LINFO2275

MARKOV DECISION PROCESS PROJECT

MATHIAS DAH FIENON (DATS2M) & GABIN KANA TSIGUIA (DATS2M)

Academic year 2021-2022

1 Introduction

In everyday life, we are called to make decision based on our expectation and experience. Mostly, we need information to schedule everything rightly. This, also, applies to the stochastic field. One of the way to make a good decision in a stochastic process, is to base the decision on the markov decision process. In the following, we will try to apply the markov chains decision process to Snakes and Ladders game.

2 The objective

The objective of this project is to put into practice some of the techniques introduced in the data mining and decision making lectures. This is done through the study of a practical case which requires the use of a scientific programming language related to the statistical processing of data. This work aims at applying algorithms (mainly the value iteration) solving "***Markov decision processes***" (**MDP**) in the framework of a Snakes and Ladders game. The report will…

- Explain, through detailed and commented python code, how with the **MDP**, we can obtain an **optimal strategy**, calculating a good estimation of the value of the minimun number of turns to completed the game.
- Show some comparison between the performance of the strategy computed by the **MDP**, and others strategies(default or randomly generated).

A schematic representation of this game is shown below:

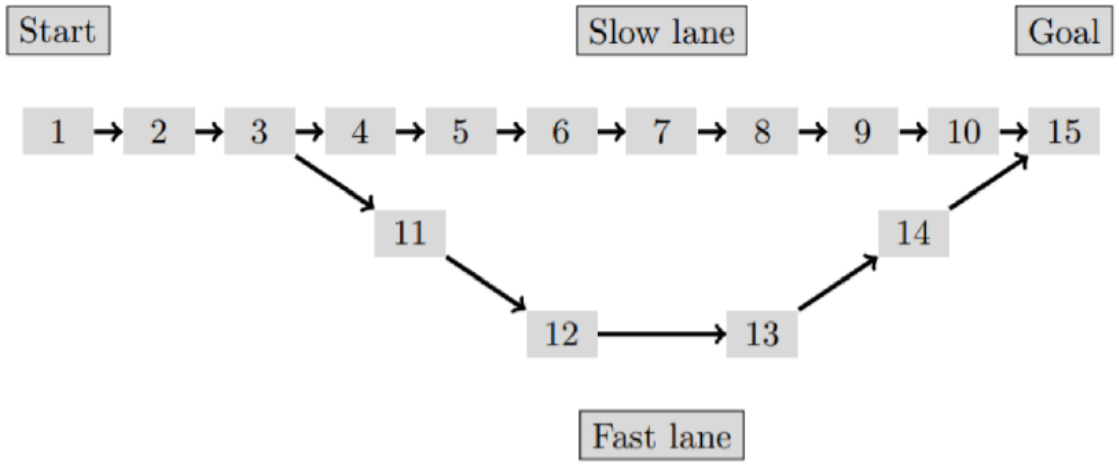

Figure 2.1: Board of the game

We have three dices (security, normal and risky) that have each one probability(1/2, 1/3 and 1/4) of letting us move forward (by 1, 2 or 3 squares), or don't move(playing "0").

3 Transition probabilities

Now that the objective is settle, we need an idea about the transition probabilities of jumping from one square to another (this dependent of the dice we use).

Recall that, a markov decision process depend on

- states ***s***,
- action ***a*** take to jump from state ***s*** to state ***s'*** with its probabibily ***p(s'/s, a)***
- the cost ***c(s'/s, a)*** or the reward $Ra_t(s_t, s')$ of this action.

4 Value iteration algorithm

The value iteration algorithm is one way of solving the markov decision problems. Markov decision processes are discrete-time stochastic control processes used for a variety of optimization problems where the outcome is partially random and partially under the control of the decision maker. Markov decision processes extend Markov chains with choice and motivations; actions allow choices, and rewards provide motivation for actions.

The state of the process in slot t is s_t and the decision maker may choose any action $a_t \in A(s_t)$ available in that state. As a result of action a_t the system moves to a new state s_t and provides the reward $Ra_t(s_t, s')$. The probability that the system moves to state s' is given by the transition function $Pa_t(s_t, s')$. The goal is to optimize a policy π maximizing a cumulative function of the random rewards, e.g., the expected discounted sum of rewards over an infinite time horizon is $\sum_{t=0}^{\infty} \gamma^t Ra_t(s_t, s_{t+1})$.

To calculate the optimal policy given P and R, the state transitions and, respectively, the rewards, one needs two arrays and indexed by state, the value and policies, respectively.

$$V(s) = \sum_{s'} P_{\pi(s)}(s, s')(R_{\pi(s)}(s, s') + \gamma V(s'))$$

$$\pi(s) = argmax_{\alpha}\{\sum_{s'} P_{(}s, s')(R_{\alpha}(s, s') + \gamma V(s'))\}$$

In the value induction proposed by Bellman the calculation of is substituted in the calculation of $V(s)$:

$$V_{i+1}(s) = max_{\alpha}\{\sum_{s'} P_{\alpha}(s, s')(R_{\alpha}(s, s') + \gamma V_i(s')\}$$

where:

- **S** is a finite set of system states,
- **A** is a finite set of actions; $As_t \in A$ is the finite set of actions available in state $S_t \in S$ in time slot t.
- **P** the set of transition probabilities; with $Pa_t(s_t, s') = P(s(t+1) = s' | s_t, a_t)$ the probability that action a_t in state s_t in time slot t will lead to state s' in time slot $t+1$.
- **R**$a_t(s_t, s')$ the immediate reward after the transition from state s_t to state s'.

- $\gamma \in [0,1]$ a discount factor representing the difference in importance between present and future rewards.

5 Implementation

In this section, will be detailed the differents particularities of the MDP. The first one is the constant "**convergence factor**", fixed to 5, used for the condition of the value-iteration algorithm. Here the absolute value of the difference between the new computed vector of expected value, and the previous one $||\hat{V}_{new} - \hat{V}_{previous}|| < \gamma$.

The second particularity, is to build the new values of expectation separately instead of building a transition probability matrix, and computing the expectation using a matrix-vector product until convergence. The probabilities used to build those values, are calculated with a function, that takes as parameters the current squares, the dice type and the roll result.

6 Simultaion and comparison

In order to make comparisons between fixed or random game strategies and the one resulting from our MDP, in this section we will perform game simulations.

We will compare the number of turns needed to complete the game, using any strategy, and the number of turns resulting from a MDP; both strategies applied on the same defined layout.

This operation will be done a series of times, with the aim of evaluating the performance of the **optimal strategy** proposed by MDP in terms of percentage.

The test strategies fixed at the base will be:

- Play with only "security dice" for each square (test A)
- Play with only "normal dice" for each square (test B)
- Play with only "risky dice" for each square (test C)
- Play with a mix of types of dice. (tests D,E,F)

6.1 Simulation with circular layout:

The plots below shows the numbers of turns to complete the game, comparing the result of a simulation of 50 games. The orange color representing the results using tests policies, and the blue one, the results using the optimal strategy returned by the MDP.

The markov score is higher than that obtained using the sub-optimal strategies. i.e. with the test A, using only security dice, the first plot shows that using Markov policy, we use very few number of turns to win. For 50 games, markov has best score 40 times. Comparing with test B, only normal dice, Markov has 34 times best scores.

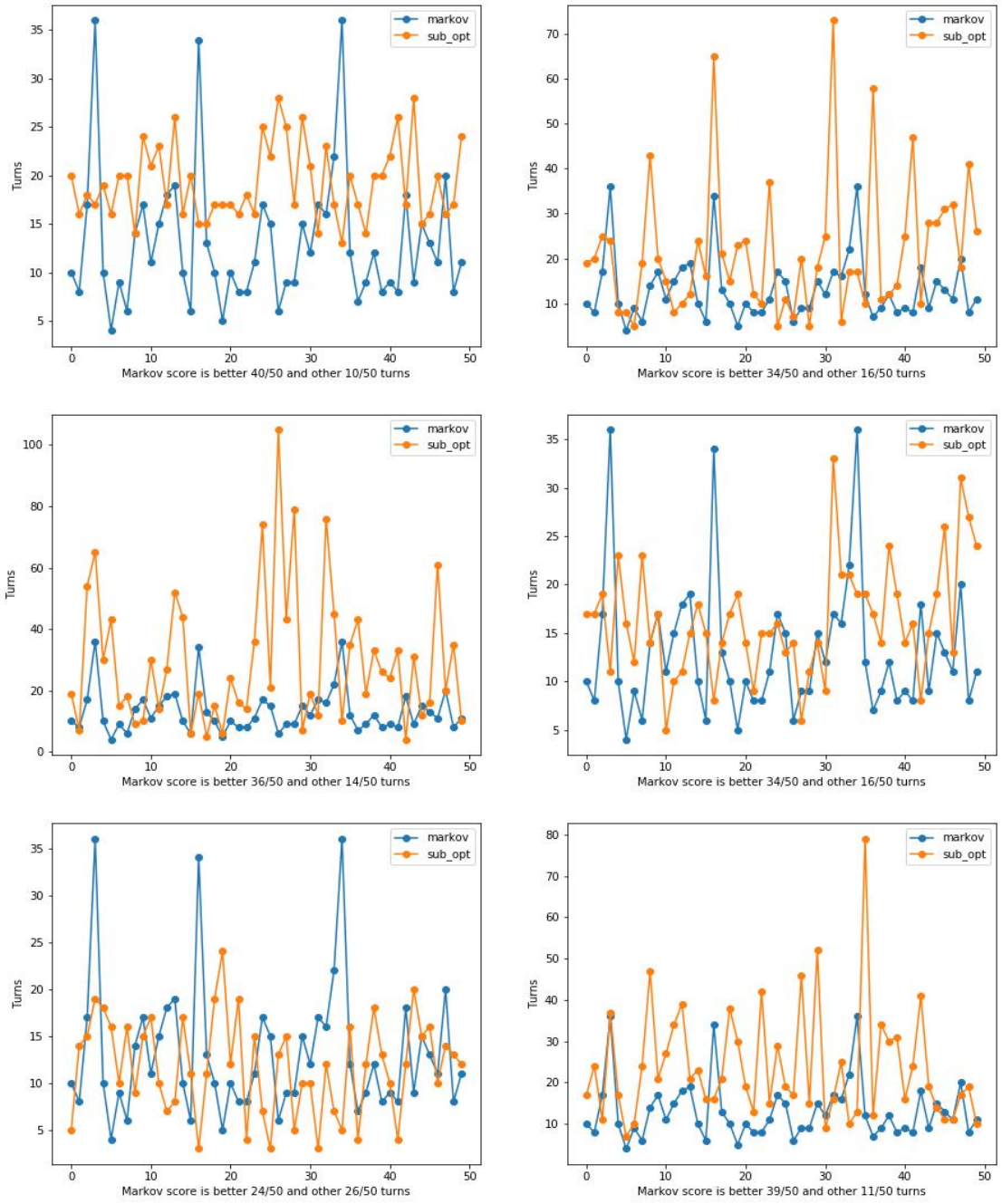

Figure 2: Simulation using circle = True

6.2 Simulation with non circular layout:

In the first plot, the MDP strategy has a best number of turns 35 times in 50 games, respect to using the sub-optimal strategy **only security dice**

The second plot, has almost the same score for both strategies (MDP and **only normal dice**). But we note that the MDP has a best score for 4 comparisons.

Making the same comparison using a circular layout, we have the following plots:

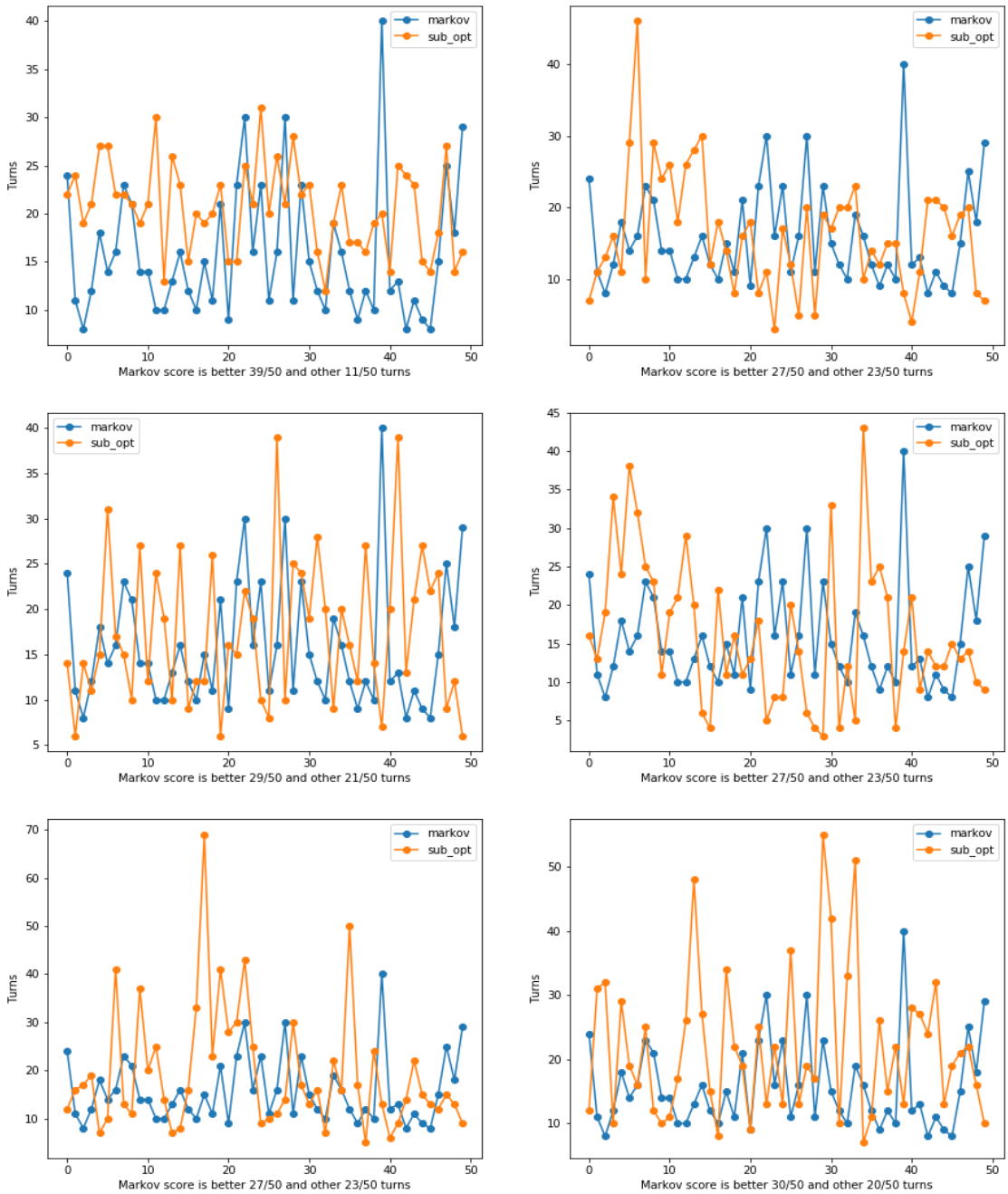

Figure 3: Simulation using circle = False

In all of those plots, we note that the MDP strategy give the better results for comparing with all the sub-optimal strategies.

7 Conclusion

According to the results obtained, we therefore observe that our Markov function predicts better game strategies, compared to defined or random strategies. It is however advisable, in order to improve its performances, to make compromises on the factor of convergence and Gamma.