Introduction to Artificial Intelligence

Dr. Thomas Keller
C. Büchner, C. Grundke, A. Kauffmann

University of Zürich
Spring Semester 2022

Exercise Sheet 9

Due: May 11, 2022

Points total: 20 marks

Exercise 9.1 – MDP Formalization

7 marks

You are in charge of a large Mediterranean fish farm that produces prawns. Once every catching season, you need to decide what part of the prawn population you will catch and commercialize, and what part you will leave in the farm and allow to reproduce. Assuming that the prawn population in any given season is denoted by x, the reward you get if you commercialize $y \leq x$ prawns is $10y$, whereas the number of prawns on the fish farm if you leave $z \leq x$ prawns to reproduce is a random variable. For the sake of simplicity,[1] we assume that with probability 0.7, the population next season will be $2z$ (a standard year), with probability 0.2, it will be $4z$ (a great year!), and with probability 0.1 it will be $0.5z$ (a not-so-good year).

(a) Assume you start with a population of 1000 prawns, and you value the reward of one catching season twice as high as the reward for the next one (after all, climate change is likely to turn your fish farm into a tropical wave pool for tourists soon enough). You might also assume that your fish farm has a maximum capacity of N. Formalize your fish farm as a *Markov Decision Process* $\mathcal{M} = \langle S, A, T, R, s_0, \gamma \rangle$, clearly and formally defining each of the elements of the problem: what are the states S in your model, what the possible actions A, what is the transition function T, what is the reward function R, what is the initial state s_0, and what is the discount factor γ.

Hint: We formalize MDPs in the lecture, but provide a brief explanation here so you can start to work on the exercise: MDPs are very similar to transition systems (as defined on slide 7 in search1), except that we have rewards instead of costs and that each outgoing edge (representing an action) can end in multiple successor states, depending on the transition probabilities.

The transition model T *of the transition system is hence replaced by a* transition function T *in the MDP, which is a function denoting for each triple* $\langle s, a, s' \rangle \in S \times A \times S$ *the probability to end in state* s' *when applying* a *in* s*. (If* $T(s, a, s') = 0$*, then it is impossible to transition between* s *and* s' *through* a*.) Furthermore, there are no* action costs *in an MDP. Instead, the* reward function *maps state* s *and action* a *to a real number. For example, if applying an action* a_1 *in state* s *incurs a reward of* -1 *and leads to state* s_1 *with probability* 0.8 *and to state* s_2 *with probability* 0.2*, we have* $R(s, a) = -1$*,* $T(s, a_1, s_1) = 0.8$ *and* $T(s, a_1, s_2) = 0.2$*.* (5 marks)

(b) Discuss the influence of the discount factor in this scenario. To do so, answer the following two questions:

- In general, what changes about your business model if you increase the discount factor?

[1] The actual law governing the growth of prawn population is more complex, but results nonetheless in a probability distribution, which is what interests us here.

- How does the *optimal policy* of your fish farm change when you increase the discount factor?

(2 marks)

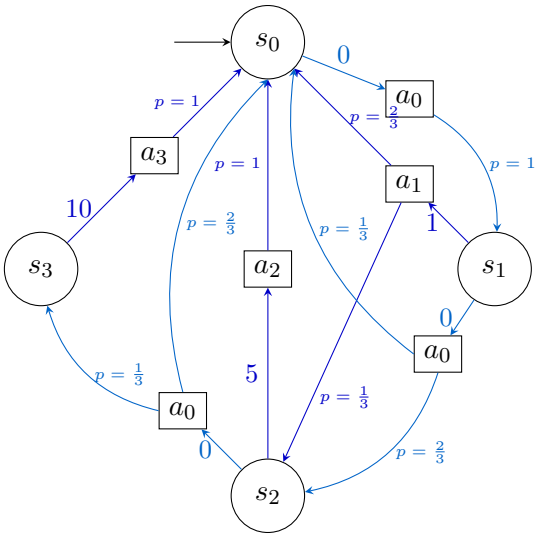
Exercise 9.2 – Value Iteration

5 marks

Consider the discounted reward MDP $\mathcal{T} = \langle S, A, R, T, s_0, \gamma \rangle$ with

- $S = \{s_0, s_1, s_2, s_3\}$,
- $A = \{a_0, a_1, a_2, a_3\}$,
- $R(s, a_0) = 0$ for all $s \in S$
  $R(s, a_1) = 1$ for all $s \in S$
  $R(s, a_2) = 5$ for all $s \in S$
  $R(s, a_3) = 10$ for all $s \in S$,
- $T(s_0, a_0, s_1) = 1$
  $T(s_1, a_0, s_0) = \frac{1}{3}$ and $T(s_1, a_0, s_2) = \frac{2}{3}$
  $T(s_2, a_0, s_0) = \frac{2}{3}$ and $T(s_2, a_0, s_3) = \frac{1}{3}$
  $T(s_1, a_1, s_0) = \frac{2}{3}$ and $T(s_1, a_1, s_2) = \frac{1}{3}$
  $T(s_2, a_2, s_0) = 1$
  $T(s_3, a_3, s_0) = 1$
  $T(s, a, s') = 0$ for all other $s, s' \in S$ and $a \in A$
- $\gamma = 0.9$.

A graphical description of $\mathcal{T}$ can be seen below:



Perform *five iterations* of value iteration with initial values $\hat{V}^0(s_0) = 10$ and $\hat{V}^0(s) = 0$ for $s \in \{s_1, s_2, s_3\}$. Provide the values for all states after each iteration in the table below, and provide the policies that result from the values after the 1st, 2nd, 3rd, 4th and 5th iteration. Would value iteration need to perform further iterations after the 5th iteration?

i	$\hat{V}^i(s_0)$	$\hat{V}^i(s_1)$	$\hat{V}^i(s_2)$	$\hat{V}^i(s_3)$
0	10	0	0	0
1				
2				
3				
4				
5				

$\pi_1(s_0) =$ _____ $\pi_1(s_1) =$ _____ $\pi_1(s_2) =$ _____ $\pi_1(s_3) =$ _____

$\pi_2(s_0) =$ _____ $\pi_2(s_1) =$ _____ $\pi_2(s_2) =$ _____ $\pi_2(s_3) =$ _____

$\pi_3(s_0) =$ _____ $\pi_3(s_1) =$ _____ $\pi_3(s_2) =$ _____ $\pi_3(s_3) =$ _____

$\pi_4(s_0) =$ _____ $\pi_4(s_1) =$ _____ $\pi_4(s_2) =$ _____ $\pi_4(s_3) =$ _____

$\pi_5(s_0) =$ _____ $\pi_5(s_1) =$ _____ $\pi_5(s_2) =$ _____ $\pi_5(s_3) =$ _____

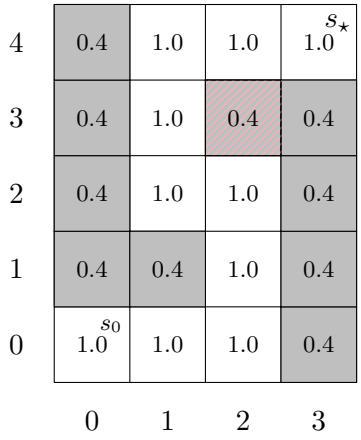
Exercise 9.3 – Asynchronous Value Iteration

8 marks

Recall the MDP example from the lecture where an agent moves in a grid world. 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. The exception is the state in the upper right corner labeled $s_\star$ where only a special noop action is applicable (we call $s_\star$ the *goal state*). All actions incur a reward of -1 except for moving away from the striped cell (2, 3), which incurs a reward of -3 and for the noop action in $s_\star$ which yields a reward of 0. We consider a discount factor of 0.9.

There is a chance that the agent gets stuck when trying to move. 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.

Note: This MDP can be modelled more naturally as a Stochastic Shortest Path Problem (SSP), which are MDPs where the reward function is replaced by action costs and the discount factor by a goal.



In this exercise, you have to implement a variant of value iteration called *asynchronous value iteration* for the grid world problem. The difference to value iteration as presented in the book and lecture is that not all states are updated simultaneously (called a synchronous backup), but only one (random) state is updated in every iteration. This can reduce the computation significantly and still guarantees to converge *if all states are selected repeatedly.*

One problem with asynchronous value iteration is that there is no obvious termination criterion. In practice, it is often used in settings where we would like to improve our solution while time is left (called any-time setting). For this exercise, we terminate based on a fixed number of iterations.

Download the file `asynchronous-vi.zip` from the exercise material. Your task is to implement all functions with a `TODO` in the file `asynchronous-vi.py`. You do not need to modify or submit other files.

Hint: You can make yourself familiar with the problem representation by looking at `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 `asynchronous-vi.py`. It takes an optional argument `--num-iterations` which specifies for how many iterations to run the algorithm (the default is 300).

Hint: 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.

- `compute_q_value`: compute the Q-value (called action-value in the lecture) for a given state and action and given state-values.
- `compute_greedy_action_and_q_value`: for a given state and given state-values, compute the greedy (=best) applicable action and the resulting Q-value of that action.
- `bellman_update_in_place`: perform an in-place Bellman update of the state-value of a random state of the MDP.
- `asynchronous_value_iteration`: perform asynchronous value 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. Make sure your PDF has size A4 (fits the page size if printed on A4). Submit your single PDF file to the corresponding exercise assignment in MOODLE.
- For programming exercises, only create those code text files 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. Create a ZIP file (ending `.zip`, `.tar.gz`, or `.tgz`; *not* `.rar` or anything else) containing the code text file(s) (ending `.py`) and nothing else. Do not use directories within the ZIP, i.e., zip the files directly.

- Do not upload several versions to MOODLE, i.e., if you need to resubmit, use the same file name again so that the previous submission is overwritten.