Foundations of Artificial Intelligence

M. Helmert, T. Keller
S. Eriksson
Spring Term 2020

University of Basel
Computer Science

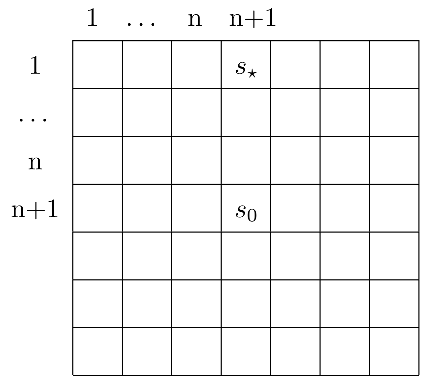
Exercise Sheet 4

Due: March 25, 2020

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 4.1 (1+1+1 marks)

Consider a search problem on a square grid of size $2n+1$ (for $n \in \mathbb{N}$). An agent is initially located in state s_0 in the grid cell with coordinates $(n+1, n+1)$ and has the goal to reach state $s_\star$ located in the grid cell with coordinates $(n+1, 1)$. The agent has the possibilities to move north, east, south, and west in the grid if there is a grid cell in the corresponding direction (otherwise, the corresponding action is not applicable). We assume that the agent uses breadth first search to compute a plan.



(a) How many search nodes are at least inserted into the open list until the agent finds a plan if duplicate detection is not used? Give an answer as a function of n and justify your answer.

(b) How does that answer differ if duplicate detection is used?

(c) Compare the number of search nodes that is inserted into the open list in the last search layer that is created completely for grids of size $n = 10$ and $n = 20$ and discuss the results.

Exercise 4.2 (1+1+1 marks)

Which of the search algorithms shown in slide 35 of the printout version of chapter 12 would you use for the following problems? For breadth-first and uniform cost also consider whether to use the tree or graph variant. Justify your answer in one to two sentences.

(a) Brute-forcing a password of unknown length, where each action (with cost 1) consists of adding a character to the string and the goal is reached as soon as we have the correct string (no enter key needed). We assume a maximal password length of 100.

(b) Reaching a location in a 2D grid (like in Exercise 4.1), where the agent can move in the 4 cardinal directions with a cost of 1.

(c) Solving a Sudoku puzzle. We assume that empty cells are filled in a specific order, i.e. from top left to bottom right, and that each assignment action has cost 1.

Exercise 4.3 (4+2 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.*

The objective of Exercise 2.4 from two weeks ago was to implement the state space of an elevators dispatch problem. This week, we extend the Elevators state space to additionally support action costs, where the cost of moving elevator i (with $0 \leq i <$ #elevators) up or down one floor is $10 + i$, and embarking and disembarking incurs a cost of 1.

On the website, you can download an implementation of the state space of Elevators.

(a) Implement *uniform cost search* to solve Elevators problems with action costs. Only create a single new file called `UniformCostSearch.java`. The new class `UniformCostSearch` must derive from `SearchAlgorithmBase`. 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`.

(b) Test your implementation on the example problem instances you can find on the website. 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 UniformCostSearch elevators elevator_inst_1
```

sets the memory limit to 2 GB. If the RAM of your computer is 2GB or less, set the memory limit to the amount of available RAM minus 256 MB instead. You are also free to use higher memory limits. In any case, describe in your solution how much RAM was used.

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 violated.

Submission rules:

- Create a single PDF file (ending .pdf) for all non-programming exercises. If you want to submit handwritten parts, include their scans in the single PDF. Put the names of all group members on top of the first page. Use page numbers or put your names on each page. Make sure your PDF prints on A4 (fits the page size).
- For programming exercises, create only those Java textfiles (ending .java) required by the exercise. Put your names in a comment on top of each file. Make sure your code compiles and test it!
- For the submission, you can either upload the single PDF or prepare a ZIP file (ending .zip, .tar.gz or .tgz; not .rar or anything else) containing the single PDF and the Java textfile(s) and nothing else. Please do not use subdirectories in the ZIP.
- Only upload one submission per group. Do not upload several versions, i.e., if you need to resubmit, use the same file name again so that the previous submission is overwritten.