Vrije Universiteit Amsterdam

Multi-Agent Systems
XM_0052

Homework Assignment 6

Author:
Oleg Litvinov (2719624) - o.litvinov@student.vu.nl

December 30, 2021

Monte Carlo Sampling

In the first task we consider the following important sequential decision problem. This problem is renting a house in Amsterdam. There is a significant shortage of affordable housing in Amsterdam. Fortunately, we have a list of appointments where we can visit and evaluate the house. It is assumed that all houses are affordable but their "fitness" is different. Right after each visit we have to make a decision if we want to rent this place because the demand is crazy and there is no way to go back to the previous option. The problem is also named as Secretary problem. We state the task as follows:

- the number of appointments is very large, n $\to \infty$;
- we have to select one place before the end;
- the goal is to maximise the probability that you will pick the best house in the list.

For this task two approaches will be applied: the threshold one and the odds-algorithm. The latter is a well-investigated popular algorithm with expected probability $1/e = 0.368$ to select the best option. The former is a very simple idea of selecting the first appearance of a score higher than the threshold defined. Having n $\to \infty$ this threshold may be relatively high.

To evaluate the suggested algorithms, Monte Carlo sampling is used. The number of houses (options) available is set to 100000 and the number of simulations is 1000. The threshold for the first model is set to 0.999. The number of steps before stop and the ratio to the max score possible are registered. Before each simulation the house scores are shuffled. The simulation results are presented on figures 1 and 2.

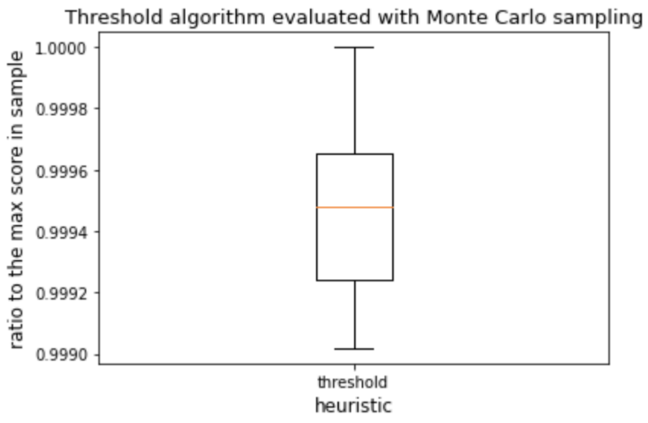


Figure 1: Threshold algorithm evaluated with Monte Carlo sampling

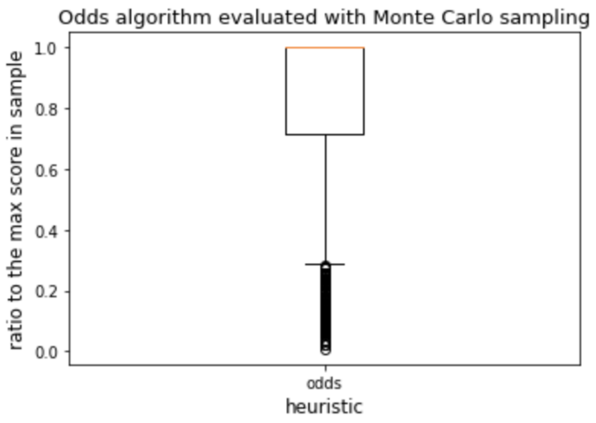


Figure 2: Odds algorithm evaluated with Monte Carlo sampling

From the distributions we see that even picking the best available flat almost every third time, the odds-algorithm has significantly worse density in terms of ratio to the max score in comparison to the threshold algorithm. This may be simply explained by the histograms 3 and 4 of the number of steps before stop. The threshold for the first model is relatively low given such long sequences of houses and therefore the needed is easily found at the beginning of the way. On the other hand, for the odds-algorithm during first $n/e = 100000/e \approx 37000$ the options are rejected and after that moment the bar is set very high and may not be beaten to the end. In such case the last option (no matter how bad it is) will be selected.

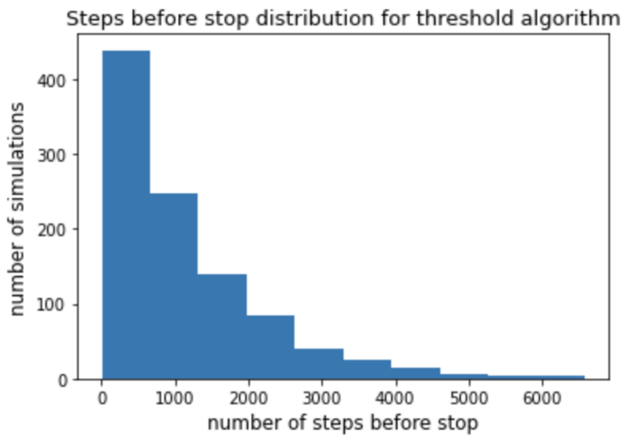


Figure 3: Steps before stop distribution for threshold algorithm

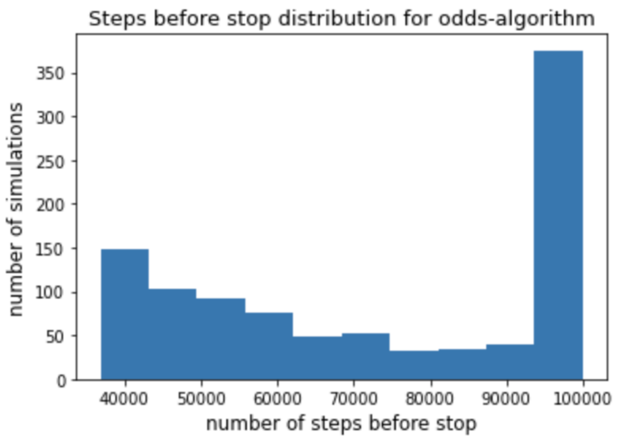


Figure 4: Steps before stop distribution for odds-algorithm

Interestingly, for odds-algorithm the ratio of the number of "best-pick" simulations to the number of simulations is 0.362 which corresponds to the theoretical results.

Apart from the renting, there are lots of similar tasks covered on the web. For example, the googol game and parking ploblem. But from the practical point of view, job/wife/employee search given the weights for their parameters and the distributions of these parameters looks like a great task to solve using Monte Carlo sampling.

Monte Carlo Tree Search

After investigating Monte Carlo sampling method, Monte Carlo Tree Search (MCTS) algorithm will be investigated and implemented.

For this, a binary tree of depth $d = 12$ was constructed. To each leaf-node a unique "address" is assigned describing the path from the root to this node.

The task setup is formulated as follows:

- a random leaf-node is picked as a target;
- for every leaf-node the edit-distance between its address and the target leaf-node address is computed;

- each leaf-node is assigned to a value from a decreasing function of the distance: $x_i = Be^{-d_i/\tau}$, where $B = 1$ and $\tau = 0.5$ are selected to keep most nodes values non-negligible. This helps distinguishing that the optimal leaf-node is found.

MCTS algorithm is used in this task to search for the optimal (i.e. highest) value. MCTS algorithm was launched with different values of hyperparameter c in $UCB(\text{ node }_i) = \bar{x}_i + c\sqrt{\frac{\log N}{n_i}}$ formula. The number of available simulations from the same root node and the number of roll-outs are also hyperparameters and were investigated.

The results of the launch with a limitation of 50 simulations per each root node are presented on figure 5. From the results we see that the number of simulations is enough to find the needed path regardless of other parameters. The hyperparameter c is responsible for the exploration part. But with the sufficient number of simulations its increase (which causes exploration increase) only delays finding the optimal leaf-node (blue line -> orange -> green). Increasing the number of roll-outs from 1 to 5 also increases the convergence time (red line vs blue; purple vs orange; brown vs green) even with the same budget of 50 simulations.

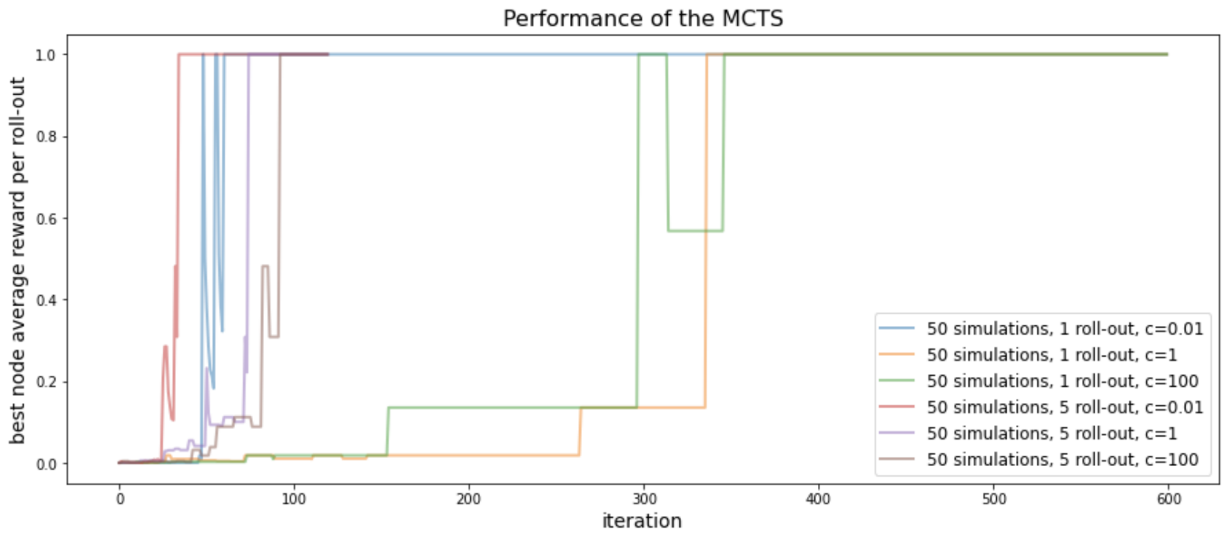


Figure 5: Performance of the MCTS with a limitation of 50 simulations per root

For the more strict limitations of 10 simulations the results 6 are more interesting. At first, we see that 1 roll-out with 10 simulations given doesn't converge to the target leaf-node. Insufficient number of roll-outs prevents from evaluating nodes properly especially on the early steps when the path to be rolled-out is relatively long. For the 5 roll-out setups, all configurations converged regardless of the exploration hyperparameter. The potential explanation of the lack of difference between red, purple, and brown lines is that c doesn't affect the exploration a lot with such small number of simulations and the diversity is reached because of the roll-outs. Finally, with 10 simulations the convergence is achieved much earlier than with 50 simulations.

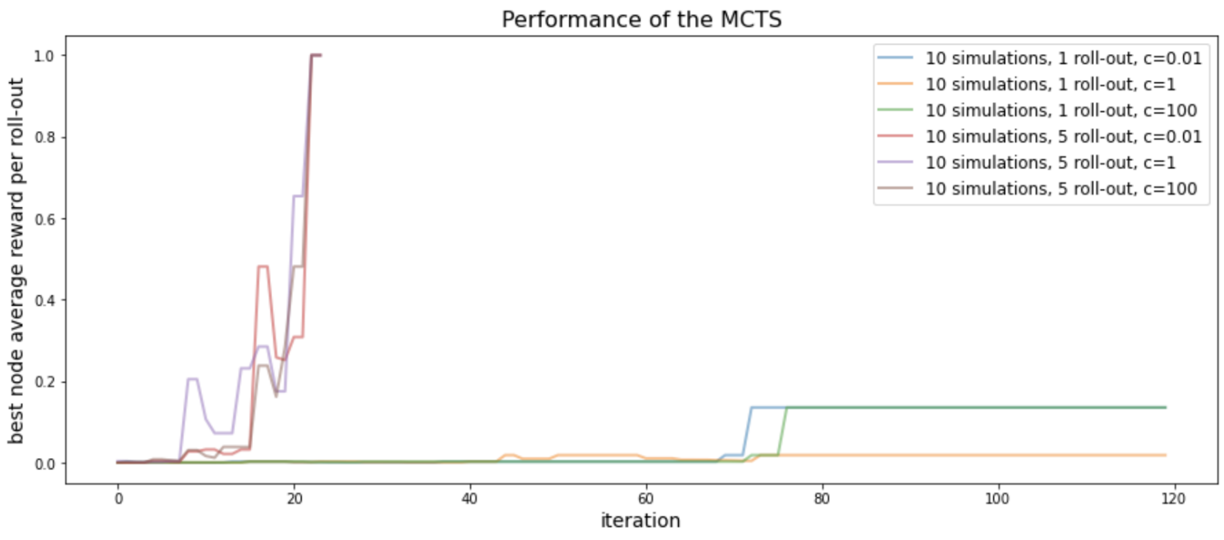


Figure 6: Performance of the MCTS with a limitation of 10 simulations per root

Reinforcement Learning: SARSA and Q-Learning for Gridworld

In the last section of the paper, SARSA and Q-Learning algorithms were investigated.

The 9 ×9 gridworld 7 was considered. The blue gridcells represent walls that cannot be traversed. The green cell represents a treasure and transition to this cell yields a reward of +50 whereupon the episode is terminated (i.e. absorbing state). The red cell represents the snakepit: this state is also absorbing and entering it yields a negative reward of −50. All other cells represent regular states that are accessible to the agent. In each cell, the agent can take four actions: move north, east, south or west. These actions result in a deterministic transition to the corresponding neighbouring cell. An action that makes the agent bump into a wall or the grid-borders, leaves its state unchanged. All non-terminal transitions (including running into walls or grid borders) incur a negative reward ("cost") of −1.

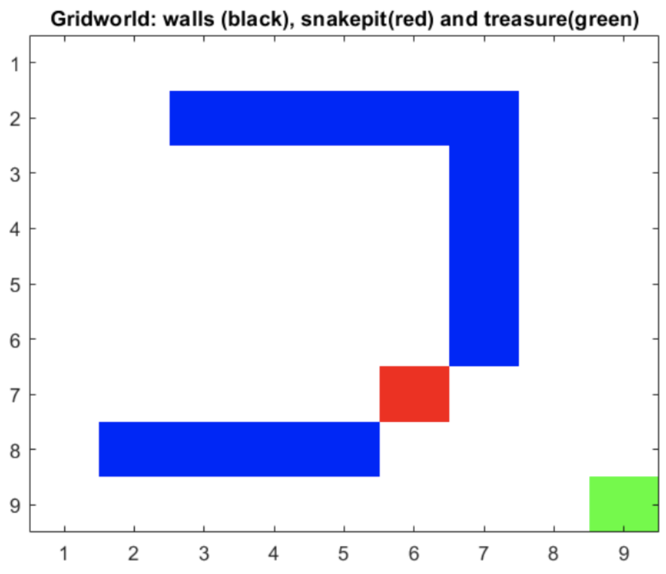


Figure 7: Grid world of size 9 ×9: Walls are blue, snakepit red, and treasure green

The environment from the above was implemented as a reinforcement learning environment and was launched "against" three different approaches:

- Monte Carlo policy evaluation to compute the state value function $v_\pi(s)$ for the equiprobable policy π

- SARSA in combination with greedification to search for an optimal policy;
- Q-learning to search for an optimal policy.

To evaluate the equiprobable policy via Monte Carlo, a sweep every-visit Monte Carlo algorithm was implemented. Sweep means that each state is taken as the initial one after another in a circle. The desired state values were evaluated over 1000 episodes for each state and are depicted on 8.

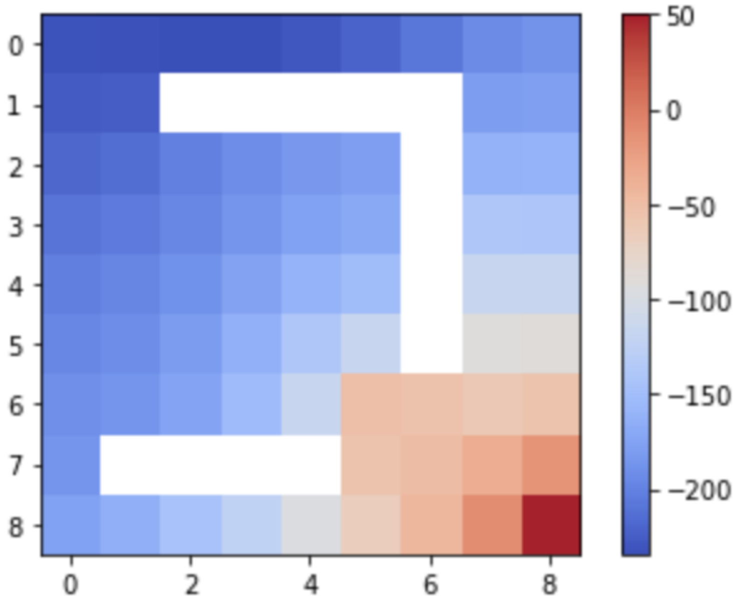


Figure 8: State values for the equiprobable policy evaluated with Monte Carlo

SARSA and Q-learning were both launched for evaluating optimal policy. Both algorithms were used with ϵ-greedy strategy and the parameter value of 0.05. For depicting their optimal state action tables, a special tool was implemented. The results were obtained on 100000 episodes for both and are presented on 9 and 10. The colour represents the optimal state values or, in other words, the best action value per state. The arrows point to the direction of the optimal action in a given state. The size of the arrow is calculated via normalizing the action vector.

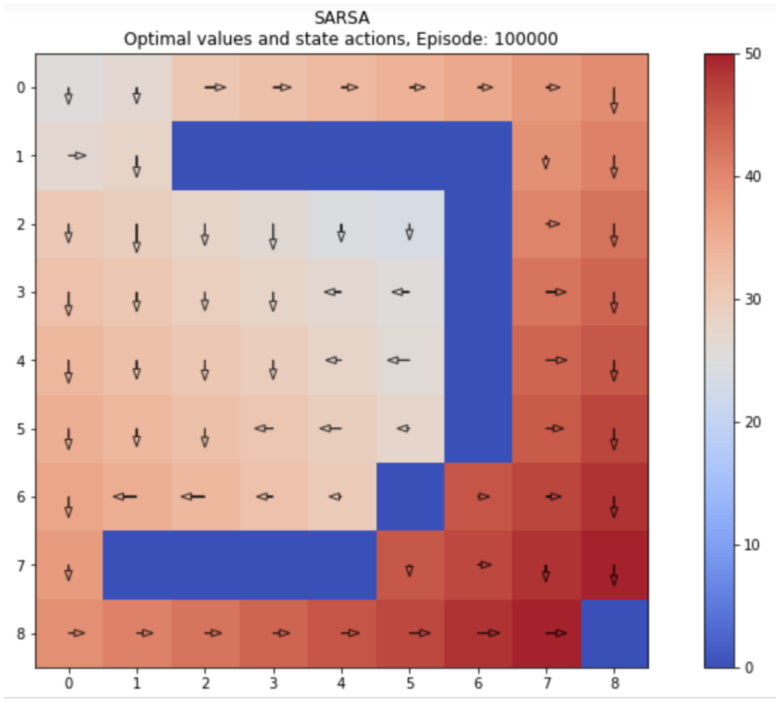


Figure 9: Optimal policy and optimal state values evaluated with SARSA on 100000 episodes

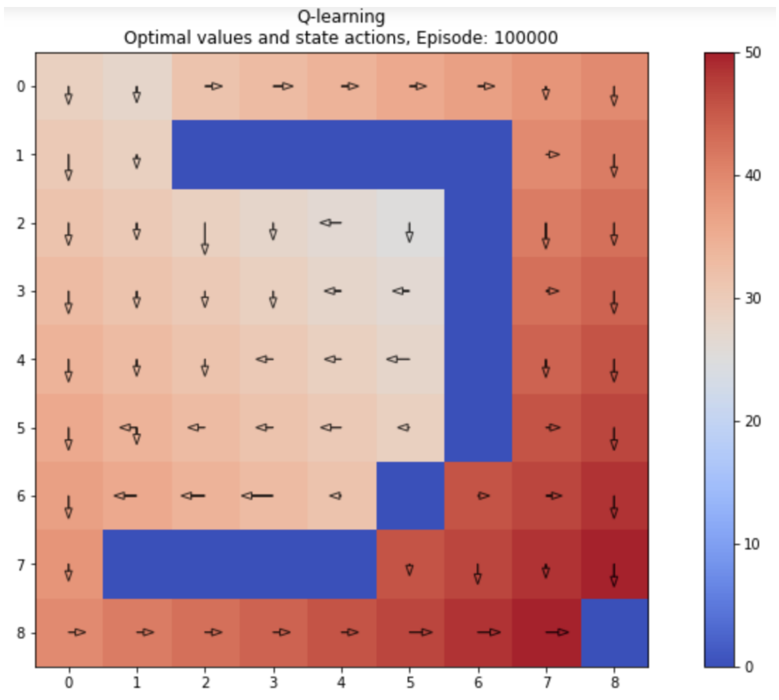
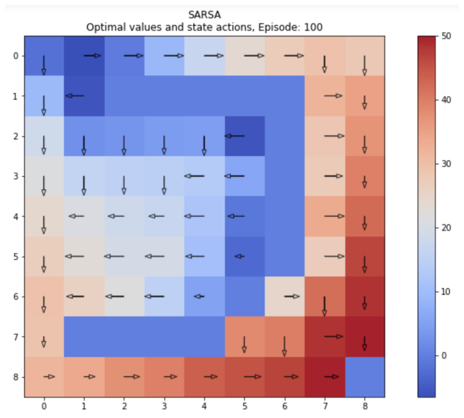
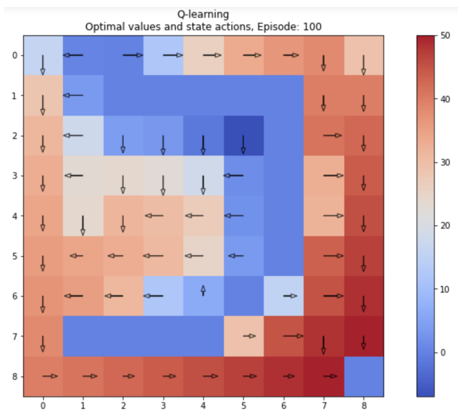


Figure 10: Optimal policy and optimal state values evaluated with Q-learning on 100000 episodes

On the sufficient amount of episodes both algorithms perform similarly. But Q-learning is more sample efficient and aimed to learn value function for optimal policy π^*. This difference is clearly visible on figure 11 where the number of training episodes is limited to 100. The arrows to the optimal directions are longer on the right image and the colours are also more intense because of more efficient sample usage.



(a) SARSA



(b) Q-learning

Figure 11: Comparison of optimal policy and optimal state values found by SARSA and Q-learning after 100 episodes