Foundations of Artificial Intelligence

M. Helmert
S. Sievers
Spring Term 2023

University of Basel
Computer Science

Exercise Sheet 5

Due: April 02, 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.

Exercise 5.1 (1+3+1+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 tutors or assistant – know *sufficiently ahead of the due date.*

(a) We extended the black box interface for state spaces by a method `h(s)` that returns the heuristic value of a given state s. Implement the heuristic based on the Manhattan distance described in Exercise 4.3. The file `SokobanStateSpace.java` in the provided archive `best-first-search.zip` already contains a stub for the method.

Notes: Do not create new files or modify existing files except `SokobanStateSpace.java`. You can add new methods within `SokobanStateSpace.java`.

Only submit your version of `SokobanStateSpace.java`.

(b) 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` $\cdot\, h(n.\text{state})$. Make sure that the value of the member variable `expandedNodes` is updated correctly. Also make sure to use suitable data structures such that your implementation is reasonably efficient. A possible implementation of the open list (yet certainly not the only one) is to use a `java.util.PriorityQueue` and one possibility for the closed list is to use a `java.util.HashSet`.

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

Only submit your version of `BestFirstSearch.java`.

(c) 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 sokoban instances/sokoban_instance_00.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.

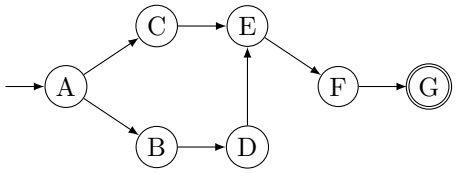
For each algorithm, report runtime, number of node expansions, solution length and solution cost for all instances that can be solved within the given time and memory limits. All of these are printed by the program if a solution was found. For all other instances, report if the time or the memory limit was violated. To do so, use a separate table of the following form for each algorithm and round values to 3 digits after the decimal:

instance	result	runtime	expansions	solution length	solution cost
0	(un)solved?	...			

(d) Discuss the results. In particular, compare the metrics of the three best-first search algorithms and also compare them to those of uniform cost search of Exercise 3.3 b). Also consider the theoretical properties of all four algorithms.

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 and explain why your heuristic has these properties. Then use A* without reopening to solve the problem and show that the found solution is not optimal. To do so, draw the search tree where each search node is annotated with a number denoting the expansion order or no number if it is not expanded, and addtionally the g-, h-, and f-values. Draw each search node as follows:

$$\begin{array}{c} X^i \\ g/h/f \end{array}$$

where the superscript i in X^i denotes that X was expanded in the ith step; do not use any superscript if X was not expanded; and $g/h/f$ denote the obvious values.

(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 (1 mark)

Would A* without reopening behave identically to A* with reopening with a consistent but inadmissible heuristic? Briefly justify your ansewr.

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.