UCLouvain

epl

Project 1
—
Markov Decision Processes
Report

LSINF2275 - Data Mining and Decision Making

Academic Year 2020-2021

Group Members:
Martin Draguet [54751700] [DATE2M]
Arno Geurts [44251700] [DATE2M]
Romain Graux [28681700] [DATE2M]

Monday 5th April, 2021

1 Introduction

The objective of this practice is to implement a practical case of Markov Decision Processes (MDP) : the Snakes and Ladders game. In this project, it is asked to apply the value iteration algorithm on different layouts in order to find the best strategy.
NB : On each board in this report, above each case are indicated the Markov decision and the integer representing the identity of the square in the following way : **dice choice | square identity**[1].

2 Theoretical Analysis

Let us first prove that the *snakes and ladders* game can be solved using MDP. It is a stochastic game : the dice is truly and fairly random whilst the board and the positions of the traps are deterministic. It is thus possible to compute a policy (i.e. strategy) to finish the game as fast as possible, avoiding all traps. The decision making will be the choice of the optimal dice for each case of the board.
Also, the Markov assumption tells that a current state of the process does not depend on past states, which is the case for the *snakes and ladders* game since the choice of the dice only depends on the current case on the board and not on the past turns. We can thus use MDP to find the optimal policy.

Finding the best strategy amounts to reach a destination from an initial state, while minimizing the total expected cost at each turn (i.e. finding the right dice to throw by taking into account all the possible outcomes). The value iteration algorithm expresses this process with the following equations :

$$\begin{cases} \hat{V}(k) \leftarrow \min\limits_{a \in U(k)} \left\{ c(a|k) + \sum\limits_{k'=1}^{n} p(k'|k,a)\hat{V}(k') \right\}, k \neq d \\ \hat{V}(d) = 0, \text{where } d \text{ is the destination case on the board, an absorbing state} \end{cases} . \tag{1}$$

We obtain the Bellman-Ford optimality conditions by convergence of Equation 1, which gives the value of the optimal dice to throw for each turn (i.e. state) k :

$$\hat{V^*}(k) = \arg\min_{a \in U(k)} \left\{ c(a|k) + \sum_{k'=1}^{n} p(k'|k,a)\hat{V}(k') \right\} \tag{2}$$

For the *snakes and ladders* game :

- $\hat{V}(k)$ is the expected number of turns from state k to reach the destination case d.
- $c(a|k)$: the function c is a cost function which expresses the cost for using a certain dice a from state k. This cost is always equal to 1 unless state k is a *prison* trap and has to wait an extra turn to play.
- $p(k'|k,a)$ is the probability to reach state k' from state k by throwing the dice a
- $U(k)$ is the set of all the possible actions that can be taken at state k, i.e. all the different dices.

3 Implementation Choices

The whole Markov decision process script is composed of 4 classes:

3.1 Trap

We have implemented a simple class `Trap` with every trap as a function, each function return the *next position* and if the next state is *freezed*. We also have a function that takes the index of the trap as argument and returns the asked trap function.

3.2 Dice

We have implemented a `Dice` abstract class so we are able to implement each dice. The class is composed of the *move bounds* and the *trap activation probability*. With these class variables, we are able to determine the randomized next number of moves but also the probability getting a particular number of moves.

[1] 0 if square is normal, otherwise the integer refers to the trap type as defined in the assignment of the project

3.3 SnakeAndLadders

We also have implemented a `SnakeAndLadders` environment that behaves as a Gym environment of our Snake and Ladders game without GUI mode (*reset*, *step*, *seed*, ...). This environment could be used as a reinforcment learning environment or any Markov process (Q-learning, DQN, DDQN, ...).

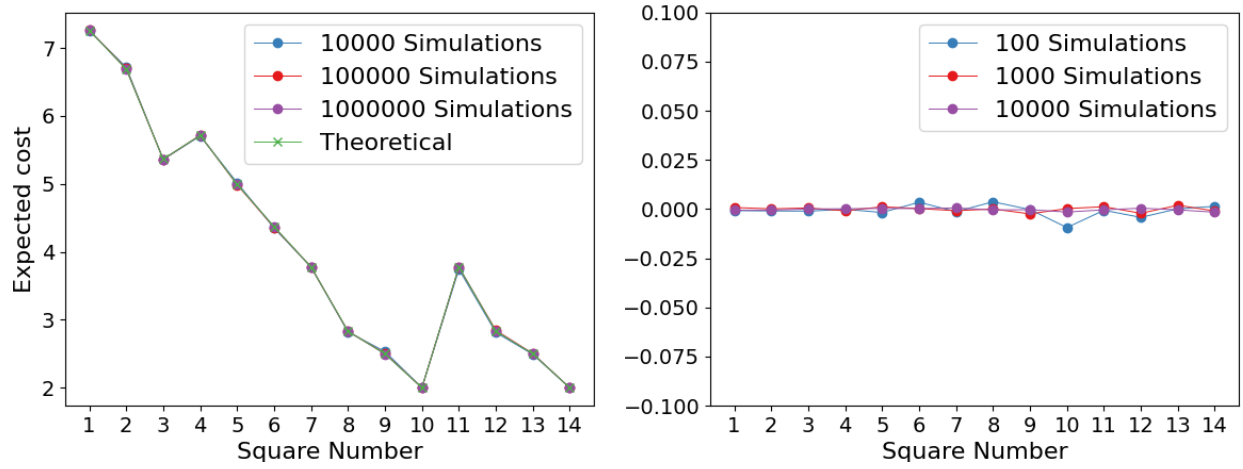
3.4 Strategy

And finally we have implemented a `Strategy` abstract class that will be the subclass for `MarkovDecisionProcess` and `ChosenDices` classes. The `ChosenDices` strategy will act with the particular dices given as argument when initializing the strategy. The `MarkovDecisionProcess` strategy contains all functions needed to compute the Markov decision process algorithm but basically 2 main functions :

- the `next_states` function computes all the next states k', with their probabilities p and if the next state is freezed starting from a state k with an action a.
- the `compute` function that will iterate until convergence over all *states* and all *actions* to compute the new Q value of that particular *state*, *action* tuple.

4 Theoretical Cost and Model Validation

The theoretical cost is the expected amount of plays to get to the final case. This cost is given by the `MarkovDecisionProcess` function. We have implemented a `simulations` function that computes these values empirically by running a significant amount of games and computes the expected cost for each square as the mean of all simulations. The relative error is given by the ratio between the empirical cost and the theoretical cost, and the theoretical cost : $\frac{\text{theoretical cost}-\text{empirical cost}}{\text{theoretical cost}}$. As seen on Figure 1 below, the empirical cost follows the theoretical values and the relative error is never greater than 0.5, which proves that our theoretical model is correct.



(a) Theory vs Simulations

(b) Relative error

Figure 1: Model Validation on basic board without traps and *circle=False*

We run the same validation test on a random board (see Figure 2) with parameter `circle` set to *True*. As we see on Figure 3, the results are convincing : the simulations curve follows the theoretical values and the relative error decreases when the number of simulations increases, which proves the consistency of the model.

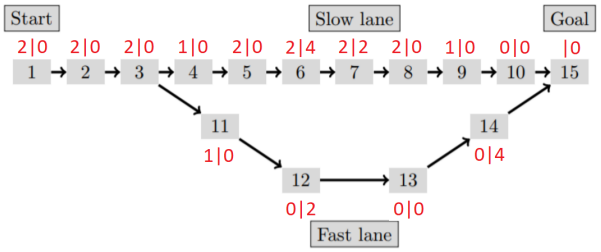
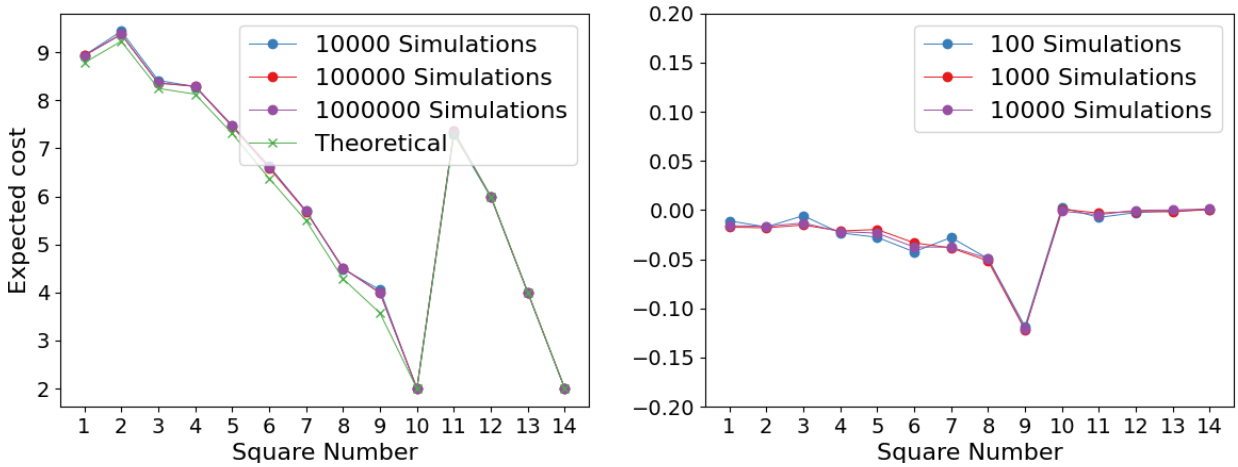


Figure 2: Board with traps



(a) Theory vs Simulations

(b) Relative error

Figure 3: Model Validation on random board with traps and *circle=True*

5 Strategy Comparison

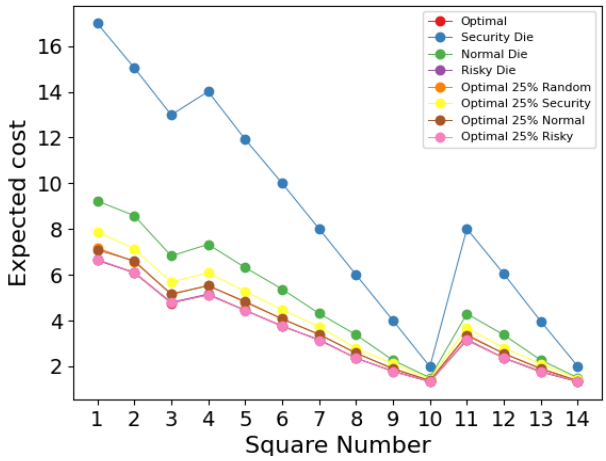
We have decided to implement several strategies :

- `optimal` : throws the dice computed by the Markov process
- `security_only` : always throws the security dice
- `normal_only` : always throws the normal dice
- `risky_only` : always throws the risky dice
- `optimal_with_random_25` : follows the optimal policy but with a 25% probability of throwing a random dice. This represents a player who as faith in his luck.
- `optimal_with_security_25` : follows the optimal policy but with a 25% probability of throwing the security dice. This represents a player who has a defensive play style.
- `optimal_with_normal_25` : follows the optimal policy but with a 25% probability of throwing the normal dice. This represents a *lambda* player, taking risks but not too much.
- `optimal_with_risky_25` : follows the optimal policy but with a 25% probability of throwing the risky. This represents a player with an aggressive play style.

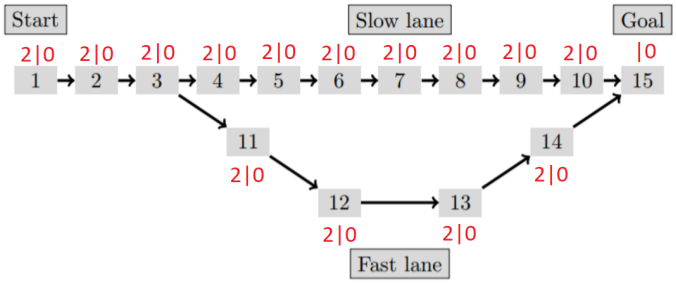
We will now observe the expected costs of the strategies on different layouts and compare them.

No-circle board without traps

As we simulate the game with this 8 different strategies with no traps and no-circle board, we can that the best strategies are the one that takes either risks (`risky`), either behaves optimally (`optimal`) or both (`optimal 25\% risky`). This makes sense since the risky dice is always chosen by the MDP when there are no traps. All other strategies are significantly less efficient.



(a) Expected cost

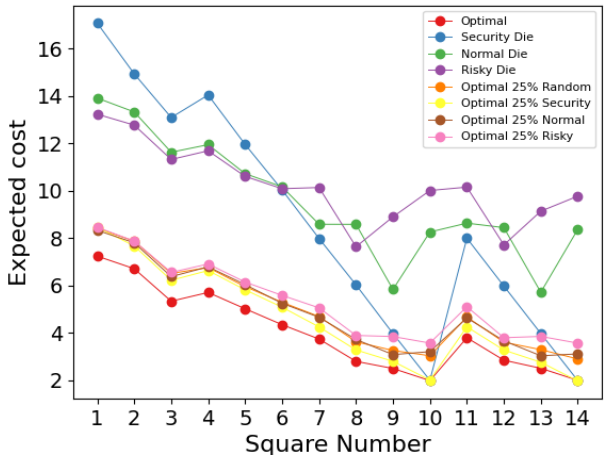


(b) Board

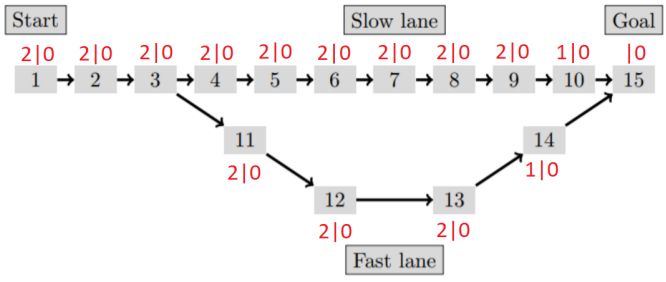
Figure 4: Strategy Comparison on board without traps and *circle=False*

Circle board without traps

This time, the best strategy are the `optimal` and every strategy using partially the `optimal` on one side and an other strategy on the other side. On the other hand, the `Normal` and `Risky` strategies behaves badly while approaching the last square. The `Optimal` strategy always chooses to use the risky dice, since it has the most chance to be the fastest solution, except at the end since the board is circular.



(a) Expected cost

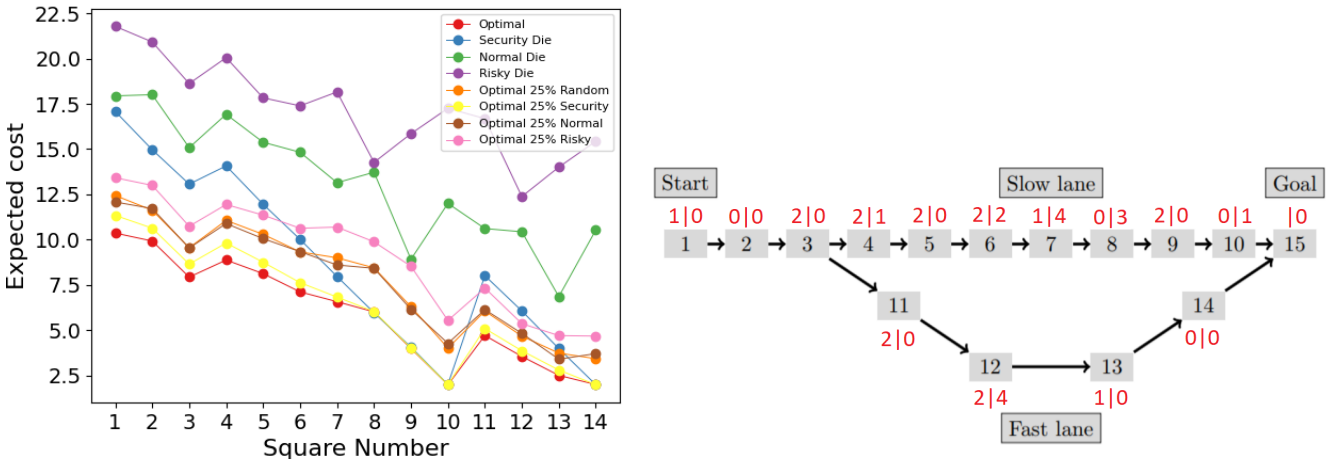


(b) Board

Figure 5: Strategy Comparison on trapped board and *circle=True*

Circle board with trapped slow lane

In this layout, we have several traps on the slow lane while the fast lane is lightly trapped. On the fast lane, one chooses to use the security dice as we get after the square 12, while on the slow lane, the MDP chooses to use the risky dice most of the time in order to avoid as much trap as possible, except for example on square 8 because we have to avoid the trap on the 10th square. Once again, we can see that the `optimal` strategy gains the best results, closely followed by the mixed strategies. `Risky` and `Normal` strategies have the worst results, due to the fact that they do not pay attention to traps.

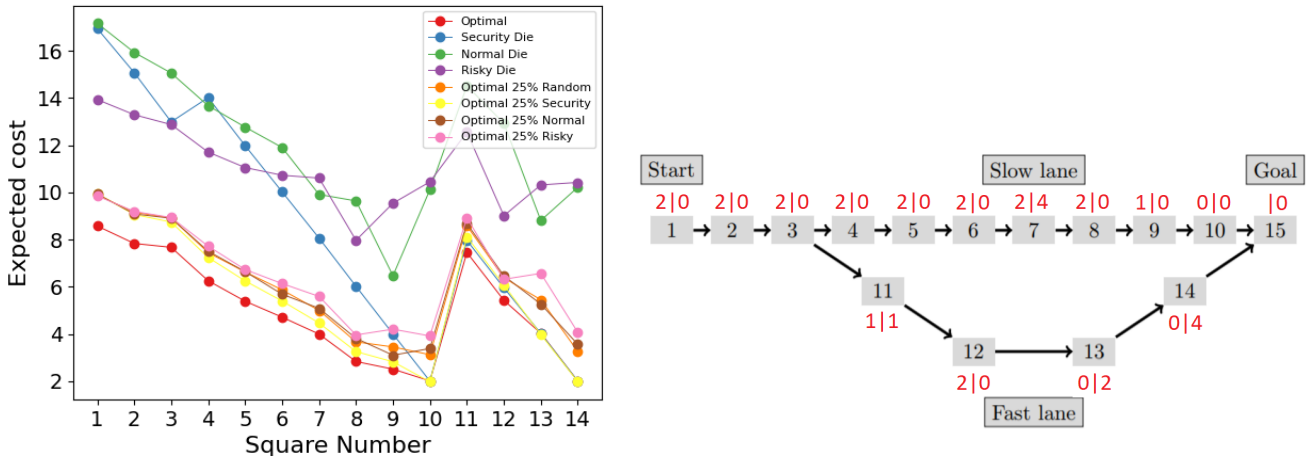


(a) Expected cost (b) Board

Figure 6: Strategy Comparison on trapped board and *circle=True*

Circle board with trapped fast lane

Here, inversely, the slow lane is heavily trapped while the slow lane not. The MDP then chooses to go risky on the slow lane by choosing the risky dice except when approaching the last square. On the other lane, the process chooses to use every dice, depending on the case and on the next trap. As for the previous layout, the `Optimal 25\% Security` is the one that performs the most closely to the `Optimal` strategy, while the `Risky` and `Normal` strategies performs bad, once again for not paying attention to the different traps and to the fact that the board is circular.



(a) Expected cost (b) Board

Figure 7: Strategy Comparison on trapped board and *circle=True*

6 Conclusion

As we observe the different results of the `Markov Decision Process` and the different comparison with simple and more complex strategies, one can conclude that the MDP, as it is theoretically proven, leads to the bests results. This does not mean that you always win with this strategy, as randomness is involved in the game, but on a big sample of games no strategy will be better.
One possible improvement would be a strategy that forces the player to take the fast lane, even if it is trapped. Sadly, this would be cheating since the rules of the game state that taking the fast lane is a random choice. But like some people would say, it is impossible to beat the optimal strategy if you are not cheating ...