UCLouvain
Faculté des sciences

Université catholique de louvain
Louvain School of Statistics

LSINF2275
Data mining and decision making

Group Project 1
Markov Decision Processes

Aurèle Bartolomeo - 2889-1700 (DATS)
Arthur Geeraerd - 1175-1600 (DATS)
Lionel Lamy - 1294-1700 (DATS)

April 5, 2021

Contents

1 Introduction

The aim of this project is to study and implement the value iteration algorithm in order to solve Markov Decision Processes (MDP). Our framework will be the well known “Snakes and Ladders” game. It will have the particular shape that is represented in Figure 1. That is, it will be composed of 15 squares including a slow and a fast lane (the player can take the fast lane with probability 1/2 if he land on square 3). Our main objective will be to find the best strategy to finish the game in an expected number of turns that should be the smallest. After having computed that, we will compare the cost of this theoretical optimal strategy and verify if it corresponds to the mean cost that we obtain by simulating a large number of games using that strategy. Furthermore, we will also verify that indeed other strategies are sub-optimal (meaning that the number of turns to finish the game is expected to be superior that the optimal one).

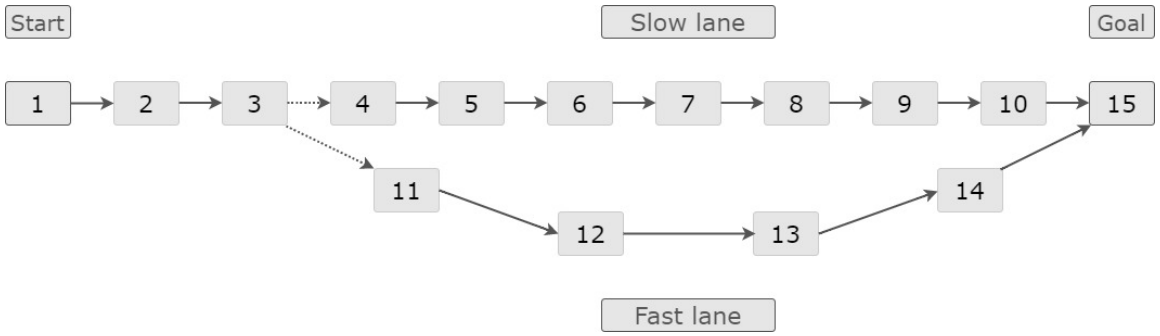

Figure 1: The game board (displayed without any trap or strategy)

1.1 Traps

Obviously, a game of this type is much more exiting if we add some traps to it. Some squares will therefore not be standard cases but traps of 4 different types. The first type is called a “restart” trap and will teleport the player back to square number 1. The second type imposes a “penalty” to the player, teleporting it three squares backward. The third trap type is a “prison” that will force the player to wait one turn before playing again. Finally, there will be the “gamble” trap, that will teleport the player with equal probability anywhere on the board.

1.2 Dices

At each turn of the game, the player will have the choice between three different dices that are listed below :

- the “security" dice. It takes value 0 or 1 and allows the player to ignore traps completely.
- the “normal" dice. It takes value 0, 1 or 2. If the player falls on a trapped square using this dice, it has a 50% chance of triggering it.
- the “risky" dice. It takes value 0, 1, 2 or 3 and automatically triggers the trap if there is one on the destination square.

For dice number i, each of its values have a probability $\frac{1}{n_i}$ to happen, where n_i is the number of different values of the dice.

2 Value Iteration Algorithm

In order to develop the intuition behind the algorithm, one need some notation.

- Let $D(k) \in \{Security_{dice}, Normal_{dice}, Risky_{dice}\}$ be the three actions available at each state (square) of the process. $d(k)$ denote the dice rolled at node k. One could also denote d(k) = a, that is, the action played at the square k.
- Let k be the node (square) where the player is and k' the node reached from k at the next step.
- Let $\hat{V}(k)$ be the expected cost at state k.
- Let n be the number of squares on the board. In our case n = 15.

As explained before, the player has a set of dices he can roll in order to reach the goal state (the last square of the board) at lower cost. Those different dices correspond to the set of actions available for the player at each square on the board. Also, we have to deal with different dices that sometimes produce undetermined results, the squares on the board may be trapped and starting from the idea that the player want to follow the set of actions that minimizes his cost function (the policy), we rely on the *Value Iteration Algorithm* to minimize the player's expected cost function that is such as :

$$\hat{V}(k) \longleftarrow \min_{a \in D(k)} \{c(a|k) + \sum_{k'=1}^{n} p(k'|k,a)\hat{V}(k')\} \tag{1}$$

Where $c(a|k)$ corresponds to the direct cost of playing the action a in state k. As opposed to the direct cost that is deterministic, there is a undetermined cost that depends on the next state, k'. For the purpose of being able to deal with the uncertainty we compute the probability of moving from state k to each state of the set of state, k' playing action available at node k and we multiply this expected transition's probability by the cost function's value of being at state k'. Also, one can see that there is no equality sign between both expressions. The left arrow indicates that V-value is founded by iterating the expression t times in order to make the V-value converging to its true value. Once a certain threshold is reached, one end up with the optimal set of action that yields to minimize the cost function. Obviously, there are some limits to this process. One needs to know the rewards/cost of ending up at each state k as well as the transition probability between k and k'.

More intuitively, the idea behind this equation is to take the direct cost of playing a certain dice, $d(k)$, when the player is on a certain square, k, on the board. Since the player is facing an odd world the other main goal of the value iteration algorithm is to consider what will append after this throw of dice. More precisely, we calculate the probability of ending up at every square k' reachable from square k and we multiply this probability with the expected cost of ending up at node k'. Since we are dealing with a cyclic board we need to iterate until the expected cost function shows convergence. A good indicator that shows convergence is the fact that the difference between the value of $\hat{V}(k)$ at the previous iteration and the current one tends to be very small. As an indication, we have chosen a threshold value of 1×10^{-6}.

3 Implementation

The first decision we made was to think and design our implementation in an object-oriented way to make it as flexible and extensible as possible. To this end, we have built a *SnakesAndLadders* class that groups the properties (such as the game board, dice and traps) and methods of the game[1]. This way of doing allowed us to understand each part of our code, make it more readable and be able to change the game settings at will. In addition, we could easily extend our class in another class called *Simulation* which, as its name suggests, was intended to simulate games and thus verify that our theoretical expectations are indeed correct.

Another particularity of our implementation concerns the transition and cost matrices. As explained in the previous section, in the Value Iteration Algorithm we need to compute the probability of moving from a square k to k' through a certain action as well as the cost of this move. Instead of doing this in a trivial way, i.e. by constructing for each of the possible actions matrices of all the probabilities and all the costs, we decided to construct only one matrix of size $N_{15} \times D_3$, where N is the number of squares and D the set of possible actions. We then store only the moves whose probability is not zero in the form of a list of tuples $\big(k', p(k'|k,a), c(a|k)\big)$. We thus avoid having several matrices largely filled with zero.

4 Simulations and comparison

In this section, we will investigate the optimal strategies computed by our algorithm for a few trap arrangements and compare the theoretical expected number of turns with the empirical average obtained from 500,000 simulations. Then, we will focus our analysis on the comparison with several sub-optimal strategies.

4.1 Layout without trap

This is the simplest board that we could build but the best strategy is not completely trivial given its configuration [1]. In fact, if the board had been composed of an unique lane, it is quite straightforward to agree that the optimal strategy is to roll the risky dice at each turn. But in our case, we have a fast lane that can be reached only by landing on the third square. We might thus expect the player to try to reach the square and only start using the risky dice after the separation.

However, when we run our *markovDecision* function, we obtain as an optimal strategy the use of the risky dice for each of the squares. The theoretical expected number of turns returned by the algorithm for this policy is 6.67, which is also the average number of turns obtained by simulating a large number of games.

Following our initial intuition, let's see empirically what would be the average number of turns to finish the game when using a strategy consisting of using the risky dice everywhere but on case 2 (normal dice) and 3 (safe dice). Our goal here is really to force the player to land on square 3. With this sub-optimal strategy, we obtained an average number of turns of 7.27 which is indeed a bit higher of what we had with the optimal one. For comparative purposes, always using the normal or safe dice will take an average of 9.21 and 17.00 turns, respectively.

[1] Diagrams of our classes are available here

One last experiment we can do is to see what happens if we play randomly (i.e the strategy is defined such that for each case, we attribute the dice i with probability 1/3). In order to calculate that quantity, we will simulate a large (1000) number of such random strategies and for each one calculate the average number of turns to finish the game by simulations. At the end, we take the mean of all these quantities. We know intuitively with the results already calculated that the quantity should be situated somewhere between ± 6.67 and ± 17.00. By computing this, we obtained a result equal to 10.16. Note that since this is the mean of a thousand average, this number can vary greatly from one game to another.

Optimal	**Intuitive**	**Safe**	**Normal**	**Risky**	**Random**
6.67	7.27	17.00	9.21	6.67	10.16

Table 1: Empirical average number of turns by strategy, blank layout

4.2 Circle layout without trap

A second very simple board consists in the same layout as in the previous example (no traps) but with an added constraint : the player has to land exactly on the last square. If he oversteps the goal, he is taken back at the beginning. The board is behaving now like a circle instead of a straight line. Intuitively, we expect the player to play the risky dice everywhere but on square 9 (normal dice), 10 (safe dice), 13 (normal dice) and 14 (safe dice). That is exactly the theoretical strategy that we obtain by using our MDP function.

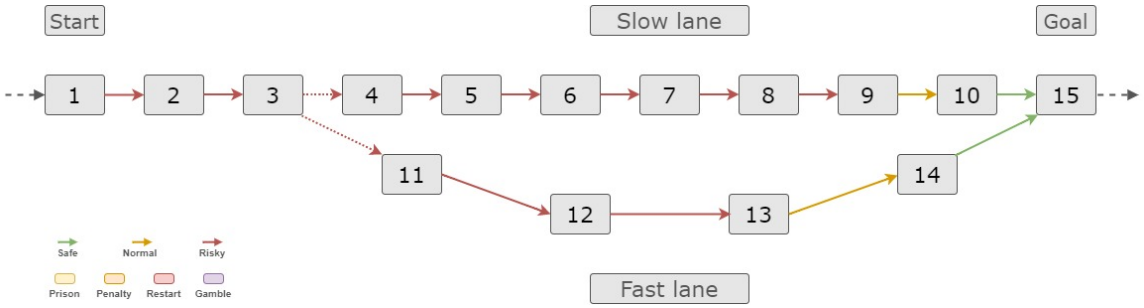

Figure 2: Optimal policy for the circle layout without trap

The expected number of turns needed to complete the game is 7.26 and is the same as what we observe when we simulate games with the optimal strategy a large number of times. Note that this number is slightly larger than in the previous example, because we are forced to slow down at the end of the board. Without doing that, meaning that we use the risky dice on every case, the cost would be of 12.35, so that would be a huge increase. We tested the same other strategies: using always the safe, the normal, the risky or random.

Optimal	**Intuitive**	**Safe**	**Normal**	**Risky**	**Random**
7.26	7.26	17.00	13.57	12.35	15.23

Table 2: Empirical average number of turns by strategy, blank circle layout

4.3 Some traps

In the previous simulation [2], we found that the algorithm did not find it particularly valuable to try to take the fast line when there was no trap. However, we would like to get an idea of how the algorithm handles the situation in the presence of traps. For this purpose, we placed some penalty traps on the starting squares as well as a prison trap and a restart trap on the slow lane.

This trap layout gave us the expected results because the optimal policy chosen by the algorithm at the beginning of the game is not to play the risky dice. Thus, there is a 50 percent chance of landing on the fast line. Otherwise, as the restart trap seems to have a huge effect, the best strategy is to play carefully on the slow lane. Note that even on the seventh case, it is important to play the safe dice because otherwise we have still a chance to trigger the trap if we get a 0 when we roll the dice. And since we are still with a circular board, the last actions are the same as in the previous one [2] and avoid going past the goal.

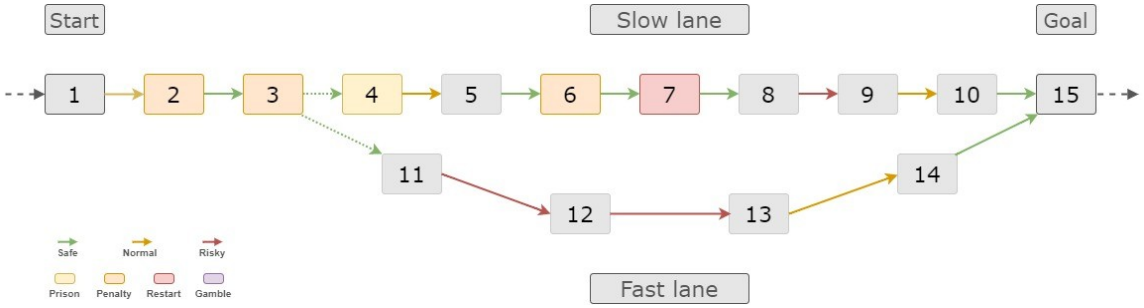

Figure 3: Optimal policy for the circle layout with some traps

The theoretical average number of turns needed to reach the goal with the optimal policy is 12.94 and corresponds to what we obtain empirically by simulating games. This cost increases when we play only the safe/normal/risky dice with respective costs of 17.00/24.62/46.32. When we compute the mean cost for 1000 random strategies, we get 29.93. As a conclusion, the risky dice is very dangerous to use at some places in this layout (especially before the restart trap) and using only that dice triples the average numbers of turns, as you can see in the table below.

Optimal	Safe	Normal	Risky	Random
12.94	17.00	24.62	46.32	29.93

Table 3: Empirical average number of turns by strategy, layout with some traps

4.4 Traps everywhere

As a last experiment we have built a board which is not circular and only composed of traps. The traps are arranged in a loop according to the following order: restart, penalty, prison, gamble and following the numbering. Note that the start and finish squares cannot be trapped. With so much traps, our guess is that playing the safe dice everywhere is actually the best choice.

However, the algorithm gives us a slightly different theoretical optimal strategy. It suggests us to prefer the normal dice on the first and fourth case. After thinking about it, the choice to roll a normal die in the first place seems interesting since we are only at the very beginning and in case of the activation of a trap, we would come back to our place and actually lose only one turn. The choice to play normal on the fourth square is less trivial to explain. We think that the gamble square is the key element. Indeed, if we fall on it and if it is activated, we still have 10 chances out of 15 to be teleported on a more beneficial square.

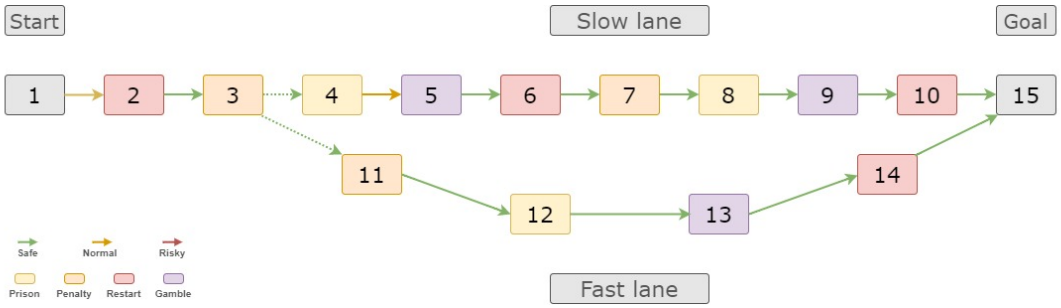

Figure 4: Optimal policy for the layout with traps everywhere

As for the previous simulations, we calculated the number of turns needed to finish the game on average using suboptimal strategies. We see that normal, risky and random strategies give very realy poor results in comparison to the optimal strategy.

Optimal	**Intuitive**	**Safe**	**Normal**	**Risky**	**Random**
16.72	17.00	17.00	32.15	40.81	31.46

Table 4: Empirical average number of turns by strategy, traps everywhere

5 Conclusion

This work allowed us to deep dive into what Markov Decision Processes are. In particular, we implemented from scratch the value iteration algorithm in a board game context. Thus, we were asked to extract theoretical material from the lessons to extend it to a real life situation. The game was “Snakes and Ladders” and the goal was to find the best strategy to finish the game in a minimal number of turns.

We then compared our theoretical results to some simulations and confirmed that indeed we had the same answer to the problem. Overall, we can assert that the simulations were instructive because our intuitive strategies were not the actual optimal ones (even though they were not too far either). Moreover, we have seen that there is a real improvement for a player to have a coherent and thoughtful strategy because the expectation of the cost of playing in a random way is really superior.

As a final word, we want to say that we are proud of our implementation of the problem as well as our results and that we would have been delighted to further our analysis. ∎