Planning and Optimization

M. Helmert, G. Röger
C. Büchner, R. Christen, S. Dold

University of Basel
Fall Semester 2022

Exercise Sheet 4

Due: October 23, 2023

Important: For submission, consult the rules at the end of the exercise. Non-adherence to these rules might lead to a penalty in the form of a deduction of marks or, in the worst case, that your submission will not be corrected at all.

Exercise 4.1 (0.5+0.5+2+1 marks)

Consider the propositional planning task $\Pi = \langle V, I, O, \gamma \rangle$ with

- $V = \{x, y, z\}$;
- $I = \{x \mapsto \mathbf{T}, y \mapsto \mathbf{T}, z \mapsto \mathbf{F}\}$;
- $O = \{o_1, o_2, o_3\}$ where
  - $o_1 = \langle x, \neg x \wedge (y \rhd z), 2 \rangle$,
  - $o_2 = \langle z, x \wedge \neg y \wedge \neg z, 1 \rangle$, and
  - $o_3 = \langle \top, (\neg y \rhd y) \wedge (y \rhd \neg y), 2 \rangle$; and
- $\gamma = \neg(x \vee z)$.

In this exercise, your task is to define a sequential SAT-encoding for Π.

(a) Provide the clauses that encode the initial state I.

(b) Provide the clauses that encode the goal γ at horizon T.

(c) Provide all clauses that encode the transitions induced by o_1 for some time step i. Simplify the clauses and omit those that simplify to $\top$. (You do not need to provide intermediate results for the simplification.) Annotate each remaining clause as precondition clause, positive or negative effect clause, or positive or negative frame clause.

(d) Assuming the precondition, effect and frame clauses (positive and negative) induced by o_2 and o_3 are given, which clauses are yet missing to obtain a complete SAT-encoding? Provide them explicitly, again parametrized for some time step i.

Exercise 4.2 (4.5+1.5 marks)

Update the course repository (`/vagrant/planopt-hs23` in your course VM) with `$ git pull`. Navigate to the new directory `sheet04` which contains the files required for this exercise.

Pyperplan (`https://github.com/aibasel/pyperplan`) is a lightweight STRIPS planner written in Python. While it does not come with as strong performance as Fast Downward, it is very easy to extend and modify.

(a) In the file `pyperplan/src/search/bdd_bfs.py` you can find an incomplete implementation of a BDD-based breadth-first search. Complete it by using the utility methods in the file `pyperplan/src/search/bdd.py`. Do not modify anything else than the file `pyperplan/src/search/bdd_bfs.py` (and do not modify the constructor of `BDDSearch` yet, this is for part (b)). Test your search on the tasks in the directory `blocks` and make sure that it can find valid plans.

You can run the code with the command

```
$ ./pyperplan/src/pyperplan.py -s bdd blocks/domain.pddl blocks/p1.pddl
```

(b) The constructor of `BDDSearch` contains a commented out alternative variable order for the variables within the BDD. Change the order by commenting out the old order and including the new order instead. Print the number of total BDD nodes after adding each operator and after each expansion step (use the provided method `print_bdd_nodes()`). Compare the two variable orders on a small task and discuss the results.

Submission rules:

- Exercise sheets must be submitted in groups of 2–3 students. Create a team on ADAM including all members of your group and submit a single copy of the exercises per group.
- 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 text file(s) 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 text file(s) and nothing else. Do not use directories within the ZIP, i.e., zip the files directly. After creating your zip file and before submitting it, open the file and verify that it complies with these requirements.
- 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.