Informatics 2D Coursework 2: Symbolic Planning

Rimvydas Rubavicius, Jay Park, and Alex Lascarides

Deadline: **12 pm (noon), Thursday 30th March 2023**

Introduction

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

- **Modelling**: creating PDDL problem and domain files for a given specification (35 marks);
- **Experiment**: designing an experiment to evaluate a 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 "Assessment" (on the LHS menu) and go down to"Coursework 2: Symbolic Planning". Alternatively, you can access them here. You will download a file `Inf2d-cw2.zip` which can be unpacked using the following command:

```
unzip Inf2d-cw2.zip
```

This will create a directory `Inf2d-cw2` 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 templates: use `report.docx` or `report.tex`[1] to produce the report.

Submision

For this coursework, we will use Gradescope[2] for both submitting and marking. To submit your work, access Gradescope via an interface on the LEARN website for the course: click "Assessment" (on the LHS menu), go to "Assignment Submission" and click the Gradescope submission link for "Coursework 2: Symbolic planning". You will be transferred to the Gradescope website where you will be asked to submit a programming assignment "Coursework 2: Symbolic planning". Open the submission area for the coursework in which the new window will appear (see Figure 1). In this window drag & drop or click to browse so as to add all the files you will produce in this coursework (see example submission in Figure 2; **do not change the names of these files!**) and press "upload". After doing this, an autograder will run tests to check if the expected files are uploaded. If you do not attempt some tasks (e.g., Task 3.4), do not upload files for them; the autograder will notice this and will pass the tests. Note that you need to compile `report.tex` into `report.pdf` or save `report.docx` as `report.pdf` and submit that. The usage of autograder makes sure that the files uploaded are syntactically valid and that the plans are produced in a reasonable amount of time (the autograder will timeout if running all planning problems instances will take more than 10 minutes).

[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.

[2] https://gradescope.com/

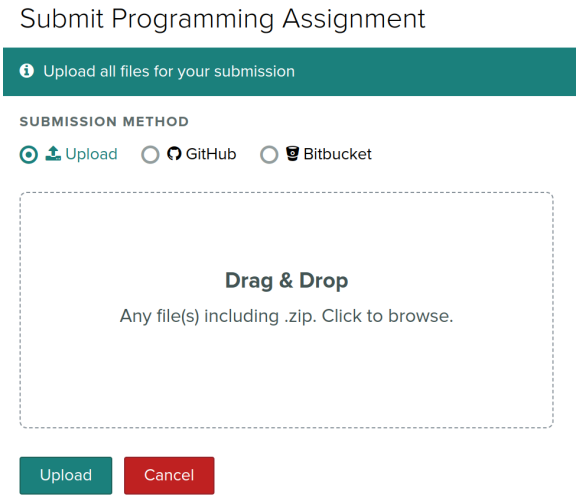

Figure 1: Submission window

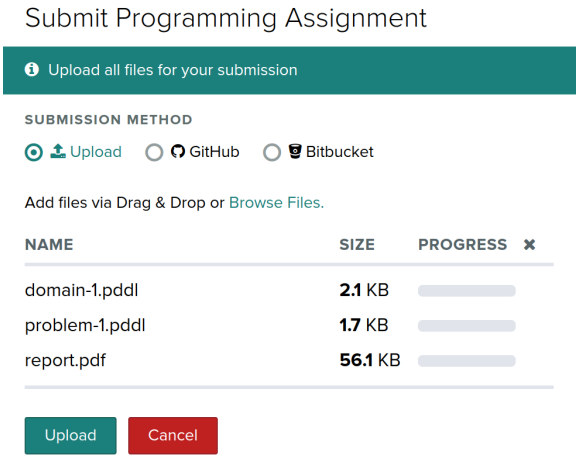

Figure 2: Example files to submit.

If you attempt all tasks in this coursework, the upload should contain the following files:

```
All possible files to submit for coursework 2
├── domain-1.pddl
├── domain-2.pddl
├── domain-3.pddl
├── domain-4.pddl
├── domain-5.pddl
├── problem-1.pddl
├── problem-2.pddl
├── problem-3.pddl
├── problem-4.pddl
├── problem-5.pddl
├── problem-hard.pddl
└── report.pdf
```

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 code 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 (35 marks)

Consider a mine in which MineBot helps to mine ore. To do this, MineBot has to bring the desired ore from the mine corridors to the lift so as to transport the ore to the surface. Ore is often blocked by rocks, in which case MineBot has to use a hammer to release it. The objective of this task is to design and test the PDDL domain and problem files for this scenario.

Task 1.1: Domain File (25 marks)

A PDDL domain file describes the predicates of the domain as well as the actions executable in it. For this subtask, create a domain file called `domain-1.pddl` and declare predicates that are used to describe the following:

- the mine, i.e. the layout of mine corridors and which cells are connected;
- the location of objects like ore, hammer, and the lift;
- whether or not there are rocks blocking an ore;
- whether or not MineBot is holding anything.

Additionally in `domain-1.pddl`, define the actions MineBot can perform. Specify the arguments, preconditions and effects for each of the actions:

- MineBot can *move* between two cells if they are connected. MineBot cannot move diagonally;
- MineBot can *pick up* an item, such as the hammer, so long as it is in the same cell as that item and it is not holding anything else. That is, MineBot can only hold one item at a time;
- If MineBot is in the same cell as an ore that is blocked by rocks and holds the hammer, it can *break* the rock;
- MailBot can *turn on* the lift if it is in (the same location);
- MineBot *mines* the ore by bringing it to the lift and turning it on.

Note that you can define additional predicates and actions to model the described domain.

Task 1.2: Problem File (10 marks)

A PDDL problem file describes the initial and the goal state. Create a `problem-1.pddl` file and in PDDL define the initial state, as depicted in Figure 3, using the predicates from Task 1.1.

Define also the goal in `problem-1.pddl`, which is to mine ores `A`,`B` and `C`.

Task 1.3: Testing (0 marks)

To test the correctness of your `domain-1.pddl` and `problem-1.pddl` files, use the metric FF planner `ff` provided with the coursework description. If you are not working on DICE or if `ff` does not work for you, you can compile the planner from the source found here. Simply follow the instructions in the README file in that directory. Please note that there is **no support** for using `ff` on any operating system other than DICE. To run `ff`, execute the following command:[3]

```
./ff -o domain-1.pddl -f problem-1.pddl
```

[3] You may need to make `ff` executable if it is not by running the command `chmod u+x ff`

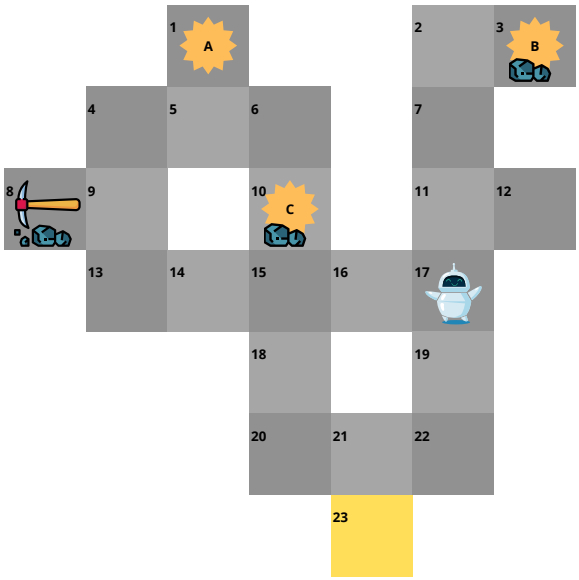

Figure 3: The visualization of the initial state of the mining problem. light/bright grey squares with numbers represent different mine cells. The orange polygon with letters represents ore. If it is the case that there is a rock symbol next to the ore, it means that the ore is blocked by the rock. The hammer is currently located in the cell marked with the number 8 and the MineBot is in the cell marked with the number 17. Finally, the lift is located in an amber cell marked with the number 23.

Task 2: Experiment (15 marks)

To find the plan, `ff` uses a best-first search in which the 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 $w_h = 5$.

The objective of this task is to design and perform the experiment to evaluate the effect of w_g and w_h on the planner's performance.

Task 2.1: Design (5 marks)

The current problem files are not challenging enough for the planner. For this subtask, design a harder problem called `problem-hard.pddl`, whilst keeping `domain-1.pddl` fixed. This problem instance will be used to benchmark the planner's performance. Describe the design of your problem instance in the report, and justify your choice.

Task 2.2: Evaluation (10 marks)

Using `domain-1.pddl` and `problem-hard.pddl`, design an experiment to evaluate the effect of different values of w_g and w_h on the planner's performance. To run the experiments with different values of w_g and w_h use the following command:[4].

```
./ff -E -g <w_g> -h <w_h> -o domain-1.pddl -f problem-hard.pddl
```

Give your experiment results in the report. Include the analysis which should be brief and have both *quantitative* and *qualitative* elements.

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

Task 3: Extensions (50 marks)

The objective of this task is to extend the domain to make it closer to the real world.

Task 3.1: Limited Power (10 marks)

In the real world, MineBot movement between the cells requires energy, which the current domain does not take into account. MineBot energy is stored in its battery, which has a capacity of 40 units. Whenever MineBot 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 3 units of energy. To help MineBot recharge its battery, the mine now has an energy station (see Figure 4). Whenever MineBot is in the same cell as the energy station, it recharges itself, instantly, to the battery's full capacity of 40 units.

In this subtask, create the extended domain `domain-2.pddl` and problem `problem-2.pddl` files containing the initial state as depicted in Figure 4 and the same goal as `problem-1.pddl` (ie, ore A, B and C are mined). Note that `domain-2.pddl` must include the additional actions and predicates for handling the energy considerations mentioned above (*hint: use numeric-fluents*). Don't forget to test your domain and problem files using the `ff` planner.

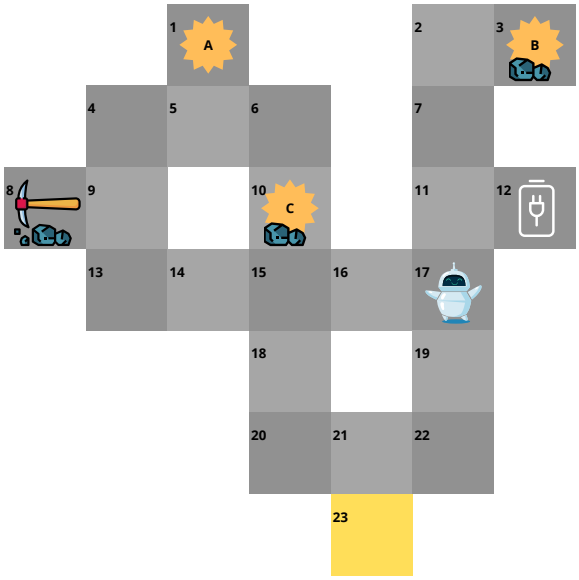

Figure 4: The mining problem scenario for Task 3.1. Now the mine has an energy station located at a cell marked with the number 12. The goal state remains the same as in `problem-1.pddl`

Task 3.2: Fire Emergency (5 marks)

In the real world, there might be a fire in the mine due to unfortunate natural gas leakage. MineBot cannot enter cells with fire. To deal with the fire, MineBot can use a fire extinguisher: this involves picking it up from wherever it is, taking it to the cell that is connected to the cell with the fire, and using the extinguisher to extinguish the fire.

Create `domain-3.pddl` and `problem-3.pddl`, which is the extended version of the domain specification in the previous question, taking into account the contingencies described above. Figure 5 depicts the initial state of the problem, and the goal state remains the same as in `problem-1.pddl`. Don't forget to test your domain and problem files using the `ff` planner.

Task 3.3: Helping Hand (10 marks)

In the real world, even if MineBot has enough energy, it may be mechanically incapable of lifting larger ores that are too heavy on their own. In other words, MineBot can lift small ores by itself,

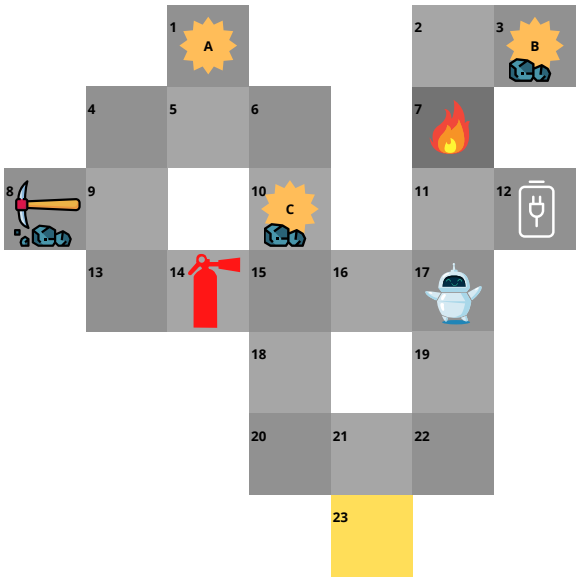

Figure 5: The mining problem scenario for Task 3.2. Now the mine has a fire located in a cell marked with the number 7 and a fire extinguisher located in the cell marked with the number 14. The goal state remains the same as in `problem-1.pddl`

but for large ores, it needs extra help. To do this, the mine now has an additional MineBot that has the same capabilities as the original one. Together, these two MineBots can pick up large ores and transport them to the lift. They can both lift a (large) ore so long as they are both in the same cell as that ore, there is no rock blocking that ore, and both MineBots aren't already holding anything. When transporting the large ore, both robots are moving in tandem.

In this task, you are asked to create `domain-4.pddl` and `problem-4.pddl` to handle the constraints mentioned above. The `problem-4.pddl` should represent the initial state given in Figure 6 and the goal of mining ores `A`, `B`, `C` and `D`. Don't forget to test your domain and problem files using the `ff` planner.

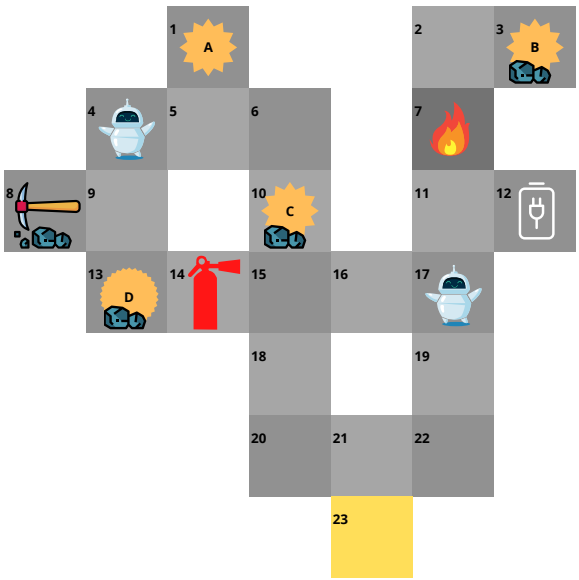

Figure 6: The mining problem scenario for Task 3.3. Now the mine has an additional MineBot in the cell marked with the number 4 as well as a large ore in the cell marked with the number 13 (represented with an orange polygon with more sides than a small ore)

Task 3.4: Your Extension (25 marks)

In the real world, there are even more considerations that the above domain specifications don't take into account. For this last subtask, you need to motivate and design extended domain and problem files that make them more realistic. In particular, for this subtask you need to:

- identify a factor that is a part of the real-world scenario but has not so far been a part of the domain specification you have formalised. Describe this factor and how it affects planning (informally, in English). Describe your extension in the report, and justify your choice.
- create `domain-5.pddl` and implement this factor. You can add predicates and actions as necessary, but you have to describe them in the report;
- create `problem-5.pddl` where this factor affects which plans are valid, and which aren't.
- test your domain and problem files using the `ff` planner.

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 maximum mark awarded for this subtask is only 3% 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 for the course.