Multi-Agent Systems

Final Homework Assignment

MSc AI, VU

E.J. Pauwels

Version: December 12, 2021— **Deadline: Thursday 30 Dec 2021 (23h59)**

IMPORTANT

- This project is an **individual assignment.** So everyone should hand in their own copy.
- The questions below require some amount of programming. In addition to the report (addressing the questions and discussing the results of your experiments), also provide the code on which your results are based (e.g. as Python notebooks). However, make sure that the **report is self-contained**, i.e. that it can be read and understood without the need to refer to the code. Store the pdf-report and the code in a zipped folder that you upload via canvas.
- This assignment will be **graded.** The max score is 4 and will count towards your final grade.
- Your **final grade (on 10)** will be computed as follows:

  Assignments 1 thru 5 (max 1) + Individual assignment (max 4) + Final exam (max 5)

1 Monte Carlo Sampling (20%)

Use MC sampling to get insight in the following important sequential decision problem. This problem surfaces in different guises in lots of real world applications, but we will phrase it in terms of house rental.

Renting a house in Amsterdam As you are probably well aware there is a shortage of affordable housing in Amsterdam. As an upward moving young professional you're interested in renting your first home. You've contacted a real estate agent who has arranged n appointments for site visits of houses that fit within your budget. These visits are scheduled at specific dates and times, so you will visit the houses in pre-fixed order (which you can think of as a random permutation of the sequence $1, 2, 3, \ldots, n$). Assume that at the end of each visit you are able to assign a score $0 \leq x \leq 1$ that captures your preference: E.g. $x = 1$: *OMG!! My dream house!!* ; $x = 0$: *no way i'm living in this dump!* ; $x = 0.2$: *hmmmm, . . . only if the rent is really low;* ; $x = 0.8$: *this house has a lot of potential, . . . etc.*

Assumptions We'll make the following assumptions:

1. You really need a place to live, so you do have to pick one of the n options.
2. If you could visit all the houses before making a decision, life would be easy: after each site visit you assign a score and when you've seen all n of them, you pick the house with the highest score (for simplicity's sake, we assume there are no ties, so there is a unique ranking).
3. However, life isn't that simple. At the end of each visit you have to decide on the spot whether or not you will rent the place. If you decide not to, then someone else will immediately snap it up. Put differently, changing your mind at at later stage is impossible: your decision at the end of the visit is final.
4. Your goal, of course, is to maximise the probability that you will pick the best house in the list.
5. Assume that n is sufficiently large, in other words, we are really interested in the asymptotic behaviour as $n \to \infty$.

Questions

1. What strategy would you use? Use MC to explore possible approaches and/or motivate your answer.
2. Can you suggest interesting variations on this question (that I could maybe use in next year's assignment :-)? No need to provide a solution, formulating the question suffices.

PS This is an exercise on MC simulation, so **no need** to cast this problem into the framework of sequential game theory.

2 Monte Carlo Tree Search (MCTS) (30%)

Construct binary tree Construct a *binary tree* of depth $d = 12$ (or more – if you're feeling lucky). Since the tree is binary, there are two *branches* (aka. edges, directions, decisions, etc) emanating from each node, each branch (call them L(eft) and R(ight)) pointing to a unique child node (except, of course, for the leaf nodes – see Fig 1). We can therefore assign to each node a unique "address" (A) that captures the route down the tree to reach that node (e.g. $A = LLRL$ – for an example, again see Fig 1).

Assign values to leaf-nodes Next, assign to each of the 2^d leaf-nodes a value as follows:

- First, pick a random leaf-node as your target node and let's denote its address as A_t.
- For every leaf-node i compute the **edit-distance** between its address A_i and A_t, i.e. $d(A_i, A_t)$.

- Recall that the **edit-distance** counts the number of positions at which two strings differ, e.g.:

$$d(LRLR, LRRR) = 1, \qquad d(LRRL, LLLL) = 2, \qquad d(RRLL, RLRR) = 3$$

- Finally, define the value x_i of leaf-node i to be a decreasing function of the distance $d_i = d(A_i, A_t)$ to the target node: e.g.

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

  where B and τ are chosen such that most nodes have a non-negligible value.

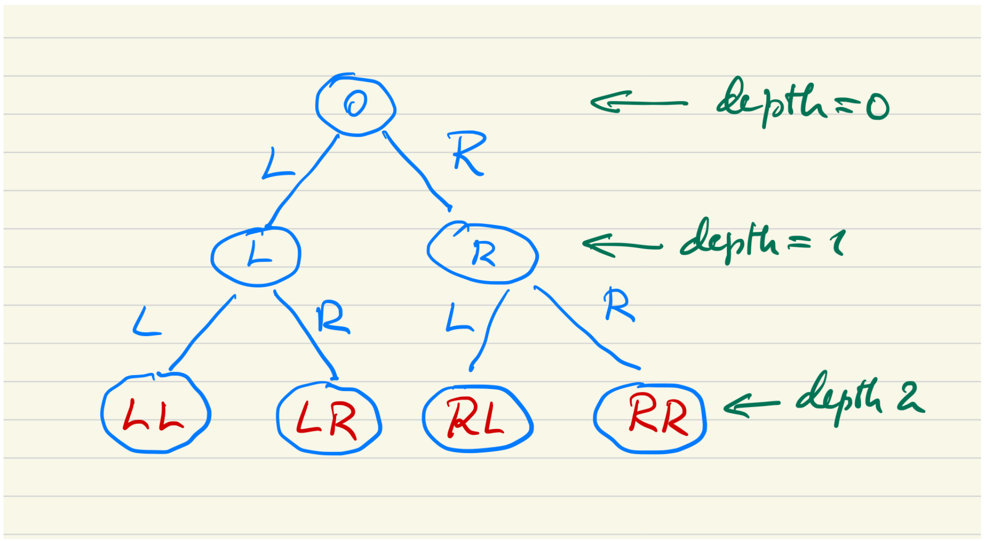

Figure 1: Example of binary tree of depth 2. The “address” of the leaf-nodes (layer 2) is indicated in red.

Questions

- Implement the MCTS algorithm and apply it to the above tree to search for the optimal (i.e. highest) value.
- Collect statistics on the performance and discuss the role of the hyperparameter c in the UCB-score.

Assume that the number MCTS-iterations starting in a specific root node is limited (e.g. to 10 or 50). Make a similar assumption for the number of roll-outs starting in a particular ("snowcap") leaf node (e.g. 1 or 5).

3 Reinforcement Learning: SARSA and Q-Learning for Gridworld (50%)

Consider the 9×9 gridworld example depicted in the Fig. 2 below. The blue gridcells represent walls that cannot be traversed. The green cell represent a treasure and transition to this cell

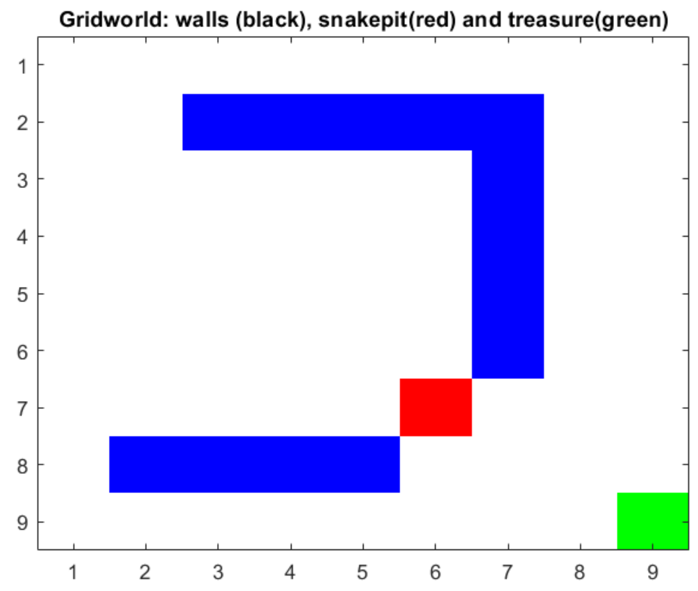

Figure 2: Grid world of size 9×9: Walls are blue, snakepit red, and treasure green.

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.

For the questions below, we assume that the agent is not aware of all the above information and needs to discover it by interacting with the environment (i.e. model-free setting). Perform, and comment on, the following experiments:

- Use Monte Carlo policy evaluation to compute the state value function $v_\pi(s)$ for the **equiprobable policy** π (i.e. all 4 actions have probability 1/4). Visualize the result (e.g. by using some sort of heat map where the color of each cell corresponds to the state value).
- Use SARSA in combination with greedification to search for an optimal policy.
- Use Q-learning to search for an optimal policy. Compare to the solution obtained by SARSA.