Foundations of Artificial Intelligence

M. Helmert, T. Keller
S. Eriksson
Spring Term 2020

University of Basel
Computer Science

Exercise Sheet 2

Due: March 11, 2020

Exercise 2.1 (1+1 marks)

Characterize the following environments by describing if they are *static / dynamic*, *deterministic / non-deterministic / stochastic*, *fully / partially / not observable*, *discrete / continuous*, and *single-agent / multi-agent*. Explain your answer.

(a) real-time strategy games (StarCraft, Age of Empires, ...)

(b) moving a robotic arm

Exercise 2.2 (0.5 + 0.5 + 0.5 + 0.5 marks)

Determine if the following statements are true for all state spaces $\mathcal{S} = \langle S, A, cost, T, s_0, S_\star \rangle$. Explain your answers.

(a) If all actions have cost 0, each solution for $\mathcal{S}$ is optimal.

(b) If $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ is a minimal cost path from state $s \in S$ to state $s' \in S$ and $\pi' = \langle \pi'_1, \ldots, \pi'_m \rangle$ is a minimal cost path from s' to state $s'' \in S$, then $\pi'' = \langle \pi_1, \ldots, \pi_n, \pi'_1, \ldots, \pi'_m \rangle$ is a minimal cost path from s to s''.

(c) It is possible for unit-cost state spaces (where all actions have cost 1) that some state has the same optimal solution cost as one of its successors.

(d) If there is no solution for some state, then there is no solution for any of its predecessors.

Exercise 2.3 (2 marks)

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.

Exercise 2.4 (4+2 marks)

The task in this exercise is to write a software program. We expect you to implement your code without using existing code you find online. If you encounter technical problems or have difficulties understanding the task, please let us know.

On the website of the course, you find Java code (`state-spaces.tar.gz`) that implements the black box interface for state spaces that was discussed in the lecture, as well as an interface for states and actions. The code includes an example implementation of the interface for the blocks world problem. You can test the implementation by invoking the `StateSpaceTest` class, which creates a set of random successor states starting from the initial state. To run the program, first compile it with

```
javac StateSpaceTest.java
```

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

```
java StateSpaceTest blocks blocks-problem.txt
```

The sole purpose of the provided blocks world implementation is to serve as an example that helps you for your own implementation.
Your task is to implement the provided interface for an elevator dispatch problem as described below. Everything should be implemented in a single file called `ElevatorsStateSpace.java`. **In your handin, only submit your `ElevatorsStateSpace.java` as a solution to this exercise, any other files will be ignored.**

(a) Implement the state space of an elevators dispatch problem where

- x passengers need to be transported by n elevators across m floors,
- the initial state describes on which floor each passenger is waiting and on which floor each elevator currently is,
- passengers can embark and disembark on an elevator, and an elevator can move one floor up or down at a time, and
- the goal is to deliver each passenger to their designated floor.

(b) Implement the `buildFromCmdline` function for the elevators dispatch problem such that it parses an input file with the following syntax:

- first line: total number of passengers, elevators and floors (each separated by a whitespace).
- second line: one integer for each passenger (in order, separated by a whitespace) indicating the floor they are waiting on
- third line: one integer for each elevator (in order, separated by a whitespace) indicating the floor they start at
- fourth line: one integer for each passenger (in order, separated by a whitespace) indicating their destination floor
- note: floors are numbered from 0 to $m - 1$

The archive on the website includes a sample input file `elevators-problem.txt`. It encodes an instance with 3 passengers, 2 elevators and 5 floors (floor 0 to floor 4). The first passenger waits on floor 3, the second on floor 1 and the third on floor 4. The two elevators start at floor 0 and floor 1 respectively. The goal is to transport the first passenger to floor 2, the second to floor 4 and the third to floor 0.

To test your implementation, uncomment the two indicated lines in the method `createStateSpace` of the `StateSpaceTest` class. You can then execute the command

```
java StateSpaceTest elevators elevators-problem.txt
```

Notice that the command uses the new keyword `elevators` (rather than `blocks` as in the example above). Also notice that the problem is not expected to be solved by the provided random walk.

Important: Solutions should be submitted in groups of two students. However, only one student should upload the solution. Please provide both student names on each file you submit. We can only accept a single PDF or a ZIP file containing *.java or *.pddl files and a single PDF.