LSINF2275 - Data mining and decision making

Markov Decision Processes

Project guidelines

Professor: Marco Saerens

Teaching Assistants: Sylvain Courtain and Pierre Leleux

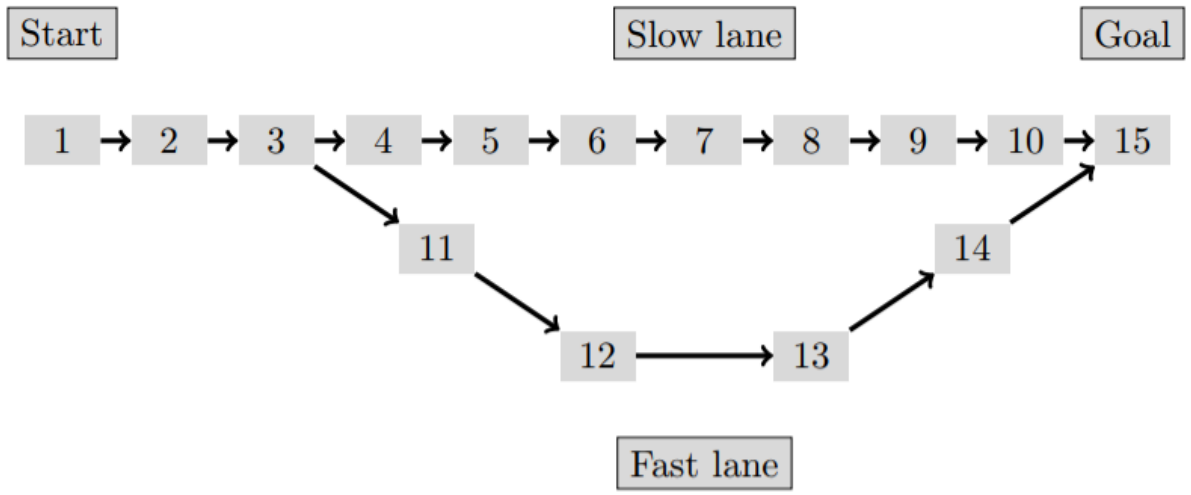
1 Objective

The objective of this project is to put into practice some of the techniques introduced in the data mining and decision making lectures. This will be done through the study of a practical case which requires the use of a software related to the statistical processing of data. This work aims at applying algorithms solving "Markov decision processes" in the framework of a Snakes and Ladders game.

2 Problem statement

The realization of the project will be carried out in groups of maximum 2 students. You are asked to determine the solution of the following problem:

Let us assume that you are playing a Snakes and Ladders game which is made of 15 squares. Square number 1 is the initial square (start) and square 15 is the winning square (arrival). If you reach square 15, you win. The board can contain traps that slows your progression if triggered. A schematic representation of this game is shown below: To move forward, you can



choose between two dices

- The "security" dice which can only take two possible values: 1 or 0. It allows you to move forward by 0 or 1 square, with a probability of 1/2. With that dice, you are invincible, which means that traps are not triggered when playing with the security dice.

- The "normal" dice, which allows the player to move by 0, 1 or 2 squares with a probability of 1/3. However, when you play with this dice, you are vulnerable to traps.

At square 3, there is a junction with two possible paths. If the player just passes through square 3 without stopping, he will continue to square 4 or beyond (5, 6, etc), as if the other path did not exist. Conversely, if the player stops on square 3, he will have, on the following turn, an equal probability of taking the slow or the fast lane. For instance, for the security dice, if the player is on square 3 and the result is 0, the player stays on square 3 with probability 1. If the result is 1, he reaches square 4 or 11, each with probability 0.5. Both paths ultimately reach the final square.

You should also define "trap" squares on the game, that are triggered when stopping exactly on that square. Recall that the player is only sensitive to traps when drawing the normal dice. There are different types of trap, each has its own effect

- Type 1 – Restart: Immediately teleports the player back to the first square.
- Type 2 – Penalty: Immediately teleports the player 3 squares backwards.
- Type 3 – Prison: The player must wait one turn before playing again.
- Type 4 – Mystery: Randomly applies the effect of one the three previous traps, with an equal probability.

Note that the first and final square can never be trapped.

We ask you to determine the optimal strategy, i.e., for each square, which dice should be used to reach the goal square in a minimal number of turns, on average (the total cost is the number of turns to reach the goal square). You should determine this solution in two different scenarios:

- You should exactly stop on the arrival square to win. The game board is designed as a circle, which means that, if you overstep the last square, you have to restart from the 1st square.
- You win as soon as you have land on or overstep the arrival square (i.e. if you are on square 15 or further).

3 Implementation

You are asked to implement, in *Python* 3, the following function

```
markovDecision(layout,circle)
```

This function launches the Markov Decision Process algorithm to determine the optimal strategy regarding the choice of the dice in the Snakes and Ladders game, using the "value iteration" method.

Inputs:

- `layout`: a vector of type `numpy.ndarray` that represents the layout of the game, containing 15 values representing the 15 squares of the Snakes and Ladders game:

  `layout[i]` = 0 if it is an ordinary square
  = 1 if it is a "restart" trap (go back to square 1)
  = 2 if it is a "penalty" trap (go back 3 steps)
  = 3 if it is a "prison" trap (skip next turn)
  = 4 if it is a "mystery" trap (random effect among the three previous)

- `circle`: a boolean variable (type `bool`), indicating if the player must land exactly on the finish square to win (`circle = True`) or still wins by overstepping the final square (`circle = False`).

Output: Your function `markovDecision` is expected to return a type `list` containing the vectors `[Expec,Dice]`

- `Expec`: a vector of type `numpy.ndarray` containing the expected cost (= number of turns) associated to the 14 squares of the game, excluding the finish one.

- `Dice`: a vector of type `numpy.ndarray` containing the choice of the dice to use for each of the 14 squares of the game (1 for "security" dice, 2 for "normal" dice), excluding the finish one.

In addition, it is also interesting to compare this strategy with other (sub-optimal) strategies, e.g. the use of dice 1, dice 2 or dice 3 only, a mixed random strategy, or a purely random choice, etc. Launch/simulate an important number of games and compare empirically the performance of each of these strategies with the optimal strategy (or policy) obtained by value iteration. You should then report (i) the theoretical expected cost, obtained by value iteration, and compare it with the empirical average cost when playing (simulating) a large number of games with the optimal strategy, and (ii) compare the empirical average cost obtained by the optimal strategy with the other (sub-optimal) strategies mentioned before (dice 1 only, etc). You could test this with a few different configurations of traps, depending on your time.

4 Report

Please do not forget to mention your affiliation on the cover page, together with your name (SINF, INFO, MAP, STAT, BIR, DATS, etc). You are asked to write a report of around 5 pages (without the code). This report will contain

- a brief explanation of the method used to determine the optimal strategy regarding the choice of the dice.

- a description of your implementation along with results/comparisons comparing the optimal strategy and the empirical results obtained for the Snakes and Ladders game with, or without, trap squares at relevant places.

- a discussion of the simulation results.

Your project will be evaluated on the following aspects:

- The quality of your report and code (including comments);
- The amount of experimental work (including potential additional experiments);
- The efficiency of your function `markovDecision`

Important: Beware that the efficiency of your function will be graded by an automated script that will execute your function several times using different board layouts to check if your function always returns the correct policy with the correct expected costs. It is therefore important that you strictly comply with the syntax of the function (name, inputs and outputs).

This report must be uploaded on March 22, 2020, before 23:55 together with the code (all files zipped together) on Moodle in the section "Assignments". Do not forget to comment your code. This first project accounts for 3 points in the final grade of the course. Your project can be handed behind schedule, but your group will get a penalty of -0.25 for each day late. For instance, a project worth 2/3, but handed on March 23 (1 day late), will get a grade of 1.75/3.

5 Practicalities

Consider the following practical details concerning the implementation:

1. Traps can be triggered when drawing a 0 from the dice. For instance, if you use the normal dice and get a 0, while standing on a trapped square, you will trigger the trap of the square you are standing on.
2. There is no "cascade triggering" of traps: if you trigger a trap that makes you teleport backwards, you ignore the potential presence of a trap on the square you got teleported to. Note that, however, if you play with the normal dice, you can still trigger the trap next tun by drawing a 0 (see point 1).
3. Penalty traps situated too close to the beginning (on square 2 or 3) that cannot make the player move 3 squares backwards, simply teleport the player on the first square like a restart trap would. This rule also applies in the case of **circular** boards (`circle = True`).