**Project 1**

**Artificial Intelligence**

Fall 2024

**Distributed: Wednesday 28 August 2024**

**Part A Due: Friday 6 September 2024**

**Part B Due: Friday 20 September 2024**

*[Solutions to both parts of this project must be submitted via CANVAS prior to midnight on the due date.*

This project may be undertaken in pairs. Please state the ***name(s)*** of the people undertaking the project and the ***contributions*** that each has made. Only ONE submission should be made per group.

**Purpose**: To gain a thorough understanding of the working of a simple, reflex agent. This will enable you to put into practice the concepts of performance measurement, environment interaction, and the use of actuators (PEAS) to assist the agent’s navigation in a stochastic setting that ***mimics a real-world environment***. Thus, this requires reading through the project specification carefully to make sure that you understand its requirements. Jot down any questions/doubts that you may have and feel free to ask me questions in class or in person. Together with your partner (if working with another person), work out a strategy before you start coding the solution in Python. Given the limited timeframe for the project, some simplifications have been applied. In all cases, when simplifications are used, they have been pointed out explicitly. You may find it useful to compile a) list of all functional requirements and b) a list of all assumptions before starting to code.

**Environment Description:** The application is a ridesharing one to be deployed in any given area of geographical coverage spanning a city. The environment is specified in the form of a graph structure as represented below (simply an example, not the entire road network). The graph that you will use consists of 100 nodes, with an average connectivity of 6.

A network of blue dots and black lines

Description automatically generated

Each node in the graph represents either a potential pickup or drop-off point. The distance between a pair of connected nodes is **0.5**th of a mile. Each edge of the graph contains a weight that reflects the density of traffic along that edge. Edge weights are floating point numbers in the range 0.0 to 1. When generating the graph, you may randomly generate edge weights in the range mentioned above.

The ABC company employs a fleet of N vehicles to service the transportation needs of its customers. Each customer enters their pickup point (node number in graph) and drop-off point (also a node number) when placing a request. The system responds by assigning a vehicle to the customer and letting the customer know an estimate of the pickup time in minutes. This estimate may be an underestimate of the actual waiting time if the assigned vehicle picks up another passenger on its way to the original passenger, but for the purposes of this project, we will let the estimate stand – i.e., the estimated passenger waiting time once computed will not be updated. All vans are identified by an integer in the range 1 to N. Van assignment to a customer must account for proximity of the vehicle to the customer’s pickup location – the van ***closest***to the customer’s pickup point will be assigned to the customer as long as that vehicle has room for another passenger. The capacity of each van is 6 and that means a maximum of 5 customers can be inside any of the vans at any given point in time. If all vans are full, the agent should display the following message: “All vans are currently busy, please try again in 15 minutes time”.

Customers can only be picked up or dropped off at certain locations which are restricted to be nodes of the graph. For the purposes of this project, we will assume that vehicles travel at a constant speed of 30 mph. Since pickup request are serviced every minute and edge lengths are always 1 mile, this means that on the minute every van in the system will be at a particular graph node and will be in a position to pick up a customer should one be requesting service at that node. The same applies to drop offs.

As the ride sharing service aims to keep its costs to customers down, it may assign as many customers as there are seats available. The passenger capacity is ***5***. In case more than one customer is present in the vehicle the AI agent needs to work out the order in which the customers will be dropped off. The simplest and fairest policy is that drop offs are done on a first come first served basis – i.e., customers who were picked up first will be dropped off first.

We will also assume that cancellations are ***not*** made by either the customer or the company once a reservation has been made.

If a vehicle is not assigned a new passenger after dropping off its last one, then ***it simply parks at the drop off point and waits for the next passenger to arrive***. We will assume that parking is always available.

Your task in this project is to implement the following requirements:

**Part A (due one week after project is handed out)**

Your task in this project is to implement the following requirements:

R1. Produce a pseudo code version of the algorithm needed for managing customer requests (generating and receiving customer service requests), scheduling customer pickups and drop offs. Pseudo code is a mix of programming statements and English. You may use *if* statements, *while* statements, *function* statements and so on, but please do **not** submit actual Python code for Part A as it is meant as a thinking, not coding exercise. If still in doubt, refer to pseudo examples presented in the lecture content.

Tutorial 1 provides the foundation for this algorithm (Tutorial 1 will be posted online on Canvas and will be discussed in class). Your algorithm should follow good design principles such as modularity, have meaningful names for variables/data structures and have comments at key points in your pseudo code.

Your pseudo code needs to cover the following 5 pieces of functionality:

1. Assignment of vans to customers.
2. Servicing of customer pickup requests.
3. Computation of waiting time for each customer where waiting time is the elapsed time between pickup and the time that the customer made the pickup request.
4. Servicing of customer drop off requests.
5. Computation of service time for each customer where service time is the elapsed time between drop off and the time that the customer made the pickup request.

(**30 marks)**

**Note:** All pseudo code must be typewritten and submitted in a pdf document. No handwritten versions will be accepted.

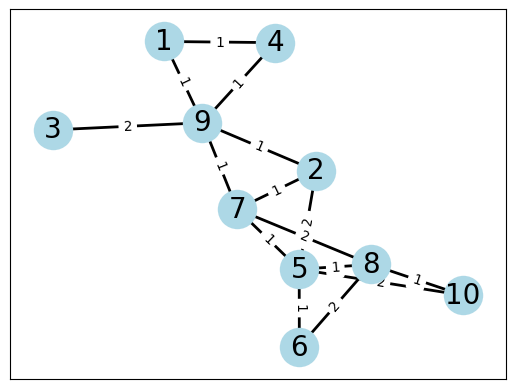
**Part B (due two weeks after Part A is submitted)**

In Part B we will model the effect of congested node. Edges with weight value 2 represent congested roads while weight value 1 represent roads with no congestion. Travel along congested roads take twice the amount of time when compared to uncongested roads. This means that a van along a congested road will take 2 clock ticks to travel along the edge whereas a van on an uncongested road only consumes one clock tick to travel along such an edge.

R2 Implement a Python program that implements the algorithm in R1 on the network given below. In this simplified scenario the graph consists of only 10 nodes and 15 edges with the weights as specified. Also, the number of vans is just 2.

Your program must show for *each* clock tick the

1. contents of the service queues S1 and S2 for both vans
2. contents of the paths P1 and P2 taken by vans 1 and 2 respectively from the start position to the position at the current clock tick

 At clock tick 1

Pickup request at 1 for customer 1, drop off at 9.

Pickup request at 2 for customer 2, drop off at 6.

At clock tick 2

Pickup request at 5 for customer 3, drop off at 7.

Pickup request at 3 for customer 4, drop off at 4.

At clock tick 3

Pickup request at 4 for customer 5, drop off at 7.

Pickup request at 2 for customer 6, drop off at 10.

**(25 marks)**

R3. To implement this requirement, you will make use of the program you developed in R2 with three changes. Firstly, you will replace the graph in R1 with a random graph of 100 nodes and a connectivity of **6**. Secondly, your program will need to work with 30 vehicles, instead of just 2.  Thirdly, you will need to randomly generate 450 to 600 reservations per hour (of simulation time). The system’s operating hours are from 8 am to 6 pm. No reservations are accepted after 6 pm but customers who are already in vans beyond 6 pm are serviced. In other words, the simulation ends when all service queues in the fleet of 30 vans are empty.

Apart from these three changes above the program functionality will remain the same as in the program for R2. As with R2 you will need to *populate* the service queues S and the routing queues R. Note that you will *not need to display* the contents of the queues on the console.

Your modified program should compute:

1. the average waiting time per customer (taken over the fleet of 30 vans) per day. **(9 marks)**
2. the average service time per customer (taken over the fleet of 30 vans) per day. **(9 marks)**
3. the average number of trips(taken over the fleet of 30 vans) per day. **(9 marks)**

Note that reservations occur at random points in the clock and may originate at random nodes in the graph. Likewise, destinations are also random.

R4. Using the same graph that you used in R3 and varying the fleet size to **60** vehicles and suppose that the same number of reservations occur per hour as mentioned in R3 above.

What are the new values of average waiting average time per customer, average service time per customer and average number of trips travelled per day taken across the entire fleet? **(8 marks)**

R5 Now run your code across a new graph having a connectivity of **3** instead of **6** whilst still having 100 nodes. What is the new value of the average number of trips when the booking rate given in R4 is applied? Why does it differ? **(10 marks, 5 for value, 5 for reason)**

Hand in for Part B:

The following should be submitted:

1. A **publicly** (make sure you test this) **accessible link** to your Collab notebook containing your code.

2. A **pdf** (an original pdf, no images please) of the copy of your code. **Do not simply save your notebook as a pdf in Collab.** This does not produce a properly formatted pdf document. Consult [Bing Videos](https://www.bing.com/videos/riverview/relatedvideo?q=creating+a+pdf+from+a+collab+notebook&mid=C77A7FED744CC90F4A03C77A7FED744CC90F4A03&FORM=VIRE) this tutorial that explains on how to save a Collab notebook as a pdf file.

3. A separate pdf (again, no images please) of the answers to requirements R3 to R5.