Readme

To execute the bus allocation and scheduling script, open the script in jupyter notebook and follow the comments.

The logic of the code has been explained in plain language in the image file in the folder.

The code mainly implements the following algorithm:

Algorithm: Bus Scheduling and Allocation in both directions for a bus route

Input: List of trips in both direction *up_trips and down_trips;* Bus travel time for the route *travel_time*in minutes

Output: Bus Schedule for the given route

1: Initialize *Bus_tag*← *0, curr_hour* ← *4, next_hour* ← *5, curr_min* ← *0*

2: Compute *up_count*= $\sum_{i=0}^{Number\ of\ Time\ Slots} up_trips[i]$and *down_count*= $\sum_{i=0}^{Number\ of\ Time\ Slots} down\_trips[i]$

3: while j ≤ Number of Time Slots do

4: If *up_trips[j]>1* then generate the schedule for *curr_hour* and append to *up_schedule* with headway between trips = Number of minutes in 1 hour / *up_trips[j]*

5: Else if *up_trips[j] == 1* then schedule the trip for *curr_hour* at 30 minutes past *curr_hour* and append to *down_schedule*

6: If *down_trips[j]>1* then generate the schedule for *curr_hour* and append to *down_schedule* with headway between trips = Number of minutes in 1 hour / *down_trips[j]*

7: Else if *down_trips[j] == 1* then schedule the trip for *curr_hour* at 30 minutes past *curr_hour* and append to *down_schedule*

8: j ← j+1, *curr_hour = curr_hour + 1*, *next_hour* ←*next_hour + 1*, *curr_min* ← *0*

9: end while

10: Create a sorted combined list of *up_schedule* and *down_schedule* named combined with *dir* ← *'U'* appended to *up_schedule* records and *dir* ←*'D'* appended to *down_schedule* records

11: *up_schedule_index* ← *0*, *down_schedule_index* ← *0*

```
12: while j ≤ len (combined) do
13:     If combined[j].dir== 'U' then down_count ← down_count – 1
14:         If down_count> 0 then
15:             If len (up_arrival)>0 then sort up_arrival by time in ascending order
16:                 If up_arrival [0].time ≤combined[j].time then
17:                     up_schedule [up_schedule_index].Bus_Tag ← up_arrival [0].Bus_tag
18:                     down_arrival.append (up_arrival [0].time + travel_time,up_arrival
                        [0].Bus_tag)
19:                     up_arrival [0].drop()
20:                     up_schedule_index ← up_schedule_index + 1
21:                 Else sort down_arrival by time in ascending order
22:                     If down_arrival[0].time + travel_time ≤ combined[j].time then
23:                         up_schedule [up_schedule_index].Bus_Tag ← down_arrival
                            [0].Bus_tag
24:                         down_arrival.append (down_arrival [0].time +
                            2*travel_time,down_arrival[0].Bus_tag)
25:                         down_arrival [0].drop()
26:                         up_schedule_index ← up_schedule_index + 1
27:                     Else create new bus for the trip
28:                         Bus_tag ← Bus_tag + 1
29:                         up_schedule [up_schedule_index].Bus_Tag ← Bus_tag
30:                         down_arrival.append (combined[j].time +travel_time,Bus_tag)
31:                         up_schedule_index ← up_schedule_index + 1
32:             Else goto step 20
33:         Else goto step 20
34:     Else up_count ← up_count – 1
35:         If up_count> 0 then
36:             If len (down_arrival)>0 then sort down_arrival by time in ascending order
37:                 If down_arrival [0].time ≤ combined[j].time then
38:                     down_schedule [down_schedule_index].Bus_Tag ← down_arrival
                        [0].Bus_tag
39:                     up_arrival.append (down_arrival [0].time +
                        travel_time,down_arrival[0].Bus_tag)
```

40: *down_arrival [0].drop()*

41: *down_schedule_index* ← *down_schedule_index + 1*

42: Else sort *up_arrival* by time in ascending order

43: If *up_arrival[0].time + travel_time* ≤ *combined[j].time* then

44: *down_schedule [down_schedule_index].Bus_Tag* ← *up_arrival [0].Bus_tag*

45: *up_arrival.append (up_arrival [0].time + 2*travel_time,up_arrival [0].Bus_tag)*

46: *up_arrival [0].drop()*

47: *down_schedule_index* ← *down_schedule_index + 1*

48: Else create new bus for the trip

49: *Bus_tag* ← *Bus_tag + 1*

50: *down_schedule [down_schedule_index].Bus_Tag* ← *Bus_tag*

51: *up_arrival.append (combined[j].time + travel_time,Bus_tag)*

52: *down_schedule_index* ← *down_schedule_index + 1*

53: Else goto step 40

54: Else goto step 40

55: j ← j+1

56: end while

57: Total_buses ← *Bus_tag*

58: Return *up_schedule, down_schedule, Total_buses*