Foundations of Artificial Intelligence

M. Helmert
T. Keller
Spring Term 2018

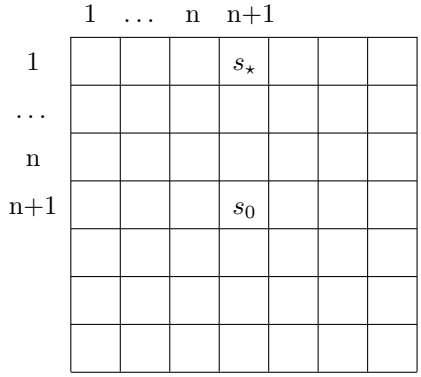
University of Basel
Computer Science

Exercise Sheet 3

Due: March 28, 2018

Exercise 3.1 (2+2+2 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 3.2 (1+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.* Please also read the *hints* below.

The objective of last week's Exercise 2.2 was to implement the state space of a Sokoban variant which differs from the original formulation in that each box has a dedicated goal position. This week, we extend the SimpleSokoban state space to additionally support action costs. We encode the action cost in the input files by providing the grid of height h and width w as h lines of w non-negative integers, where a 0 stands for a wall (as in the representation used in Exercise 2.2), while any other integer gives the cost of *entering* that grid cell, i.e. if grid cell c is assigned a cost of 5, each move action *to* c and each push action that pushes a box *from* c to one of its neighbors incurs a cost of 5.

On the website, you can download an implementation of the state space of SimpleSokoban (since this is a valid solution to Exercise 2.2, the files will not be available before Thursday, March 22, when the deadline for last week's exercise sheet has passed).

(a) Extend the SimpleSokoban state space to allow action costs as described above and adapt the `buildFromCmdline` function such that files in the described format can be parsed.

(b) Implement *uniform cost search* to solve SimpleSokoban problems with action costs in a file `UniformCostSearch.java`. To do so, you may inherit from the provided class in `SearchAlgorithmBase.java`, which provides the means to measure search statistics among other things.

Hint: For your implementation of uniform cost search, a posible implementation of the open list (yet certainly not the only one) is to use `java.util.PriorityQueue` and one possibility for the closed list is to use a `java.util.HashSet`. Depending on your implementation, it is furthermore possible that you have to implement comparison and/or hashing methods (`equals` and `hashCode`) for all classes that are used to describe a state.

(c) Test you 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 sokoban sokoban_inst_5_6_2
```

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 and number of expanded nodes 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.

Hint: Please *test* your solution. Your code must be compilable and we must be able to run it!

Important: The exercise sheets can be submitted in groups of two students. Please provide both student names on the submission. Please create a PDF for exercise 3.1 and a directory containing the Java files for exercise 3.2. Afterwards, please create a zip file containing the PDF and the directory and submit it.