Informatics 2D Coursework 2: Symbolic Planning (v.1.1)

Rimvydas Rubavicius and Alex Lascarides

Deadline: **3pm, Thursday 31st March 2022**

Introduction

This coursework is about designing and evaluating symbolic planning domains and problems using Planning Domain Definition Language (PDDL). It is marked out of 100 and worth 15% of the overall course grade. It consists of four tasks:

- **Modelling**: formalization of the domain, given a specification (25 marks)
- **Implementation**: writing your formalisation in PDDL and testing it on problem instances within an existing PDDL planner (10 marks)
- **Experiments**: designing an experiment to evaluate a PDDL planner (15 marks)
- **Extensions**: extending the domain to deal with real world challenges (50 marks)

The files you need are on the course LEARN website: click on "Assessment" (on the LHS menu) and go down to"Coursework 2: Symbolic Planning". Alternatively, you can access the files here. You will download a file `Inf2d-coursework2.tar.gz` which can be unpacked using the command:

```
tar -xf Inf2d-coursework2.tar.gz
```

This will create a directory `Inf2d-coursework2` which contains: example blocks world domain and problem files (`EXAMPLE-blocks-world-domain.pddl`,`EXAMPLE-blocks-world-problem.pddl`), the metric FF planner (`ff`), and a report template (`report.tex`).[1]

[1] If you have not used LaTeXbefore, you may find https://computing.help.inf.ed.ac.uk/latex and https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes useful.

Submision

Create a directory `Inf2d-cw2-s<matric>` where `<matric>` is your matriculation number (e.g 1234567). This directory should include the following files for the submission (if you do not attempt some tasks, do not include associated files):

```
Inf2d-cw2-s<matric>
├── domain-ext-1.pddl
├── domain-ext-2.pddl
├── domain-ext-3.pddl
├── domain.pddl
├── problem-1.pddl
├── problem-2.pddl
├── problem-3.pddl
├── problem-ext-1.pddl
├── problem-ext-2.pddl
├── problem-ext-3.pddl
└── report.pdf
```

Please note you need to compile `report.tex` into `report.pdf` and submit that.

Before submitting, you should compress the directory into `Inf2d-cw2-s<matric>.tar.gz`. You can do this using the following command:

```
tar -cvzf Inf2d-cw2-s<matric>.tar.gz Inf2d-cw2-s<matric>
```

You can check that this file stores the intended data with the following command, which lists all the files one will get when one extracts the original directory (and its files) from this file:

```
tar -t Inf2d-cw2-s<matric>.tar.gz
```

Also, you can check the size of the above file using the following command, to check that the file stores the intended data:

```
ls -l inf2d-cw2-s<matric>.tar.gz
```

Submit this file via LEARN by uploading the file using the interface on the LEARN website for the course. If you have trouble please refer to the blogpost here.

Failure to comply with this specification will result in reduction of 5 marks awarded for your submission.

The deadline for submission is **3pm, Thursday 31st March 2022**.

You can submit more than once up until the submission deadline. All submissions are timestamped automatically. Identically named files will overwrite earlier submitted versions, so we will mark the latest submission that comes in before the deadline.

If you submit anything before the deadline, you may not resubmit afterward. (This policy allows us to begin marking submissions immediately after the deadline, without having to worry that some may need to be re-marked).

If you do not submit anything before the deadline, you may submit exactly once after the deadline, and a late penalty will be applied to this submission, unless you have received an approved extension. Please be aware that late submissions may receive lower priority for marking, and marks may not be returned within the same time frame as for on-time submissions.

For information about late penalties and extension requests, see the School web page here.

Do not email any course staff directly about extension requests; you must follow the instructions on the web page.

Good Scholarly Practice: Please remember the School's requirements as regards all assessed work for credit. Details about this can be found at: http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct

Furthermore, you are required to take reasonable measures to protect your assessed work from unauthorised access. For example, if you put any such work on a public repository (e.g., GitHub), then you must set access permissions appropriately (for this coursework, that means only you should be able to access it).

Task 1: Modelling (25 marks)

Problem Specification

Consider a post office in which MailBot helps a teller to serve customers. When a customer requests a package, the teller sends the request to MailBot to deliver it (or them). To do this, MailBot has to bring the desired packages from the warehouse to the delivery belt and turn it on, so as to transport the package to the teller. Before putting the package on the delivery belt, MailBot has to scan it using portable scanner. In this coursework, you will model the warehouse and the MailBot actions using Planning Domain Definition Language (PDDL). In later tasks, you will implement your specification to be executable by the planner `ff`. To successfully execute teller's requests, MailBot must have the capability to move through the warehouse, be able to carry and use the scanner, pick up and put packages onto the delivery belt and use the belt to deliver the packages to the teller.

Task 1.1: Describing The World State (5 marks)

Your first subtask is to define the predicates which will describe the world state in your domain. Your world model should be able to describe the following:

- Warehouse, i.e. layout of the warehouse and which cells are *connected*.
- The location of MailBot and other objects such as the portable scanner, packages, and delivery belt.
- Whether or not the delivery belt is *turned on*.
- Whether or not a particular package has been *scanned* by MailBot.
- Whether or not MailBot is holding an object.

In `report.tex`, name all predicates and constants that you will use to describe the world, and give a brief description in English of what they represent. If you need additional predicates to model the actions in task 1.2 please also describe them here as well. Additionally, in the report write down an initial state as visualized in Figure 1. Ensure the initial state is a valid PDDL initial state which could be used to create a plan for completing the problem given Task 1.2. actions: in particular, make sure that it complies with the syntactic constraints on representing states in PDDL.

Task 1.2: Actions (10 marks)

In this subtask, define the actions MailBot can perform. Define the arguments, preconditions and effects of each of the actions and include them in `report.tex`:

- MailBot can *move* between two tiles if they are connected. MailBot cannot move diagonally or through the wall.
- MailBot can *pick up* objects, such as the scanner, if it is in the same tile as the object and it is not holding anything else. That is, it can only hold one object at any one time.

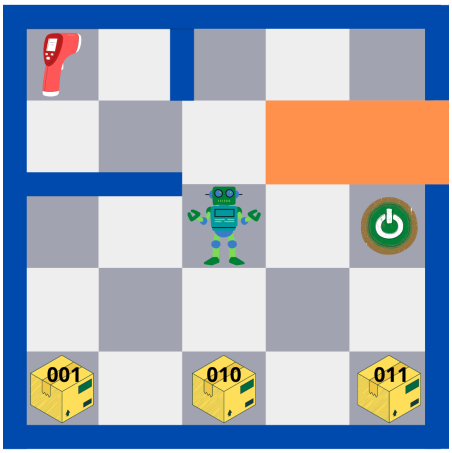

Figure 1: Initial world state of the domain. The warehouse includes 5x5 grid of tiles (dark/light gray), a portable scanner (red), delivery belt (orange) with a switch (brown outline) which is green when delivery belt is turned off and red when it is turned on, walls (blue), packages (yellow boxes) which have green labels when they are not scanned, and red labels when the packages is scanned., and MailBot located at the center of the warehouse.

- If MailBot is in the same location as a package and holds the portable scanner, it can *scan* the package (it should only scan the package once for delivery).
- MailBot can *turn on* the delivery belt switch if it is in the same tile as a switch.

Task 1.3: Backwards state-space search (10 marks)

In this question you must perform the backwards state space search algorithm as described in the lectures and include it in `report.tex`. The starting state is given in Figure 2 and the goal is to deliver package 011. Represent this goal in your PDDL formalisation of the domain. Spell out the working of the algorithm by specifying at each step which are the relevant actions and what the current goal state is. State why the relevant actions are selected as relevant. State clearly which action the algorithm selected at each step. For the purpose of this task you may assume the search always picks the correct action when there is a choice, so you do not need to simulate dead ends and backtracking. Finally, state clearly why the algorithm terminates when it does, and state the valid plan that the algorithm generated for getting from the initial state to the goal.

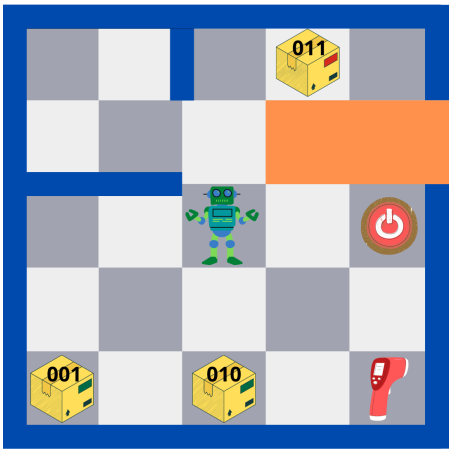

Figure 2: Task 1.3. initial state. Package 011 is already scanned by Mailbot.

Task 2: Implementation (10 marks)

Implement your domain model in PDDL using the example files in `Inf2d-coursework2` as a starting point.[2] To test your plan correctness, use the metric FF planner `ff`.[3] To run the planner execute the following command:[4]

```
./ff -o <domain-file>.pddl -f <problem-file>.pddl
```

Task 2.1: Test Problem #1 (5 marks)

write `problem-1.pddl` in which

- Initial state: as described in Figure 1
- Goal state: deliver package 001

Task 2.2: Test Problem #2 (5 marks)

write `problem-2.pddl` in which

- Initial state: as described in Figure 1
- Goal state: deliver package 001 and package 011.

For both of these subtasks, report the plans found.

Task 3: Experiment (15 marks)

To find the plan, the PDDL planner `ff` uses best-first search in which the world state s is evaluated using a weighted evaluation function:

$$f(s) = w_g g(s) + w_h h(s)$$

where $g(s)$ is the cost so far to reach s, $h(s)$ is the estimated cost to get from s to the goal state, and $w_g, w_h \in \mathbb{Z}$ are weights. By default in `ff`, $w_g = 1$ and $w_h = 5$.

In this task, you are asked to design and perform the experiment to evaluate the effect of w_g and w_h to the planner's performance.

Task 3.1: Design (5 marks)

The current problem files are not challenging enough for the planner. For this subtask, design a harder problem instance `problem-3.pddl`, whilst keeping `domain.pddl` fixed. This problem instance will be used to benchmark the planner's performance. Describe the design of the problem instance in `report.tex`.

Task 3.2: Evaluation (10 marks)

Given the `problem-3.pddl`, design an experiment and evaluate the effect of different values of w_g and w_h to planner's performance. To run the experiments with different values of w_g and w_h use the following command [5]:

[2] You may find https://planning.wiki/ useful.

[3] If you are not working on DICE or if `ff` does not work for you, you can compile the planner from source found here. Simply follow the instructions in the README file in that directory. Please note that we offer **no support** for using the FF planner on any operating system other than DICE.

[4] You can make `ff` executable by running the command `chmod u+x ff`

[5] `-E` flag turns-off hill climbing which by default is used before best-first search as described in the original `ff` paper found here

```
./ff -E -g <w_g> -h <w_h> -o domain.pddl -f problem-3.pddl
```

Report your experiment results in `report.tex`. Your experiment analysis should be brief and have both *quantitative* and *qualitative* elements.

Task 4: Extensions (50 marks)

In this part you will extend the domain to support additional actions and considerations to make the domain model closer to the real world scenario.

Task 4.1: Energy Station (10 marks)

In a real word, MailBot movement between the tiles has a cost, which the current domain model does not taken into account. MailBot energy is stored in its battery, which has a capacity of 15 units. Whenever MailBot moves from one cell to the next without holding anything, it loses 1 unit of energy. Whenever it moves from one cell to the next whilst holding something, it loses 2 units of energy. To help MailBot recharge its battery, the warehouse now has a power station (see Figure 3). Whenever MailBot is in the same tile as the power station, it recharges itself, instantly, to the battery's full capacity of 15 units.

In this task, you are asked to create the extended domain `domain-ext.pddl`, which must include the additional actions and predicates to handle the energy considerations mentioned above (*hint: use numeric-fluents*). To test your implementation, create `problem-ext-1.pddl`, for which the initial state is given in Figure 3. The goal in this planning problem is that package 001 and package 010 are delivered to the teller. In `report.tex`, include the plan that `ff` found.

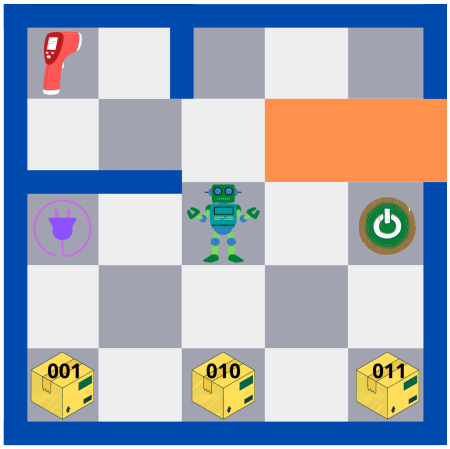

Figure 3: Initial state for Task 4.1. The power station is coloured in purple. The initial energy level of MailBot is 10 units.

Task 4.2: Limited Power (15 marks)

In a real world, even if MailBot has enough energy, it may be mechanically incapable of lifting objects that are too heavy on its own. In other words, MailBot can pick up the scanner and small packages by itself, but for large packages, it needs extra help. Previously, the teller would help MailBot, but recently the post office got a DeliveryBot that has the same capabilities and constraints as MailBot, except it cannot use a portable scanner. MailBot and DeliveryBot can together move large packages to different cells and move it onto the delivery belt.

In this task, you are asked to extend `domain-ext.pddl` to handle the lifting constraints mentioned above. To test your implementation, create `problem-ext-2.pddl` for which the initial state is given in Figure 4. The goal in this planning problem is that package 011 is delivered to the teller. In `report.tex`, include the plan that `ff` found.

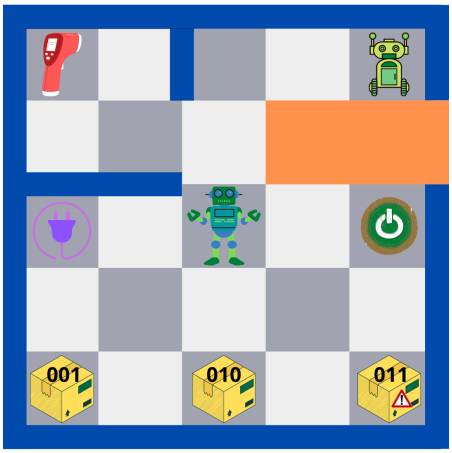

Figure 4: Initial state for Task 4.2. The DeliveryBot is located in the top right corner of the warehouse. The large package 011 is marked with a danger sign.

Task 4.3: Your Extension (25 marks)

In a real world, there are even more considerations that the current domain model is not taking into account. For this last subtask, you need to motivate and design the extension to the domain model to make it more realistic. In particular, for this subtask you need to:

- identify a factor that is a part of the real world scenario but not so far a part of the domain model you have formalised. Describe this factor and how it affects planning (informally, in English). Put your answer in `report.tex`.
- extend your formalisation of `domain-ext-2.pddl` in a new domain file `domain-ext-3.pddl` to include this factor. You can add predicates and actions as necessary, but you have to describe them in `report.tex`.
- identify a planning problem where this factor affects which plans are valid, and which aren't. Implement this planning problem as `problem-ext-3.pddl`. In `report.tex`, identify the plan that `ff` found.

WARNING: you will only get credit for this task if your extension is well motivated, correctly implemented and clearly explained. This task is intended to challenge students who already feel that they have mastered the course material, and want to go further. Do not attempt this question unless you have completed all the previous subtasks, and are sure that you have done a good job on those. Also, do not attempt this subtask if you have personally spent 12 or more hours on the coursework already. The mark awarded for this subtask is likely to be less than 1% of your overall course mark. Unless you really whizzed through the earlier subtasks, please stop now and spend your time and energy revising other course materials or getting more sleep. Both are likely to have a much bigger impact on your final mark.