Planning and Optimization

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

University of Basel
Fall Semester 2022

Exercise Sheet 1

Due: October 02, 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 1.1 (2+1 marks)

Play around with the Fast Downward planner. The files required for this exercise are in the directory `sheet01` of the course repository (`/vagrant/planopt-hs23` in your course VM). Update your clone of the repository with `$ git pull` to see the files.
For this exercise, set a time limit of 1 minute and a memory limit of 2 GB. Using Linux, such limits can be set with `$ ulimit -t 60` and `$ ulimit -v 2000000`, respectively.
The directory `sheet01/tile` contains four variants of the 15-Puzzle:

- original formulation (`puzzle.pddl`, `puzzle01.pddl`)
- variant with weighted tiles (`weight.pddl`, `weight01.pddl`)
- variant with glued tiles (`glued.pddl`, `glued01.pddl`)
- variant with cheating action (`cheat.pddl`, `cheat01.pddl`)

To run Fast Downward, use the script `fast-downward.py` with the corresponding domain and problem files, specifying the search algorithm and the heuristic. Example for the original formulation, greedy best first search and the FF heuristic:

```
$ ./fast-downward/fast-downward.py tile/puzzle.pddl tile/puzzle01.pddl \
    --heuristic "h=ff()" --search "eager_greedy([h])"
```

(a) Run Fast Downward on the original formulation of the 15-puzzle, using greedy best first search and different heuristics:

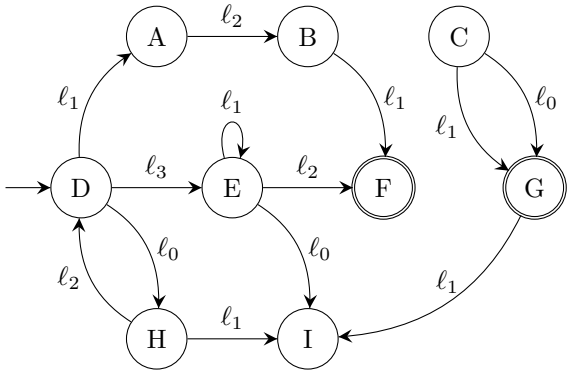
- additive heuristic: `add()`
- blind heuristic: `blind()`
- causal graph heuristic: `cg()`
- FF heuristic: `ff()`

Summarize your results with respect to time (`Total time`), number of expanded and generated states (`Expanded` and `Generated`), and solution quality (`Plan Cost`) in a table.

(b) Run Fast Downward again to solve the other variants of the domain. Discuss in 2–3 sentences how the results differ from those for the original formulation.

Hint: You do not need to report explicit numbers as in part (a).

Exercise 1.2 (3+2 marks)
Consider the following transition system:



$$c(\ell_0) = 0, c(\ell_1) = 1, c(\ell_2) = 2, c(\ell_3) = 3$$

(a) Formalize the depicted transition system as a 6-tuple $\mathcal{T} = \langle S, L, c, T, s_0, S_\star \rangle$.

Hint: When writing down something formally, always ask yourself what kind of mathematical object you are using where, then use correct notation. For instance, a tuple such as $\mathcal{T}$ is formalized using angle $\langle \cdots \rangle$ or round $(\cdots)$ brackets; a set such as S is using braces $\{\cdots\}$; a function such as $c\colon L \to \mathbb{R}_0^+$ describes how objects in L are mapped to an object in $\mathbb{R}_0^+$; and a single object such as s_0 is using no brackets at all.

(b) Answer the following questions about the depicted transition system.

i) Is $\mathcal{T}$ deterministic? Justify your answer.

ii) Which states of $\mathcal{T}$ are *unreachable* from state H?

iii) What are the *predecessors* of state E in $\mathcal{T}$?

iv) What is the cheapest *solution* of $\mathcal{T}$?

Exercise 1.3 (2 marks)
Consider the following formula over propositions $\{X, Y, Z\}$:

$$\varphi = ((X \wedge \neg Y) \vee (\neg Z \wedge \neg(X \vee Y)))$$

(a) Transform φ into an equivalent formula that is in conjunctive normal form (CNF). Provide your transformation step by step so that it is easy to verify equivalence from one step to the next.

Hint: If you need a reminder of equivalence rules, you may consider slides 17–19 of Chapter E2 from last year's lecture "Discrete Mathematics in Computer Science": `https://ai.dmi.unibas.ch/_files/teaching/hs22/dmics/slides/dmics-e02-handout.pdf`,

(b) Provide two interpretations $\mathcal{I}$ and $\mathcal{J}$ of propositions $\{X, Y, Z\}$ such that $\mathcal{I} \models \varphi$ and $\mathcal{J} \not\models \varphi$.

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.