Introduction to Artificial Intelligence

Assignment 1

1. [20 points] Suppose we have an agent who works for the Hunt Library. The agent is assigned a task of transferring a set of boxes one-by-one by lifting them from location A and placing them in location B inside the building. Sensors can tell it whether the agent is near its destination or not. The rooms have stationary obstacles whose locations are not initially known. If the agent bumps into an obstacle, it will drop the box it is carrying. Some boxes contain fragile contents which will break if dropped. The environment contains obstacle-free paths, of various lengths. Answer the following questions about the agent. Wuf.

a. Define a PEAS specification for the agent.

The PEAS specification for the agent is as follows:

Agent Type: Box Transferring Robot

Performance Measure:

1. Make sure all boxes reach the destination.
2. Minimize breakage and prioritize box safety(especially the fragile ones). Transfer the boxes to the destination unscathed.
3. Learn and analyze the shortest path from location A to location B based on the percept history to increase efficiency and reduce the amount of energy spent.

Environment:
Library, Boxes, Obstacles, Rooms, Obstacle-Free paths

Actuators:
Lift Fork, Wheels

Sensors:
Distance Sensor, Direction Sensor(whether going towards A or going towards B), Package drop Sensor(to notify that the package dropped of, and it needs to pick the package up again).

b. Is it sufficient for the agent to be simple reflex? Why or why not?

If we talk about it being 'SUFFICIENT', then yes we can say that its sufficient for the agent to be simple reflex, but this comes at the cost of decreased efficiency and increased chances of good's breakage.

So, ideally it should **not** be a 'Simple Reflex Agent' so that it can avoid collision with obstacles and work with increased efficiency. This can be achieved by storing the percept input, analyzing the percept history and learning from the previous data. Learning how to avoid collisions, taking the shortest route possible.

c. Would the ability to move randomly help agent performance or not? Identify possible disadvantages to this sort of movement.

- If the agent is a "SIMPE REFLEX" agent, then the random movements might help. Since the agent is unaware of the location of the obstructions, we can say that this is a partially observable environment, which can result into infinite loops for the agent. We can escape from infinite loops by randomizing the actions of the agents.

  For example, if there is an obstruction near location A at a particular coordinate and if the agent choses that path every time, then the collision chances would be 100%. On the other hand, if it moves randomly but towards the direction, it wont collide with the same obstruction all the times.

  Moving randomly also comes with some disadvantages like decreased efficiency for one. The agent is moving randomly and the direction of movement might not necessarily be towards the destination. This can result in longer time to complete one trip and hence increased energy consumption.

- On the other hand, if the agent is a sophisticated agent which can analyze the shortest path and/or avoid collisions using percept history by maintaining an internal state, then randomizing will only worsen the performance of the agent. For example: it might deviate from the shortest path as a result of randomization. Or it might collide with an obstruction even after knowing their location. Or it might not even reach the destination.

d. Suggest one improvement to the agent design. Since every improvement carries drawbacks, what are the drawbacks to yours?

The agent should definitely have a sonar or cameras which will help it detect obstructions on the way. This will minimalize collisions and damage to the goods, which increases the output and completes the goal of the agent.
So, when the agent is moving forward, using sonar it can continue to monitor if there is an obstruction in front and it can get the distance from that obstruction. As soon as the distance is less than some threshold value, the agent can change course and avoid collision.

The disadvantage of this improvement would be **increased costs** and **energy consumption**.

2.a. Describe a PEAS (R&N Ch. 2) specification for Watson.

Performance

1. Watson should be able to comprehend the input and convert the speech into fragments to information which can be deciphered by it.
2. It should be able to quickly search for the answer in the database. It might not find direct answers to questions, so it should comprehend the gist of the question and then quickly find the answer.
3. It's important that the results are searched accurately and quickly as it's a game show.
4. Watson should also be aware of the game rules

Environment

Game show audience, host, fellow contestants(competitors), audio input, video input. The environment is observable, stochastic, sequential, dynamic, continuous and multi-agent.

Actuators

1. A robot arm to buzz the buzzer
2. Voice output to speak out the answer.

Sensors

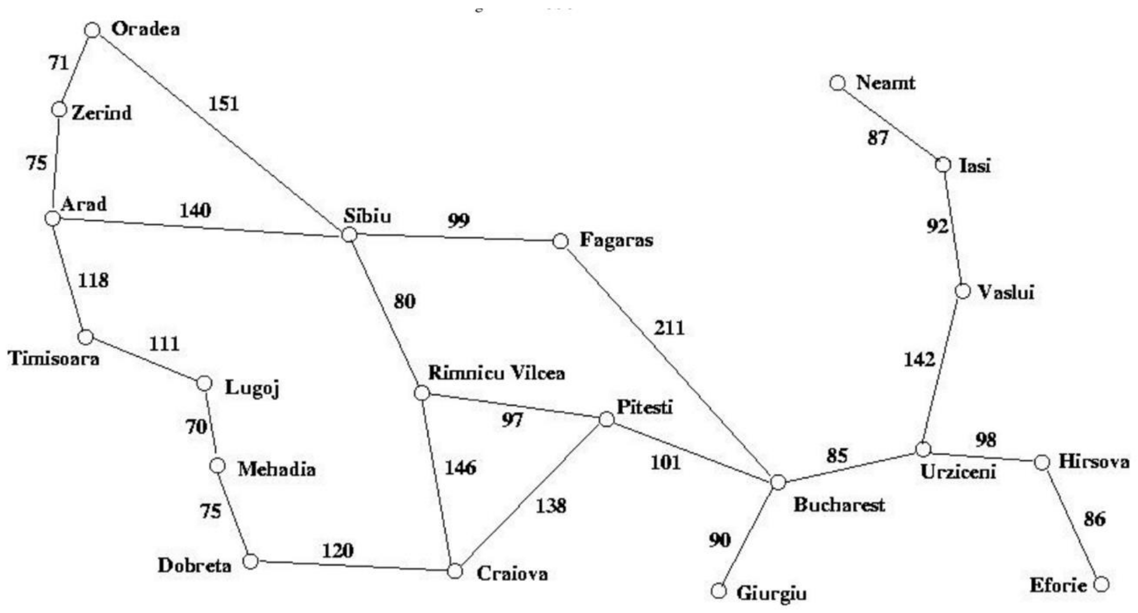
1. Audio Input
2. Video Input to answer video/image related questions.

c. According to our various spectra for characterizing environments, state whether Watson's environment is

1. fully or partially observable,
2. deterministic or stochastic,
3. episodic or sequential,
4. static or dynamic,
5. discrete or continuous, and
6. single or multi-agent.

Stochastic, partially observable, sequential, continuous, multi-agent environment and dynamic

3



a. [20 points] Consider the path from Zerind to Craiova and the path from Craiova to Zerind. Run your algorithms and show the paths returned by DFS and BFS results for each case.

Depth-first(Zerind-Craiova):-

Zerind Sibiu Fagaras Bucharest Giurgiu Pitesti Craiova

Breadth-first(Zerind-Craiova):

zerind arad oradea sibiu timisoara fagaras rimnicu_vilcea lugoj bucharest Craiova

We can see that since the destination is pretty deep down in the tree structure, Depth-first is better than Breadth-first.

Breadth-first(Craiova-Zerind):

craiova dobreta pitesti rimnicu_vilcea mehadia bucharest sibiu lugoj fagaras giurgiu urziceni arad oradea timisoara hirsova vaslui zerind

Depth-first(Zerind-Craiova):-

craiova dobreta mehadia lugoj timisoara arad sibiu fagaras bucharest giurgiu pitesti rimnicu_vilcea urziceni hirsova eforie vaslui iasi neamt oradea zerind

b. [10 points] Is there a case where Depth-First performs worse (in terms of number of cities visited in the path, not the distance) than Breadth-First in your implementation? If yes, what is the case? If not, explain why.

There are some cases where Breadth-First easily perform better than Depth-first. For example, in cases where the city is closer to the top of the tree.

Lets say if we start searching from Zerind and we would want to reach Sibiu or Timisoara, we can quite clearly see that these cities are quite close to the top of the tree(which is Zerind). So when we expand the nodes breadth first rather than depth first we will encounter Sibiu(or Timisoara) with fewer no. of city traversals.

c. [10 points] Is there a case where Breadth-First performs worse (in terms of number of cities visited in the path, not the distance) than Depth-First in your implementation? If yes, what is the case? If not, explain why.

There are many cases where Depth-first performs better than Breadth-first. For example, in cases where the city is located deep down in the tree and we need to minimalize the no. of traversals through cities, we will opt for Depth-First as it focuses on expanding the nodes depth-wise.
Depth first is also better in terms of space consumption when compared with breadth-first as it is a linear function of nodes while breadth first is an exponential function.

As we proved in the (a) part of this question, if we want to go from Zerind to Craiova, since Craiova is pretty deep down in the tree, depth first will give you better results with minimal city traversals.

d. [20 points] For the same graph, perform a hand-execution of Depth-First Iterative Deepening (DFID) with increment and cutoff initialized to 1, starting at Bucharest. List the nodes in the order expanded for the first five iterations of DFID, and the state of the data structure (stack) after each iteration. Expand the nodes alphabetically and insert them in nondecreasing alphabetical order. How does this list compare with the list of expansions in Breadth-First Search?

- For cutoff = 1:
  The stack is initialized with Bucharest.
  Bucharest is removed from the tree and put into the stack. Then it is checked, if Bucharest is at the max depth bound(which is 1). So, we don't expand Bucharest in this case.
  Stack: {Bucharest}
  Order of expansion: Bucharest

- For cutoff = 2:

The stack is initialized with Bucharest, then Bucharest is pushed into the stack and checked for the following.

Stack: {Bucharest}
Order of expansion: Bucharest

Since Bucharest is not at the maximum depth bound we expand it and get the children {Fagaras, Giurgiu, Pitesti, Urziceni}. We remove Bucharest from the stack and add the children in alphabetical order.

Stack: {Fagaras, Giurgiu, Pitesti, Urziceni }
Order of expansion: Bucharest, Fagaras

Now, we check for Fagaras, since its at the max depth, we cannot expand it, we remove it from the stack and check for Giurgiu.

Stack: { Giurgiu, Pitesti, Urziceni }
Order of expansion: Bucharest, Fagaras, Giurgiu
Similarly we check for Giurgiu, Pitesti and Urziceni, we find that all of them are at the maximum depth, so we don't expand them, we remove them from the stack till our stack is empty.

After Giurgiu is removed:
Stack: { Pitesti, Urziceni }
Order of expansion: Bucharest, Fagaras, Giurgiu, Pitesti

After Pitesti is removed:
Stack: { Urziceni }
Order of expansion: Bucharest, Fagaras, Giurgiu, Pitesti, Urziceni

After Urziceni is removed:
Stack: { }
Order of expansion: Bucharest, Fagaras, Giurgiu, Pitesti, Urziceni

And that's when we stop when we empty the stack.

- For cutoff = 3:
  The stack is initialized with Bucharest, then Bucharest is pushed into the stack and checked for the following.

  Stack: {Bucharest}
  Order of expansion: Bucharest

Since Bucharest is not at the maximum depth bound we expand it and get the children {Fagaras, Giurgiu, Pitesti, Urziceni}. We remove Bucharest from the stack and add the children in alphabetical order.

Stack: {Fagaras, Giurgiu, Pitesti, Urziceni }
Order of expansion: Bucharest, Fagaras

Now, we check for Fagaras, since its not at the maximum depth we will expand it and remove it from the stack. On expanding we get, {Sibiu}.

Stack: {Sibiu, Giurgiu, Pitesti, Urziceni }
Order of expansion: Bucharest, Fagaras, Sibiu

Now, we check for Sibiu, since its at the maximum depth, we can't expand it, we remove it from the stack and check Giurgiu next.

Stack: {Giurgiu, Pitesti, Urziceni }
Order of expansion: Bucharest, Fagaras, Sibiu, Giurgiu
Now we check Giurgiu , since its not at the maximum depth, we can expand it and remove it from the stack. On expanding we get nothing, since it doesn't have any child nodes.

Stack: { Pitesti, Urziceni }
Order of expansion: Bucharest, Fagaras, Sibiu, Giurgiu

Now we check Pitesti, since its not at the maximum depth, we can expand it and remove it from the stack. On expanding we get {Romnieu Vilcea, Craiova}.

Stack: { Craiova, Romnieu Vilcea, Urziceni }
Order of expansion: Bucharest, Fagaras, Sibiu, Giurgiu, Craiova

Now we check Craiova, since its at the maximum depth, we can't expand it and remove it from the stack.

Stack: { Romnieu Vilcea, Urziceni }
Order of expansion: Bucharest, Fagaras, Sibiu, Giurgiu, Craiova, Romnieu Vilcea

Now we check Romnieu Vilcea, since its at the maximum depth, we can't expand it and remove it from the stack.

Stack: { Urziceni }
Order of expansion: Bucharest, Fagaras, Sibiu, Giurgiu, Craiova, Romnieu Vilcea, Urziceni

Now we check Urziceni, since its not at the maximum depth, we can expand and remove it from the stack. On expanding we get {Hirsova, Vaslui}.

Stack: { Hirsova, Vaslui}
Order of expansion: Bucharest, Fagaras, Sibiu, Giurgiu, Craiova, Romnieu Vilcea, Urziceni, Hirsova

Now we check Hirsova, since its at the maximum depth, we can't expand it we just remove it from the stack.

Stack: { Vaslui}
Order of expansion: Bucharest, Fagaras, Sibiu, Giurgiu, Craiova, Romnieu Vilcea, Urziceni, Hirsova, Vaslui

Now we check Vaslui, since its at the maximum depth, we can't expand it we just remove it from the stack.

Stack: { }
Order of expansion: Bucharest, Fagaras, Sibiu, Giurgiu, Craiova, Romnieu Vilcea, Urziceni, Hirsova, Vaslui.

- Similarly, for cutoff = 4:

  Order of expansion would be:

  Bucharest, Fagaras, Sibiu, Arad, Oradea, Rimnueu Vilcea, Giurgiu, Pitesti, Craiova, Dobreta, Urziceni, Hirsova, Eforie, Vaslui, Iasi.

- Similarly, for cutoff = 5:

  Order of expansion would be:

  Bucharest, Fagaras, Sibiu, Arad, Timisoara, Zerind, Oradea, Rimnueu Vilcea, Craiova, Pitesti, Giurgiu, Urziceni, Hirsova, Eforie, Vaslui, Iasi, Neamt.