Foundations of Artificial Intelligence

M. Helmert
T. Keller
Spring Term 2018

University of Basel
Computer Science

Exercise Sheet 4

Due: April 11, 2018

Exercise 4.1 (2+2 marks)

(a) Consider the "missionaries and cannibals" problem from the lecture. As a quick reminder: states in this problem are triples $\langle m, c, b \rangle \in \{0, 1, 2, 3\} \times \{0, 1, 2, 3\} \times \{0, 1\}$, where m gives the number of missionaries, c the number of cannibals and b the number of boats that are at the *wrong* river bank in the given state. Keep in mind that the boat can carry no more than two persons.

Now Consider the heuristic h_1 for the "missionaries and cannibals" problem where

$$h_1(\langle m, c, b \rangle) := \max\{m + c - b, 0\}.$$

Determine if h_1 is *safe*, *goal-aware*, *admissible* and/or *consistent*. Justify your answer for each property.

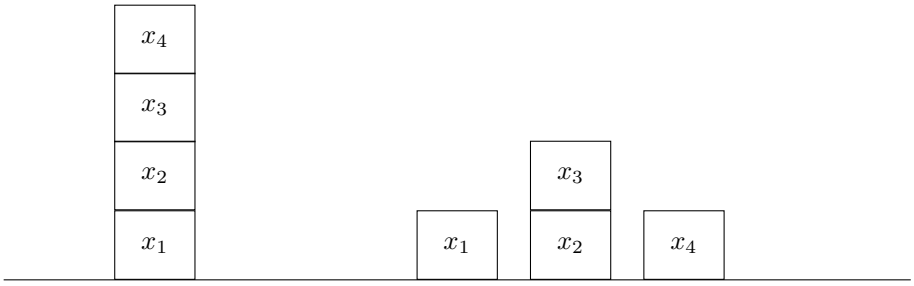
(b) Let $x_1, \ldots, x_n$ denote the blocks in a blocks world problem. Consider the heuristic

$$h_2(s) := \sum_{i=1}^{n} f(x_i)$$

for blocks world, where the function f is defined as:

$$f(x_i) := \begin{cases} 1 + |\{x_j \mid x_j \text{ is anywhere above } x_i\}| & \text{if } \mathit{goalpos}(x_i) \neq \mathit{pos}(s, x_i) \\ 0 & \text{otherwise.} \end{cases}$$

The expression $\mathit{goalpos}(x_i) \neq \mathit{pos}(s, x_i)$ holds if block x_i is on block y in state s, but should be on block $z \neq y$ in the goal (as discussed in the lecture, y and z may also represent the table). The heuristic h_2 hence determines all blocks that are not yet at their goal position and adds all blocks above those blocks (since they have to be moved before those blocks can be moved). To illustrate the heuristic, consider the following example:

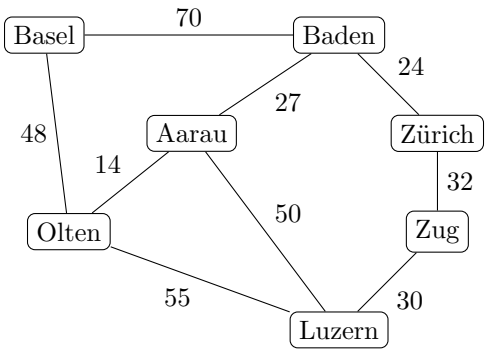


The initial state s_0 is depicted on the left side, and the goal state on the right. The heuristic value in the initial state is $h_2(s_0) = 0 + 3 + 0 + 1 = 4$ as x_1 and x_3 are on the correct block already and hence $f(x_1) = f(x_3) = 0$. Block x_2 is on x_1 instead of the table with 2 blocks above x_2, so $f(x_2) = 3$, and block x_4 is on x_3 instead of the table and there are no blocks above x_4, so $f(x_4) = 1$.

Determine if h_2 is *safe*, *goal-aware*, *admissible* and/or *consistent*. Justify your answer for each property.

Exercise 4.2 (2.5+2.5 marks)

Consider the following map:



Let the air-line distance between Zug and the other cities be given by the following table:

city	**distance**
Aarau	44
Baden	38
Basel	83
Luzern	21
Olten	51
Zug	0
Zürich	23

Consider the heuristic that maps each state to its air-line distance to Zug.

(a) Provide the search tree of A* (without reopening) when queried for the shortest path from Basel to Zug. Indicate the order in which nodes are expanded and annotate each node with its f-, g-, and h-values.

(b) Provide the search tree of greedy best-first search (without reopening) when queried for a path from Basel to Zug. Indicate the order in which nodes are expanded. Compare the result to the result of (a).

Exercise 4.3 (5+4 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 tutors or assistant – know *sufficiently ahead of the due date.*

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

The objective of last week's Exercise 3.2 was to implement uniform cost search for the SIMPLE-SOKOBAN problem with action costs. This week, we are going to work with informed search algorithms. To this end, we have extended the interface `StateSpace` with a method that returns a heuristic value for the given state (the method is called `public int h(State s)`). You can find the code on the website of the course. Note that we consider the variant without action costs that was already used in Exercise 2.2.

(a) Implement the following three admissible heuristics for the SIMPLESOKOBAN problem for a given state s, where the boxes in $\{b_1, \ldots, b_n\}$ are located in position $\text{loc}_s(b_i)$ for $i = 1, \ldots, n$. The goal position of the boxes $b_1, \ldots, b_n$ is $\text{goal}(b_i)$.

- Let $\text{dist}^{MD}(c_1, c_2)$ be the Manhattan distance between two grid cells c_1 and c_2. Then, h_1 computes the Manhattan distances of the current position and the goal position for each box and sums these values up, i.e.,

$$h_1(s) = \sum_{i=1}^{n} \text{dist}^{MD}(\text{loc}_s(b_i), \text{goal}(b_i)).$$

- Let $\text{dist}^{SP}(c_1, c_2)$ be the length of the shortest path between two grid cells c_1 and c_2, which differs from the Manhattan distance only in the fact that the shortest path must

be in non-wall grid cells only. h_2 is computed analogously to h_1, but the length of the shortest path is used instead of the Manhattan distance, i.e.,

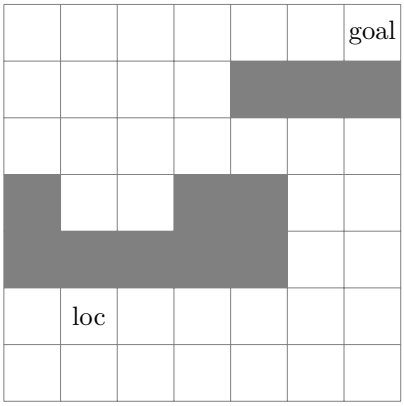
$$h_2(s) = \sum_{i=1}^{n} \text{dist}^{SP}(\text{loc}_s(b_i), \text{goal}(b_i)).$$

- Let $\text{dist}^{Sok}(c_1, c_2)$ be the length of the shortest path between two grid cells c_1 and c_2 that is such that it consists of non-wall grid cells only, and that for each grid cell in the path, the grid cell on the opposite site of the successor grid cell in the path is also accessible (i.e., if the path goes from cell 4/4 to cell 4/5, then cell 4/3 must also be accessible). h_3 is computed analogously to h_1 and h_2 as

$$h_3(s) = \sum_{i=1}^{n} \text{dist}^{Sok}(\text{loc}_s(b_i), \text{goal}(b_i)).$$

  Hint: There are reachable states where dist^{Sok} evaluates to infinity. You may return a sufficiently large heuristic value instead.

Example: Consider the following SIMPLESOKOBAN state s with just a single box located in the cell labeled with "loc" and with goal position "goal":



Gray cells indicate walls while white cells can be entered. In this example, we have $h_1(s) = 10$. The shortest path between the position of the box and its goal position that does not ignore walls is an S-shaped path of length $h_2(s) = 14$ with edges in cells 2/2, 6/2, 6/5, 4/5, 4/7 and 7/7 (assuming that the bottom left box is cell 1/1). Since it is not possible to push the box up from cell 4/5 to cell 4/6 since 4/4 is a wall, we have $h_3(s) = 16$ and an S-shaped path with edges in cells 2/2, 6/2, 6/5, 3/5, 3/7 and 7/7.

The class `SimpleSokobanStateSpace` provided on the website already contains the methods `private int h1(SimpleSokobanState s)`, `private int h2(SimpleSokobanState s)` and `private int h3(SimpleSokobanState s)` which you should implement according to the description above.

To be able to select the used heuristic as a parameter, a `SimpleSokobanStateSpace` now requires a file that contains the task (as before) and the name of the heuristic (in addition), i.e., any of the strings "blind", "h1", "h2", or "h3". The `StateSpaceTest` class has been adapted to this and prints heuristic values in addition to the state information. We have already implemented the *blind* heuristic that assigns 0 to goal states and 1 to all other states.

Provide heuristic values of the initial state for all four heuristics for all instances of the provided example problems. You can use the provided `StateSpaceTest` class to obtain these values. For instance, calling

`java StateSpaceTest sokoban sokoban_inst_6_5_2 h1`

will give you the heuristic value of the h_1 heuristic on the smallest instance (among other output).

(b) Implement A* without node reopening in a file `AstarSearch.java`. To do so, you may inherit from the provided class in `SearchAlgorithmBase.java`, which also provides code to measure search statistics.

Hint: Note that as for Exercise 3.2, a possible 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.

Test your implementation of A* and the heuristics h_1, h_2 and h_3 on the example problem instances you can find on the website. Set a time limit of 5 minutes and a memory limit of 2 GB for each run. On Linux, you can set a time limit of 5 minutes with the command `ulimit -t 300`. Running your implementation of A* on the smallest instance with

`java -Xmx2048M AstarSearch sokoban sokoban_inst_6_5_2 h1`

sets the memory limit to 2 GB and uses the heuristc h_1. 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 solution cost, 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. Use all four heuristics for these experiments.

Finally, compare the results of all four heuristics with each other. Knowing that A* with the blind heuristic is an uninformed algorithm (similar to uniform cost search or breadth-first search when using unit cost problems), what can you observe when you compare the results of using A* with the blind heuristic and the heuristics you implemented?

Important: The exercise sheets can be submitted in groups of two students. Please provide both student names on the submission and at the top of each Java file you have changed. Please create a PDF for exercises 4.1 and 4.2 and a directory containing the Java files for exercise 4.3. Afterwards, please create a zip file containing the PDF and the directory and submit it. Please, do not include a copy of the original exercise sheet in your submission.