Assignments MDPs (2 parts)

Authors: Eugenio Bargiacchi, Pieter Libin, Roxana Radulescu, Diederik M. Roijers, Denis Steckelmacher, Timothy Verstraeten

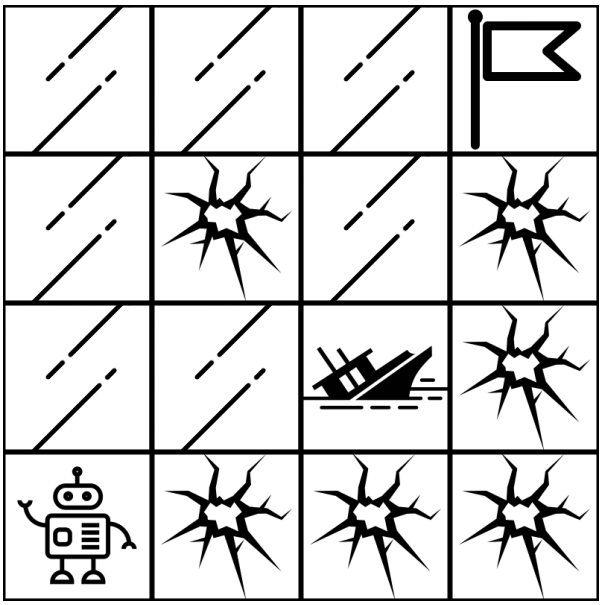
This document describes a simple (yet challenging) environment for which you will develop reinforcement learning agents. The first section presents the environment, the next one contains questions and suggestions about what to do next.

World Description

The ice world is a small, 4x4 icy grid. A robot is tasked to cross through the ice to reach the opposite side of the small world, while, if possible, collecting some treasure from a shipwreck embedded in the ice.

Unfortunately, the ice has been melting, is slippery, and is already cracked in several places. The robot must avoid falling through the cracks! Due to the slippery nature of the ice, the robot risks sliding on it at each step.

The world can be represented as follows:



(icons from thenounproject.com)

The robot tries to take one step on the ice, one cell at a time, vertically or horizontally (not diagonally). At each time-step, the robot has a 5% chance of slipping on the ice, and go all the way to the side of the environment. For instance, if the robot tries to go up from any cell in the first column, it may slide across the environment and end up on the top-left cell (or in any crack

it may encounter while sliding). The robot cannot move outside the environment, so trying to go up when in the first row has no effect.

Reaching the goal rewards the robot with 100 points. Passing on the shipwreck will allow the robot to collect some of the treasure inside: each time the robot passes there it will get an additional 20 points reward. If the robot falls through the cracks is destroyed (ending the episode), and so it's penalized with -10 points.

More formally, the environment has a total of 16 states (one for each possible position of the robot in the ice world). The robot has 4 actions available ('UP', 'RIGHT', 'DOWN' and 'LEFT'). Each action moves the robot in its direction with probability 0.95. When the robot tries to move outside of the grid, the action will have no effect with probability 1.

Slipping will happen with probability 0.05, and it results in the robot moving in the direction specified by the action up to either the grid border, or to the first crack (into it). (NB: it is possible for both slipping and not slipping as a result of an action to result in the same state, i.e., when the robot is one step away from the edge in the direction of movement, or when the agent is already at the edge and cannot move in that direction any further. Therefore, the transition function for states near the edge (for a given action/direction) looks different from states further away from the edge.)

Assignments explanation

Implementations of the environment described above are provided in Python, Java and Matlab. You can also reimplement the environment yourself in any language you want, but we strongly advise you to use our implementation.

These assignments consist of two parts, each consisting of must-haves and optional assignments. Each part is graded separately. Perfectly doing the must-haves leads to a 12/20 score for that part of the assignments. The remaining 8 points can be attained by doing optional assignment. Each optional assignment has a maximum number of additional points. NB: even though your number of points can exceed 20, the maximum possible score per part is capped at 20. However, note that it is possible to lose points by making mistakes in the assignments, therefore, when aiming for a 20, it is advised to aim for a total number of attainable points higher than 20.

Assignments part 1:

1. **Must-have:** Implement Value Iteration[1] on the environment. Our implementation of the environment provides transition probabilities and rewards, so you can focus on the VI algorithm without having to fully understand what the environment does. Your

1 http://incompleteideas.net/sutton/book/ebook/node44.html

implementation should output the optimal deterministic policy for this environment, along with the value of all 16 states.

2. **Must-have:** Implement Howard's Policy Iteration[2] and compare the output and runtime of Value Iteration and Policy Iteration.
3. **Optional assignments**: Implement the following algorithms and compare the results to the other algorithms you implement(ed), in terms of runtime. NB: always check wheter your output is correct as well.
   a. Simple Policy Iteration (max 3 points)
   b. Planning by Linear programming (max 10 points) – see the primal linear program formulation in Section 4.1 of Littman Dean and Kaelbling, UAI 1995 (https://arxiv.org/ftp/arxiv/papers/1302/1302.4971.pdf). For this method, you will need to use a linear programming library. Python has built-in linear programming libraries, but they are not very fast. Could alternatives are lpsolve 5.5, and Gurobi. Gurobi is a commercial package, but as a student you can request an academic license.
4. **Optional assignment (max 10 points)**: Read Van Seijen and Sutton – Efficient planning in MDPs with small backups, ICML 2013 (http://proceedings.mlr.press/v28/vanseijen13.pdf) and answer the following questions: a) Why do small backups make planning more efficient? b) The paper focusses on model-based learning rather than just planning. Why would this algorithm be especially useful in a model-based learning context rather than onlyplanning? c) How would you adapt value iteration in order to make use of small backups? Please write an adapted algorithm in pseudocode, accompanied by a line-by-line explanation in text.

Assignments part 2:

Implementations of the environment described above are provided in Python, Java and Matlab. You can also reimplement the environment yourself in any language you want, but we strongly advise you to use our implementation.

5. **Must-have:** Q-Learning[3] on the environment. Compare the values output by Value Iteration to your Q-Values. Does Q-Learning lead to the optimal policy? Does it find the same values as Value Iteration?
6. **Must-have:** Implement SARSA[4] and compare the performance (i.e., cumulative reward) during learning of Q-learning and SARSA. Is there a difference, and if so, how can this be explained?
7. **Optional (4 points):** Extend your Q-Learning implementation with Experience Replay[5] and compare the performance with standard Q-learning and SARSA. Does it still find the

2 http://incompleteideas.net/sutton/book/ebook/node43.html
3 http://incompleteideas.net/sutton/book/ebook/node65.html
4 http://incompleteideas.net/sutton/book/ebook/node64.html
5 http://busoniu.net/files/papers/smcc11.pdf

optimal policy? How many episodes does your agent need in order to learn (with and without ER)?

8. **Optional (6 points):** Extend Q-Learning with Eligibility Traces[6] and compare the performance with standard Q-learning and SARSA. Combining eligibility traces with experience replay is difficult (so don't do it at first). Why, and how would you do it?
9. **Optional (2 points + 6 for implementation):** Do you see any other way to increase the sample-efficiency of Q-Learning? Why is sample-efficiency so important in real-world applications of reinforcement learning? If you want to implement your idea for additional points (max 6), please discuss with Denis about the idea first.

Plotting

Debugging a reinforcement learning agent is quite difficult. The first and most useful step consists of plotting learning curves. Those curves show the cumulative reward per episode obtained over time, and should go upwards. Plotting those curves is extremely easy using **gnuplot**.

1. Your program should write cumulative rewards in a file. For instance, after each episode, write the episode number and cumulative reward obtained during this episode in a file. The file looks like this:

   ```
   1,1.283764
   2,0.938478
   3,1.383632
   4,1.029384
   5,0.29474
   ```

   The file is a valid CSV file, that you can open in Excel/Calc/etc to plot, if you are familiar with one of these programs. Or you can use gnuplot to produce plots very quickly.
2. Install **gnuplot** (gnuplot and gnuplot-qt on most Linux distributions), an Open-Source plotting software available on every major operating system. Gnuplot is actually a tool you should discover, if you are not yet using it. It is extremely quick at plotting big datasets, and produces Nature-ready, state-of-the-art-looking plots (with some configuration).
3. Launch Gnuplot, by typing `gnuplot` on the command line.
4. In gnuplot, type these commands (the first one allows Gnuplot to read CSV files)

   ```
   set datafile separator ","
   plot "file.csv" using 2 smooth bezier with lines title "Learning Curve"
   ```

   Translated to English, this long command tells Gnuplot to plot data from "file.csv", with the data in the second column, smoothing the data so that it is easier to read, using lines

6 http://incompleteideas.net/sutton/book/ebook/node72.html

instead of dots or points, with the given title. Our reference implementation produces the following plot: