Project 1
Artificial Intelligence

Fall 2022

[Solutions to this assignment must be submitted via CANVAS prior to midnight on the due date. These dates and times vary depending on the milestone to be submitted. Submissions up to one day late will be penalized 10% and a further 10% will be applied for the next day late. Submissions will not be accepted if more than two days later than the due date.]

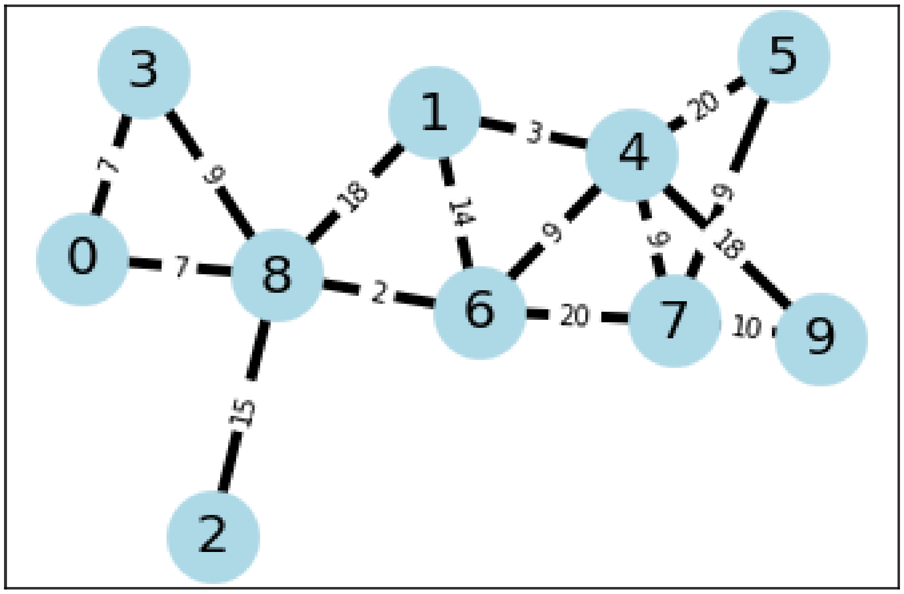
This project may be undertaken in pairs or individually. If working in a pair, state the ***names*** of the two people undertaking the project and the ***contributions*** that each has made. Only ONE submission should be made per group.

Purpose: To gain a thorough understanding of the working of an agent that adapts to changing traffic conditions and returns the shortest path from a given starting location to a destination. This will extend the shortest path methods that we discuss in the course lectures.

We will make use of a network that ***mimics a real-world environment*** such as road network or a computer network. Both types of environments are dynamic with time varying traffic and the goal is to ***adapt*** to varying traffic levels. The network/graph will be implemented with the NetworkX tool which is the subject of Tutorial 1 of this course.

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, ***work out a strategy before you start coding the solution in Python.*** Given the limited timeframe for the project, some simplifications have been assumed. In all cases, when simplifications are used, they have been pointed out explicitly. You may find it useful to compile: a) a list of all functional requirements and b) a list of all assumptions before starting to code.

Environment Description: The environment is a network that is specified in the form of a graph structure as represented below (simply a sample, not the entire network). The graph that you will use consists of 100 nodes, with an average connectivity of 3.



Each node in the graph represents either a potential start, destination, or transition point. The edges on the graph are labelled with weights that represent the time in minutes taken to travel from the starting node on the edge to the ending node. Thus, in the above graph we can conclude that it takes twice as long to travel from node 1 to node 8 as it does to travel from 3 to 8. In addition to the edge weight another factor that affects the time taken is the *traffic loading factor* (not shown in the Figure above) which represents the degree of traffic congestion on an edge. The loading factor is subject to change with time. For the purposes of this project, we will assume that changes take place *every 15 minutes*. The changes reflect either an increase or a decrease on the base level for that hour. The table below gives the *base levels* for the 24-hour clock.

Hour	Base Level
8 am	1.0
9 am	0.95
10 am	0.9
11 am	0.9
12 noon	0.9
1 pm	0.9
2 pm	0.95
3 pm	0.95
4 pm	0.9
5pm	1.0
6 pm	1.0
7 pm	0.85
8 pm	0.7
9 pm	0.6
10 pm	0.45
11 pm	0.3
12 am	0.1
1 am	0.1
2 am	0.1
3 am	0.15
4 am	0.25
5 am	0.3
6 am	0.65
7 am	0.85

At each hour the actual traffic level is the multiple of the base level for that hour with the change that takes place at every 15 minutes within that hour. Changes are implemented by a normal distribution having a mean of 0 and a standard deviation of 0.25. Thus, for example the % *change* in the actual traffic load for the edge (3,8) between 9 am and 9.15 am is 0.95×p×9, where p is the value returned by the normal distribution function (0.95 is the base value for 9-9.15 am timeslot and 9 is the edge length of (3,8)). This updated traffic load will then take effect and remain at the same level until 9.15 am. This means that the shortest path for a pair of nodes (N, M) may change depending not just on the time of the day but also on the timeslot within the hour (i.e., whether it is the first 15 minutes, the second, and so on).

Instead of recomputing the shortest path every 15 minutes for a given pair of nodes we will use an adaptive approach to modify the shortest path that was computed for the hour. Thus, for example suppose that we compute the shortest path between nodes (2, 4) at 10 am traffic levels as 2, 8, 1, 4. Due to changes that took place at 10.15 am the new shortest route between (2, 4) could now be 2, 8, 6, 7, 4. The new route reuses some of the edges from the old route and this enables us to improve efficiency. One of the tasks for this project is to design and implement an algorithm that will implement this adaptation. A separate document, available from here discusses some ideas for designing an adaptive algorithm.

The simulation will be driven by a clock that ticks every 15 minutes. Starting at 12 midnight, 10 requests are made for a shortest path from a randomly selected starting (source) node to another randomly selected destination node at the beginning of each 15-minute time slot. In addition, vehicles that have not finished their journeys will need to also make requests as their existing paths may be outdated as traffic conditions change. Thus, your algorithm needs to track each journey from its start until it reaches its destination. This tracking will involve identifying which nodes it has visited so far as well as the next node to be visited from the last visited node so that adjustments can be made to its path due to the changed traffic load.

Your task in this project is to implement the following requirements. The project has three milestones, with milestone 1 comprising requirements R1 and R2: milestone 2 consisting of requirement R3 and milestone 3 consisting of requirement R4. The milestones have different deadlines as shown below.

Milestone 1 R1

Produce a pseudo code version of the algorithm needed to adapt the shortest path between a given pair of nodes. Please note that pseudo code is ***not*** the same as actual code, *and so Python code will not be acceptable here.* ***Due at midnight on Wednesday 14 September*** **(15 marks)**

Milestone 1 R2

Using the pseudo code that you produced in Q1 above, *extend* the functionality to include the following requirements. Produce an extended version of your pseudo code from R1.

1. generate new navigation requests (i.e., start and destination nodes),
2. revise existing journeys (i.e., change nodes on path for nodes not already visited, if necessary)
3. keep track of journey time for each vehicle that has completed its journey,
4. keep a running count of total journey time taken across all vehicles that have completed their journeys.

Also due at midnight on Wednesday 14 September **(20 marks)**

Notes:

1. Produce ONE pdf document that contains algorithms for requirements R1 and R2 above.
2. If you finish R1 and R2 before the milestone 1 deadline, start working on milestone 2 (R3) immediately afterwards.

Milestone 2 R3

Produce Python code that implements the requirements R1 and R2 above. Your Python code should be supplied in the form of a Google Colab notebook ***as well as a copy in pdf form***. Use the same pdf that you used for R1 and R2 above.

Due at midnight on Sunday 25 September **(40 marks)**

Milestone 3 R4

This part will not need any extra programming (apart from some trivial changes in parameter values).

1. Change the value of the connectivity parameter from 0.3 to 0.6 and report on the new value of the total journey time that you computed in R3.
2. Change the value of the standard deviation of traffic load from 0.25 to 0.125 and report on the new value of the total journey time that you computed in R3.
3. Explain ***how*** changes in connectivity affect changes in journey time.

4. Explain ***how*** changes in standard deviation affect changes in journey time.

Due at midnight on Wednesday 28 September **(25 marks)**

End of project specification