Tutorial 2

Clock Tick 1

Initialize routing queues R1 and R2

R1={0} R2={0}
tail(R1)=0 tail(R2)=0
Calculate distance D for next pickup request from destination point of each van.

Compute distance of pickup location 8 to van 1 location 0 D(8,0) and compare with distance of pickup location 8 to van 2 location 0 D(8,0). Since both distances are the same, we break ties randomly and assign customer 1 to the first van empty van which is van 1.

Next do the same for next customer. Compare D(3,0) with D(3,0). Since both are the same, we break ties in favour of the van which is empty and has the lowest id. This happens to be van 2.

S1={(id1,p,8),(id1,d,9)}
S2={id2,p,3),(id2,d,6)}

Now compute the next node in the path for each van. This is done by issuing a call to Dijkstra's function or Astar function calls.

P1=get_next_node(0,8)=8
P2= get_next_node(0,3)=3

Clock Tick 2

R1={0,8} R2={0.3}

Tail(R1)=8, Tail(R2)=3

D(4,8)<D(4,3), thus we allocate customer 3 to V1

D(2,8)<D(2,3), thus we allocate customer 4 to V1

Since more than 1 customer is allocated to van 1, we need to determine the best scheduling order for pickups of customers 3 and 4.

Van 1 is currently at 8; so, we need to determine the best scheduling order for 8, 2 and 4. Node 2 is closer to 8 than 4 and so the optimal schedule is 8, 2 and 4.

S1={(id1,d,9),(id4,p,2),(id4,d,4),(id3,p,4),(id3,d,7)}

S2 remains the same as no allocation to van 2, S2={(id2,d,6)}
P1= get_next_node(8,9)=6
P2= get_next_node(3,6)=8

Clock Tick 3

R1={0,8,6} R2={0,3,8}

Tail(R1)=6, Tail(R2)=8

D(1,6)=D(1,8), thus we allocate customers 5 and 6 to V1 (as we break ties by assigning the van with the lowest id number).

Since more than 1 customer is allocated to van 1, we need to determine the best scheduling order for pickups of customers 3, 4, 5 and 6.

Van 1 is currently at 6; so, we need to determine the best scheduling order for 6, 2, 4, 1 and 1. Node 4 is closest to 6 than node 4 or node 1, and so we proceed from 6 to 4 first. Now we need to plot a path from 4 to nodes 2 and 1. Node 1 is closer to 4 than node 2. So, we proceed from 4 to 1. So, the optimal schedule is 6, 4, 1, 1 and 2.

S1={(id1,d,9), (id3,p,4),(id3,d,7),(id5,p,1),(id6,p,1),(id5,d,7),(id6,d,9),(id4,p,2),(id4,d,4)}

S2 remains the same as no allocation is made to van 2, S2={(id2,d,6)}
P1= get_next_node(6,9)=7

Clock Tick 4

R1={0,8,6,7} R2={0,3,8,6}

S1 is the same as in tick 3.

S2={}

P1= get_next_node(7,9)=9

Clock Tick 5

R1={0,8,6,7,9} R2={0,3,8,6}

S1={(id3,p,4),(id3,d,7),(id5,p,1),(id6,p,1),(id5,d,7),(id6,d,9),(id4,p,2),(id4,d,4)}

P1= get_next_node(9,4)=7

Clock Tick 6

R1={0,8,6,7,9,7} R2={0,3,8,6}

S1 is the same as in tick 5.

P1= get_next_node(7,4)=4

Clock Tick 7

R1={0,8,6,7,9,7,4} R2={0,3,8,6}

S1={(id3,d,7), (id5,p,1),(id6,p,1),(id5,d,7),(id6,d,9)} (id4,p,2),(id4,d,4)}

P1= get_next_node(4,7)=7

Clock Tick 8

R1={0,8,6,7,9,7,4,7} R2={0,3,8,6}

S1={id5,p,1),(id6,p,1),(id5,d,7),(id6,d,9),(id4,p,2),(id4,d,4)}

P1= get_next_node(7,1)=4

Clock Tick 9

R1={0,8,6,7,9,7,4,7,4} R2={0,3,8,6}

S1 is the same as in tick 8.

P1= get_next_node(4,1)=1

Clock Tick 10

R1={0,8,6,7,9,7,4,7,4,1} R2={0,3,8,6}

S1={(id5,d,7),(id6,d,9),(id4,p,2),(id4,d,4)}

P1= get_next_node(1,7)=4

Clock Tick 11

R1={0,8,6,7,9,7,4,7,4,1,4} R2={0,3,8,6}

S1 is the same as in tick 10.

P1= get_next_node(4,7)=7

Clock Tick 12

R1={0,8,6,7,9,7,4,7,4,1,4,7} R2={0,3,8,6}

S1={(id6,d,9),(id4,p,2),(id4,d,4)}

P1= get_next_node(7,9)=9

Clock Tick 13

R1={0,8,6,7,9,7,4,7,4,1,4,7,9} R2={0,3,8,6}

S1={(id4,p,2),(id4,d,4)}

P1= get_next_node(9,2)=7

Clock Tick 14

R1={0,8,6,7,9,7,4,7,4,1,4,7,9,7} R2={0,3,8,6}

S1 is the same as in tick 13.

P1= get_next_node(7,2)=6

Clock Tick 15

R1={0,8,6,7,9,7,4,7,4,1,4,7,9,7,6} R2={0,3,8,6}

S1 is the same as in tick 14.

P1= get_next_node(6,2)=8

Clock Tick 16

R1={0,8,6,7,9,7,4,7,4,1,4,7,9,7,6,8} R2={0,3,8,6}

S1 same as in tick 15.

P1= get_next_node(8,2)=2

Clock Tick 17

R1={0,8,6,7,9,7,4,7,4,1,4,7,9,7,6,8,2} R2={0,3,8,6}

S1={(id4,d,4)}

P1= get_next_node(2,4)=8

Clock Tick 18

R1={0,8,6,7,9,7,4,7,4,1,4,7,9,7,6,8,2,8} R2={0,3,8,6}

S1 is the same as in tick 17.

P1= get_next_node(8,4)=6

Clock Tick 19

R1={0,8,6,7,9,7,4,7,4,1,4,7,9,7,6,8,2,8,6} R2={0,3,8,6}

S1 is the same as in tick 17.

P1= get_next_node(6,4)=4

Clock Tick 20

R1={0,8,6,7,9,7,4,7,4,1,4,7,9,7,6,8,2,8,6,4} R2={0,3,8,6}

S1={}

Summary

1. Your algorithm for R1 will use the same reasoning as the trace above.
2. At each clock tick functionality is:
    a. Update the route taken for each van (queue R).
    b. Gather new service requests.
    c. Allocate vans to pickup requests – this will involve inserting pickups and drop off entries into the service queue (S) for the van that was assigned.
    d. Determine the optimal schedule for requests in the service queue.
    e. Determine the next node in the path taken for each van.

Note: If you are unsure about any of the requirements or how to design the algorithm, contact me during office hours.

Out of scope for Project 1 but relevant to a real-world ridesharing application

Due to time constraints the following functionality is not to be implemented but is needed in real world situation.

1. We can optimize scheduling further by subjecting drop offs (not just pickups) to the optimization process.
2. We can model the effects of time (traffic densities vary not only in space but also in time) by making the edge weights on the graph dynamic.
3. There is a small (but not non-zero) risk of a customer staying indefinitely in the van or at least a very long time. This is because our scheduling policy is not constructed on a first-come-first served basis. To address this issue, we would have to track waiting time in the van for each customer and push those customers whose waiting time is 5 clicks or more to the head of the service queue.