Introduction to Artificial Intelligence

Dr. Thomas Keller
C. Büchner, C. Grundke, A. Kauffmann

University of Zürich
Spring Semester 2022

Exercise Sheet 3

Due: March 16, 2022

Points total: 20 marks

Exercise 3.1 – Heuristics

4 marks

Consider again the *Wolf, Goat, and Cabbage* problem from exercise 2.1.

(a) Define a heuristic for the *Wolf, Goat, and Cabbage* problem. The heuristic should be non-trivial, i.e., we do neither accept constant functions (e.g., $h(s) = 0$ for all s) nor the blind heuristic that assigns a constant c to every goal state and a larger constant $c' > c$ to every non-goal state.

Hint: You may use your problem formalization from last exercise sheet but you can also solve this task without a problem formalization. (2 marks)

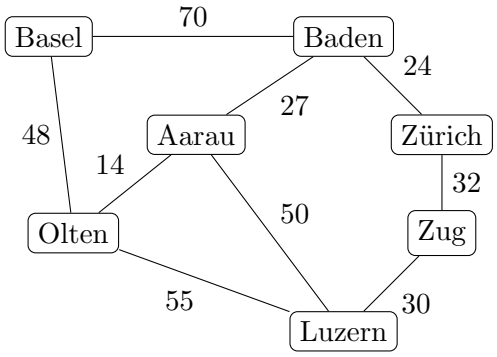
(b) Is your heuristic admissible? Justify your answer. (1 mark)

(c) Is your heuristic consistent? Justify your answer. (1 mark)

Exercise 3.2 – Informed Search Algorithms: Practice

7 marks

Consider the following map:



Let the air-line distance between Zug and the other cities be given by the following table:

city	**distance**
Aarau	44
Baden	38
Basel	83
Luzern	21
Olten	51
Zug	0
Zürich	23

Consider the heuristic that maps each state to its air-line distance to Zug.

(a) Provide the search tree of A* (without reopening) when queried for the shortest path from Basel to Zug. Indicate the order in which nodes are expanded and annotate each node with its f-, g-, and h-values. (3 marks)

(b) Provide the search tree of greedy best-first search (without reopening) when queried for a path from Basel to Zug. Indicate the order in which nodes are expanded. (3 marks)

(c) Compare the results in (a) and (b) and discuss your observations in 1–2 sentences. Are the results as expected? Justify your anser (1 mark)

Exercise 3.3 – Informed Search Algorithms: Programming **9 marks**

Consider the search framework provided in `sheet03-programming.zip`. Note that it is slightly different from the one provided in exercise 2.4, but breadth-first and depth-first search as well as uniform-cost search should still work here.

We introduce the *pancake problem* in `pancake_problem.py` which describes a family of search problems as depicted in Figure 1. Given a stack of n pancakes of different and unique sizes, the goal is to stack them from largest on the bottom to smallest on top. The available actions are $\{\texttt{flip-x} \mid \texttt{x} \in \{1, \ldots, n\}\}$ where `flip-x` takes the top x pancakes and puts them back on top in reversed order.

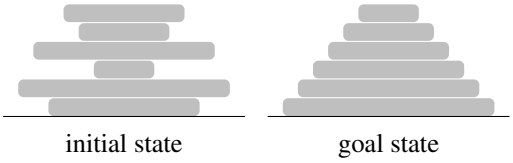


Figure 1: Example instance for the pancake problem.

Our implementation provides a heuristic for this problem which is in the literature called the *gap heuristic*[1]: there is a "gap" between two neighbouring pancakes if the absolute difference in size is larger than one. Counting all gaps in a state is an admissible estimate of the goal distance because every gap requires at least one action to fix that gap and each action fixes at most one gap.

(a) Implement *weighted* $A^\star$ search in the provided framework. (5 marks)

(b) Analyze the search behavior of different search algorithms considered in this lecture. Execute the provided `experiment.py` script to compare *runtime*, *plan cost*, *node expansions*, and *node generations* in the specified pancake problems.

Hint: The script executes the uninformed searches only on the simpler set of problems because uninformed search otherwise quickly runs out of memory.

Summarize the results in a useful way (e.g., in tables or plots) in your PDF submission and discuss your observations. Are the values as expected? Justify your answers. (4 marks)

Submission rules:

- Exercise sheets must be submitted in groups of three 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. Make sure your PDF has size A4 (fits the page size if printed on A4). Submit your single PDF file to the corresponding exercise assignment in MOODLE.

[1] "Landmark Heuristics for the Pancake Problem", M. Helmert, in Proceedings of the Third Annual Symposium on Combinatorial Search (SoCS 2010).

- For programming exercises, only create those code text files 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. Create a ZIP file (ending `.zip`, `.tar.gz`, or `.tgz`; *not* `.rar` or anything else) containing the code text file(s) (ending `.py`) and nothing else. Do not use directories within the ZIP, i.e., zip the files directly.
- Do not upload several versions to MOODLE, i.e., if you need to resubmit, use the same file name again so that the previous submission is overwritten.