MAS - Individual Assignment

Max Faber (2676067),
m.b.m.faber@student.vu.nl,
Multi-Agent Systems (XM_0053), Faculty of Science, Vrije Universiteit Amsterdam,
Amsterdam

December 30, 2021

Contents

1 Monte Carlo Sampling

Monte Carlo (MC) methods are versatile statistical techniques for estimating properties of complex systems via random sampling (Pauwels, 2021). One of these methods, MC-sampling, is a method which can be utilized in order to get the estimate of a given function.

Problem definition

We consider the following sequential decision problem: We want to rent a house in Amsterdam. However, nowadays there is a major shortage of affordable housing in Amsterdam. n appointments for site visits of houses have been made (where $n \to \infty$). These houses have to be visited in a pre-fixed order (1, 2, 3, ..., n). Each house gets assigned a score x, where $0 \leq x \leq 1$ (the closer x gets to 1, the better). After each visit a decision needs to be made whether or not we want to rent the house. It is not possible to revert this decision. Therefore, the dilemma of letting go a potential renting option (let's say $x = 0.82$) or renting it is hard: it might be possible that an even better option will come up later. The goal is to maximize the probability of picking the best house out of all the appointments.

Secretary Problem

The problem defined above is an example of the Secretary Problem (GeeksforGeeks, 2019b). This problem can be solved with the $1/e$ law of optimal strategy (where e is a mathematical constant ($e \approx 2.718$)). The name of this strategy is derived from the fact that on average, the probability of finding the best option from a set of numbers is equal to $1/e \approx 0.3679$, which we can evaluate using MC-sampling.

It does so by first obtaining the highest score for the first n/e houses in the list. All of those houses get rejected. Afterwards the first house which is better will be chosen to rent. If no house is better, the last house of the list will be chosen. It is very likely that this house is not the best house of the list and therefore we 'lose' in this case.

Experiment

Using MC-sampling we obtain whether the $1/e$ law of optimal strategy holds for our house-renting problem. This has been done by generating a list of house scores of size 10,000 from a uniform distribution. After applying the specified strategy, we check whether the best house of the list has been rented or not. This process is sampled for 100,000 episodes. After sampling, the probability of renting the best house was equal to 0.3676, which is almost identical to $1/e$.

2 Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS) is an probabilistic and heuristic driven search algorithm (GeeksforGeeks, 2019a) which can be used to find an optimal path in a search tree. MCTS operates in the following four stages (Figure. 1):

- **Selection**: In this stage MCTS uses the Upper Confidence Bound (UCB) formula (Eq. 1) in order to select the node it wants to explore:

$$UCB(node_i) = \overline{x_i} + c\sqrt{\frac{\log N}{n_i}} \tag{1}$$

  Where $\overline{x_i}$ represents the mean node value, n_i the number of visits of $node_i$ and N the number of visits of the parent of $node_i$. C is a constant and impacts the exploration of the search process.

- **Expansion**: A random child of the current node $node_i$ is picked.
- **Simulation**: (Random) Roll-out from the picked node until a leaf-node has been reached in order to retrieve a reward value.
- **Back-propagation**: Back-propagate from the roll-out leaf-node up to the root node. Where the obtained reward is added to the node its value ($\overline{x_i}$) and the number of visits (n_i) is incremented by 1.

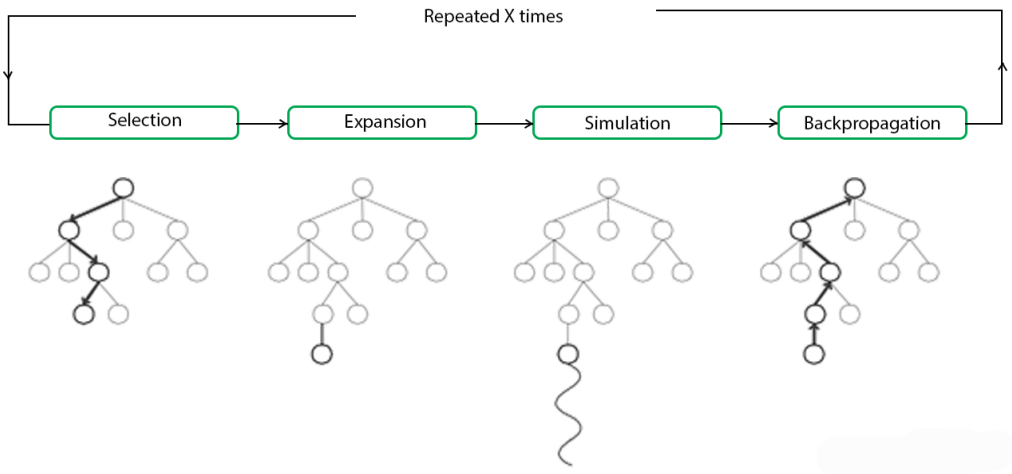

Figure 1: Stages of MCTS (GeeksforGeeks, 2019a)

Problem definition

Given a symmetric Binary Tree (BT) of depth $d = 12$, where a node's address (A) can be specified by sequentially specifying whether the left (L) or right (R) node needs to be accessed (e.g. $A = LLRRLR$). A random leaf-node (A_t) is selected as a target, of which the edit-distance $d(A_i, A_t)$ can be computed for every leaf node A_i by counting the number of positions in which the two addresses differ from each other.

Using this edit-distance, we can compute the following value x_i (Eq. 2) of leaf node i, which is a decreasing function of the distance d_i:

$$x_i = Be^{-d_i/\tau} \tag{2}$$

Where B and τ are chosen such that the leaf-nodes have a non-negligible value.

Experiment

The hyperparameters for the value x_i (B and τ) have been chosen to be set to $B = 1$ and $\tau = 1$. This way all values for x_i were non-negligible and separable from each other. The MCTS algorithm has been implemented and has ran for 1000 episodes in order to be able to take an average and compute the standard deviation. The algorithm's budget was defined such that it performed 50 iterations per episode and did it did 5 roll-outs per iteration. During these episodes, the value x_i is obtained. Where $x_i = 1$ (the higher, the closer to the target-node A_t) implies that the target has been reached. Next to that, the percentage of visited nodes (relative to the total number of nodes) is obtained in order to measure the exploration (the higher the percentage, the more explorative the search). This process has been repeated for the following assignments of the exploration-rate $C = 0, 0.5, 1, ..., 5$. Sampling this data led to the following observation:

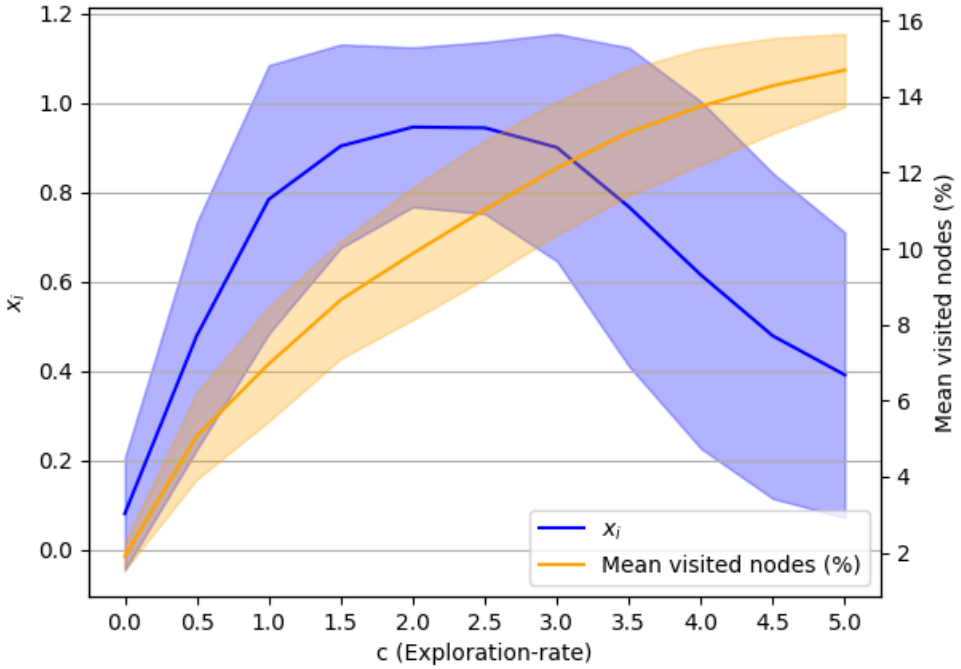

Figure 2: Caption

In the figure above (Figure. 2) we see the value for x_i and the visited node percentage per exploration-rate C, with its corresponding standard deviations. For $C = 2, C = 2.5$, the target value x_i is at its highest, almost at 1, which indicates that for around those values, MCTS performs at its best. However, we should keep in mind that the performance of these values for C might change as we change the values for B and τ or the depth d of the Binary Tree.

We also see a confirmation of the impact of C on the exploration rate, as the visited node percentage rises, as the value for C increases. The perfect balance between exploration and exploitation seems to take place for $C = 2, C = 2.5$, where the visited nodes percentage is around 10-11%.

3 Reinforcement Learning: SARSA and Q-Learning for Gridworld

SARSA (State Action Reward State Action) and Q-Learning are reinforcement learning algorithms and are designed to evaluate policies. The function for SARSA (Eq. 3) is defined as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{3}$$

Whereas the definition of the function of Q-Learning (Eq. 4) is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{4}$$

The difference between the two algorithms is that SARSA uses its current (ϵ-greedy) policy to determine the next action in the next state, whereas Q-Learning takes the best action in the next state.

Problem definition

Consider the 9x9 Gridworld illustrated in the figure below (Figure: 3):

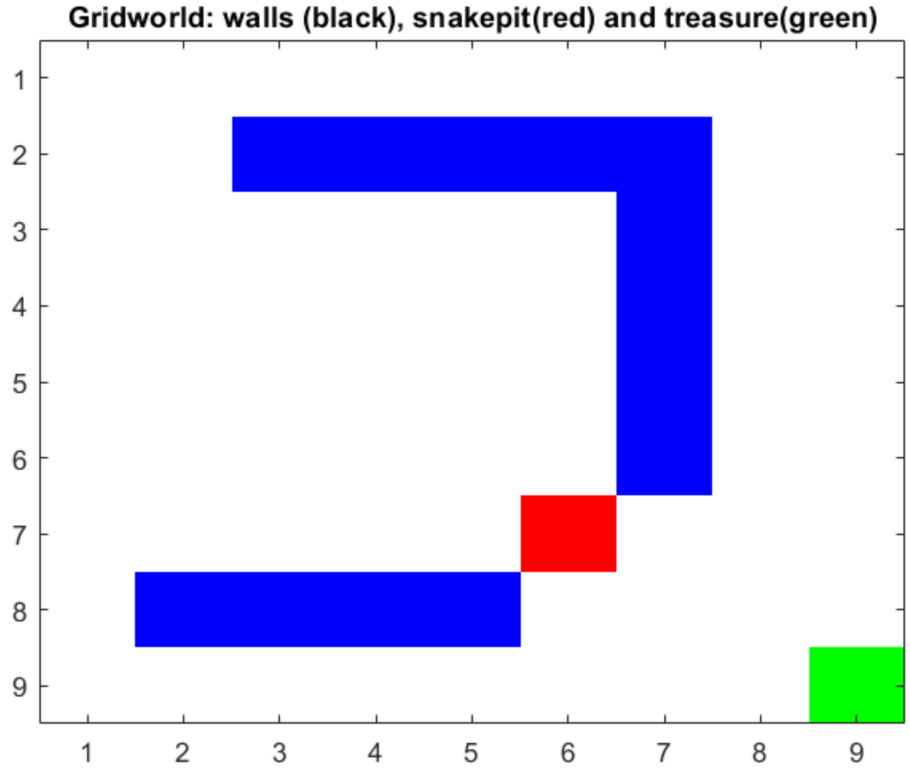

Figure 3: 9x9 Gridworld

In this Gridworld, the blue cells represent walls that cannot be traversed. The green cell represents a treasure and transition to this cell yields a reward of +50 and termination of the game. The red cell is the snakepit, is also terminating and yields a reward of -50. All white cells are accessible, are non-terminating and yield a reward of -1. The goal of the game is to terminate the game as efficient as possible and to maximize the total reward as much as possible.

Experiments

Monte Carlo policy evaluation

The state value function $v_\pi(s)$ is estimated using Monte Carlo policy evaluation. During this experiment, an equiprobable policy π has been utilized. For 5000 episodes, a game has been

played using the equiprobable policy π for every possible starting position in the Gridworld (every position except for the borders). This is also known as a systematic sweep approach. The average value function has been taken from those samples. This led to the following state value function $v_\pi(s)$ for every position in the Gridworld (Figure. 4):

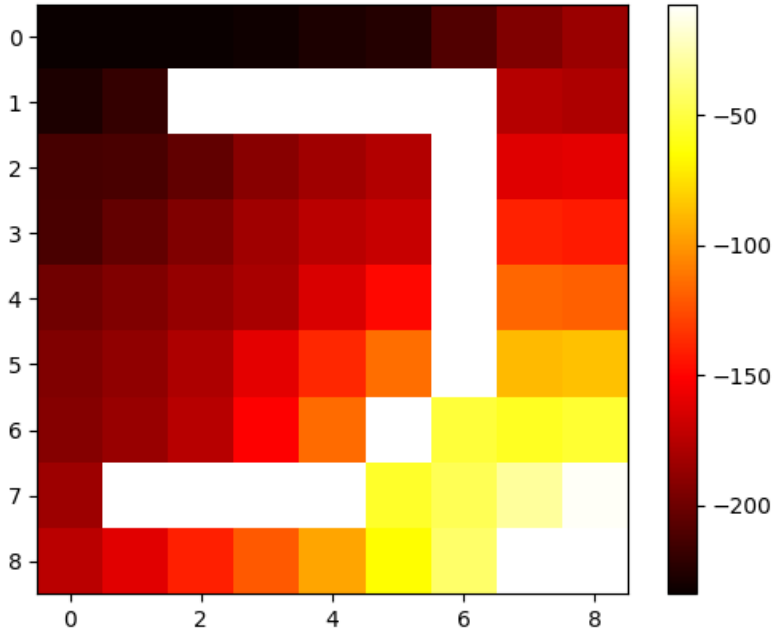

Figure 4: State value functions $v_\pi(s)$ for all positions in Gridworld, where the color represents its corresponding average total reward

As we can see, $v_\pi(s)$ increases as the position gets closer to a promising position (around (9, 9)) and decreases as we get closer to a wall, the snakepit or get further away from the treasure. From the figure above, the white cells that represent the borders can be ignored.

Optimal policies using SARSA and Q-Learning

In order to search for an optimal policy, SARSA and Q-Learning have been implemented. Both algorithms utilize an ϵ-greedy policy π. That is, given a random drawn number r, if $r \leq \epsilon$ then a random action is picked and the best action is picked otherwise. Both algorithms utilize the following hyperparameters:

- $n\_episodes = 1000$
- $\alpha = 0.1$
- $gamma = 0.99$
- $\epsilon = 0.1$

After running searching for an optimal policy using SARSA and Q-Learning with the specified hyperparameters, the following results have been obtained:

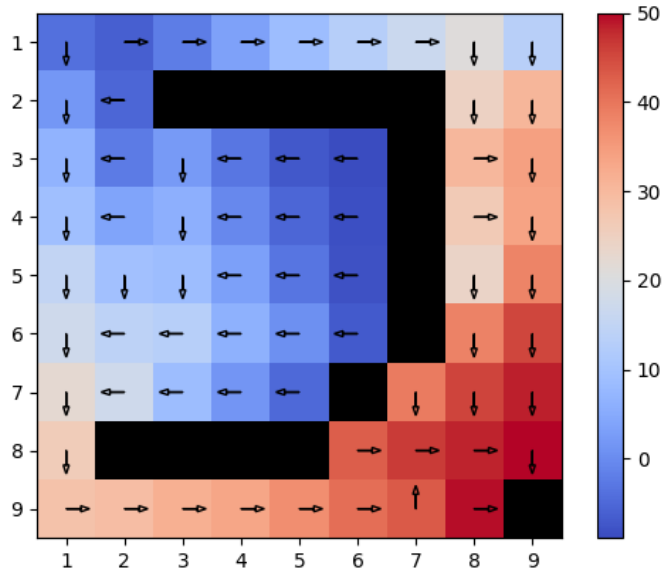

Figure 5: Optimal policy π^*, obtained using SARSA

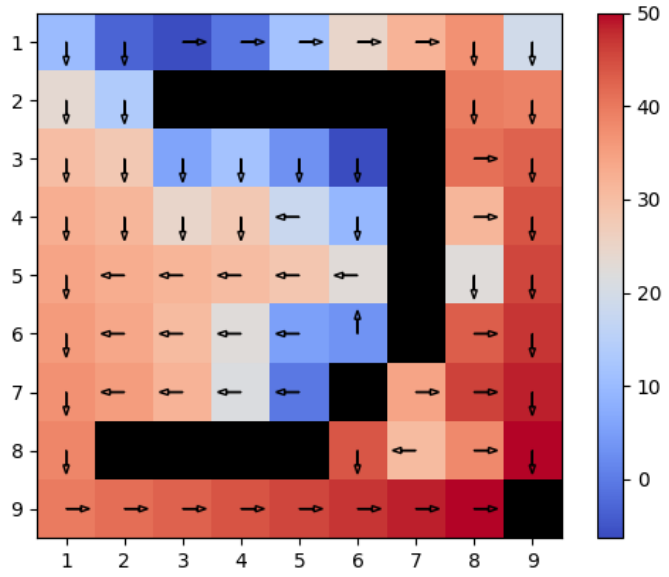

Figure 6: Optimal policy π^*, obtained using Q-Learning

In heatmaps (Figure. 5 & 6) we see the optimal policies π^* for Gridworld. The arrows represent the direction where to move towards given the current position. The colors in the heatmaps represent the Q-values of the actions indicated by the arrows. The higher this Q-value, the more 'certain' the optimal policy π^* is about the action given the state.

In the heatmap of SARSA (Figure. 5) we see that all arrows point in a direction which will eventually get the agent towards the treasure, the same holds for the heatmap for Q-Learning

(Figure. 6). However, the directions of the arrows for Q-Learning seem to be a bit more efficient. Next to that, the Q-values for Q-Learning are higher compared to those of SARSA, which indicates that the optimal policy π^* obtained using Q-Learning is more 'certain' about its actions, which is justified in this case.

References

GeeksforGeeks. (2019a). Ml — monte carlo tree search (mcts). https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/

GeeksforGeeks. (2019b). Secretary problem (a optimal stopping problem). https://www.geeksforgeeks.org/secretary-problem-optimal-stopping-problem/

Pauwels, E. (2021). Model-free methods: Monte carlo, sarsa and q-learning.