Data Mining and Decision Making

LINFO2275

Markov Decision Processes

Group 4

Grognard Simon, 37811700, INFO

Guerrero Cano Juan Valentin, 36092200, EPL- IN

Marques Mathias, 13181700, INFO

February 21, 2024

1 Introduction

Before explaining how the project has been developed, we should first make a quick introduction of what is "Markov Decision Processes".

We should introduce the **stochastic dynamic programming**, which is a technique for modelling and solving problems within the context of decision making under **uncertainty**. Specifically, this technique deals with problems in which the current period reward and/or the next period state are random. And trivially, the decision maker's goal is to maximise the expected reward.

Therefore we can define **Markov decision processes** (MDP) as a representation of a special class of stochastic dynamic programs where the stochastic process is a stationary processes that exhibits the **Markov property**; which refers to the memory less property of a stochastic process (in the field of probability theory and statistics). We will explain more precisely later.

So, the objective of this project is to implement one empirical case of the mentioned MDP; the Snakes and Ladders game. We will determine the optimal strategy regarding the choice of the dice using the "value iteration" method.

2 Theoretical Explanation

In order to solve our problem, the Snakes and Ladders game, we should prove first that it satisfies the assumptions needed for a problem being solve using an algorithm for MDP (in our case **value-iteration method**).

A problem could be classified as a MDP when it represents situations where outcomes or reward are partly random and partly under the control of the decision maker. Lets give a more rigorous definition of this assumptions:

- The problem should have a finite or countable set of **states**, which in our case are the possible squares on the board.

- The problem should have a finite or countable set of **actions**, that can be taken by the decision maker in each state. In our case the possible actions the choices between the dices.

- The problem should have a **reward function** that allow us to establish a preference between states and actions.

- The problem should have a **state transition function** that describes the probability of transitioning from one state to another given a particular action.

- The problem should satisfy the **Markov Property**, which holds that the current state of the game should be enough to determine the optimal actions, or in other words; that the probability distribution over future states must be independent of past states. It is clear that our problem satisfy this property as the choice of the dice does not depend on past states or past actions

Before explaining how have we applied the value-iteration method, we need to present some notation to understand the following equations.

- Set of states: S={1,2, ..., n}. We will use the notation: $s_t = k$ that means that the process is in state k in at time t.

- Set of actions: in each state we have a set of available actions (in our case all the possible actions for each state) which is denoted as U(k). So $a \in U(k)$ denotes an action available for state k.

- When we choose an action $a = u(s_t)$ at a time t and a state s_t, a bounded cost is generated $0 \leq c\left(u\left(s_t\right) \mid s_t\right) < \infty$, and the systems jumps to state $s_{t+1} = k$ with a probability mass of $\mathrm{P}\left(s_{t+1} = k' \mid s_t = k, u\left(s_t\right) = a\right) = p\left(k' \mid k, a\right)$

Explaining this notation we are able to continue with the theoretical explanation:

Once we have proved that our problem is a MDP, we can expose our objective, which is find the **optimal sequence of actions** (policy or strategy) to reach the final state (obviously taking in count the cost of an action given the current state). In order to do this we will try to minimize the total expected cost (1) until we reach the final state, which is a absorbing state, where we stay "forever" or we assume the game concluded.

$$V_\pi\left(s_0 = k_0\right) = \mathbb{E}_{s_1, s_2 \ldots}\left[\sum_{t=0}^{\infty} c\left(u\left(s_t\right) \mid s_t\right) \mid s_0 = k_0, \pi\right] \tag{1}$$

Therefore, we have to determine **the best policy** π^* that minimizes the equation (1). Thanks to some recurrence relations we can compute the optimal expected cost, and so, use the value-iteration algorithm (2). It is remarkable that in our problem, the total expected cost is actually the total expected turns to reach the final state from a predefined state k.

$$\begin{cases} \hat{V}(k) \leftarrow \min_{a \in U(k)} \left\{ c(a \mid k) + \sum_{k'=1}^{n} p\left(k' \mid k, a\right) \hat{V}\left(k'\right) \right\}, k \neq d \\ \hat{V}(d) \leftarrow 0, \text{ where } d \text{ is the destination state} \end{cases} \tag{2}$$

Some calculus in this equations lead us to the **Bellman optimally conditions**, which should be verified on all states different from the final state. The following equation (3), resulting from these calculus, gives us the value of the optimal dice to throw for each state k :

$$V^*(k) = \min_{a \in U(k)} \left\{ c(a \mid k) + \sum_{k'=1}^{n} p\left(k' \mid k, a\right) V^*\left(k'\right) \right\} \tag{3}$$

We will explain more specifically how have we implemented this algorithm on the next section.

3 Implementation

To implement the Markov Decision Process and the value-iterative algorithm, we used different functions that allowed us to code the board, the behaviour of the decision maker and also reach the Bellman recurrence; to find the number of expected turns to reach the destination square, and the best possible action for each square.

3.1 markovDecision function

The $markovDecision(trap, circle)$ function is the main function in our project, where we call the rest of the auxiliary functions we created to solve the game. This method takes as argument the layout of the map, "trap" and the boolean "circle" that allows us to know if the player wins by overstepping the final state or not.

In this function we define the possible dices and for each of them we call the function $proba(dice, trap, circle)$. This method compute the transition matrix for a certain dice. As we know, we will use this matrix later to make the equation 2 converge with the Bellman recurrence.

After, we also define a function $getreward(trap)$ that will return a matrix of size (3,15) for the dice and the size of the board. It take in argument the layout of the board. The whole matrix is zeros except for some case where it exist the trap 3 which means that we stay on the same case. At this moment the value of the matrix at the index dice 2 and trap 3 will be 0.5 and for the dice 3 it will be 1.0. This means that when we reach this case we have a probability to activate the trap of 0.5 or 1.0.

Finally, we call the second main function of the project, $bellman(k, U, reward, trap, de)$. This function allow us to apply the recurrence to the equation 2 and find the expected turns and the best dice for a state k (in our problem the square k). The arguments are the state where we start, k, the set of the 3 different confusion matrix, (which indeed is a 3-dimensional matrix), U, the reward matrix explained before, reward, the layout of the game, trap, and the three possible actions (in our case the three possible dices), de.

3.2 bellman function

This function is also a key part of the project. Here, by defining a threshold to get the convergence of the equation 2, we are able to compute the best expected value (optimal number of turns to reach the end), and the dice with best expected value for every state (without the goal one).

We apply to every square in the layout, all the possible actions (choosing one dice) and compute the cost of applying the action to the given square plus the reward. This reward is always 1 because it cost one turn. Sometimes, this reward can be greater than 1 if we have the trap 3 so we sum the probability to reach this case coming from state 's' with the corresponding index in the reward matrix. After, we storage the best expected value, and the best dice for each of the 14 squares. We repeat this loop until reach some convergence, what actually means that the difference between the results from two iterations is lower than the predefined epsilon.

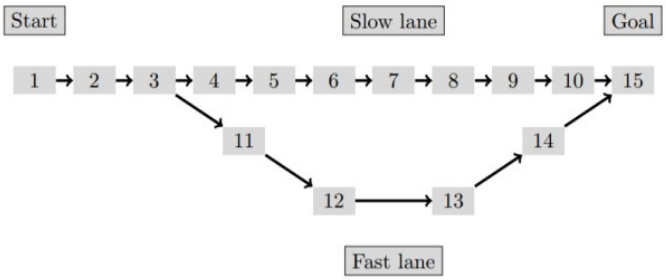
3.3 Auxiliary functions

As we mentioned before, it is quite important the role of the transition matrix. To compute it we used for each of the possible actions, one auxiliary function that allows us to modify the initial values of a predefined transition matrix by taking in count the possible traps in the map. These functions are $addtrapdeux(trap, proba)$ and $addtraptrois(trap, proba)$.

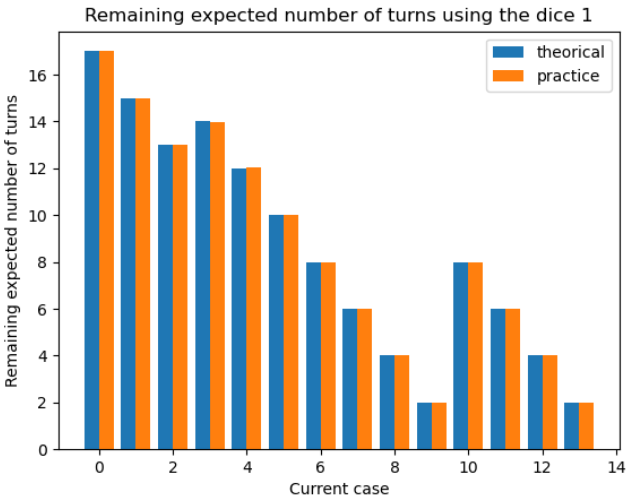
Also we used a function $Map()$ in order to create different layouts with traps in different positions to test the implementation of the MDP.

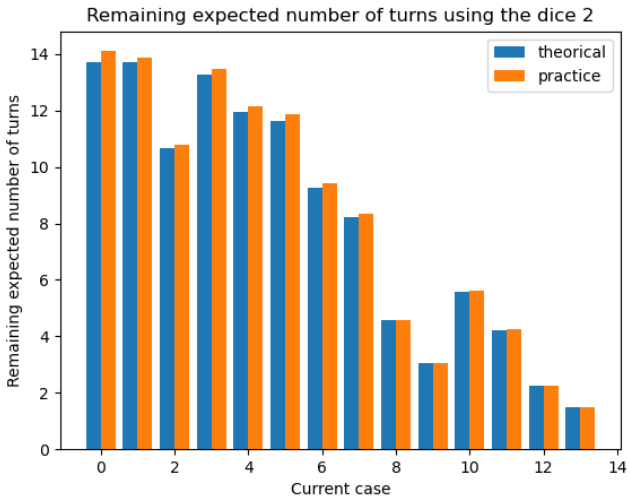
4 Results and Analysis

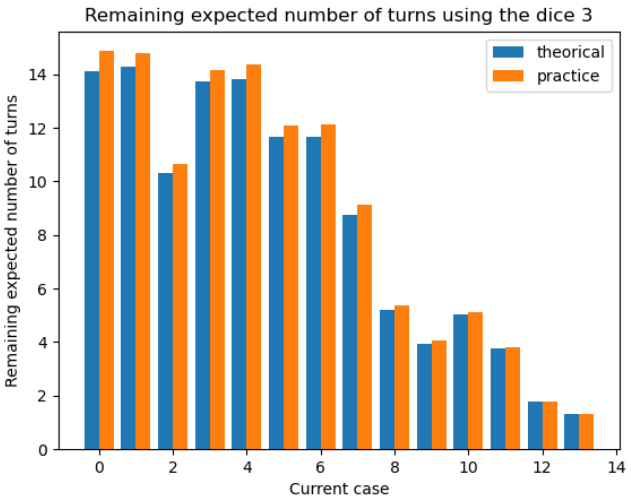
All of the tests described below were conducted using a specific map. The map features a trap located on square 8, which we refer to as trap 1. Additionally, there is a trap on square 10 (trap 2), a trap on square 2 (trap 3), and a trap on square 11 (trap 4).

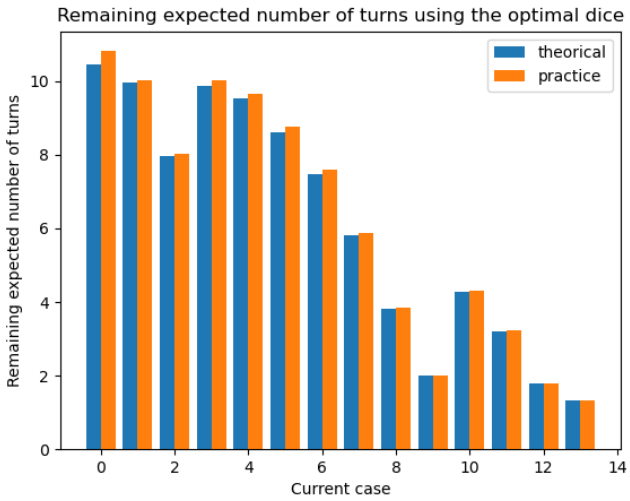


After implement the Markov Decision Processes we used an auxiliary file to test it with different strategies and different situations. In the following graphs we have used the expected number of turns as axis of ordinates and the position of the player as abscissa.



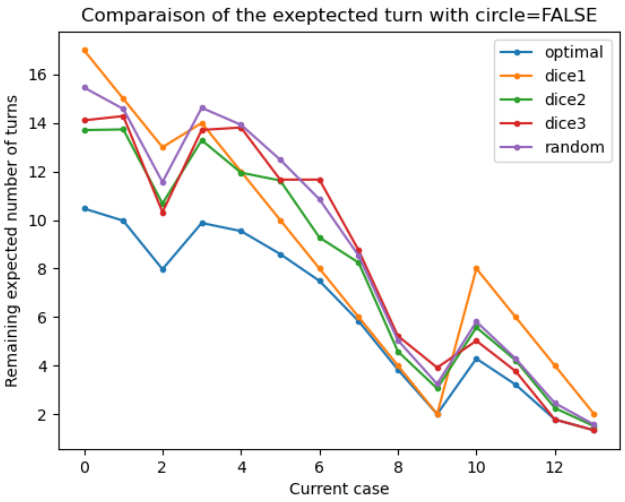


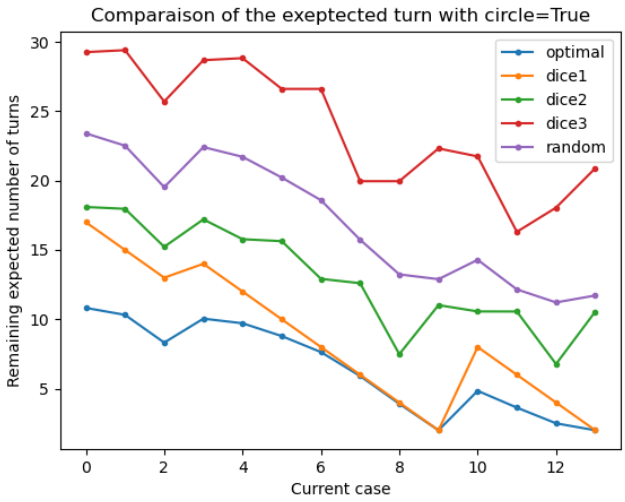




Unsurprisingly, we can see in Figure (a) how the theory match perfectly the practice, as for the dice 1 there is not a lot of randomness (only the dice 0 or 1). We can check that for dice 2, Figure (b), and for dice 3, Figure (c), the theory and the practice are almost perfectly matched. This mean that our implementation get the expected results according to the theory.

In the following images, it is clear that when we compute the expected number of turns when the boolean "circle" is false we get lower values, as just overstepping the final state means win and end the game. Instead, looking at the graph for the case of the boolean circle being true, the expected number of turns needed to end the game is quite high. It is also remarkable that for the dice 3 is the worst dice when circle it's True because you can overtake easily the solution. The dice 1 is close to the best solution when approaching the final position because our algorithm will say to take the dice 1 to avoid overtaking the final square.





Taking in count the different results obtained and the theory behind this problem, we can ensure that our implementation match nicely what the theory holds.

5 Conclusion

After regarding carefully the theory behind Markov Decision Processes and the results obtained with our implementation we can ensure that with a well-defined problem which satisfy the Markov property and the conditions predefined above, the decision maker is able to get some optimal policy. This does not mean that we can always win by following this policy as there is some randomness involved in the problem. However, in the most part of the cases we can ensure that the optimal strategy obtained will be the best.