Foundations of Artificial Intelligence

M. Helmert
S. Eriksson
Spring Term 2021

University of Basel
Computer Science

Exercise Sheet 2

Due: March 17, 2021

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 answers.

(a) Freecell

(b) Tetris

Exercise 2.2 (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.

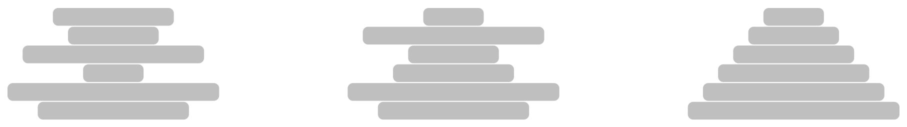
(a) If a solution for state s exists, then a solution for any successor s' of s exists.

(b) If a state space contains a transition $\langle s_1, a, s \rangle$ then it cannot contain a transition $\langle s_2, a, s \rangle$.

(c) There are state spaces that contain no actions but still have a solution.

Exercise 2.3 (1.5 marks)

In the *pancake* problem we have n pancakes with size $1, \dots, n$ on a pile. Action `flip-i` with cost 1 flips the top i pancakes. The goal to order the pile by size, i.e. the largest pancake is on the bottom, the second largest on top of the largest and so on. The following figure shows an example instance:



initial state — apply `flip-4` to initial state — goal state

Formalize the state space of the above pancake problem instance. Specify all parts of the state space, i.e. the set of states, the set of actions, the cost function, the set of transitions, the initial state and the set of goal states.

Exercise 2.4 (3.5+1.5 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 the *pancakes* problem. Everything should be implemented in a single file called `PancakesStateSpace.java`. **In your handin, only submit your `PancakesStateSpace.java` as a solution to this exercise, any other files will be ignored.**

(a) Implement the state space of the pancakes problem (for a variable number of pancakes).

(b) Implement the `buildFromCmdline` function for the pancakes problem such that it parses an input file where the pancakes are listed by size from top to bottom, separated by whitespaces. The file `pancakes-problem.txt` from the archive encodes the instance from Exercise 2.3. Make sure that your code recognizes invalid inputs such as missing or duplicate pancakes.

You can test your implementation by uncommenting the two indicated lines in the method `createStateSpace` of the `StateSpaceTest` class and then executing the command

```
java StateSpaceTest pancakes pancakes-problem.txt
```

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

Submission rules:

- Create a single PDF file (ending .pdf) for all non-programming exercises. If you want to submit handwritten parts, include their scans in the single PDF. Put the names of all group members on top of the first page. Use page numbers or put your names on each page. Make sure your PDF has size A4 (fits the page size if printed on A4).
- For programming exercises, create only those Java textfiles (ending .java) required by the exercise. Put your names in a comment on top of each file. Make sure your code compiles and test it!
- For the submission, you can either upload the single PDF or prepare a ZIP file (ending .zip, .tar.gz or .tgz; not .rar or anything else) containing the single PDF and the Java textfile(s) and nothing else. Please do not use directories within the ZIP, i.e., zip the files directly.