Planning and Optimization

M. Helmert, T. Keller
S. Eriksson, F. Pommerening, S. Sievers

University of Basel
Fall Semester 2019

Exercise Sheet F

Due: December 8, 2019

The files required for this exercise are in the directory `exercise-f` *of the course repository (*`https://bitbucket.org/aibasel/planopt-hs19`*). All paths are relative to this directory. Update your clone of the repository with* `hg pull -u` *to see the files.*

Exercise F.1 (3+5+2 marks)

Push your Luck is a simple dice game where the player repeatedly rolls a single fair die to get rewards. The player can roll the die 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 with a six-sided die 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*.

(a) Formalize the game with an N-sided die as an MDP $\mathcal{M}_{N,\gamma} = \langle S, L, R, T, s_0, \gamma \rangle$ where N and the discount factor γ are parameters. For the set of states S, use the powerset of the set $\chi = \{1, \ldots, N\}$, i.e., $S = 2^{\chi}$. The interpretation is that χ contains all possible die rolls and a state $s \in S$ contains exactly the die rolls obtained since the last reset.

(b) The file `mdps/push-your-luck.py` contains a skeleton of a script to generate the LP model for computing V^* and an optimal policy for $\mathcal{M}_{N,\gamma}$. Complete the script for your MDP model from (a).

The script uses PySCIPOpt, a Python interface to an optimization suite including SoPlex. Use `install-scip.sh` *to install it and have a look at* `scipdemo.py` *for example code.*

(c) Use your script from (b) to experiment with the game for a six-sided die ($N = 6$) and a discount factor $\gamma = 0.9$. Discuss the following questions:

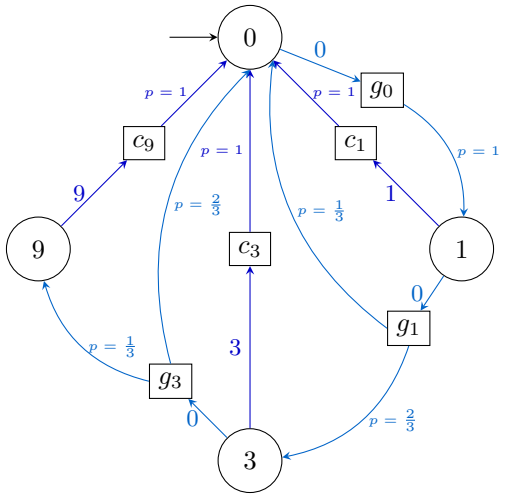
What is the optimal state value $V^*(s_0)$ corresponding to the initial state? What is the optimal action to take in that state? What is the optimal action to take after having obtained the sequence of die outcomes $1, 2, 3$? And the sequence $4, 5, 6$? How do the above answers depend on the discount factor that you use? Play a bit with the factor and observe the differences, if any.

Exercise F.2 (4+4+2 marks)

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

- $S = \{0, 1, 3, 9\}$,
- $L = \{g_0, g_1, g_3, c_1, c_3, c_9\}$,
- $R = \{g_0 \to 0, g_1 \to 0, g_3 \to 0, c_1 \to 1, c_3 \to 3, c_9 \to 9\}$,
- $T = \{\langle 0, g_0, 1\rangle \to 1, \langle 1, g_1, 3\rangle \to \frac{2}{3}, \langle 1, g_1, 0\rangle \to \frac{1}{3}, \langle 3, g_3, 9\rangle \to \frac{1}{3}, \langle 3, g_3, 0\rangle \to \frac{2}{3}, \langle 1, c_1, 0\rangle \to 1, \langle 3, c_3, 0\rangle \to 1, \langle 9, c_9, 0\rangle \to 1\}$,
- $s_0 = 0$ and
- $\gamma = 0.9$.

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



(a) Consider policy $\pi = \{0 \to g_0, 1 \to c_1, 3 \to c_3, 9 \to c_9\}$. Perform *two iterations* of policy iteration with initial policy π and $\epsilon = 0.1$ as a termination criterion for policy evaluation (you may round intermediate results to two decimals). In the first iteration of the algorithm, use initial values $V_\pi(0) = 4.7$, $V_\pi(1) = 5.2$, $V_\pi(3) = 7.3$, $V_\pi(9) = 13.3$ for policy evaluation (in principle, you can use an arbitrary initialization, but these values allow policy evaluation to converge faster). In the second iteration of the algorithm, initialize the values for policy evaluation to the last values of the previous iteration. You do not need to provide all intermediate values of policy evaluation, but after both iterations of the algorithm, write down the final values of policy evaluation as well as the updated policy. Would you need to perform further iterations of policy iteration?

(b) Perform *three iterations* of value iteration with initial values $V(s) = 0$ for all states $s \in S$. Provide the values for all states after each iteration, and provide the policy that results from the values after the third iteration. Would you need to perform further iterations?

(c) If you performed asynchronous value iteration in (b) and you wanted to update all four states in the first four value updates, in which order would you prefer to update them? In which values would those four updates result? Briefly justify your choice.