Foundations of Artificial Intelligence

T. Keller, F. Pommerening
S. Sievers
Spring Term 2023

University of Basel
Computer Science

Exercise Sheet 9

Due: May 7, 2023

Important: for submission, consult the rules at the end of the exercise. Non-adherence to the rules will lead to your submission not being corrected.

Note: due to the holiday on May 1, this exercise sheet covers only one lecture day and therefore contains exercises worth only 5 marks.

Exercise 9.1 (1.5+0.5+1 marks)

In this exercise we consider a PDDL representation of the Sokoban problem. As a reminder, in the Sokoban problem, there is an agent who can move in a grid of which a subset of cells is blocked by walls. In the general form of the problem (see, e.g., `https://en.wikipedia.org/wiki/Sokoban`), there is a set of boxes, located at their initial cells in the grid, that must be pushed to goal cells in the grid. The agent can freely move to any cell not occupied by a wall or a box. The agent can only *push* boxes: if they stand next to a box and the cell behind the box from the point of view of the agent is unoccupied, they can push the box to that free cell, which also moves the agent to the cell the box was previously. The objective is to get the boxes to the goals with as few pushes as possible. The number of moves of the agent do not matter.

The provided file `pddl.zip` contains an incomplete PDDL description of the Sokoban domain and two problem instances. In particular, the domain file `sokoban-domain.pddl` describes the variables (called predicates) and actions of Sokoban, both independent of the concrete problem instances, while the concrete problem instances are described in `sokoban-problem-01.pddl` (incomplete) and `sokoban-problem-02.pddl` (complete example problem). The problem instances contain a graphical representation at the top. Empty squares represent locations that can be entered, a '#' sign represents a wall, the '@' sign represents the initial location of the agent, a '$' sign represents an initial position of a box, and a '.' sign represents a goal location. Cell 1/1 is in the upper left corner, where x/y denotes the horizontal and vertical coordinates.

It might also be helpful to consider the example PDDL representation of blocks world, which can be found on the course website under "Supplementary Material" chapter 34.

(a) Fill the marked gaps in the file `sokoban-domain.pddl`. The two PDDL actions for pushing a box must be such that one can be used only to push a box to a goal location, and the other one only to push a box to a non-goal location.

(b) Fill the marked gaps in the file `sokoban-problem-01.pddl`.

(c) Obtain the domain-independent planning system *Fast Downward*. You can download either the sources or one of various container files from

`http://www.fast-downward.org/Releases/22.12`

The page also provides links to instructions for compilation and container usage. If you encounter technical problems, please let us know *sufficiently ahead of the due date*.

Use Fast Downward with a configuration that performs A* with the delete relaxation heuristic $h^{\max}$ to test your solution of parts (a) and (b). To do so, invoke the planner with

`./fast-downward.py DOMAIN PROBLEM --search "astar(hmax())"`,

where DOMAIN and PROBLEM refer to paths to the PDDL domain and problem files, respectively.

The plan found by *Fast Downward* should consist of 35 actions and incur a total cost of 9.

Apply greedy best-first search with the delete relaxation heuristic FF to solve both problem instances by executing

```
./fast-downward.py DOMAIN PROBLEM --search "eager_greedy([ff()])"
```

and report the runtime, the number of expanded states and the cost of the plan that was found.

Exercise 9.2 (2 marks)

In the *Gripper* planning domain, there is a robot with a gripper and two balls. The robot and the balls are initially located in the left room, and the goal is to have the balls in the right room. To achieve this, the robot can move between the two rooms and it can carry a ball using its gripper. The gripper can only carry a single ball at a time. It can pick up a ball in a room if both the robot and the ball are in the same room. It can drop the ball it carries in any of the two rooms. All actions are of equal cost 1.
Formalize the Gripper domain as a STRIPS planning task.

Submission rules:

- Exercise sheets must be submitted in groups of two students. Please submit a single copy of the exercises per group (only one member of the group does the submission).
- Create a single PDF file (ending .pdf) for all non-programming exercises. Use a file name that does not contain any spaces or special characters other than the underscore "_". If you want to submit handwritten solutions, include their scans in the single PDF. Make sure it is in a reasonable resolution so that it is readable, but ensure at the same time that the PDF size is not astronomically large. Put the names of all group members on top of the first page. Either use page numbers on all pages or put your names on each page. Make sure your PDF has size A4 (fits the page size if printed on A4).
- For programming exercises, only create those code textfiles required by the exercise. Put your names in a comment on top of each file. Make sure your code compiles and test it. Code that does not compile or which we cannot successfully execute will not be graded.
- For the submission: if the exercise sheet does not include programming exercises, simply upload the single PDF. If the exercise sheet includes programming exercises, upload a ZIP file (ending .zip, .tar.gz or .tgz; *not* .rar or anything else) containing the single PDF and the code textfile(s) and nothing else. Do not use directories within the ZIP, i.e., zip the files directly.
- Do not upload several versions to ADAM, i.e., if you need to resubmit, use the same file name again so that the previous submission is overwritten.