Planning and Optimization

G. Röger, T. Keller
G. Francès

University of Basel
Fall Semester 2018

Exercise Sheet F

Due: December 9, 2018

The files required for this exercise are in the directory `exercise-f` *of the course repository (*`https://bitbucket.org/aibasel/planopt-hs18`*). All paths are relative to this directory. Update your clone of the repository with* `hg pull -u` *to see the files.*

Exercise F.1 (**MDP Modeling and Value Iteration**: 3+3+4 marks)

(a) **The Fish Farm.** 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 $y < x$ prawns is $10y$, whereas the number of prawns on the fish farm if you leave $z < x$ 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 $\lceil 1.4z \rceil$ (a standard year), with probability 0.2, it will be $\lceil 1.8z \rceil$ (a great year!), and with probability 0.1 it will be $\lceil 0.9z \rceil$ (a not-so-good year).

Assume you start with a population of 1000 prawns, and you consider the course of action only over the next 10 catching seasons (after all, climate change is likely to turn your fish farm into a tropical wave pool for tourists after that). You might also assume that your fish farm has a maximum capacity of N. Formalize the above problem as a *finite-horizon Markov Decision Process* $\mathcal{M} = \langle S, A, T, R, s_0, H \rangle$, clearly and formally defining each of the elements of the problem: what are the states in your model, what the possible actions, what are the transition probabilities and rewards, and what is the initial state.

(b) **A Hawaiian Retreat.** Your fish farm has gone bankrupt because of a silly modeling mistake that led you to sell all of your prawns during the first season. After that, you decide that MDPs are not your thing, and try to forget about your prawn story taking some holidays in Hawaii. You have however two conflicting constraints: on the one hand, you want to take a break as soon as possible. On the other hand, your business failure left you with little money, so you want to book as cheap as possible. Furthermore, you need to attend a meeting with your disgruntled investors in exactly two weeks, so you don't have too much flexibility on the flight dates: you want to leave Basel *any day during the next 7 days*, and fly back exactly one week after that, and you want to buy a single round-trip ticket. You know that ticket prices fluctuate a lot, hence waiting a few days to book your tickets could get you a better deal. After a bit of research, you discover that there are two rival, independent companies that operate flights between Basel and Honolulu. You only have time to look at prices and buy tickets once per day. To make things interesting, we will assume that last-minute tickets (i.e. tickets to fly on the same day they are purchased) tend to be significantly cheaper.

Model the above problem as a *Stochastic Shortest Path* problem $\mathcal{M} = \langle S, A, T, c, s_0, S_\star \rangle$. You have some freedom when it comes to specifying your model, but the decisions you make in your model should be justified and reflect the above constraints; your transition probabilities should be Markovian, and your cost function should take into account the ticket price, but also the fact that you want to fly as soon as possible. You might find useful the possibility of somehow assigning some very high cost to action trajectories leading to a no-flight situation.

[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.

(c) **Push your luck.** The Hawaiian break made you recover your inner peace and your usual lucidity (or lack thereof), and you have decided that not all is lost yet, and that you should put your deep knowledge of MDPs to good use, go to the Basel casino and try your luck. The latest trend up there seems to be *Push your Luck*, a simple dice game where the player has the chance of repeatedly rolling a single fair die in order to get some rewards. The player can roll a fair day repeatedly, until she decides to collect her "accumulated" reward. The accumulated reward is defined as the product of all die outcomes *since the last game reset*, or 0, if no die has been rolled yet since that reset (the beginning of the game is also considered a game reset). The game "resets" when either the player collects the accumulated reward, or *the die shows a number that had already appeared since the last reset*. As an example, imagine that after three rolls the outcomes have been $1, 3, 4$. The player might decide to collect now, in which case her reward will be $1 \cdot 3 \cdot 4 = 12$, and the game will reset. Or she might be tempted to wait a bit, since perhaps she gets a 6 on the next roll, in which case she could collect a much more attractive reward of $1 \cdot 3 \cdot 4 \cdot 6 = 72$. On the other hand, waiting is risky: if she gets any outcome in $\{1, 3, 4\}$, then the game resets, but without her getting any reward. In any case, the player can continue playing *forever*.

- Provide the formalization of the above problem as a *discounted reward* MDP $\mathcal{M} = \langle S, A, T, R, s_0, \gamma \rangle$. The discount factor you use is not relevant for the correctness of your model, only for its solutions, so you can assume any factor you prefer for now.
- Discuss briefly what should be changed in your model in order to work for games with dice having an arbitrary number N of sides.
- Discuss briefly what modifications could be done to model the following variation of the game, or justify why it is not possible to model that variation. In this variation, the player can only play until the first reset, that is, either she rolls the die a few times without getting any repeated outcome, collects the reward, and goes home, or a repeated outcome shows up before she has decided to stop, in which case she goes home with empty pockets.

Exercise F.2 (**The Bellman Optimality Equation**: 3+4 marks)

(a) The Bellman optimality equation can be used to derive the optimal state value function V^* of any MDP $\mathcal{M} = \langle S, A, T, R, s_0, \gamma \rangle$. In the case of discounted-reward MDPs, this optimal V^* is the unique solution to the set of equations given by

$$V(s) = \max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V(s') \right],$$

for each state $s \in S$.

A possible way of solving the above set of equations would be to cast it as a Linear Program and use an LP solver, if it wasn't for the annoying max operator, which makes the equations non-linear. Think (or do a bit of research) about some way of circumventing this small obstacle, perhaps by using extra LP variables, or a reasonable objective function. Describe formally the Linear Program whose solution gives us the optimal value $V^*(s)$ of every state, and briefly justify why it works. Does this LP feature any integer variable? What is the size of the LP? What is the cost of solving the LP? What is the cost of computing V^* through this method?

(b) Take your MDP model for the *Push your Luck* game above. Write a Python script that generates the LP specification for the formulation of V^* that you developed in question (a) corresponding to the push-your-luck MDP model. You should write the LP in any format that you prefer (e.g., the input format of `lp-solve` that we saw in class), *as long as the format is readable and clear*: part of the evaluation of this question will be on how easy it

is to understand that your LP problem corresponds to the formulation in (a). Hence, we recommend that you use meaningful names for the LP variables and, if possible, for the LP constraints. Your script should be parametrized by two different things: the number of die sides in the game, and the discount factor. This means that it should be straight-forward to try out different combinations of number of sides and discount factor. You have a possible skeleton of the script in `exercise-f/scripts/generate_lp.py`.

Once your script is working, solve the LP corresponding to $\gamma = 0.9$ and the standard 6-side game with your solver of choice. What is the optimal state value $V^*(s_0)$ corresponding to the initial state? What is the optimal action to take in that state? What is the optimal action to take after having obtained the sequence of die outcomes $1, 2, 3$? And the sequence $1, 2, 6$? How do the above answers depend on the discount factor that you use? Play a bit with the factor and observe the differences, if any.

Please, submit (1) your Python code, (2) the resulting LP encoding(s), and (3) an answer to the above questions.

Exercise F.3 (**Value Iteration:** 3+1+2 marks)

This exercise makes use of the excellent Berkeley Pacman Project materials. We want you to answer Questions 1–3 of the Reinforcement Learning project.[2] For this, you should download the Pacman framework (available from the URL), which is written in Python, and you should implement the Value Iteration algorithm as described in Q1. You can use the provided "autograder" script to be confident in your solutions, but we will grade this question by checking the code ourselves. In questions Q2 and Q3, you are asked to experiment a bit with the different parameters of an MDP in order to understand how that affects the optimal solution. Submit the relevant code files so that we can execute your solution, *and also* briefly discuss your choices of parameters for question Q3 and the reasoning behind these choices. You can ignore the scores assigned by the Berkeley course designers, they are obviously too generous 😊.

Exercise F.4 (**Determinizations & PPDDL:** 2 marks)

In `models/tireworld` you can find the PPDDL benchmark set of the Tireworld problem from the 2006 International Probabilistic Planning Competition.[3] Have a look at the domain and a few instances, and describe in your own words what problem the encoding is representing, being precise enough so that anyone could come up the formal MDP model by reading your description. Additionally, specify the most-likely-outcome and all-outcomes determinizations of Tireworld *as deterministic PDDL* domain files. Submit your domains as `tireworld-mlo.pddl` and `tireworld-ao.pddl`. Discuss briefly the advantages and shortcomings of each determinization for this particular domain.

Exercise F.5 (**FF-Replan:** 5 marks)

The use of determinizations to perform online action selection within a *plan + execute + monitor* approach for probabilistic planning showed very good performance on the international probabilistic planning competitions in 2004 and 2006. The top performant planning in both competitions was `FF-Replan`, a simple determinization-based planner build on top of the classical (deterministic) `FF` planner. In this exercise, we ask you to read and discuss the main ideas behind `FF-Replan`:

> Sungwook Yoon, Alan Fern, and Robert Givan. FF-Replan: A Baseline for Probabilistic Planning. In *ICAPS'07* pp. 352–359, 2007.

Read and summarize the main ideas of the paper in your own words (between 300 and 600 words). Your answer should briefly discuss the following:

[2] `http://ai.berkeley.edu/reinforcement.html`

[3] PPDDL is an extension of PDDL to accommodate probabilistic planning. The language specification can be found at `http://reports-archive.adm.cs.cmu.edu/anon/2004/CMU-CS-04-167.pdf`, but you'll probably find that the language is close enough to PDDL to be easily understood. Additionally, the paper in exercise (F.5) has a short description of the language that should be enough for your needs.

- How does `FF-Replan` work? What kind of determinizations does it use? How does it use them? How often does `FF-Replan` compute a determinization of the original probabilistic problem?
- How does `FF-Replan` determinize PPDDL nested probabilistic effects? How is the partial state-action policy that `FF-Replan` keeps in memory built?
- What is the relation between `FF-Replan`'s main techniques and Hindsight optimization?
- What do the impressive results of `FF-Replan` tell you about the type of problems used in the competition? How are the significant empirical differences between the two variants of `FF-Replan` explained?

The exercise sheets can be submitted in groups of three students. Please submit one single copy of the exercises per group (only one member of the group does the submission), and provide all student names on the submission.