Foundations of Artificial Intelligence

M. Helmert, G. Röger
J. Seipp, S. Sievers
Spring Term 2017

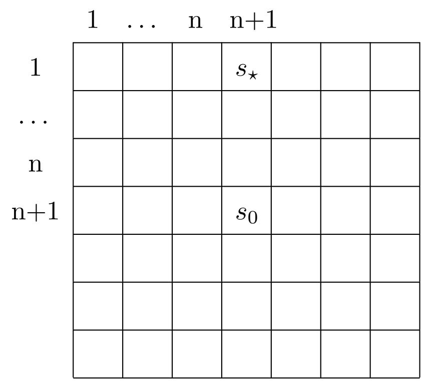
University of Basel
Computer Science

Exercise Sheet 3

Due: March 22, 2017

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 (5+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). If you encounter technical problems or have difficulties understanding the task, please let us – the tutor or assistants – know *sufficiently ahead of the due date*. Please also read the *hints* below.
The objective of last week's Exercise 2.3 was to implement a blocks world variant where blocks were associated with a size, blocks were only allowed to be stacked on larger blocks, and there was a fixed number of ordered table positions where towers of blocks could be built. This restricted blocks world corresponds to a closely related problem: the *Tower of Hanoi* (https://en.wikipedia.org/wiki/Tower_of_Hanoi). In this problem, towers (or rather tower positions) are called *pegs* and blocks are called *disks* (each disk still has a unique size and disks can only be moved to a peg if that peg is empty or if there are only larger disks stacked onto it). For this exercise, we consider the *4 peg* variant of the Tower of Hanoi, i.e. there are exactly four pegs, all disks are initially on the first peg and the goal is to move them to the last (fourth) peg. Furthermore, we consider a variant where each action has non-unit cost: an action moving disk i has cost i (again assuming an integer-based representation of disks as blocks in last week's Exercise 2.3).

You can find the code for the 4-pegs Tower of Hanoi state space on the website of the course. (Note that with the above renaming scheme, this also provides you with a possible solution for last week's programing project, except for the parameterized input specification and the action cost. Therefore, the code will be uploaded after the deadline for last week's exercise sheet has passed.)

(a) Implement *uniform cost search* to solve 4-pegs Tower of Hanoi problems in a file `UniformCostSearch.java`. To do so, you may inherit from the provided class in `SearchAlgorithmBase.java`, which among others provides the means to measure search statistics.

    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.

(b) 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 tower-of-hanoi 4toh_inst_5
    ```

    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.