LSINF2275 - Data mining and decision making

PROJECT 1

Markov Decision Processes

Radia El Amraoui (88361900 DATS2MS/G)
Adrien Delhez (16641700 MAP)
Nathan Cloos (19761700 MAP)

April 5, 2021

1 Introduction

A Markov decision process (MDP) is a classical and very powerful tool in stochastic decision making. The idea is that there is an agent interacting with some environment through actions, and as a consequence of those actions, the state of the environment changes and the agent receives an associated cost. What makes the task of finding the optimal strategy rather difficult is that the environment can be stochastic and that the current action can have consequences on the future. This means that we have to reason in terms of probabilities and formulate the objective as minimising the total expected cost. The expectation is necessary because only scalar functions can be optimized, and the sum of costs is taken so as to account for the future costs resulting from the current action. With the assumption that the state obeys the Markov property, it is possible to solve this problem in an iterative way, using for example the value iteration algorithm as we did in this project.

The environment that we will consider is a Snakes and Ladders game where the goal is to reach the final state in the least amount of turns as possible. We will start by giving an overview of our implementation of the game as well as the value iteration algorithm. Then, we will compare the total expected cost obtained theoretically with estimates of this cost obtained by taking an empirical average over a large number of simulations. More precisely, we first consider a game layout with not traps and a constant dice strategy and then, we compare the optimal policy to other sub-optimal policies for layouts containing traps. We will also analyse and visualise the optimal policy in details. In particular, we will see how adding a trap to the slow lane affect the optimal policy.

2 Implementation

In this section, we give an overview of the different parts of our code and their structure, without going into the details of the implementation. There are two main parts, one for computing numerically the optimal solution of the MDP, and the other to generate estimates from simulations of the game.

2.1 Theoretical part

In this part of the code which is concerned with the computation of the optimal solution, we have a class `SnakesAndLaddersProb` and the function `markovDecision`. We follow the notation of the book of Sutton and Barto (2018) "Reinforcement learning: an introduction" (second edition), where for example the objective is to maximize the total reward, instead of minimizing the total cost. In the class `SnakesAndLaddersProb` we implemented the transition probability function $p(s'|s,a)$, which gives the probability of the next state s' given the current state s and the action a for the Snakes and Ladders game. Note that in this game, the state is the number of the square on which the player is and the action is the dice that is chosen. This class provides as well the expected reward associated with such a transition.

The value iteration algorithm is a method from dynamic programming to compute the optimal value function, denoted v^*. It consists in iterating the following update equation to progressively improve the estimated value function V.

$$\begin{aligned} V(s) &\leftarrow \max_a \mathbb{E}[R_{t+1} + V(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s'} p(s'|s,a)\big[r + V(s')\big] \end{aligned}$$

Once the optimal value function is well enough approximated[1], $v^* \approx V$, the optimal policy can be computed in this way:

$$\pi^*(s) = \arg\max_a \sum_{s'} p(s'|s,a)\big[r + V(s')\big]$$

where $\pi^*(s)$ is the optimal action for the state s.

[1] In practice we can stop the iterations when the increments in the update equation are smaller than some tolerance, e.g. 10^{-9}.

2.2 Simulation part

This part contains a class `SnakesAndLaddersSim` that implements the dynamics of the game, a bunch of agent classes for different types of strategy, and a function to run simulations. We deliberately tried to implement the class `SnakesAndLaddersSim` used for the simulations as independently as possible from the class `SnakesAndLaddersProb` used to compute the transition probabilities. For example, we decided to assign the two implementation tasks to two different members of our group. This allows us to have some degree of certainty concerning their correctness in the case where the probabilities estimated through simulations match the manually specified probabilities.

The function to run simulations takes an agent and simulates the interaction of the agent with the game. It produces transitions that can be used to approximate empirically the value function or the transition probabilities for example.

3 Results

3.1 No traps and constant dice

In order to check that the results obtained theoretically and empirically match, we start by comparing them on a simple case. The layout of the game doesn't contain any traps and the agent plays always the same dice. The figure 1 shows that there is indeed a very good matching between the two results. Observe that when circle is true, there is a small discrepancy for the risky dice as the total expected cost is a bit underestimated by the empirical estimation. It is understandable that in that case the value of the states is more difficult to estimate empirically since with the risky dice, the agent can circle many times before stopping right on the final square.

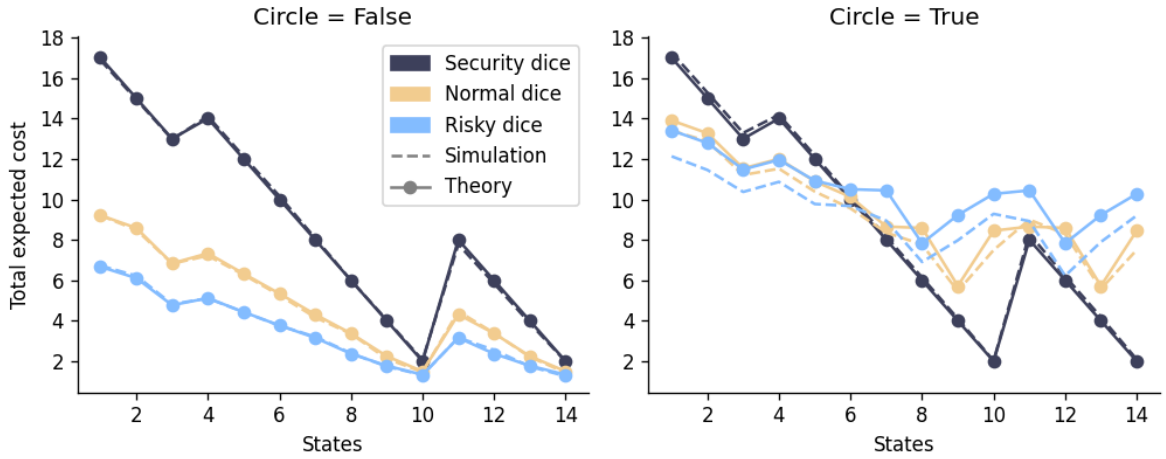

Figure 1: Comparison of the total expected cost for each state obtained by value iteration with the estimates obtained empirically from 1000 simulations, for an agent that is always playing the same dice. The layout of the game doesn't contain traps.

3.2 Analysis of the optimal policy

The figure 2 shows a visualisation of the optimal policy obtained by value iteration for different configurations of the game. The links between the squares have been omitted for simplicity but they are exactly the same as in the visualisation of the game provided in the project statement. The shape of the boxes determine whether circle is set to false or to true, i.e. square if false and circle if true. The letters in the boxes indicate the dices chosen by the optimal policy. Finally, the traps are represented by a colored edge around the boxes

where the color corresponds to the type of the trap (see the legend on figure 2).

Let's start with the two configurations on the top of figure 2. They contain no traps and the one on the left does not circle while the one on the right does. As expected, the optimal policy when circle is false is simply to always choose the risky dice to move as quickly as possible to the goal state. When circle is true, we see that it slows down near the goal state in order to not overstep it and restart from the beginning. Three squares away from the goal, it still plays the risky dice as the maximal displacement is three and so it is not possible to overstep it. Two squares away from it, it plays the normal dice and one square away, it plays the security dice. It has indeed found the optimal strategy that completely prevents from overstepping the goal.

Then, let's look at the four configurations at the bottom of figure 2, where circle is set to false for all of them. The idea of this experiment is to see which trap, placed on the slow lane, will make the agent slow down in order to reduce the probability of triggering the trap. We start by considering the case where a restart trap is placed on the slow lane just before the goal state (Config 1 - dark grey trap). We observe that the there is a slow down just before the trap:

- Three squares away, the agent plays the normal dice to avoid making a three and triggering the trap.
- Two squares away, surprisingly it is better to play the risky dice even though there is probability of 1/4 to trigger the trap.
- One square away, it is the safest option that is optimal as the agent plays the security dice which do not trigger the restart trap.

Note that it is possible to completely avoid the triggering of the trap by taking the security dice when the agent is two squares away from it, but these results tell us that it is not the optimal strategy, which may seem at first a bit counter-intuitive. Another interesting observation is that the agent also slows down at the beginning so as to increase its chances to take the fast lane and avoid the lane with the trap. It starts with a normal dice in square 1 and then a security dice in square 2 to make sure that it will step on square 3, at the junction of the two lanes. Acting this way maximises its probability to take the fast lane and avoid the high cost due to the restart trap. In the case of a penalty trap (Config 2 - blue trap) or a gamble trap (Config 4 - orange trap), the optimal agent only slows down by taking a normal dice instead of a risky one when it is three squares away from the trap. There is no endeavour to take the other lane. Finally, the prison (Config 3 - green trap) placed on the slow lane don't effect the optimal policy as it remains the same as in the case with no traps, i.e. always choose the risky dice.

3.3 Comparison with sub-optimal policies

We now compare the total expected cost obtained with the optimal strategy against other sub-optimal strategies. We consider two different configurations, each has one trap of each type placed randomly and circle is false for the first one, true for the second. These configurations and the corresponding optimal policies are represented on the top of figure 3. The two plots below reports the total expected cost estimated with an empirical average over simulations for various policies. There is also a dashed red line showing for the optimal policy that the total expected cost computed theoretically with value iteration is the same as the empirical estimation.

As we have just seen, the strategy of always taking the risky dice can still be optimal in the presence of traps. This is confirmed here where it is the optimal policy for Config 1. We can also observe on the bottom left plot that a purely random strategy, where each dice has an equal probability to be chosen, has a total expected cost that is very similar to the one obtained with a constant normal dice strategy. When circle is true as in Config 2, the ranking of the policies changes a lot. As we can see on the bottom right plot, it is now the security dice that is the best among the sub-optimal policies. The reason is that it is impossible to circle with this policy. The optimal agent knows when it is safe to play the more risky dices by taking into account the layout of the game, which is why it achieves the best total expected cost.

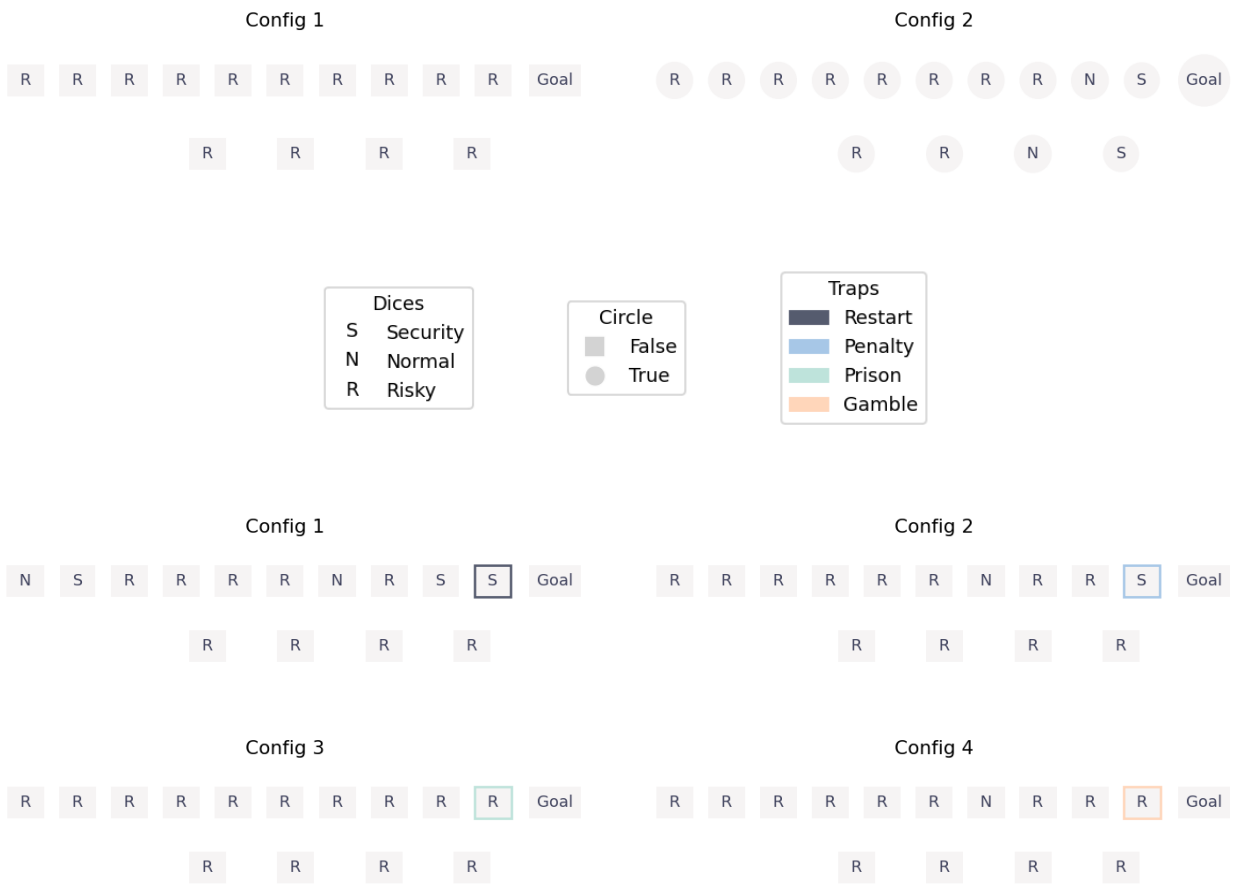

Figure 2: Visualisations of the optimal policy for different game configurations. The links between the boxes have been omitted for simplicity but there are exactly the same as the visualisation of the game provided in the project statement. The letters in the boxes indicate the dice, their shape indicates whether circle is true or not, and a colored edge corresponds to a trap.

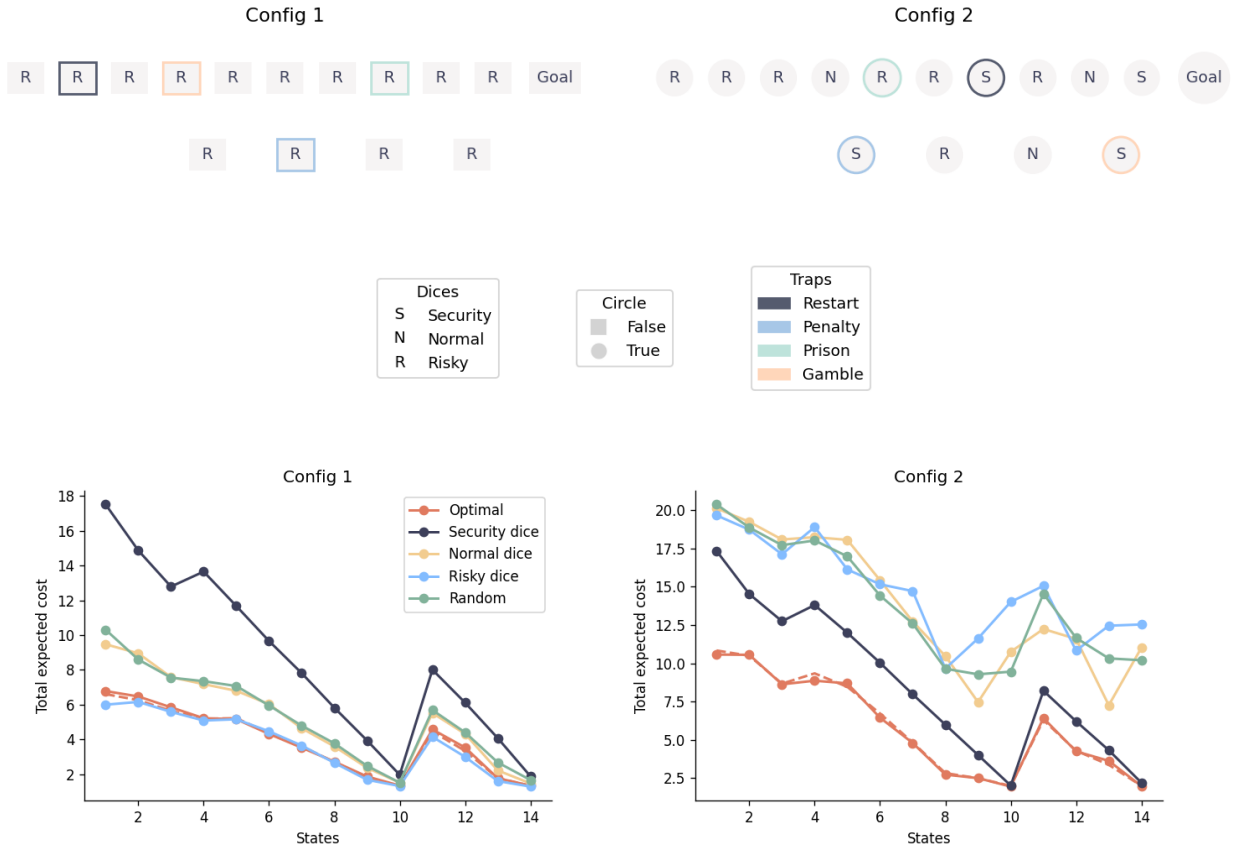

Figure 3: On the top, two different game configurations and the corresponding optimal policies. On the bottom, comparison of the empirical total expected cost for different policies, in each of the two configurations. The estimates are obtained from 1000 simulations. The dashed red line shows the total expected cost obtained theoretically with value iteration for the optimal policy. Note that the colors of the traps are not related to the colors of the different policies.

4 Conclusion

In this project, we implemented and solved a Markov decision process where the agent found the best strategy in a Snakes and Ladders game to reach the final state in the least amount of turns. We compared the total expected cost obtained theoretically with the value iteration method, and found that it is consistent with empirical estimates obtained from simulations of the game. By analysing the optimal policy, we observed that the agent is able to take into account the configuration of the game, i.e. the position of the traps and whether it circles, in its strategy. For example, we saw that when a restart trap is positioned on the slow lane just before the goal state, the agent slows down at the beginning to increase its probability to take the other lane and avoid the trap. It is an example where current actions can influence the future and that sometimes it is better to sacrifice a few turns at the beginning to achieve a lower total expected cost. We also saw that it can be difficult to intuitively reason in terms of probabilities as it was not at all obvious that taking the risky dice two squares away from that restart trap placed just before the goal state, is indeed the optimal strategy. A Markov decision process is very useful tool to tackle such problems where our intuition reaches its limits.