

Markov Decision Processes

Project guidelines

Professor: Marco Saerens

Teaching Assistants: Diego Eloi and Nicolas Szelagowski

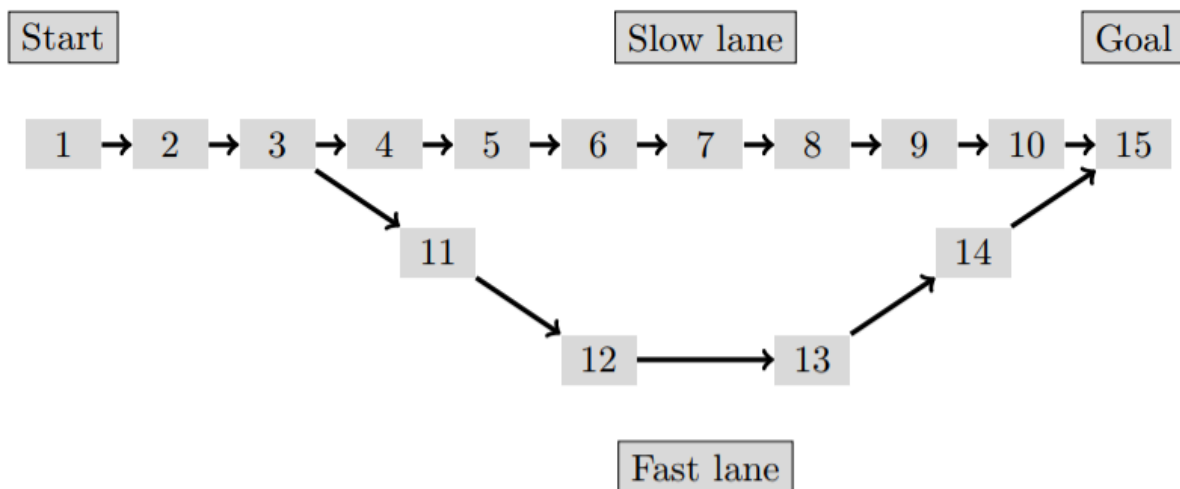
1 Objective

The objective of this project is to put into practice some of the techniques introduced in the data mining and decision making lectures. This is done through the study of a practical case which requires the use of a scientific programming language related to the statistical processing of data. This work aims at applying algorithms (mainly the value iteration) 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 3 students (2 students only if 3 is not possible). You are asked to solve 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 (goal). If you reach square 15, you win. The board can contain traps that slow down your progression if triggered and bonuses that can help you to reach the destination faster. A schematic representation of this game is shown below:



At any time during the game, to move forward, you can choose between three dices:

- The “security” dice which can only take two possible values: 0 or 1. 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 you ignore the presence of traps when playing with the security dice – you are immune to traps and bonuses (if any).

- The “normal” dice, which allows the player to move by 0, 1 or 2 squares with a probability of $1/3$. If you land on a trapped square using this dice, you have 50 % chance of triggering the trap/bonus.
- The “risky” dice which allows the player to move by 0, 1, 2 or 3 squares with a probability of $1/4$. However, when playing with the risky dice, you automatically trigger any trap or bonus you land on (100 % trigger probability).

At square 3, there is a junction with two possible paths. If the player just passes through square 3 without stopping, he continues to square 4 or beyond (5, 6, etc), as if the other path does 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 $1/2$. Both paths ultimately lead to the final, goal, square.

You should also define “trap” squares and “bonus” squares on the game, that can be triggered when stopping exactly on that square. Recall that the player is only exposed to traps and bonuses when drawing the normal or the risky dice. Trapped squares are of three different types. Bonus squares (only one type) allow the player to play a second time in a row, without having to wait one turn. Concerning the traps, there are different types of trap, each one 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 backward.
- Type 3 – Prison: The player must wait one turn before playing again.

Note that the first and final squares cannot contain a trap or a bonus.

We ask you to compute 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 cost represents 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, goal, 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 land on or overstep the final 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 “bonus” trap (play again, without waiting next turn)

- **circle**: a boolean variable (type `bool`), indicating if the player must land exactly on the final, goal, square 15 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 two 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 goal square. The vector starts at index 0 (corresponding to square 1) and ends at index 13 (square 14).
- **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 and 3 for “risky”), excluding the goal square. Again, the vector starts at index 0 (square 1) and ends at index 13 (square 14).

Please, follow exactly these guidelines because your code will be evaluated by executing it a large number of times with different trap configurations and computing the number of successful executions (providing the right policy).

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, as well as for the different scenarios (circle or not), depending on your time.

Finally, if you have some time left, and thus *as a bonus*, feel free to investigate the Q -learning reinforcement learning method on the same problem. What are your conclusions, in comparison with the use of the standard MDP value iteration?

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) and group number. You are asked to write a report (PDF) in English of maximum 6 pages (everything included, but may contain appendices). This report must have a professional look & feel, like a scientific or a technical report. Therefore, your report can integrate plot (if pertinent), but no screenshot of equations and/or screenshot of code outputs. Do not forget to provide references for your sources of information¹ and be consistent in your notations.

Your report must at least contain:

- a short introduction;
- a brief theoretical explanation of the method used to determine the optimal strategy regarding the choice of the dice;
- a short description of your implementation²;
- for various trap layouts (trap squares at relevant places), comparisons between theoretical and empirical results for the optimal strategy as well as between the optimal and suboptimal strategies. Analyze and discuss your results.
- a conclusion.

Your project will be evaluated on the following aspects:

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

Important: Beware that the correctness and 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 and expected costs. It is therefore important that you strictly comply with the syntax of the function (name, inputs and outputs). Moreover, your code must be a .py, not a Jupyter notebook.

This report must be uploaded before March 30, 2025, 23:55, together with the code (all files

¹A good reference available online is the book of Sutton and Barto (2018) “Reinforcement learning: an introduction”, second edition.

²Do not go into details about your code itself. If you still wish to provide details about the code and the functions of your implementation, do not hesitate to integrate a ReadMe file in your zip, but your report should not focus on the code.

zipped together) on Moodle in the section “Assignments”. Do not forget to comment your code. This first project accounts for 5 points in the final grade of the course. Your project can be handed behind schedule, but your group will get a penalty of -0.5 (on 5) for each day late. For instance, a project worth 4/5, but handed on March 31 (1 day late), will be graded 3.5/5. Submissions will no longer be accepted after 4 days late.

5 Practicalities

Consider the following practical details concerning the implementation:

1. Traps and bonuses can be triggered when drawing a 0 from the dice. For instance, if you use the normal or the risky dice and get a 0, while standing on a trapped square, you (may) trigger the trap of the square you are standing on.
2. There is no “cascade triggering” of traps: if you trigger a trap that teleports you (type 2 or 4 traps), you ignore the potential presence of a trap/bonus on the square you got teleported to. Note that, however, if you play with the normal or risky dice, you may still trigger the trap next turn by drawing a 0 (see previous point).
3. Penalty traps (type 2) 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 (type 1) would. This rule also applies in the case of **circular** boards (`circle = True`).
4. When landing on a bonus square and triggering the bonus, the player is free to select any dice to play with (the player is not forced to roll the same dice again).

6 Some references

- Bertsekas & Tsitsiklis (1996) “Neuro-dynamic programming”. Athena Scientific. (chapter 8 describes some applications).
- Brandimarte (2021). “From shortest paths to reinforcement learning: a Matlab-based tutorial on dynamic programming”. Springer.
- Kochenderfer, Wheeler & Wray (2022) “Algorithms for decision making”. MIT Press.
- Powell (2011) “Approximate dynamic programming”, 2nd ed.”. Wiley.
- Powell & Ryzhov (2012) “Optimal learning”. Wiley.
- Puterman (1994) “Markov decision processes”. Wiley. (discusses applications in management sciences).
- Russell & Norvig (2022) “Artificial intelligence: a modern approach”, 4th ed. Prentice-Hall.
- Plaat (2020). “Learning to play: reinforcement learning and games”. Springer.
- Plaat (2022). “Deep reinforcement learning”. Springer.

- Sutton & Barto (2018) “Reinforcement learning: an introduction”, 2nd ed.”. MIT Press.
- Szepesvari (2010) “Algorithms for reinforcement Learning”. Morgan and Claypool Publishers.

Appendix: Common pitfalls

For your information, here are some common issues that frequently appeared in the the last years corrected reports. This list is for all types of reports, asked in several different courses; therefore some of these remarks might not apply in your special case.

- Please follow strictly the instructions. For instance, if we ask a theoretical description of the models, do not forget to include it. If it is asked, it means that we assign a mark on it.
- The report should be written in the same way as a scientific report or paper and the submitted version must be in PDF.
- Also, if we describe the structure of the procedure(s) or function to be implemented in the code (signature, etc), or the name of the files that have to be delivered, in the instructions, please follow precisely these guidelines. We usually write a program that automatically tests your procedure or reads your file; if the guidelines are not applied correctly, it crashes. It results in a waste of time and penalties...
- No copy/paste of equations or algorithms (images) from the web or from any other source. This is non-professional. Also, add a reference to the sources in the caption or the text if you copy a figure.
- In the theoretical part, be coherent in your mathematical notation (use consistently the same notation and style for the same quantity in all the equations and the text).
- Clearly explain the intuition behind the main equations.
- The look & feel of the report must be professional, like a scientific or a technical report.
- Do not forget the references / sources in the explanations – any claim must be justified by references.
- Do not cite fancy web sites in the references if there are good reference textbooks or scientific papers covering the material. Web sites are changing all the time and are themselves copying their material from more reliable sources like technical books, technical or scientific reports. Moreover, a good textbook uses the same notation and conventions all along which is more convenient.
- Use exactly the same format and structure for all your bibliographical references
- Clearly explain the validation methodology when evaluating the models or comparing them. The procedure should be sufficiently clear and detailed so that it could be reproduced by the reader.
- Provide a clear presentation and discussion of the obtained experimental results.

- For the tables and plots of the results, indicate the quantities that are reported as well as their units in the caption.
- The explanation of the detailed code (with copy/paste of code examples) in the report should not be too long (the code is anyway provided). The algorithm(s) could be explained in detail, but then in the theoretical section.
- Do not forget to comment your code.

Good work !!
