LINFO2275: Data mining and decision making

Using Markov Decision Processes to study the case of a Snakes and Ladders game

Group 9 :
CAPON Nathan (DATS)
HEROUFOSSE Gauthier (BIR)
SCHERPEREEL Colin (BIR)

Teaching :
Pr. SAERENS Marco
COURTAIN Sylvain
LELEUX Pierre

Academic year 2021-2022

1 Introduction

Markov Decision Processes (MDPs) are a kind of models generally used in optimization problems that can be solved via dynamic programming or reinforcement learning algorithms. They aim to model decision processes involving a decision maker (the agent) evolving in an environment which state changes partly randomly and partly following the action choices made by the agent. The state of the environment affects the immediate reward obtained by the decision maker following his actions as well as the probabilities of future transitions. The goal of the agent is to find the optimal policy under which the total reward will be maximized (or minimized if the reward is considered as a cost).

In the context of this project, we apply the value iteration algorithm to solve a MDP designed for the case of a Snakes and Ladders game. The objective of this implementation is to find the optimal policy, i.e. which dice to roll at which state, to reach the final square of the board in the smallest number of turns. More particularly, we are interested in comparing the theoretical expected cost (the expected number of turns to end the game) computed by value iteration with an empirical average cost using the same optimal strategy. In a second time, we also compare empirical average costs obtained with different sub-optimal strategies for different board configurations (including the presence of traps, bonuses and circular/linear layouts).

2 Methods

Markov Decision Processes

In a general perspective, MDPs are composed of the following elements:

- A set of n states, $S = \{1, 2, ..., n\}$,
- For each state $s = k$, a set of m admissible actions, $U(k) = \{u_1, u_2, ..., u_m\}$ that can be chosen by the agent,
- A bounded cost, $0 \leq c(u(s_t)|s_t) < \infty$, incurred when the agent chooses the action $a = u(s_t)$ at time t in state s_t,
- A probability mass $P(s_{t+1} = k' \mid s_t = k, u(s_t) = a) = p(k' \mid k, a)$ defining the jump of the system from state $s_t = k$ to the new state $s_{t+1} = k'$ when action $a = u(s_t)$ is taken.

In addition to these components, several assumptions can be formulated:

➢ The set of admissible actions depends on the current state k and is independent of time.

➢ In a similar way, the probability distribution defining the jump of the system only depends on the current state k and not on past states (Markov assumption).

➢ The costs are supposed non-negative so that the network of states does not contain any negative cycle.

➢ The goal state ($s = d$) is reachable from each state.

The objective of the approach is to reach the destination state when starting from some initial state $s_0 = k_0$ at time $t = 0$ with the smallest cost.

This purpose is achieved by finding the optimal policy π^* obtained by minimizing the total expected cost $V_\pi(k_0)$. The optimal total expected cost writes:

$$V^*(k_0) = \underset{\pi}{min}\,\{V_\pi(k_0)\} \tag{1}$$

The computation of the optimal total expected cost can be approached thanks to a recurrence relation derived from the value-iteration algorithm. It consists in updating the approximation of the expected cost $\hat{V}(k)$ for each state k iteratively until this value converges. The value-iteration algorithm can be described as follows:

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

Modelling a Snakes and Ladders game

The Snakes and Ladders game that we consider can be modelled with a MDP. The player is represented by an agent and his purpose is to land his pawn on the last square of the board in the fastest way possible (with a minimal number of turns). The player evolves on a board with 15 squares and has a choice of action between three dices (security, normal and risky). Each action costs one turn in the game and the probabilities of jumping from one state to another are defined by the type of dice chosen, the layout of the board (fast and slow lanes & circular/linear configuration), as well as the presence of traps and bonuses that impact the player's progression.

The purpose of the approach is to find the optimal policy, i.e. the best dice to choose at each square, as well as to compute the expected cost when starting from each square, for a given layout. The position of traps and/or bonuses, as well as the circular/linear configuration of the board are known in advance. Mathematically, the situation is represented as follows:

- S = {1, 2, ..., 15},
- For each state k, $U(k) = \{S, N, R\}$ with S, N and R being the security, normal and risky dices respectively,
- Excepted when a prison trap (the player must wait one turn, $c(u(s)) = 2$) or a bonus (the player can play again, $c(u(s)) = 0$ for one turn only) is triggered, the following statement holds:
$$\begin{cases} c(u(s)) & = 1, \qquad \forall\, s \in S \setminus \{15\} \\ c(u(s)) & = 0, \qquad \text{when}\, s = 15 \end{cases}$$
- $P(s_{t+1} \mid s_t, u(s_t))$ is defined by the choice of the dice and the layout of the board.

3 Implementation

The implementation of the previously described model is done in *Python*. The whole process consists in a 'while' loop computing the approximation of the expected cost for each square, as well as the associated policy, by successively calling 5 different functions. The iteration process is repeated until convergence is reached. More details can be found in the code itself, as many annotations and a complete description of each function, as well as of their inputs and outputs, are provided.

4 Results

In this section, we will first discuss theoretical results obtained by the Markov Decision Process after convergence. Going through different configurations, we will measure the relative weights of each modification of the board. This process starts with the simplest configurations up to the most advanced ones. Note that the optimized Markov Decision Algorithm will be named as computer in this section.

4.1 First remarks and influence of attribute *circle*

The first parameter studied is the 'circle' attribute. This parameter influences the game only on the last square of the board. The player is here obliged to land exactly on the last square to win the game, whereas in the case of a linear board he only has to pass the last square to win. Our first guess is that the algorithm will understand the danger of missing its landing, and will thus adapt its game-play to force its path to the last square.

The results of the a game with a board empty of traps/bonuses is summarized in Table 1 below. We can make some observations:

- The number of expected turns logically decrease with the progress on the board ;
- The optimal strategy for the computer is almost the same in each configuration. Because there is no trap, the player chooses to maximize its chances of moving forward by using the risky dice each time. As expected, in the circle game, the algorithm decides to secure its landing on the goal square, avoiding the risky dice when it is close to the last square ;
- The possible shortcut on the third square does not seem to be a relevant alternative for the computer in this configuration. It does not try to force its way to this path using the same strategy as mentioned above for the circle game ;
- We also notice that the average number of turns at any position is always higher in a circular game than in a linear one, which seems relatively obvious as the player has to exactly reach the goal square to win the game with this configuration.

Circular Layout			
Square	Layout	Expec	Dice
1	0	6.67	3
2	0	6.11	3
3	0	4.77	3
4	0	5.12	3
5	0	4.43	3
6	0	3.77	3
7	0	3.16	3
8	0	2.37	3
9	0	1.78	3
10	0	1.33	3
11	0	3.16	3
12	0	2.37	3
13	0	1.78	3
14	0	1.33	3

Linear Layout			
Square	Layout	Expec	Dice
1	0	7.26	3
2	0	6.69	3
3	0	5.36	3
4	0	5.71	3
5	0	5	3
6	0	4.37	3
7	0	3.78	3
8	0	2.83	3
9	0	2.5	2
10	0	2	1
11	0	3.78	3
12	0	2.83	3
13	0	2.5	2
14	0	2	1

Table 1: Results for a game with an empty board

4.2 Comparison between different layouts

In this part, we would like to identify the impact of the choice of the layout on the outputs of our algorithm. Table 2 below shows the results of the expected number of turns starting from the first square for different layouts. A few observations can be made:

- The type of square contained in the layout has a significant impact on the expected number of turns. Indeed, the first three types, which correspond to penalty cases (restart, penalty, prison), seem to rise the expected number of turns, where the fourth, which represents a bonus (play again), makes it decrease. Nevertheless, it is interesting to point out that the three penalty traps do not have the same impact. One can see that the third trap seems to be less inconvenient by looking at the result associated to layout 3 in comparison with layouts 1 & 2.
- Moreover, the results associated to layout 4 show us that the association of bonuses with the third dice is very powerful. Indeed, a series of bonuses allows the players to finish the game sometimes in only one turn !
- Layouts 5, 6 and 7 give us an insight of the type of outputs obtained with our MDP algorithm for various cases, especially when there is either no bonus, or only bonuses, and when the layout is almost full of traps/bonuses.
- Furthermore, the position of the traps/bonuses also has an influence on the output. Indeed, even though layouts 8 and 9 contain the same amount of traps/bonuses of the same kind, their expected cost is different. This is only due to the position of these squares.

Label	Composition	Expec (True)	Expec (False)
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	6.67	7.26
1	0 1 1 1 1 1 1 1 1 1 1 1 1 1 0	17	17
2	0 2 2 2 2 2 2 2 2 2 2 2 2 2 0	17	17
3	0 3 3 3 3 3 3 3 3 3 3 3 3 3 0	11.54	11.89
4	0 4 4 4 4 4 4 4 4 4 4 4 4 4 0	1.33	2.17
5	0 2 3 0 1 2 0 1 2 0 3 3 0 1 0	14.03	14.1
6	0 4 4 0 0 0 0 0 4 0 4 0 0 0	4.97	5.73
7	0 4 2 1 3 2 4 3 2 0 1 3 4 1 0	14.52	14.68
8	0 0 4 2 1 0 3 0 3 2 1 4 0 0 0	9.81	10.22
9	0 4 0 0 1 1 3 2 4 0 3 0 0 2 0	11.61	11.87

Table 2: Excepted number of turns using different layouts

5 Validation

In order to validate our model, we will compare the theoretical results described above with the empirical results after simulating a large number (10'000) of independent games with the same layout. Sub-optimal strategies will also be studied to confirm the domination of the optimal policy computed with the value-iteration algorithm.

5.1 Comparison of theoretical and experimental results using the optimal strategy

Table 3 shows the comparison between expected and experimental results using layout 7 presented in Table 2. This layout was chosen because it contains all kinds of traps and bonuses.

We can see that for both circular and linear scenarios, empirical costs are extremely similar to the expected ones found by value-iteration. The maximum divergence between expected and empirical results can be found for the 9^{th} square of the circular board and only reaches 2.26%. This proves the consistency of our model. It must be noted that this operation was repeated using different levels of layout complexity: the same conclusion could be drawn each time.

Circular Layout					Linear Layout				
Square	Layout	Expected	Empirical	Dice	Square	Layout	Expected	Empirical	Dice
1	0	14.68	14.68	2	1	0	14.52	14.58	2
2	4	13.68	13.71	1	2	4	13.52	13.54	1
3	2	11.68	11.63	1	3	2	11.52	11.35	1
4	1	11.36	11.37	1	4	1	11.04	11.08	1
5	3	9.36	9.31	3	5	3	9.04	9.07	3
6	2	8.10	8.16	2	6	2	7.75	7.89	2
7	4	6.37	6.39	3	7	4	5.92	5.98	3
8	3	5.03	4.97	3	8	3	4.69	4.71	3
9	2	3.62	3.54	2	9	2	3.27	3.26	3
10	0	2	2.01	1	10	0	1.33	1.33	3
11	1	8	8.00	1	11	1	8	8.00	1
12	3	6	5.99	1	12	3	6	6.02	1
13	4	4	3.95	1	13	4	4	3.96	1
14	1	2	2.00	1	14	1	2	2.00	1

Table 3: Comparison between theoretical and empirical costs using the optimal strategy

5.2 Comparison of different strategies of players

Considering previous outputs of our algorithm, it appears that the optimal strategy differs radically as a function of the given layout. We found that it could be interesting to define different profiles of players and to observe the different outputs that arise with their participation in the game. We thus defined 6 different profiles and made them match with real player behavior. Probabilities associated with their playing style (see the table below) were sometimes computed randomly but stay constant between each simulation.

- Profile 1 : Only use first dice
- Profile 2 : Only use second dice
- Profile 3 : Only use third dice
- Profile 4 : security policy
- Profile 5 : random policy
- Profile 6 : risky policy

Players	Probabilities		
Dice :	1	2	3
1	1	0	0
2	0	1	0
3	0	0	1
4	0.5	0.42	0.08
5	0.33	0.33	0.33
6	0.38	0.12	0.5

Table 4 shows respectively the expected and experimental costs obtained by the optimal strategy, as well as results for strategies described above. We can observe that mostly the best policy is to use only either dice 1 or dice 3. Players with low risk aversion will get better results when the layout is almost full of traps. Intermediate strategies seem to win almost in no configuration, but never finish last. However, we can focus our attention on more realistic layouts (with all types of traps and bonuses): layouts 7 and 8. It appears that the best strategies for layout 7 correspond to the worse ones for layout 8, which means that playing with a risky policy is only gainful when the layout contains few traps. By this way, one can conclude by saying that the best strategy to use is highly correlated with the contents of the layout and so there is no policy that could perform well with any layout, which is something that we could guess.

Thus, the only way to plan the best strategy to apply is to resort to computation. Then, we observe that the number of turns brought by the strategy promoted by computer is very close from the theoretical number of turns. So, it seems clear that our algorithm works pretty good and brings us a good approximation of the optimal policy.

Note : In the following table, the value X echoes to a strategy which doesn't succeed. This can happen when the layout is full of trapped squares, using of dice 3 will prevent the player to go forward.

Circle = False								
	Optimal		Sub-optimal strategies					
Layout	Expected	Computed	1	2	3	4	5	6
0	6.67	6.68	17.06	9.22	6.67	11.56	9.51	9.15
1	17	17.08	17.09	143.87	X	X	X	X
2	17	17.09	17.16	92.19	X	X	X	X
3	11.54	11.52	17.2	13.03	X	X	X	X
4	1.33	1.34	17.07	5.35	1.34	8.79	5.64	5.01
5	14.03	13.99	17.07	21.82	20.31	21.61	23.22	24.1
6	4.97	5.01	17.08	8.55	5.93	10.22	8.48	8.23
7	14.52	14.51	17.07	21.06	27.36	20.47	22.42	23.24
8	9.81	9.82	17.07	12.30	11.79	14.82	13.98	14.32

Table 4: Player results comparison using different layouts

Remark : We decide only to show results for non-circular game because our previous observations are still unchanged whether for circular game or not.

5.3 Statistical behavior for realistic games

In this last section, we will focus our analysis on a realistic layout for a Snakes and Ladders game. Even though a few different layouts were tested, the results that are presented in Table 2 correspond to the 8^{th} one. This layout takes 2 traps of each kind and 2 bonuses in a linear configuration.

Again, a few observations can be drawn with Figure 1:

- The computer always have the smallest mean value regardless of the layout used ;
- Player 1 always has the same distribution, regardless of the layout used, since he always chooses the first die. His standard deviation is therefore also the smallest (**5.06** turns). By contrast, standard deviation for common 3rd-die players is higher than **9** ;
- The results of players from 2 to 5 depend strongly on the layout used, its traps and bonuses and their positions ;
- As the computer is also subject to the randomness of the dice, its victory is not always assured. In this configuration, Player 3, the 3rd dice spammer, is the closest to its brilliant results.

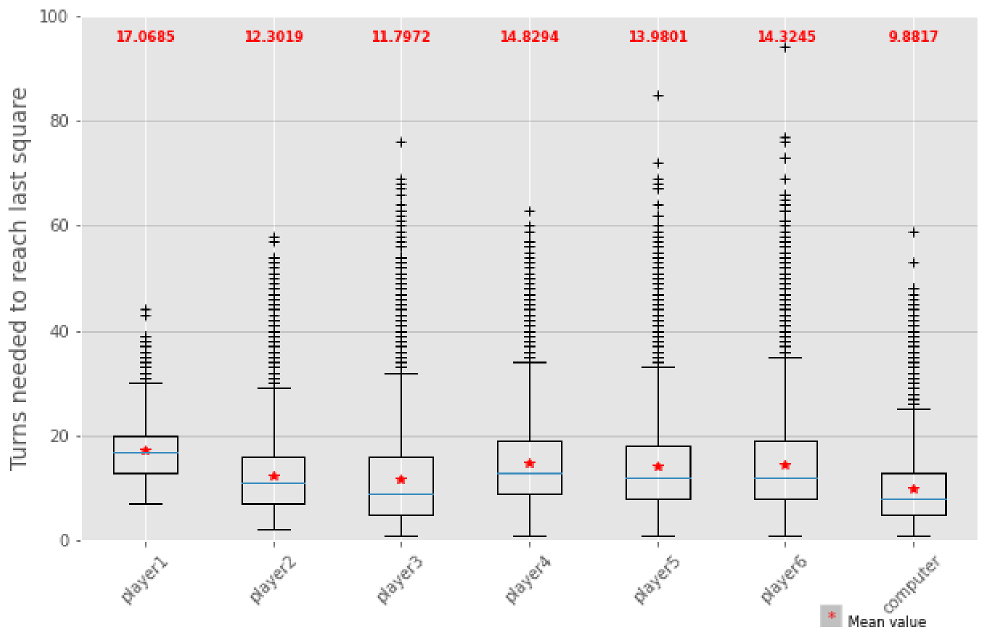

Figure 1: Results distribution for each player in a realistic simulation

6 Conclusion

The aim of this project was to apply the value iteration algorithm to solve a MDP designed for a Snakes and Ladders game. Our implementation allows us to find the expected cost and the best die to roll, for any layout and at each position. The validation was carried out using an empirical method, simulating a large number of games with different layouts and configurations. We also tried to measure the influence of the layout on the optimal die to roll. It appears that the number of traps and bonuses really impact the results, as well as their position on the board. The 'circle' attribute has a forcing effect: the computer always prefer to secure its landing on the goal square, avoiding the risky die when it is close to the last square.

We then tried to simulate real cases using traditional player types and realistic boards. We found that, although the choice of die is important, the results will always be primarily influenced by the layout: a strategy that pays off in one layout will probably fail in the next.

7 References

Sutton and Barto (2018) "Reinforcement learning: an introduction"

Decision making course (2022) - Marco Saerens, UCLouvain

Value Iteration Algorithm - Dynamic Programming Algorithms in Python, YouTube.
https://www.youtube.com/watch?v=hUqeGLkx$_{z}sab_{c}hannel = CodingPerspective$