·	X	X
·	·	·	·	·	X	·	X	·	X	·	X	X	X	X	X	X	·	·	·
·	X	X	X	X	X	·	X	·	X	·	·	·	·	·	·	X	X	X	·
·	X	·	·	·	·	·	X	·	X	X	X	X	X	X	·	X	·	·	·
·	X	X	X	X	X	X	X	·	X	·	·	X	·	·	·	X	·	X	·
·	·	·	·	·	·	·	·	·	X	·	X	X	·	X	X	X	·	X	·
X	X	X	X	X	X	X	X	X	X	·	·	·	·	X	·	·	·	X	·
·	·	·	·	·	·	·	·	·	X	·	X	X	X	X	·	X	X	X	·
·	X	X	X	X	X	X	X	X	X	·	X	·	·	X	·	·	·	X	·
·	·	·	·	·	·	·	·	·	·	·	X	·	X	X	X	X	·	X	X
·	X	X	X	X	·	X	X	X	X	X	X	·	X	·	·	·	·	·	·
·	·	·	·	X	·	·	·	·	·	·	·	·	X	·	X	X	X	X	·
X	X	X	·	X	X	X	X	X	X	X	X	X	X	·	·	·	·	X	·
·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	X	X	·	X	·

2 Keys and doors

Next we will extend the navigation problem with keys and doors. Some of the blocked grid locations represent **doors** that can be opened using **keys**. Each door is marked with an uppercase letter, and the key that opens this door is marked with the same letter in lowercase. For these problems, the agent has two additional actions available:

- (`pickup kc`): pick up the key (`k=key`) represented by the letter `'c'`. The agent has to be in the **same location** as the key.
- (`unlock dC kc`): open the door (`d=door`) represented by the letter `'C'` with the key `'c'`. The agent has to be **adjacent** to the door in either of the four cardinal directions (`up, down, left, right`).

The agent can only carry one key at a time, and this key will automatically be consumed when it is used to open a door. Only then can the agent pick up another key.

You can use the PDDL domain from the navigation problems as a starting point, since the actions (`up`), (`down`), (`left`) and (`right`) are almost identical. As before, it is possible to write a program that translates the initial configuration of a problem to a PDDL instance file.

Problem G

The problem is still to navigate from the bottom right corner to the top left corner. In the problems with keys and doors, a lowercase letter represents a key, and an uppercase letter (different from X) represents a door.

·	·	X	a
·	·	X	·
A	X	X	·
·	·	·	·

Problem H

·	·	X	·	c
C	X	X	·	b
·	·	X	·	a
B	X	X	·	·
·	·	A	·	·

Problem I

·	·	·	X	·	·	c
·	·	·	X	·	·	·
·	·	·	X	·	·	·
X	C	X	X	X	B	X
b	·	·	X	a	·	·
·	·	·	A	·	·	·
·	·	·	X	·	·	·

Problem J

·	·	·	W	·	·	X	w	X	r
X	X	X	X	·	·	X	·	X	·
v	X	u	X	·	p	X	U	X	V
·	X	·	X	·	·	X	o	X	q
O	X	P	X	Q	R	X	S	X	T
s	X	t	X	i	j	X	l	X	k
I	X	J	X	K	L	X	M	X	N
f	X	n	X	c	·	X	e	X	d
C	X	D	X	E	F	X	G	X	H
m	A	g	B	·	·	a	b	h	·

3 Generalized planning

The third exercise consists in completing the PDDL domain definition of Gripper, and executing a generalized planner called BFGP (Best-First Generalized Planning) in order to produce a generalized plan that solves **all instances** in a given set. The supplied code contains a notebook (`exercise3.ipynb`), the incomplete domain definition of Gripper (`domain.pddl`), and a set of five Gripper instances (`p01.pddl--p05.pddl`).

To solve the exercise you need to perform the following steps:

- Complete the definition of the PDDL domain of Gripper ("`domain.pddl`"). To test whether the definition is correct, you can use `editor.planning.domains`.
- Upload the notebook (`exercise3.ipynb`) to `Colab` and run the Installation part of the code.
- In the file system (the folder symbol in the left sidebar), right-click and create a new folder "`gripper`".
- Upload the PDDL files to the "`gripper`" folder (including the complete "`domain.pddl`").
- Run the Exercises part of the code, and modify the number of program lines until BFGP finds a generalized plan.
- Download the generalized plan located at "`tmp/gripper/gripper.prog`".
- Add your domain file "`domain.pddl`" and solution "`gripper.prog`" to your submission zip file.