Foundations of Artificial Intelligence

M. Helmert
S. Sievers
Spring Term 2023

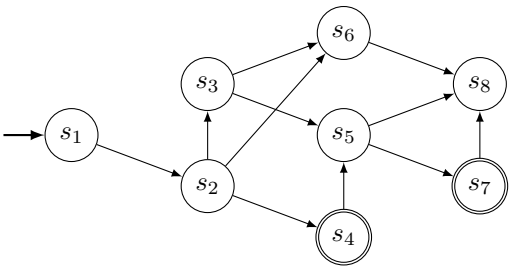
University of Basel
Computer Science

Exercise Sheet 3

Due: March 19, 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 3.1 (1+1+1 marks)



Consider the search space depicted above. (Note: We omit labels in the state space for clarity since they are irrelevant for the question.)
Are the following statements correct? Justify your answer.

(a) Tree search expands a finite amount of nodes, no matter in which order nodes are expanded.

(b) If the algorithm continues exploring the search space after finding a goal (meaning the goal test is omitted and nodes are expanded until the open list is exhausted), tree and graph search expand the same number of nodes.

(c) With any expansion order, graph search finds a goal with less than 10 expansions.

Exercise 3.2 (1+1+1 marks)

Show that nontrivial state spaces with the following properties exist by providing a corresponding state space and explaining why it satisfies the required property. Your state space must have at least 5 states reachable from the initial state (including the initial state itself) and all solutions must have at least length 2.
Note: You can omit action names and instead directly use action costs as labels in the graph.

(a) Tree search and graph search expand the same amount of nodes (when using the same expansion strategy).

(b) Breadth-first search expands $|S| - 1$ nodes.

(c) Breadth-first search finds a suboptimal solution.

Exercise 3.3 (3+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.*
We consider the Sokoban problem again, however, adapted such that actions have more diverse costs: cells are associated with a positive integer denoting the cost entering that cell. That is, if

the agent moves to a cell, the cost of that action equals the cost of the target cell. If the agent pushes a box, the cost of that action equals the cost of the cell where the agent stops (and the box was initially). We provide an updated implementation of the Sokoban state space in the provided file `uniform-cost-search.zip`.

(a) Using the provided file `uniform-cost-search.zip`, 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`.

Only submit the file UniformCostSearch.java

(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 sokoban instances/sokoban_instance_00.txt
```

sets the memory limit to 2GB.

Note that the JVM per default uses multi threading. This means that the 10 minutes time limit can be reached very quickly, within less than two minutes of wall clock time if your computer has many cores.

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. Use a table of the following form for this task and round values to at 3 digits after the decimal:

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

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.