COSC 6384 HW2 Report

Vedant Vohra

Problem Analysis

The Selfless Traffic Routing problem is an optimization problem. Each vehicle has a start point, destination, and deadline.

The model consists of a map, containing paths on which the vehicles travel, a set of uncontrolled vehicles with pre-configured routes and a set of controlled vehicles with 4 parameters: (1) start point (an edge), (2) destination (an edge), (3) time to set off, (4) deadline

The required task is to implement a routing policy which ensures the following:

1. All vehicles reach their respective destinations
2. Deadlines should not be missed
3. The time spent on the road by each vehicle must be minimized

To achieve these goals, the policy will need to decide the direction of travel for each vehicle at every step. The decision space is: [LEFT, RIGHT, STRAIGHT]

This must go on until all vehicles have either reached their destination or have been removed from the map by TraCI

Algorithm

The algorithm that I am proposing is based on Dijkstra's shortest path algorithm. In Dijkstra's algorithm, we take a greedy approach to minimizing sum of all edges which fall on the path taken by the vehicle.

The shortcoming of this approach is that taking only edge lengths into account can cause multiple vehicles to choose the same edge at the same time. This can cause congestion on certain edges, which in turn will increase the time spent by the vehicles on the road.

My proposed modification to this approach aims to reduce the chance of congestion by also taking into consideration how busy the nearby edges are, in addition to their lengths/weights. If the shortest edge is already occupied by one or more vehicles, then the current vehicle will pick the next shortest path which is either free of vehicles or has the least number of vehicles on it.

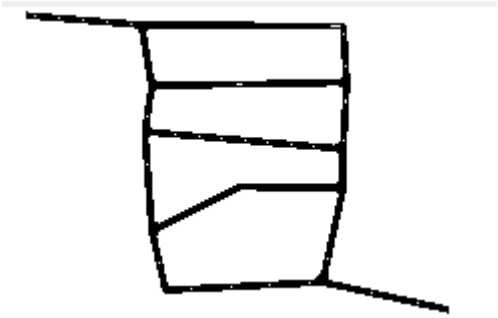
The following is a high-level representation of this algorithm:

```
for vehicle in vehicles:
    dist[v] ← INFINITY
    prev[v] ← UNDEFINED
    add v to Q
    dist[source] ← 0

    while Q is not empty:
      sort(Q, level_1: edge_vehicles_count[u], level_2: dist[u])
      u ← Q[0]
      remove u from Q

      for each neighbor v of u still in Q:
          alt ← dist[u] + Graph.Edges(u, v)
          if alt < dist[v]:
              dist[v] ← alt
              prev[v] ← u
```

Experiments

To evaluate this algorithm, I have used simple_grid1 with 10 controlled vehicles and 50 uncontrolled vehicles with the following configuration:

Map:



Vehicles (controlled):

id: 100, destination: gneE12, start time:0, deadline: 578;
id: 101, destination: gneE12, start time:5.0, deadline: 594;
id: 102, destination: gneE12, start time:10.0, deadline: 673;
id: 103, destination: gneE12, start time:15.0, deadline: 903;
id: 104, destination: gneE12, start time:20.0, deadline: 595;
id: 105, destination: gneE12, start time:25.0, deadline: 705;
id: 106, destination: gneE12, start time:30.0, deadline: 567;
id: 107, destination: gneE12, start time:35.0, deadline: 752;
id: 108, destination: gneE12, start time:40.0, deadline: 944;
id: 109, destination: gneE12, start time:45.0, deadline: 715;

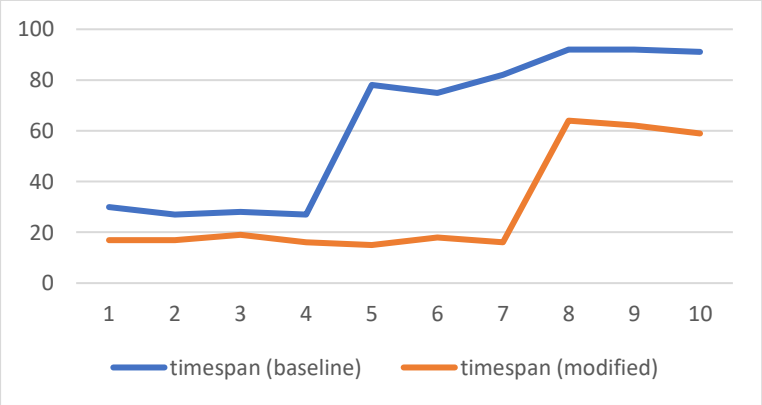
To obtain a baseline, Dijkstra’s algorithm was used.

Results:

Testing Dijkstra's Algorithm Route Controller
Vehicle 100 reaches the destination: True, timespan: 30.0, deadline missed: False
Vehicle 101 reaches the destination: True, timespan: 27.0, deadline missed: False
Vehicle 102 reaches the destination: True, timespan: 28.0, deadline missed: False
Vehicle 103 reaches the destination: True, timespan: 27.0, deadline missed: False
Vehicle 104 reaches the destination: True, timespan: 78.0, deadline missed: False
Vehicle 105 reaches the destination: True, timespan: 75.0, deadline missed: False
Vehicle 106 reaches the destination: True, timespan: 82.0, deadline missed: False
Vehicle 107 reaches the destination: True, timespan: 92.0, deadline missed: False
Vehicle 108 reaches the destination: True, timespan: 92.0, deadline missed: False
Vehicle 109 reaches the destination: True, timespan: 91.0, deadline missed: False
Average timespan: 62.2, total vehicle number: 10
0 deadlines missed.

Testing Modified Dijkstra's Algorithm Route Controller
Vehicle 100 reaches the destination: True, timespan: 17.0, deadline missed: False
Vehicle 101 reaches the destination: True, timespan: 17.0, deadline missed: False
Vehicle 102 reaches the destination: True, timespan: 19.0, deadline missed: False
Vehicle 103 reaches the destination: True, timespan: 16.0, deadline missed: False
Vehicle 104 reaches the destination: True, timespan: 15.0, deadline missed: False
Vehicle 105 reaches the destination: True, timespan: 18.0, deadline missed: False
Vehicle 106 reaches the destination: True, timespan: 16.0, deadline missed: False
Vehicle 107 reaches the destination: True, timespan: 64.0, deadline missed: False
Vehicle 108 reaches the destination: True, timespan: 62.0, deadline missed: False
Vehicle 109 reaches the destination: True, timespan: 59.0, deadline missed: False
Average timespan: 30.3, total vehicle number: 10
0 deadlines missed.

Comparison of time spent by each vehicle on the road:



It is evident in the findings above that the modified algorithm has a marked improvement in the time spent by the vehicles on the road.

However, it was observed that the new policy is less stable than Dijkstra's. On random occasions, many or all vehicles fail to reach their destinations with the following error:

"Not enough decisions provided to compute valid local target. TRACI will remove vehicle."

My theory on this behavior is that the scheduler is getting stuck in a loop where the next vacant edge might lead to a dead-end and preventing the vehicles from reaching their destinations. It might be possible to resolve this via backtracking, but that will require more analysis.

References

https://github.com/DDeChoU/Selfless-Traffic-Routing-Testbed