UCLouvain

École polytechnique de Louvain

LINFO2257 : Data Mining and Decision Making

Markov Decision Processes

Professor : Marco SAERENS

Aymeric WARNAUTS (DATS) - 8703 1800
Nathan NEPPER (SINF) - 1428 1800
Tanguy LOSSEAU (INFO) - 4578 1400

Group 22

2 April 2022

1 Introduction

This project aims at solving the optimal strategy of "Snake and Ladder" games using Markov decision processes. A "Snake and Ladder" board has 15 tiles that are disposed in 2 lanes, a Fast and a Slow lane; each ending on the 15^{th} tile, the tile we have to join in order to win the game. The player has to roll a dice to determine its progression on the board. Moreover, a tile can contains a bonus or various traps that slows our progression or a bonus.

Implementing a Markov decision process for such games may be usefull given the impact of the randomnes of dice rolling on the optimal strategy; we will discover the efficiency of the *Value Iteration Algorithm* in such random conditions.

2 Algorithm

We will use the *Value-Iteration algorithm*, which will determine the best strategy π^* for each state in order to reach the goal state $d = 15$ from initial state $s0 = k0 = 1$ with minimal cost (the less turns as possible), which can be done by using an optimal strategy which solves:

$$V^*(k_0) = \min_{\pi} \mathbb{E}_{s_1,s_2,\dots}\left[\sum_{t=0}^{\infty} c(a \in A \mid s_t) \mid s_0 = k_0, \pi))\right] \tag{1}$$

Where A is the set of actions $\{$*Security*, *Normal*, *Risky*$\}$, $c(a \in A \mid s_t)$ is the cost function of the action a conditional on the state at time t; this cost function is the number of turn. We aim at minimizing the cost function conditional on the starting state $s_0 = k_0$ and the strategy π.

From this minimization problem, we can derive the value iteration's recurrence relation given the game, which is the following Bellman equation:

$$\begin{cases} \hat{V}(k) = \min_{a \in A} \sum_{k'=1}^{15} p(k' \mid k, a) * (c(a, k' \mid k) + \hat{V}(k')) & \forall i \in [1..14] \\ \hat{V}(d = k_{15}) = 0 & \text{(absorbing state)} \end{cases} \tag{2}$$

where $p(k' \mid k, a)$, $c(a, k' \mid k)$ are the probability / cost of landing on tile k' from tile k using dice a. Those probabilities depend on the dice, the probability of triggering a trap and the probability to go on the fast or slow lane. Those probabilities will be discussed in details in the next section.

3 Implementation

The aforementioned *Value-Iteration algorithm* has a key requirement: it needs all possible consequences resulting in the application of any valid action applied to a state. As such, our implementation is structured upon the following three classes:

- Action: summarizes the name, range, and trap trigger likelihood of each usable dice.
- Consequence: contains the cost, probability, and resulting state of applying an action to a state.
- State: a position on the board of given category (trapped, bonus...) holding for each possible Action a list of possible Consequences.

Any board of the Snake and Ladder game consists in an array of State, defined by a Layout/Circling combination, and built using the *build_state* function.

3.1 Value-iteration

With these ingredients, the Bellman equation (2) can be implemented as shown in listing 1.

```
#Absorbing state?
if state.position == 15:
    continue

#Not end state
best_cost = sys.float_info.max
for action in Action.names:
    cost = 0
    for cons in state.results_of[action]:
        cost += cons.probability * (cons.cost+expect[cons.end_state.position-1])

    if cost < best_cost:
        best_cost = cost
        best_action[state.position - 1] = action

expect[state.position - 1] = best_cost
```

Listing 1: Value-iteration implementation

This implementation simply picks, for each state (tile on the board), the action (selected dice) which minimize it's expected required number of turn to finish the game.

Though this **expect** value is coarsely initiated to the tile's distance from goal node, it is refined at each iteration; and we consider convergence was reached when the maximal expectancy modification reaches a predefined minimal value, 10^{-6}.

3.2 Simulation

In order to asses the performance of Optimal Strategy, and therefore our implementation; we were also tasked with simulating, for a given board layout and a given set of dice, the number of steps required from each tile to win the game.

After building the required Snake and Ladder board, we simulate n_it times for each square the number of steps required to reach square 15.

The step of simulating the result of an Action on a State is done by picking a random value val in the range [0,1] according to a uniform distribution. The first Consequence whose cumulative probability exceed this number is picked.

Since val follows a uniform distribution, we have that:

$$\forall a, b \in [0,1] : p(a < val \leq b) = b - a \tag{3}$$

Defining $C = state.results\_of[actions[state.position - 1]]$, we obtain:

$$p(\sum_{k=0}^{i-1} C[i].probability < val \leq \sum_{k=0}^{i} C[i].probability) = \sum_{k=0}^{i} C[i].probability - \sum_{k=0}^{i-1} C[i].probability = C[i].probability \tag{4}$$

Hence, using the explained technique, each consequence has it's own probability of being picked when simulated.

The resulting implementation is shown in Listing 2.

```
for _ in range(n_it):
    while True:
        if state.position == 15:
            # End position reached: save current number of steps
            break

        val = random.uniform(0, 1)
        cum_sum = 0.0
        for c in state.results_of[actions[state.position-1]]:
            cum_sum += c.probability
            if cum_sum >= val:
                step += c.cost
                state = c.end_state
                break
```

Listing 2: Simulation implementation

4 Results

Now that the theoretical basis for solving the problem are explained, we will evaluate our implementation with multiple scenarios and compare the Optimal and Sub-Optimal strategies regarding their mean steps numbers. For validation purposes, we decided to draw a random ladder using a *PRNG* (Pseudo-Random Number Generator). This ladder is uniformly distributed of Traps, *Bonus* and Ordinary tiles; we will use to following configuration for all our tests:

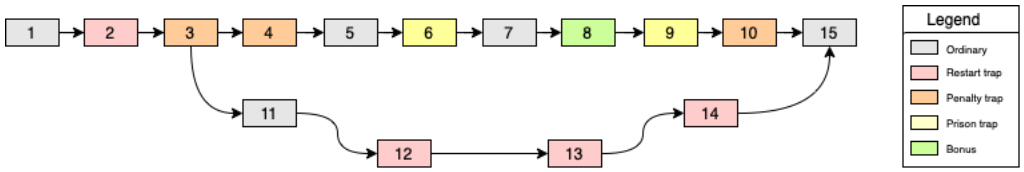


Figure 1: Random ladder

4.1 Optimal strategy

The Optimal strategy computed for this specific ladder in the situation of non-circle board is the following. We can see that the move on the ladder are mostly Security move given the concentration of *Restart* traps; which are the harshest ones.

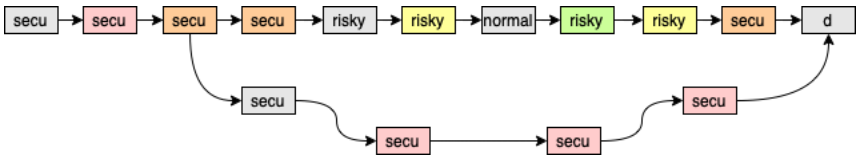


Figure 2: Optimal strategy in non-circle board

Now we can estimate the Optimal strategy for the circle board. The only element changing between the two strategy is the move made on the 9^{th} tile, which changed from a *Risky* to a *Normal* move because there is a

probability that the *Risky* move get back to the first tile; while it wasn't the case previously because going beyond the destination wasn't penalized by a restart.

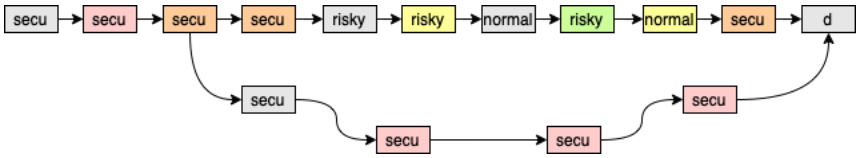
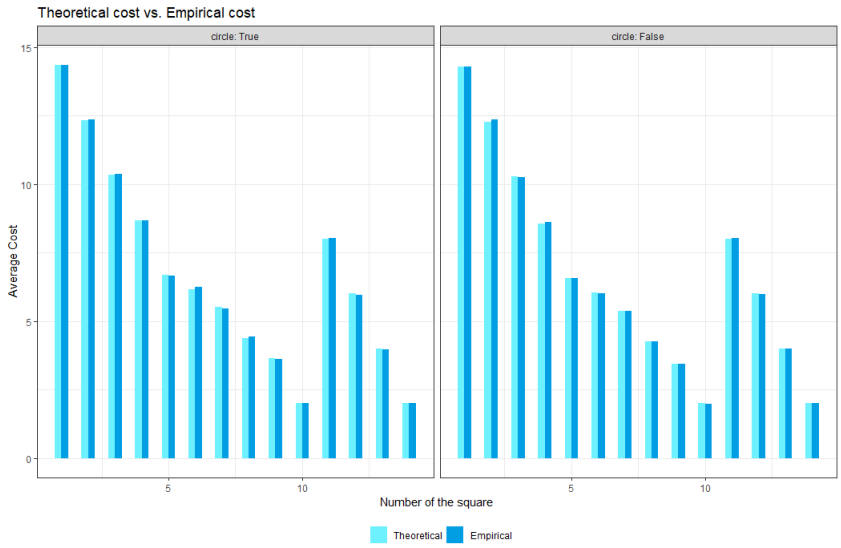


Figure 3: Optimal strategy in circle board

Finally, the fact that the two Optimal Strategies are close can be explained by the fact that the board layout is a hard problem given the position of the traps. Actually, the fast-lane is highly trapped and most of the traps are at the end of the travel; which increase the risk of loop-back for non-circle board, and therefore gives a similar number of Security moves.

Given the Optimal Strategies, we want to prove they are efficient in reality; this is when simulations come on stage. In fact, we will compare the **Theoretical expected cost** with the **Empirical cost** measured on 10000 simulations of game with the Optimal Strategy for *circle* and *non-circle* board.



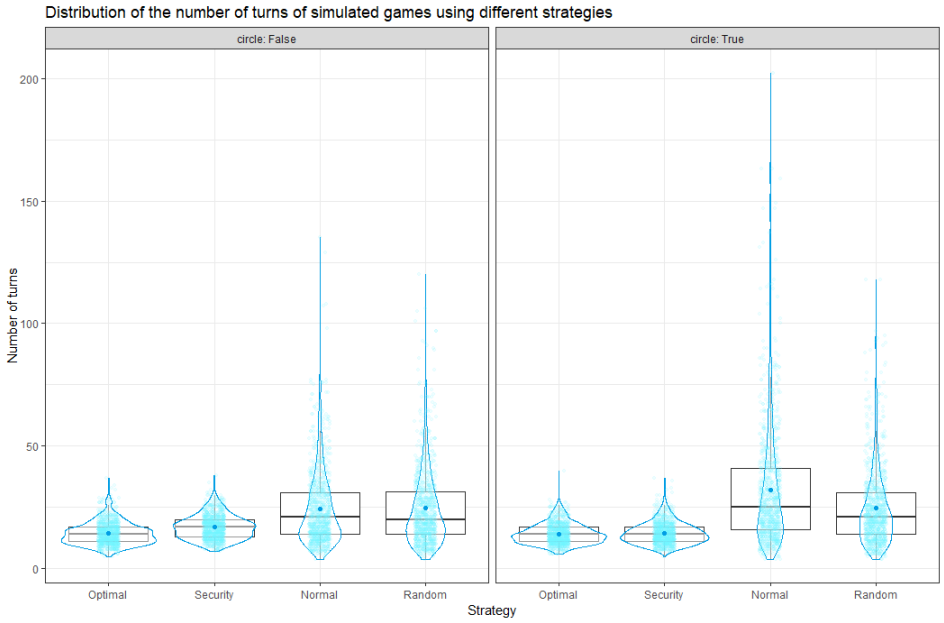
As we can see, the Theoretical cost we obtain by the Value-Iteration Algorithm seems to be extremely close to our Empirical cost derived by simulation of a great quantity of games. The fact that our estimated cost is close to the simulated one gives us a good appreciation of the performance of our implementation.
Moreover, the results on the *circle* and *non-circle* boards are really close; this may be explained by the fact that the strategy only slightly differs. In fact, we can see that only one move changed between the two scenario; this phenomenon has been explained earlier (Section 4.1).

4.2 Comparison with sub-Optimal strategy

Now that we proved the efficiency of our Market Decision Process, we may want to compare the Optimal strategy to other Sub-Optimal strategy that player may use in a standard game. For this purposes, we decided to do 1000 simulations of game with the following Sub-Optimal strategies:

- **Security** The player only use *Security* move
- **Normal** The player only use *Normal* move
- **Random** The player pick the move to be used randomly between [*Security*, *Normal*, *Risky*]

We decided to avoid simulation of a only *Risky* strategy because of a lack of quick convergence due to the layout configuration and the very bad performance of the strategy.



Remembering the choices made in case of circular and non-circular layouts, we can make some developments over the strategies distributions. It leads to an *Optimal* strategy very close to the *Security* one because of the risky configuration of the board. In fact, the *Optimal* strategy promote the utilization of *Security* moves in both cases.
What's interesting is that we observe a huge concentration of points around the mean of our *Optimal* and full *Security* strategies and so we can assume a low variance in our expectations. Thanks to this concentration of the distributions we can say that the mean is a good unbiased estimator if we choose both strategies.

5 Conclusion

From our multiple simulations, we can see that the Optimal strategy stays the best way to achieve in the smallest amount of steps. Our estimate of the cost is accurate and seems to be a unbiased estimator of the mean cost of the Optimal Strategy given our simulation of the Empirical cost in regards to the Theoretical one.
Moreover, even in harsh layout condition our Markov Decision Process seems to be efficient and find a better strategy than the heuristic based ones.
Finally, regarding the ways of improvement, we may try to compare our Optimal Strategy to a better Human-Though Sub-Optimal Strategy, but the results are sufficiently great to assume that such mixed strategy are bound to fail to counter a MVP based Optimal Strategy. A another way to improve our project would be to expand the states management to even bigger Board and therefore generalize to any board type.

References

[1] **Douglas J.White** (1985) *Real Applications of Markov Decision Processes*, Institute for Operation research and the management sciences

[2] **Martin L. Puterman** (1994) *Markov decision processes: Discrete Stochastic Dynamic Programming*, A JOHN WILEY AND SONS, INC., PUBLICATION

[3] **Saerens M.** (2021) *Decision Making*, Uclouvain LSINF 2275 Moodle [Slides]

[4] **Sutton, R. Barto, G.** (2018) *Reinforcement Learning: An Introduction*, Adaptive Computation and Machine Learning, 2nd Edition