Foundations of Artificial Intelligence

M. Helmert
S. Eriksson
Spring Term 2022

University of Basel
Computer Science

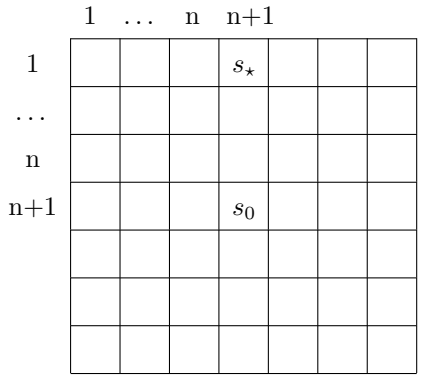
Exercise Sheet 3

Due: March 20, 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 3.1 (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 solution.

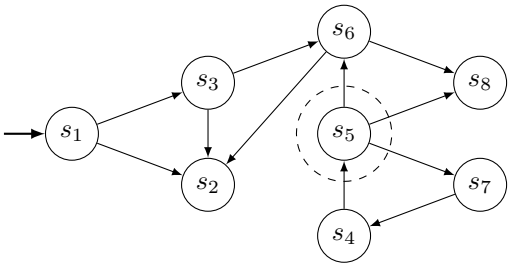


(a) How many search nodes are at least inserted into the open list until the agent finds a solution using tree search (BFS-Tree)? Give an answer as a function of n and justify your answer.

(b) How does that answer differ when using graph search (BFS-Graph)?

(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 with $n = 10$ and $n = 20$ for tree search and graph search and discuss the results.

Exercise 3.2 (1+1 marks)



For the state space depicted above, are the following statements correct? Justify your answer.

Note: The initial state is depicted with an incoming arrow, and the goal state(s) with a dashed circle (as in chapter 5 slide 12). We omit labels in the state space for clarity since they are irrelevant for the question.

(a) BFS-Graph will expand the node containing s_2 before the node containing s_6 independent of the order in which successors are generated.

(b) BFS-Tree will terminate.

Exercise 3.3 (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) Implement *uniform cost search*. Only create a single new file called `UniformCostSearch.java`. The new class `UniformCostSearch` must derive from `SearchAlgorithmBase`. Make sure that the value of the member variable `expandedNodes` is updated correctly. 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 in the folder `instances`. 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 lights-out instances/lights-out_prob_01
```

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. 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:

- 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.