Foundations of Artificial Intelligence

M. Helmert
T. Keller
Spring Term 2018

University of Basel
Computer Science

Exercise Sheet 2

Due: March 21, 2018

Exercise 2.1 (3+1 marks)

(a) Formalize the state space of the 15-puzzle that was introduced in Chapter 5 of the lecture. Specify the set of states, the set of actions, the action costs, and the set of transitions. Provide the initial state and the set of goal states as depicted on slide 5 of the print version of Chapter 5.

(b) How many states are in the set of states of the 15-puzzle? How many states are in the set of goal states?

Exercise 2.2 (5+3 marks)

Note: This is the same exercise as Exercise 1.2 on the presence exercise sheet of this week.
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 know. Please also read the *hint* below.
Download the file `state-spaces.tar` from the website of the course. The Java code contains the black box interface for state spaces (`StateSpace`) that was presented in the lecture, and also an interface for actions (`Action`), states (`State`), and a wrapper for state-action pairs (`StateActionPair`). We also provide an example implementation of the blocks world state space that was presented in the lecture (`BlocksStateSpace`). It implements the given interface. Finally, for running and testing the code, `StateSpaceTest` contains methods that read input files specifying concrete blocks world state spaces in a particular format: The first line specifies the number n of blocks (identified by integers $0, \ldots, n-1$), and the second and third line specify the initial and goal state, respectively, where a state is described by lists of blocks that are stacked above each other (a so called "tower"), separated by -1 to denote different towers. For example, the initial state specified in the file `blocksworld-problem` has block 4 above block 2 above block 0 on one tower, and block 1 above block 3 on another one. To run the program, first compile it with

```
javac StateSpaceTest.java
```

from a shell (on Linux) and then run it with

```
java StateSpaceTest blocks blocksworld-problem.
```

The sole purpose of the provided blocksworld implementation is to serve as an example that helps you for your own implementation. The task of this exercise is to implement the state space of a variant of the game SOKOBAN, where an agent has to push boxes from an initial position in a grid to a goal position. Unlike in the original version (see `https://en.wikipedia.org/wiki/Sokoban`), where any assignment of boxes to goal positions is a valid goal state, each box has a dedicated goal position in our variant.

(a) Implement the variant of the Sokoban state space, including actions, states and state-action pairs, in the provided file `SimpleSokobanStateSpace.java`. As for the blocks world state space, the parameters of the state space must be specified by the user, including the grid size, number of boxes, positions of walls, the initial position of the agent and initial and goal positions for each box.

(b) Implement the `buildFromCmdline` function for SIMPLESOKOBAN that parses an input file that specifies a concrete SIMPLESOKOBAN instance in the following syntax:

- three space separated numbers in the first line indicate width w and height h of the grid (8×7 in the sample instance in the file `simple-sokoban-problem`) and the number of boxes n (7 in the example)
- the following h lines with w zeroes or ones per line indicate if the respective grid cell can be entered by the agent (1) or if there is a wall (0)
- the next line gives the initial grid cell of the agent
- the final n lines indicate the initial grid cell of each box, followed by their respective goal grid cell

The method `createStateSpace` in the class `StateSpaceTest` is such that, if given the keyword `sokoban` (rather than `blocks` as in the example), it calls `buildFromCmdline` of `SimpleSokobanStateSpace`. You can therefore test your implementation by executing the command

`java StateSpaceTest sokoban simple-sokoban-problem`.

Hint: It is sufficient to create one new Java file `SimpleSokobanStateSpace.java` in the spirit of the example state space implemented in `BlocksStateSpace.java`. In particular, no changes to the other files are required!