Queen Mary University of London

ECS629U-759P: ARTIFICIAL INTELLIGENCE

2019/20 – Semester 2

Brice Denoun
Dr. Arman Khouzani, Dr. Georgios Tzimiropoulos

Coursework 1 (12%): *Search Problems*

Winter 2020

DUE DATE: PLEASE REFER TO THE EXACT DATE FROM QMPLUS
NO LATE submission is allowed: submission link will disappear upon deadline!

Instructions: The coursework is composed of three questions. The first two involve coding exercises. The last one is a writing exercise about adversarial search. You are expected to submit two files ONLY through the QM+ submission link:

1. A single **PDF** file that must be **at most 5 pages long**. Any file submitted under the wrong format (e.g. `doc`, `docx`, `odt`, etc) or exceeding the length limit may be penalised. The PDF can include scans of handwritten material as long as it is legible but a typewritten version is encouraged (even better if you use $\LaTeX$). **ATTENTION: Do not include the text of the question in your answers. Failure to do so may disqualify your submission!**

2. A *single compressed folder* containing all your (well documented and sufficiently commented) codes. The codes can be in any language, although `Python/java/C/julia/JS/Lisp` are preferred in that order. Your codes should run (no bugs!). There must be an easy to follow `readme.txt` file inside the folder.

It is allowed to use codes from online resources. However, this has to be clearly cited with reference in your report. Collaboration is NOT permitted when attempting the answers. High level discussion when not attempting the questions may be fine, but discouraged. A better means is using our discussion forum on QM+. There is zero tolerance policy for cases of plagiarism. If in doubt, ask! Please be aware that systems can be busy and slow to respond shortly before deadlines. So you should aim to submit at least one hour before the announced deadline.

Question	Points	Score
Let's Travel! (Uniform Cost Search)	35	
It's on the tip of my tongue! (Genetic Algorithm)	35	
Let's play a game! (Adversarial search)	30	
Total:	100	

Question 1: Let's Travel! (Uniform Cost Search) . . . .

A friend of yours gave you a map of UK, but encoded as a `json` file: `UK_cities.json`. The file encodes all the roads between major cities of the UK with their length of the road (in km). After numerous arguments, you both decided to go from London to Aberdeen. Although you would like to go there while visiting as many cities as possible, neither you nor your friend can afford the time! You are then trying to reach Aberdeen from London taking the shortest path (the total driving distance).

(a) (15 points) Following the UCS algorithm, manually execute it for three iterations for this problem. As a reminder, the pseudo-code of the UCS is provided in Figure 1, which matches Figure 3.13 in the textbook (3rd edition).

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier ← INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child
```

Figure 1: Pseudo-code for the USC (Uniform-Cost Search) algorithm on a graph. This is borrowed from [1, Fig. 3.13].

- By iteration, we are referring to the outer "loop". So, in your trace, at least three nodes should be selected for expansion.
- In particular, you should provide (only) the content of the following variables, with the following specific data structure:
  (i) the "frontier": a queue of "nodes" ordered by their path-cost
  (ii) the "explored": a set of "states"
- You should represent each "node" in the frontier as a tuple of the following information: ("state", "path-cost", "best-path-to-reach-that-state"). Note: the "state" in our problem is just the name of the city you are at! The "path-cost" is just a positive real number; and the "best-path-to-reach-that-state" is a list of states starting from the initial state to that node's state.
- For the trace, you should provide only one value per each line. Write down only the new value each time any of the above two variables change. The order definitely matters.
- If you encounter any tie-breaking situation in the executing, e.g. in adding the nodes to the frontier, go with the alphabetical (lexicographical) order based on the name of the city.
- No explanation or comments are necessary. For instance, you don't even need to tell which line/part of the pseudo-code was responsible for an update. We only check whether the trace is correct (including the order of updates).

- An example few first steps of the trace with the above description would be as follows:

```
frontier = [('london', 0, ['london'])]
explored = {}
frontier = []
explored = {'london'}
frontier = [('birmingham', 110,['london','birmingham'])]
...
...
...
```

(b) (10 points) Implement the UCS for this problem. Run it with the provided data. You don't need to include the code in your report (you submit it separately). Instead:

(b-1) Provide the optimal path found by the algorithm (both the path and its length);

(b-2) Make the code print the trace (in the same format as requested in part (a) but only provide the *last two* iterations (of the outer loop) in your report.

Note: you don't have to follow the provided pseudo-code in your implementation necessarily. However, if the logic of your code deviates significantly from this version, then you need to provide your pseudo-code as well.

(c) (10 points) Suppose both you and your friend are environmentally aware. So you would like to find the path that has the lowest overall cost, which is *the sum of the overall driving time plus the overall cost of the air pollution due to your driving.*

Here are the details:

- You are free to choose your driving speed on each road of the path (but assume that you don't change your speed on a given road). So e.g. if you choose to drive at 40 km/h on the road between London to Birmingham, you maintain that speed on this road. But you are free to choose a different speed on the next road, etc.).
- If you drive at speed v km/h (on any road), you are responsible for an air pollution cost of $(0.00001v^2)$ units *per hour*.

Using your answer to the previous part, provide the best path in this scenario (both the optimal cost and the path). Again, you don't need to provide the code in your report, but you need to briefly explain your approach.

(d) (10 points (bonus)) Now, for a bonus 10 points!, suppose you have rented a super-car with the maximum speed of 300 km/h with the rental fee of 100 units per hour. Suppose you have no regard for the environment! Moreover:

- As in the previous part, you can choose your driving speed per each road.
- However: suppose each road now has a speed-limit. For simplicity, assume that the speed limit of a road is the same as the value provided for the length of that road in the `json` file.
- On each road, you can even choose to go over the speed limit! However, there is a likelihood that you get caught on a speed-camera and get a fine of 1000 units. In particular, if you drive on a road with speed limit of v_{lim} at speed v, then the likelihood (i.e., probability) of getting the fine is as follows:

$$\begin{cases} 1 - e^{-(v - v_{\text{lim}})}, & \text{if } v > v_{\text{lim}} \\ 0, & \text{if } v \leq v_{\text{lim}} \end{cases}$$

- You may only be fined at most once on a given road, but fines on different roads accumulate.

Using your UCS code, find the best path in this problem, assuming the overall cost is the sum of the car rental fee plus the total (likely) fines.

Question 2: It's on the tip of my tongue! (Genetic Algorithm). . . . .

Becoming old is certainly difficult. And one of the signs of growing older is some form of amnesia. As a matter of fact, I cannot exactly remember my two favourite passwords, but still, I have a hazy idea about what it looked like!

Your goal is to help me remember these two passwords using a genetic algorithm: It turns our even though I am amnesiac, I can still tell how close to my password a candidate is. Let me give you the few details I remember with certainty: the password was exactly **10 characters long** and could only contain upper-case English letters, numbers between 0 and 9, and the underscore symbol _. For instance, `1_PASSWORD` and `MYPA55W0RD` are legitimate candidates.

So here is the deal: For each password you show me, I am going to give a score between 0.0 and 1.0 indicating how close to my forgotten password I feel it is (where 1.0 means that the password is exactly correct) and despite my memory, I have a trustworthy intuition!

To be more precise, you have access to a function called `blurry_memory` as follows:

input: a list of candidate passwords, e.g. ["`___TEST___`", "`IS_I7_L0V3`"], along with your own student number (e.g 190123123), and the index of the password you are trying to find: 0 or 1 (recall: there are two passwords to recover).

output an ordered dictionary containing the passwords as keys and the associated scores as values (e.g "`___TEST___`": 0.045, "`IS_I7_L0V3`": 0.247)

To access this function, you can add the following line to the header of your python3 code:

```
from amnesiac import blurry_memory
```

Implement a Genetic Algorithm to recover the two forgotten passwords.

What should you provide in your report:

(a) (10 points) What did you get for the two passwords? (recall: you should use your own student ID number whenever invoking `blurry_memory`).

(b) (5 points) Explain *briefly* how specifically you chose to do the state encoding, selection, crossover, and mutation.

(c) (10 points) Report on the number of reproductions you had to do to converge to the password? Ideally, run your code multiple times and report on the average and variance instead of a single run (since Genetic Algorithm involves randomisation).

(d) (10 points) Explore the effect of at least one hyper-parameter on the performance of your algorithm. This could for instance be the choice of the mutation rate, or the selection/pairing scheme, the population number, etc. Ideally, provide a table or a figure with a very concise discussion.

Question 3: Let's play a game! (Adversarial search) . . . .

You are now engaged in a thrilling two-player adversarial game which is represented by the following tree. Suppose you are the MAX player (who goes first).

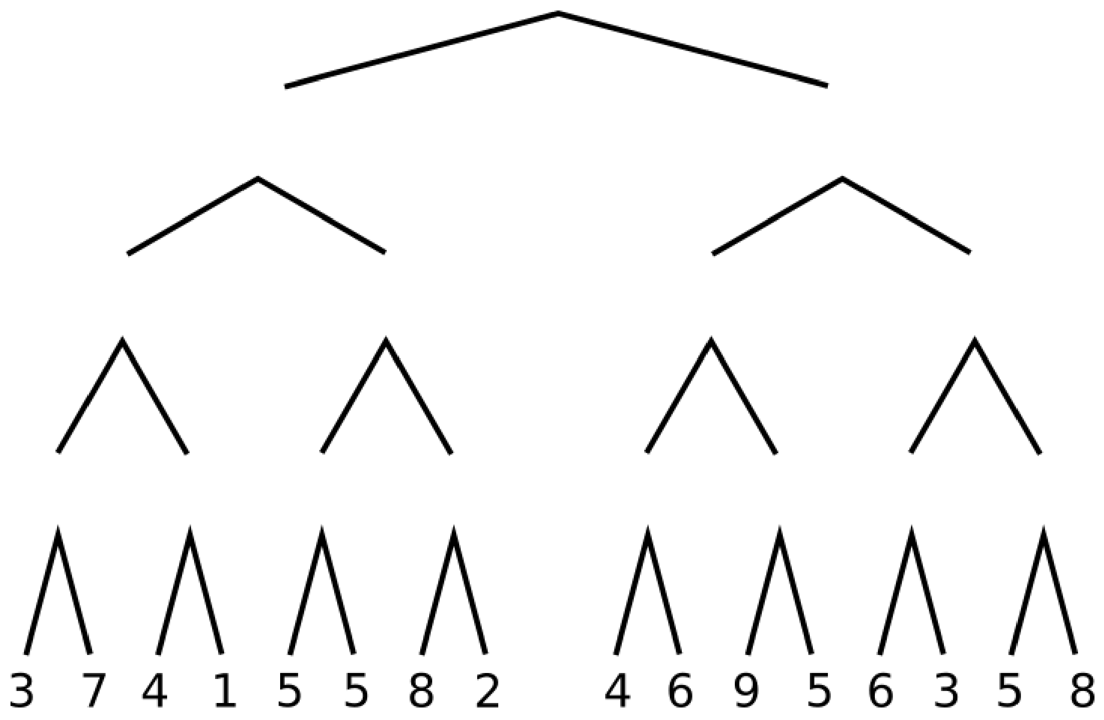

Figure 2: Tree of the game

(a) (10 points) **Play optimally (MINIMAX algorithm)**

Knowing that you start (top of the tree is you), complete the blank nodes in the provided tree using the MINIMAX approach.

(b) (10 points) **Play optimally, quicker thinking! (α-β pruning)**

Suppose there is a mean arbiter that doesn't like too much hesitation at the beginning. Perform α-β pruning on the tree you filled, searching from left to right. Which branches are pruned and why?

(c) (10 points) Suppose the entry fee to play the game is $x \geq 0$ units. In particular, this is the money that the MAX player pays the MIN player to be allowed to start the game. The rest of the game is as before. So for instance, if the terminal state of the game is the left-most leaf in Figure 2, then the MAX player wins from the MIN player 3 units (so the MIN player has to pay MAX player 3 units). Note that this is independent of the entry fee. So effectively, in this scenario, the MAX player has won $3 - x$ units overall, exactly how much the MIN player has lost.

c-(1) What are the ranges of values for x that will be still worth playing the game? (to enter the game at all). less than eq 5

c-(2) For a fixed value of x, do you prefer to be the first player (MAXimiser) or the second player (MINimiser)? Very briefly explain your answer.

depends on x

References

[1] Stuart J Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 2009.