Planning and Optimization

M. Helmert, G. Röger
C. Büchner, T. Keller, S. Sievers

University of Basel
Fall Semester 2021

Exercise Sheet X

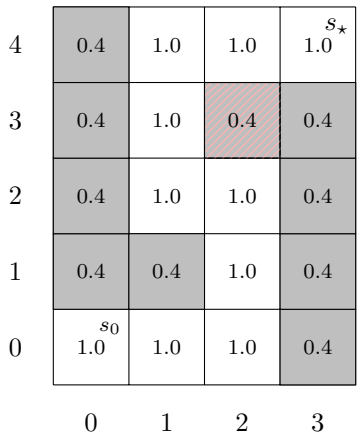
Due: December 21, 2021

Important: for submission, consult the rules at the end of the exercise. Non-adherence to these rules might lead to a penalty in the form of a deduction of marks or, in the worst case, that your submission will not be corrected at all.

This is a bonus exercise sheet. The marks on this exercise do not count towards the total number of marks and therefore neither towards the minimum number of marks you have to achieve.
The files required for this exercise are in the directory `exercise-x` *of the course repository (*`https://github.com/aibasel-teaching/planopt-hs21`*). All paths are relative to this directory. Update your clone of the repository with* `git pull` *to see the files. In the virtual machine,* `/vagrant/plan-opt-hs21` *is the repository.*

Exercise X.1 (1+1+1+1+1+2+1+1 marks)(Lecture F6)

Recall the SSP example from the lecture where an agent should move from an initial state to the goal state in a grid world. The figure below shows the grid with the initial state s_0 and the goal state $s_\star$. Possible moves are N, E, S, and W for the four directions, but actions are only applicable if they would not lead the agent out of the grid. In the goal state, no actions are applicable. The numbers in the grid indicate the probabilities p of an action to succeed, i.e., to move the agent in the intended direction. With probability $1 - p$, the agents stays where she is. The costs of applying an action is 1 in all cells except in the striped cell (2, 3), where it is 3.



In this exercise, you have to implement RTDP and LRTDP for the grid world problem. Complete all functions with a TODO in the file `rtdp/rtdp.py`. You do not need to modify or submit other files.
You can make yourself familiar with the problem representation by looking at `rtdp/instance.py`*. Executing that file prints the probabilities and costs of all cells as well as the applicable actions for all states and the resulting successors.*
To test your implementation, you can execute the file `rtdp/rtdp.py`*. It takes a mandatory argument which is the choice between* `rtdp` *and* `lrtdp`*. For RTDP, there is an optional argument* `--num-iterations` *which specifies for how many iterations to run the algorithm (the default is 30). For LRTDP, there is an optional argument* `--epsilon` *which specifies the termination threshold for change in state-values.*

We recommend to implement the functions in the following order, since you should never need to use a function further down in the order in any of the earlier functions.

- `heuristic`: implement the Manhattan distance heuristic. For state (2, 3), the heuristic should be hard-coded to return 4 instead of the Manhattan distance.
- `sample_successor`: for a given state and action, sample a successor of all possible successors according to the probability.
- `perform_trial`: implement the trial function of RTDP.
- `rtdp`: implement the main loop of the RTDP algorithm.
- `change`: compute the change of state-values of a given state when playing the greedy action compared to the given state-value.
- `check_solved`: implement the CheckSolved function of LRTDP.
- `visit`: implement the visit function of LRTDP.
- `lrtdp`: implement the main loop of the LRTDP algorithm.

Exercise X.2 (6 marks)(Lecture F7)

Execute four iterations of MCTS for the task from Exercise X.1. Some steps of MCTS rely on policies and random sampling of outcomes during selection and simulation. Use the following policies and choices:

- The tree policy always chooses the action with lowest current expected cost.
- The default policy always goes north if this action is applicable, or east if going north would leave the grid.
- If multiple applicable actions are not explicated, create a search node for north before east before south before west.
- Assume that outcomes are sampled in the following order
  (✓ means that the movement is successful, and ✗ means that the movement fails):
  - First iteration: ✓, ✓, ✓, ✓, ✗, ✗, ✗, ✓, ✗, ✗
  - Second iteration: ✗, ✗, ✗, ✗, ✓, ✗, ✗, ✓, ✓, ✓
  - Third iteration: ✗, ✓, ✓, ✗, ✓, ✓, ✗, ✗, ✗, ✗
  - Fourth iteration: ✓, ✗, ✗, ✗, ✗, ✓, ✗, ✓, ✗, ✓

Submission rules:

- Exercise sheets must be submitted in groups of three students. Please submit a single copy of the exercises per group (only one member of the group does the submission).
- Create a single PDF file (ending .pdf) for all non-programming exercises. Use a file name that does not contain any spaces or special characters other than the underscore "_". If you want to submit handwritten solutions, include their scans in the single PDF. Make sure it is in a reasonable resolution so that it is readable, but ensure at the same time that the PDF size is not astronomically large. Put the names of all group members on top of the first page. Either use page numbers on all pages 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, only create those code textfiles required by the exercise. Put your names in a comment on top of each file. Make sure your code compiles and test it. Code that does not compile or which we cannot successfully execute will not be graded.
- For the submission: if the exercise sheet does not include programming exercises, simply upload the single PDF. If the exercise sheet includes programming exercises, upload a ZIP file (ending .zip, .tar.gz or .tgz; *not* .rar or anything else) containing the single PDF and the code textfile(s) and nothing else. Do not use directories within the ZIP, i.e., zip the files directly.
- Do not upload several versions to ADAM, i.e., if you need to resubmit, use the same file name again so that the previous submission is overwritten.