Tube Simulator - Stochastic V2

Jacob Delveaux

June 26, 2020

Contents

1 Introduction

This is a program developed to simulate the London underground and over-ground transportation services, using agents taking a directed walk in the network. A large extent of the program functions based on real data from TFL, including station popularity, direction, nodes/links, etc.

2 Data

2.1 List of Data

From http://crowding.data.tfl.gov.uk/ for nodes, links, and enter/exit data (including platform popularity):

1. 2018MTT_Link_Frequency.xlsx
2. 2018MTT_Station_Link_Flows.xlsx
3. 2018NUMBAT_Definitions.xlsx

GPS data from: https://data.london.gov.uk/dataset/tfl-station-locations and Wikipedia Train speed from: https://tfl.gov.uk/corporate/transparency/freedom-of-information/foi-request-detail?referenceId=FOI-0228-1819 and Wikipedia

2.2 Data Processing

First, each station was given a GPS coordinate based on its NLC code, a 4-5 digit code that references the entire station. The links from the two data files (1) and (3) were merged using the start and end code, a three letter identifier (e.g. London Bridge Station - LBG), the start and end NLCs, the line used with direction (UP, DN, NB, SB, EB, WB). National rail data did not contain which line linked stations - these were added by hand.

2.3 Making the Graph

The links were added to a python networkx directed graph (Figure 1) using the nodes defined in data file (3) which included every platform. The nodes were labeled with the corresponding station NLC and GPS coordinates. Using the position and speed data, it was possible to get the time it takes to travel between platforms in the LU/LO system. Additionally, platforms belonging to the same station (those with the same NLC) were linked to one another and to the station entrance with a static 'transfer time' of 5 minutes.

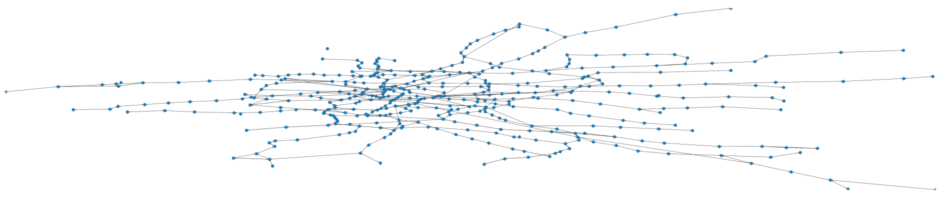


Figure 1: London Underground/Overground Graph

2.4 Station and Platform Desirability

Included in data file (2) are enter/exit numbers for Monday-Thursday at each station and for every platform. These are broken up by time of day: Early, AM, Midday, PM, Evening, Late Station popularity was obtained by taking the total number of people entering that station, divided by the total number of people entering any station at that time (i.e. the probability of entering the station). Likewise, platform desirability was computed by taking the total number of people entering that platform (e.g. DLR_SB) divided by the total number of people entering the station.

3 The Simulation

The simulation is an agent based model where agents are directed along the network stochastically, i.e. drawing their movements from a probability distribution rather than pre-defined origins and destinations.

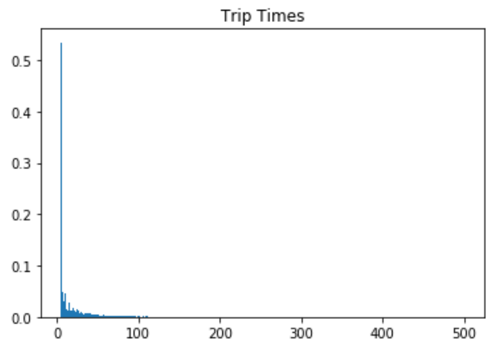
First, agents are created and placed on the network at a random entrance (drawn according to the station popularity). Agents are given a travel time (set to zero), a list of locations visited, a list of journeys taken, an exit probability check, and a swap probability check. The two probability checks are to save time generating random numbers.

At each time-step, each agent acts as follows:
1) Check if exit The quantity 1 - exp(- travel time / characteristic time) is compared to the exit probability check of the agent. This is a simple decay model. Essentially, the longer an agent is traveling, the more likely they are

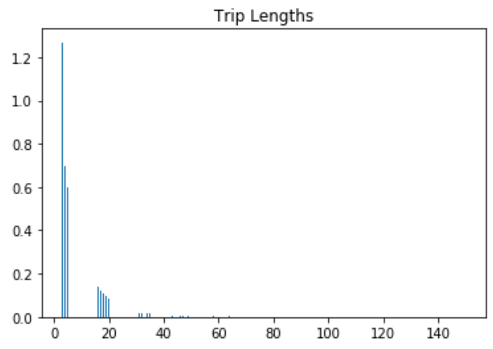
to exit the transport system. The characteristic time is set as to give an average trip time of 20-30 minutes.
1a) If the agent exits If the agent exits, then their trip is ended. Their set of locations and time traveled is added to their journeys taken (for later statistics). They are then given a new starting location and their travel time is reset to zero.
1b) If the agent doesn't exit, continue
2) Check if trains are swapped In the program, there is a XX% chance the agent swaps trains (currently set to 10%). For example, this might be a agent riding the Central line to Holborn, and then swapping to the Picadilly line. If the agent's last location was a station entrance, they must always swap.
2a) If the agent swaps trains If the agent swaps trains, then they choose a new platform drawn according to the platform popularity for that station, and their travel time increases by 5 minutes.
2b) If the agent doesn't swap trains If the agent doesn't swap trains, then they travel to the next available platform on the same line (E.g. Jubilee Eastbound West Ham -¿ Jubilee Eastbound Stratford).

The main code can be found in the Appendix 4.1.

3.1 Average Travel Time and Length

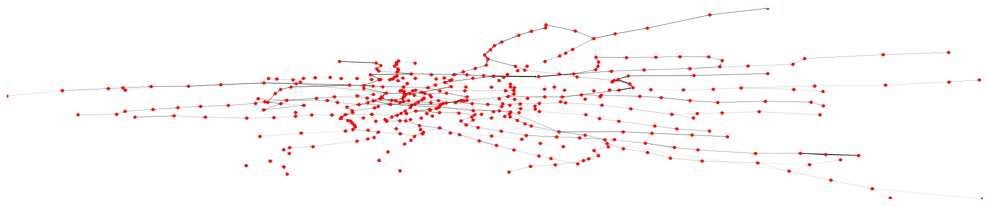


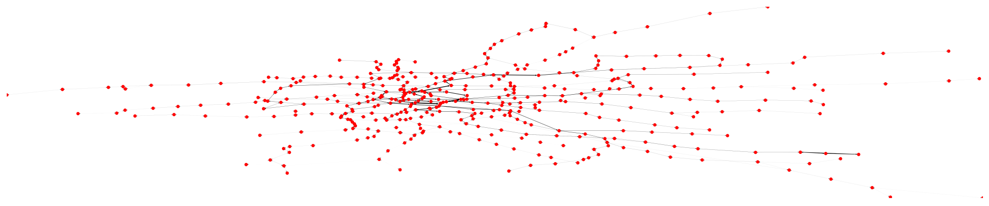
(a) Travel Time Distribution



(b) Number of Stops Distribution

3.2 Early and Evening Graphs


(a) Early Graph


(b) Evening Graph

4 Appendix

4.1 Main Code

```
"PARAMETERS"
N_people = 1000
N_iter = 1000
change_chance = 0.10
current_nodes = nx.get_node_attributes(tube_graph, 'nlc')

"RUN PROCESS"
start = time.time()

#Set agents and initial locations
people = [Person() for _ in range(N_people)]
for agent in people:
    agent.enter_station(entrance_popularity)

#Run for several iterations
for _ in range(N_iter):
    for agent in people:
        #if exit
        if agent.exit_bool() and len(agent.locations) > 2:
            agent.end_trip(entrance_popularity)
        else:
            transition_dict =
             ↪  pltfm_des[current_nodes[agent.locations[-1]]]
            #if last station was entrance
            if agent.locations[-1] in list(entrances.keys()):
                #change trains
                agent.swap_train(transition_dict,
                 ↪  entrance_popularity)
            #else
            else:
                if rnd.rand() < agent.swap_prob:
                    agent.swap_train(transition_dict,
                     ↪  entrance_popularity)
                #if train not changed
                else:
                    valid_stops =
                     ↪  connections[connections['SrtCode'] ==
                     ↪  agent.locations[-1]]
                    agent.next_stop(valid_stops, edge_time,
                     ↪  transition_dict, entrance_popularity)

#Print time
print('%i people moved %i times in %.3f seconds'%(N_people,
 ↪  N_iter, time.time()-start))
```

```
class Person():

    average_time = 500 #average train/tube journey time
    swap_time    = 5  #average time to swap platforms

    def __init__(self):
        #initialize person with zero travel time at random
         ↪  station
        self.travel_time = 0.0
        self.locations   = []
        self.journeys    = []
        #pre-allocate exit probability
        self.exit_prob   = rnd.rand()
        self.swap_prob   = rnd.rand()

    def end_trip(self, entrance_popularity):
        self.journeys.append([self.locations,
        ↪   self.travel_time])
        self.enter_station(entrance_popularity)

    def enter_station(self, entrance_popularity):
        self.travel_time = 0.0
        self.locations   =
        ↪   [str(rnd.choice(list(entrance_popularity.keys()),
        ↪   p = list(entrance_popularity.values())))]

    def exit_bool(self):
        #Poisson waiting time
        return self.exit_prob < 1.0 -
        ↪   np.exp(-self.travel_time/self.average_time)

    def swap_train(self, transition_dict,
    ↪   entrance_popularity):
        #strip current line
        transition_dict = {key_:val_ for key_, val_ in
        ↪   transition_dict.items() if not key_.split('_')[1]
        ↪   == self.locations[-1].split('_')[1]}
        #if no lines, just exit
        if len(transition_dict.keys()) > 0:
            #renormalize
            transition_dict = {k: v / total for total in
            ↪   (sum(transition_dict.values()),) for k, v in
            ↪   transition_dict.items()}
            #new_platform =
            ↪   rnd.choice(list(transition_dict.keys()), p =
            ↪   list(transition_dict.values()))
            new_platform = list(transition_dict.keys())[0]
            self.locations.append(new_platform)
            self.travel_time += self.swap_time
        else:
            self.end_trip(entrance_popularity)

    def next_stop(self, valid_stops, edge_time,
     ↪  transition_dict, entrance_popularity):
        #If at a terminus (or no valid stops), try to swap
         ↪  trains
        if len(valid_stops) == 0:
            self.swap_train(transition_dict,
             ↪  entrance_popularity)
        #Otherwise, pick a random next stop and add travel
         ↪  time
        else:

             ↪  self.locations.append(rnd.choice(valid_stops.EndCode.values))
            try:
                self.travel_time +=
                 ↪  edge_time[(self.locations[-2],
                 ↪  self.locations[-1])]
            except KeyError:
                self.travel_time +=
                 ↪  edge_time[(self.locations[-1],
                 ↪  self.locations[-2])]
```