

Foundations of Artificial Intelligence

M. Helmert
S. Eriksson
Spring Term 2022

University of Basel
Computer Science

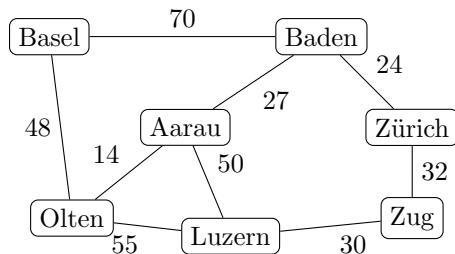
Exercise Sheet 5

Due: April 3, 2022

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.

Exercise 5.1 (2 marks)

Consider the following map and a heuristic mapping each state to its air-line distance to Zug:



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

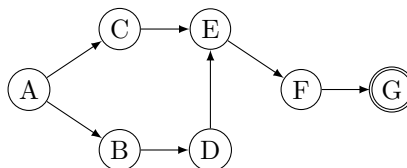
Execute the first two iterations of IDA* when starting from Aarau and trying to reach Zug, expanding successors in alphabetical order. For each iteration, specify in a nested manner with which arguments the recursive function was called and what was returned.

Example: the first iteration when starting from Basel can be specified in the following way.

```
f_bounded_search(Basel, 0, 83)
  f_bounded_search(Baden, 70, 83)
    return <108, none>
  f_bounded_search(Olten, 48, 83)
    return <99, none>
  return <99, none>
```

Exercise 5.2 (2+1 marks)

- (a) Consider the problem where the goal is to find the shortest path from A to G in the following directed graph under unit cost, i.e., where each transition incurs a cost of 1.



Show that A* *without* reopening used with an admissible but *inconsistent* heuristic can find suboptimal solutions: first provide a heuristic with the required properties for the depicted problem, then use A* without reopening to solve the problem and show that the found solution is not optimal. Draw the performed search as a graph where each search node is annotated with the expansion order, g -, h -, and f -values as follows:

X^i
 $g/h/f$ (X^i means node with state X was the i th node to be expanded)

- (b) Which part of the proof of optimality of A* without reopening (chapter 19) becomes invalid if using an inconsistent heuristic? Justify your answer.

Exercise 5.3 (4+1 marks)

The task in this exercise is to write a software program. We expect you to implement your code on your own, without using existing code (such as examples you find online) except for what is provided by us. If you encounter technical problems or have difficulties understanding the task, please let us – the tutor or assistant – know *sufficiently ahead of the due date*.

- (a) The file `BestFirstSearch.java` in the provided archive `best-first-search.zip` is a skeleton for the implementation of best-first search. It expects two parameters `gMultiplier` and `hMultiplier` which are passed through the command line. Implement the method `run` such that it performs a best-first search *without reopening* with $f(n) = gMultiplier \cdot g(n) + hMultiplier \cdot h(n.state)$. Make sure that the value of the member variable `expandedNodes` is updated correctly. You can access the heuristic value of a state through a new method in the `StateSpace` interface called `public int h(State s)`.

Notes: You are not allowed to create new files or modify existing files, but you can (and should) create new functions and / or nested classes within `BestFirstSearch`. For example, a nested class `SearchNode` can be very useful.

Only submit your version of `BestFirstSearch.java`

- (b) We altered `LightsOutStateSpace` to implement the heuristic described in Exercise 4.3. Test your implementation of `BestFirstSearch` on the example problem instances provided in the `instances` directory with the following parameters:

- `gMultiplier = 1, hMultiplier = 1` (A*)
- `gMultiplier = 1, hMultiplier = 10` (WA* with weight 10)
- `gMultiplier = 0, hMultiplier = 1` (GBFS)

Set a time limit of 10 minutes and a memory limit of 2 GB for each run. On Linux, you can set a time limit of 10 minutes with the command `ulimit -t 600`. Running your implementation on the first example instance with

```
java -Xmx2048M BestFirstSearch lights-out instances/lights-out-prob01.txt
gMultiplier hMultiplier
```

sets the memory limit to 2GB. If the RAM of your computer is 2GB or less, set the memory limit to the amount of available RAM minus 256MB instead. In any case, describe the used memory limit in your solution.

For each parameter configuration, report runtime, number of node expansions, solution length and solution cost for all instances that can be solved within the given time and memory limits. For all other instances, report if the time or the memory limit was hit.

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.