System-Optimal Congestion-Aware Routing for Multi-Class Mobility on Demand Service

Semester 8 Project Thesis

By: Niharika Shrivastava (IIT2016501)
Supervisor: Prof. Malika Meghjani
Co-Supervisor: Prof. O.P. Vyas

Department of Information Technology
Indian Institute of Information Technology
Allahabad, India

May 21, 2020

Candidate's Declaration

I declare that the project work titled "System-Optimal Congestion-Aware Routing for Multi-Class Mobility on Demand Service" done at the Singapore University of Technology and Design, Singapore and submitted at the Indian Institute of Information Technology, Allahabad is the bonafide work of Niharika Shrivastava(IIT2016501). It is a genuine record of my study carried out from January 2020 till present under the guidance of Prof. Malika Meghjani and Prof O.P. Vyas. Due acknowledgements have been made in the text to all the materials used.

Supervisor:

.....

Prof. Malika Meghjani

Co-Supervisor:

.....

Prof. O.P Vyas

Date: ..../..../.....

Place: Allahabad

Abstract

Recent developments in urban mobility have demonstrated the potential capacity of road networks to accommodate increasing traffic demand. However, all of the approaches are either sub-optimal or inefficient to perform real-time network analysis. This paper addresses the problem of real-time customer routing in a congested network. First, our aim is to provide congestion-aware routes for mobility-on-demand services that operate using a combination of a heterogeneous multi-class fleet. Every customer trip is distributed into multiple legs, i.e., walking or cycling for the first and last leg of the trip; cars or public transport for the middle leg of the trip. Second, we provide optimal transit points from one leg to another for this inter-modal (multi-class) service, such that the overall travel time is least within a capacity-bound transportation network. Finally, we build this framework into a social model by fulfilling all trip demands in a system-optimal way and in real-time. We showcase the effectiveness of our framework by using Singapore's network data from OpenStreetMap coupled with its real-time traffic data. We achieve 0% congestion for demand sizes of 70 (off-peak), 275 (moderately-peak), 2884 (high-peak), and a waiting queue of 2.1% demands during high-peak. There is a 65.5% query-save on finding the optimal transit points for the entire trip. Moreover, our solution also recommends alternate paths that are user-optimal.

Contents

1 Introduction

1.1 Motivation

Traffic congestion on urban road networks has become increasingly problematic since the 1950s [1]. With an increase in vehicles on the road and capacity of the roads remaining approximately constant, the speed of the traffic stream slows [2]. This results in higher travel times and increased pollution. With the increasing popularity of on-demand mobility services [3, 4, 5] like Uber, Grab, Yulu and Lyft, the traffic demand has started to reach road capacities more often, thereby enabling congestion to set in. All passengers inherently opt for the shortest route in the network in order to save travelling time, which ultimately results in blocked roads and very slow traffic movements [6]. All these factors lead to under-utilization of a city's intricately-built road network.

Many countries adopt the "odd-even rule" [7] that allows only a section of the vehicles to operate on a given day, for e.g., vehicles with odd number plate on Monday. In the UK, they reduced congestion by creating "out of town" [8] commercial areas and pedestrianising city centres, thereby reducing driveable routes in the most populated areas. Many works involving autonomous systems, where a central operator can manage system-wide actions, have tried to increase the throughput of a network. [9, 10] have shown that autonomous vehicles may be able to drive faster and follow other vehicles at closer distances without compromising safety, thereby effectively increasing the capacity of a road. Others have suggested ride-sharing services such as UberPool and LyftLine, or public transport to lessen the traffic burden of the [11, 12, 13]. Rerouting empty vehicles to potential regions of higher customer requests based on historical data has also shown to lessen region-wise network congestion [14, 15].

Multi-class fleets [16] allow breaking a customer trip into 3 classes: first mile, middle mile and last mile. The customer can choose to either walk on foot on pedestrian paths, use dedicated cycle-ways, or on-demand scooters for their first and last mile. The middle mile will cover up to 90% of the customer trip using the main road networks of a city and make use of fast-speed cars or public transports as per convenience. By using all three classes in an optimal combination, customers can maneuver through crowded paths and congested roads with ease. This also gives customers flexibility in choosing their preferred mode of transport based on accessibility and cost.

1.2 Problem Definition

In this paper, we present an efficient algorithm to route customer trips in a congestion-aware manner using a multi-class fleet of vehicles. Each leg of the trip is served by a different mode of transport depending on travel time and ease of the customer. The algorithm provides optimal transit points for each trip, i.e., till what point should a person walk/cycle and then take a car/public transport, in order to get the least travel time for the entire trip. All customer requests are routed into system-optimal paths in order to get the least travel time for the entire system. The optimal points are however user-centric and can therefore, act as a recommendation or alternate path.

2 Literature Review

1. In [11], car-pooling with 2-3 riders per vehicle was introduced in order to serve more customers with lesser vehicles on the road. [12] presented a more general model for real-time high-capacity ride-sharing that had rider-capacity of up to 10 simultaneous customers per vehicle. Their results showed that 98% of the taxi rides currently served by over 13,000 taxis (of capacity one) could be served with just 3,000 taxis (of capacity four). This can result in greater traffic throughput for increasing demand or lesser congestion for constant demand.

2. [17] tackled the problem of congestion-aware routing for a network consisting of Autonomous Mobility on Demand (AMoD) systems as they allow for system-wide coordination. It showed that intelligently routing and re-balancing each vehicle on capacitated roads did not increase congestion for small demand sizes. However, customer trips were considered event-based and routed greedily using an A* approach. This results in a Wardrop User equilibrium [18] which is considered sub-optimal. Moreover, the solutions could not be used for a real-time analysis [19].

3. [20] proposes congestion-aware route-planning policies for AMoD systems under mixed traffic conditions. Customer and re-balancing flows are routed in a system-centric fashion. However, for high levels of demand, pure AMoD travel can be detrimental due to the additional traffic stemming from its rebalancing flows. Private vehicles are routed in a user-centric way in order to adapt to the AMoD flows, which in turn, hampers systems to reach an optimal solution. By combining AMoD rides with moderate levels of walking, the overall system performance improved by 50%. However, there are no optimal transit points present to guide when to stop walking and start using an AMoD system. It also did not provide any route-recovery strategies for the flow solution.

4. In [16], multi-class fleets were introduced that serviced a request using a combination of heterogeneous vehicles (cars, buggies, scooter, walking). However, there was a constraint on the fleet size due to its cost. Different vehicles were assigned to each customer sequentially and with the objective of minimizing travel time for each mile. Due to this greedy approach, when the demand is much greater than the expected demand used to optimize the fleet size, the total average travel time for multi-class is worse than that of single-class.

5. Many works demonstrated new computationally efficient ways to replace the traditional process of the Traffic Assignment Problem (TAP). Frank Wolfe optimization [21] and Contraction Hierarchies [22] were used in [23] in order to achieve a system-optimal solution to the congestion-aware routing problem. There was a 20% improvement in computational time over the traditional TAP assignments. Even though the convergence rate of Frank Wolfe is known to be slow, its sub-steps could be easily made faster with parallel computing. In [24], shortest distances in a network are found using a hybrid algorithm that uses Euclidean distances as a comparative base line. The time complexity for this is $O(n^2)$. It's highly useful in cases of large cities where the total number of possible routes is huge.

6. in [20], various cost functions such as the Bureau of Public roads (BPR), 2-line approximation, 3-line approximation, and Davidson's heuristics were compared. In case of convex problems, BPR was an effective choice.

7. In [14], the network is optimally partitioned into re-balancing regions. Real-time demand estimate for every region is determined using incoming requests based on which idle vehicles are optimally assigned to these regions. There was a reduction in the average travel delay by 86%, the average waiting time by 37%, and the amount of ignored requests by 95% compared to [12] at the expense of an increased distance travelled by the fleet. The algorithm could not incorporate the existing public transportation infrastructure and congestion information of the network.

8. In [25], ride-sharing was explored in terms of how many vehicles were needed, where they should be initialized, and how they should be routed to service all the demand. There was a reduction in the fleet size by 69% and travel delay of 1.7 mins (for up to 2 customers per vehicle) and by 77% and 2.8 mins (for up to 4 passengers per vehicle). However, it's computationally expensive for online adaptation.

9. [13] computed future demands of passenger requests, based on historical data. The predictions improved the positioning of the vehicles towards satisfying future requests, reduced waiting and travel time. This lessened region-wise congestion by performing re-balancing of the network. However, seasonal changes was not taken into account while predicting future requests and the analysis was static in nature.

3 Proposed Methodology

3.1 Timeline

Timeline	Tasks
15 - 31 January	Study the previous work on ride-sharing, congestion-management, fleet sizing, assignment and re-balancing.
1 - 14 February	Build a road network from existing opensource GIS (e.g. open street maps)
15 - 29 February	Develop tools for querying APIs for public transport schedules and dynamic congestion flow information.
1 - 14 March	Integrate existing tools for graph queries and path planning algorithms.
15 March - 30 June	Develop novel algorithms for congestion-aware routing of multi-class fleet.

3.2 Method Overview

3.2.1 Phase 1

1. Setting up of Singapore city's road network (GIS) using the OpenStreetMap (OSM) database [26]. It consists of pedestrian paths, cycle-ways, and driveable roads, all interconnected via nodes.

2. Collection of Traffic Speed Band dataset using LTA Datamall [27], an API for retrieving static and dynamic network information of Singapore.

3.2.2 Phase 2

1. Integrate the traffic speed band dataset with our OSM GIS and calculate congestion information for the entire network, time-wise.

2. Propose an algorithm that finds optimal transit points for each mile (first, middle, last) for every customer request, such that the overall travel time for each customer is least.

3. Formulate a linear programming problem (LPP) that satisfies all origin-destination (OD) demands along with its capacity constraints, while minimizing travel time of the entire system.

3.2.3 Phase 3

1. Create a set of OD trip demands and run the proposed algorithm to find optimal transit points from the first to the middle mile, and from the middle to the last mile.

2. Solve the LPP using Frank-Wolfe algorithm to get system-optimal flows for all demands for the middle mile.

3. Decompose these flows into dedicated routes for each OD trip using a heuristic-based shortest successive path algorithm [28].

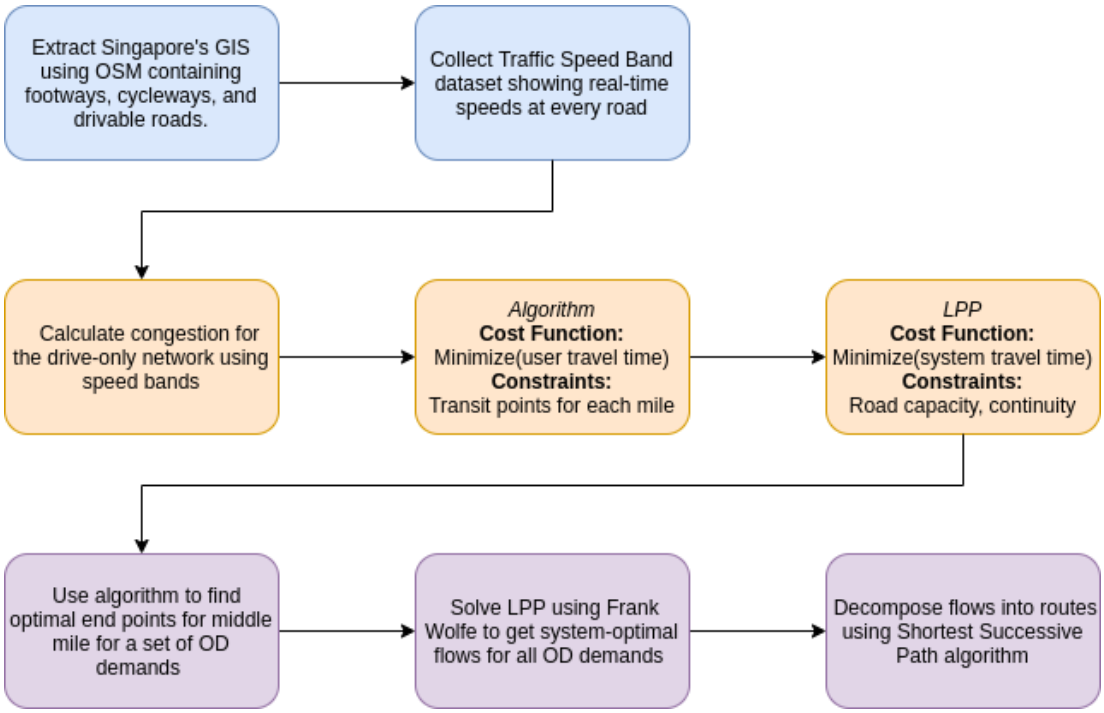


Figure 1: Project life cycle

4 Requirements

4.1 Data set

1. **Singapore's GIS:** Open Street Map (OSM) was used to extract Singapore's geographic information system (GIS) using the Overpass API.
   (a) The network is tagged with different types of roads or edges like pedestrian, cycle-ways, and drive-able routes. They constitute of corridors, cycle lanes, highways, service roads, etc.
   (b) The network is tagged with different types of nodes like bus stops, elevators, residential areas, parking lots, motorway junctions, etc.
   (c) Routing is possible from one node to another using a series of edges.
2. **Traffic Speed Band Data set:** It was collected from Land Transport Authority (LTA) Datamall owned by the Singapore Government.
   (a) It is a dynamic data set that updates itself every five minutes.
   (b) It shows speed bands (maximum and minimum observed speed) for every road link in Singapore giving real-time information about congestion on that link.
   (c) It was collected for two weeks including weekends, for time intervals of 09:00 A.M., 12:00 P.M., 03:00 P.M., 06:00 P.M., 09:00 P.M. .

4.2 Hardware and Software Requirements

1. A GPU with 12 GB RAM for fast computation .
2. **OSMnx 0.11.4:** Used to download, model, project, visualize, and analyze complex street networks from OpenStreetMap.
3. **NetworkX 2.4:** Used for path planning, routing and studying the structure and functions of Singapore's complex networks.
4. **Rtree 0.9.4:** Provided advanced spatial indexing features like Nearest neighbors search and Intersection search.
5. **Open Source Routing Machine(OSRM):** Its an open source router designed to compute the shortest path using contraction hierarchies or multi-level Dijkstra's.
6. **Python 3.6:** Interpreter language on top of which all these libraries would function.

5 Implementation

1. **Collect and combine data:** Extract Singapore's GIS using OpenStreetMap with the place name as *"Singapore, Central, Singapore"*. We get a drive-only network by specifying the network type as *"drive"*, and a pedestrian/cycle-way network by specifying the network type as *"walk"* or *"bike"*. Integrate the drive-only network with the traffic speed band data set for one slice of time, e.g., 6:00 P.M. The entire Singapore network can be visualized as a densely-connected directed graph $G(V, E)$ with V vertices (nodes) and E edges.

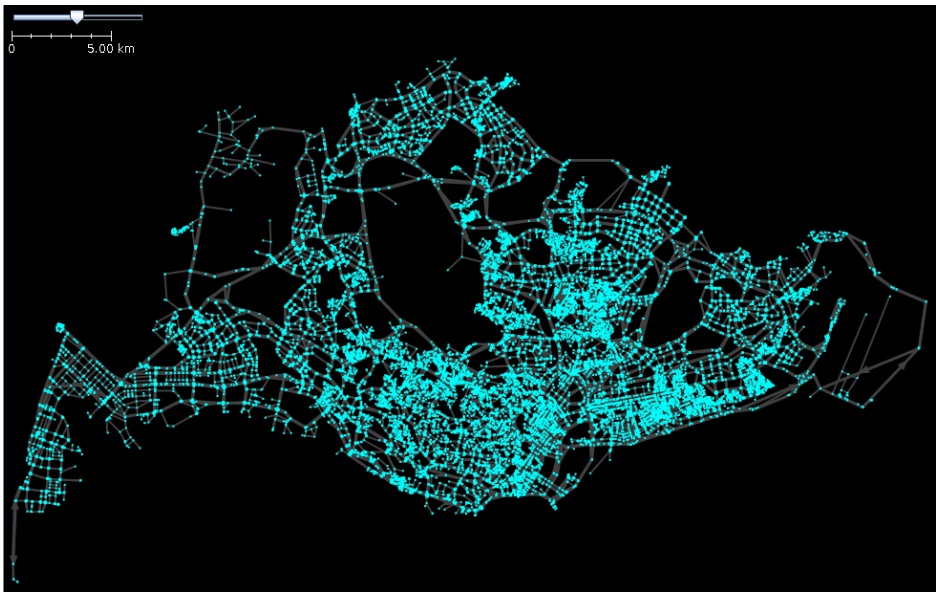


Figure 2: Nodes (V)

Figure 3: Edges (E)

2. **Compute congestion:** Calculate congestion information for the entire drive-only network. It is calculated in terms of how much travel time is required to cross a road link in the network.

   (a) If vehicles present on a road link (u, v) are within the link capacity $c(u, v)$, they are assumed to be operating at free flow speed. Thus, the time taken for vehicles to cross a road link at free flow speed is called the free flow time $t(u, v)$. However, if vehicles present on a road link exceed the link capacity, congestion starts to set in. This lowers the the speed at which they are operating and hence, travel time to cross that link increases.

   (b) The travel time $t_d(u, v)$ along each edge is computed using a heuristic delay function that is related to the current volume of traffic on each edge. We use the Bureau of Public Roads (BPR) delay model [29], which computes the travel time on each edge (u, v) $\in$ edges (E) as:

$$t_d(u, v) := t(u, v) \left(1 + \alpha \left(\frac{f(u, v)}{c(u, v)} \right)^{\beta} \right)$$

   where $f(u, v) := \sum_{m=1}^{M} f_m(u, v)$ is the total flow on edge (u, v), and α and β are usually set to 0.15 and 4 respectively.

3. **Compute middle mile transit nodes:** We now propose an efficient algorithm [1,2] that will be used to find optimal transit points for each mile (first, middle, last) for every customer request. Each customer request is composed of multiple legs(in this case, 3) served by different transportation modes (walking, cycle, taxi, public transport). The first and last miles can use only micro-mobility options like walking, cycling, or scooters. Whereas only private taxis or public transport can be used for the middle mile.

   For example, a customer will walk to a certain node in the graph for the first mile, after which they would alight a taxi till another node in the graph for the middle mile, and would cycle further until their destination for the last mile. These transit points would be selected such that the overall travel time for each customer is the least. Thus, every customer trip would be like:

$$\mathbf{X} \longrightarrow \mathbf{A} \longrightarrow \mathbf{B} \longrightarrow \mathbf{Y}$$

   X: Origin of customer request, start node of the first mile
   A: End node of the first mile, start node of the middle mile

B: End node of the middle mile, start node of the last mile

Y: End node of the last mile, destination of customer request

X and **Y** are constant for every customer request. Therefore, we only need to compute **A** and **B**. In order to exercise maximum flexibility in choosing our combination of transit points while minimizing the travel times for every mile, we first route for the middle mile, i.e., (**A** $\longrightarrow$ **B**) such that the overall travel time for each customer is least.

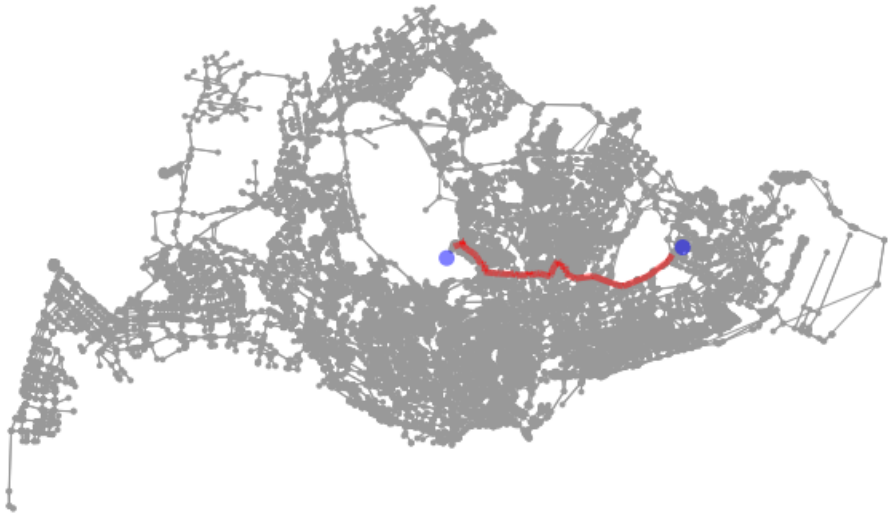


Figure 4: Routing shown for the middle mile (A —> B)

Algorithm 1: Find optimal transit nodes: A,B

Result: Demand set containing middle mile transit nodes

1: $Demand \leftarrow EmptyList()$;
2: $OD \leftarrow S, T$ {OD is set of all demands};
3: **for** $s, t \in OD$ **do**
4: $CandidateSources \leftarrow$ `GetNearestNodes(s, 720m)`;
5: $CandidateTargets \leftarrow$ `GetNearestNodes(t, 720m)`;
6: $EuclideanDict \leftarrow$ `SortLengthDict(`$CandidateSources, CandidateTargets$`)`;
7: $AB \leftarrow$ `ModifiedHybridSearch(`$EuclideanDict$`)`;
8: `Append` AB `to` $Demand$;
9: **end for**
10: $FirstMile \leftarrow$ `ShortestPath(`X, A`)`;
11: $LastMile \leftarrow$ `ShortestPath(`B, Y`)`;
12: **return** $Demand$

(a) To find an optimal starting **A** and ending node **B** for the middle mile, an assumption cum flexibility for the customer is made. According to it, all nodes within a 720 metre radius (euclidean distance) from the customer's starting coordinates **X**, are considered as potential sources and targets for the middle mile, respectively. This means that a customer is allowed to walk or cycle to/from their middle mile endpoints.

$$PotentialSources(A) = \{s_1, s_2, \ldots, s_n\}$$

$$PotentialTargets(B) = \{t_1, t_2, \ldots, t_m\}$$

A total of $m * n$ combinations of {source, target} pairs are made. We choose that {source, target} pair as **A** and **B** which takes the least travelling time from *source* to *target*.

(b) We use a modified version of the hybrid algorithm [2] used in [24] to query through the set of {source, target} pairs. The algorithm uses a base distance with which it compares all other distances to find the shortest path, in order to save time on querying. We impose an extra cutoff on the shortest path search by using an extension of the Secretary problem [30], i.e., we stop the search for the shortest path after at most 37% of {source, target} pairs have been seen.

(c) We use euclidean lengths as base distances for comparison with real lengths. Since our parameter is travel time, we calculate the euclidean length between each {source, target} pair and divide it by the permissible average speed of vehicles in Singapore (50 km/hr).

(d) To calculate the real-time travel time of each {source, target} pair route, we use Contraction Hierarchies with edge weights/cost as the Bureau of Public Roads delay heuristic. This outputs a {source, target} pair with the least travel time between them, given real-time congestion information.

4. **Compute system-optimum flows for middle mile:** In order to convert this framework into a social model and satisfy all OD demands, we formulate a linear programming problem. The objective or cost function is to minimize the travel time of the entire network, given a set of customer requests M such that, $m \in M$, where $m := (start, target)$. The capacity of the roads is used as a constraint for the upper bound on number of vehicles that can be at an edge E at the same time. Hence, the LP problem formulates as:

Algorithm 2: Modified Hybrid Search

```
Data: Dictionary X with key:(A,B); value: Euclidean distance, sorted in
      increasing order of value
Result: Nodes A,B s.t. travel time between them is least
Cutoff ← Floor(n/e); // Only search till first 37% of entries
Source ← First source of X ;
Dest ← First destination of X ;
minTime ← ∞;
for i ← 1 to Length(X) do
    travelTime ← ContractionHierarchies(Source, Dest);
    if i > Cutoff then
        if travelTime < minTime then
            Return current (A,B) pair as its optimal;
        else
            Return (A,B) pair optimal till now;
        end
    else
        if travelTime < minTime then
            Current (A,B) pair is optimal till now;
            if i < N then
                if travelTime < Euclidean distance of next pair then
                    Return current (A,B) pair as its optimal;
                else
                    Assign next pair as Source, Dest;
                end
            else
                Return current (A,B) pair as its optimal;
            end
        else
            if i < N then
                if minTime < Euclidean distance of next pair then
                    Return (A,B) pair optimal till now;
                else
                    Assign next pair as Source, Dest;
                end
            else
                Return (A,B) pair optimal till now;
            end
        end
    end
end
```

$$\text{minimize:} \quad \sum_{m \in M} \sum_{(u,v) \in E} t(u,v) f_m(u,v)$$

$$\begin{aligned}
\text{subject to:} \quad & \sum_{u \in V} f_m(u, s_m) + \lambda_m = \sum_{w \in V} f_m(s_m, w), && \forall m \in M \\
& \sum_{u \in V} f_m(u, t_m) = \lambda_m + \sum_{w \in V} f_m(t_m, w), && \forall m \in M \\
& \sum_{u \in V} f_m(u, v) = \sum_{w \in V} f_m(v, w), && \forall m \in M, v \in V \backslash \{s_m, t_m\} \\
& \sum_{m \in M} f_m(u, v) \leq c(u, v), && \forall (u, v) \in E
\end{aligned}$$

Constraints (2), (3) and (4) enforce continuity of each trip (i.e., flow conservation) across nodes. Finally, constraint (5) enforces the capacity constraint on each link.

Solving this LPP is NP-Hard. Hence we make use of a conditional gradient descent algorithm, Frank-Wolfe Optimization [31] using PyTrans. The OSM GIS is converted into *.tntp* [32] format. Flow assignment is made satisfying every demand to every road in the network in a system-optimal fashion. Hence, the solution of Frank-Wolfe optimization is feasible integral customer flows on congestion-free road links. It is to note that these customer flows belong solely on the drive-able network, since congestion is being monitored only for vehicles and public transport. These flows can be then decomposed into routes using a flow routing algorithm.

5. **Decompose system flows into path:** In order to decompose customer flows into dedicated routes:

   (a) We first sort the OD pairs using a "Maximum Neighbours" heuristic. We calculate the neighbouring nodes for every node in the shortest path (length-wise) of an OD pair. OD pairs having the least number of neighbours correspond to having the least route options. Therefore, these pairs are routed first, and so on.

   (b) The system-optimal flows act as the new capacity of the network. These capacities are used as an upper bound for a variant of the Successive Shortest Algorithm [28] [3].

Algorithm 3: Capacity-bound weighted Dijkstra

```
Data: source, target
Result: System-optimal route for an OD trip
paths ← {source: [source], target: emptyList()};
create vertex priority queue Q;
for each vertex v in Graph do
    dist[v] ← ∞;
end
dist[source] ← 0;
Q.add_with_priority(source, dist[source]);
while Q is not empty do
    u ← Q.pop_min();
    for each neighbor v of u do
        if capacity(u, v) > 0 then
            dist_vu ← weight(u, v);
            alt ← dist[u] + dist_vu;
            if alt < dist[v] then
                dist[v] ← alt;
                Q.add_with_priority(v, dist[v]);
                paths[v] ← path[u] + [v]
    end
end
return paths[target]
```

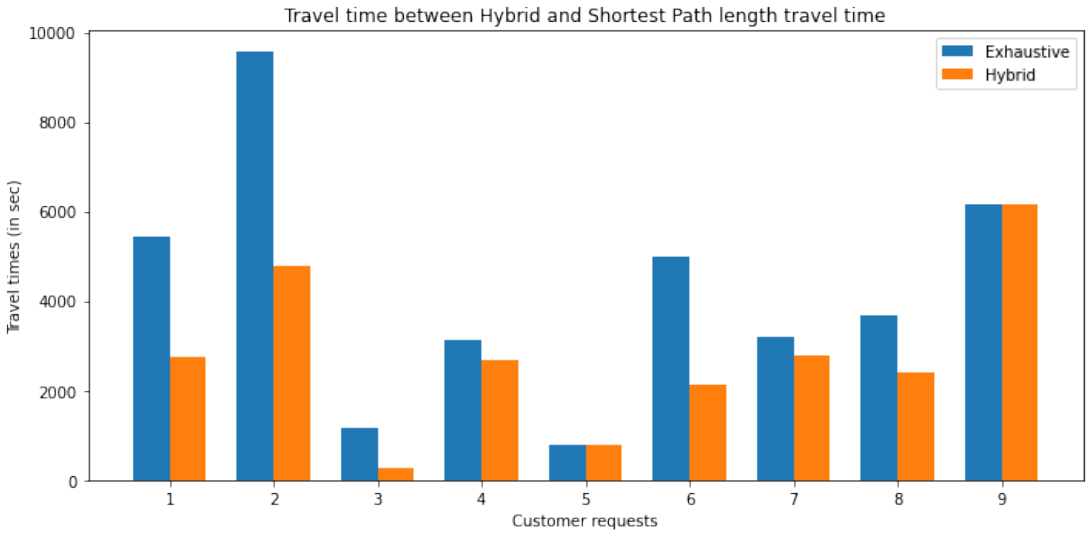
6. **Compute first and last miles:** To compute the first and last mile routes, only pedestrian roads and cycle-ways are used for routing. The customer is expected to travel these routes by either walking, cycling, or using on-demand scooters (like Yulu). These distances are calculated keeping in mind that a customer may want to (i) walk for a maximum of 10 minutes or 720 metres (assuming walking speed as 1.2 m/s), or (ii) cycle for a maximum of 15 minutes or 2 Km (assuming cycling speed as 5.4 m/s).

   (a) For the first mile, the starting node is **X**. The customer is expected to reach from **X** $\longrightarrow$ **A** as part of their first mile journey. This route is calculated using weighted-Dijkstra using edge weights as length of the paths.

   (b) For the last mile, the starting node is **B**. A route from **B** to **Y** is calculated using weighted-Dijkstra using edge weights as length of the paths. The customer is expected to reach from **B** $\longrightarrow$ **Y** in order to finish their entire trip.

6 Results

1. **Minimum distance path V.S. Minimum travel-time path:** A set of 10 OD trips is taken. (i) The blue bars represent the time taken to travel on its shortest-distance path, and (ii) The orange bars represent the time taken to travel on the least congested path.



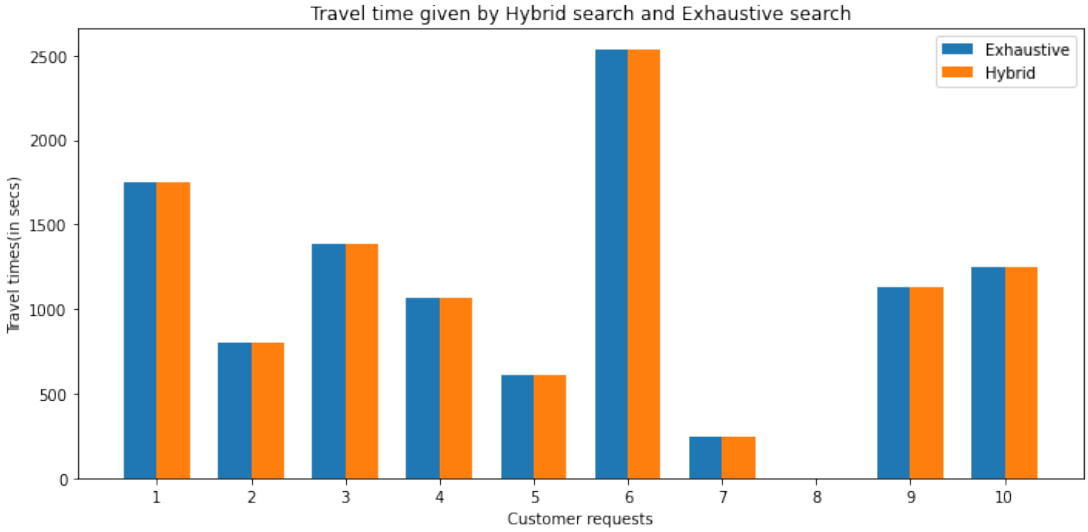
It shows that for 80% of the trips, the shortest distance path travel-time was very high. Hence, there existed a route that was greater in distance than the shortest path possible (length-wise), but the time required to travel this route was less. Therefore, the shortest-distance paths are not always the least congested paths.

2. **Exhaustive search V.S. Hybrid search:** In order to find a route with the least travel-time, given a set of OD pairs of size N, such that,

$$OD = \{(s_1, t_1), (s_2, t_2), \ldots, (s_n, t_n)\}$$

we can either search exhaustively through every OD pair till we find the minimum, or we can use a Hybrid approach, using base distances as a comparison.

   (a) **Variance between the travel times:** The exhaustive search always gives an OD pair with the minimum travel-time. We plot the travel times given by solutions of the Hybrid search to compare its performance against the Exhaustive approach.



Its seen that Hybrid Search also outputs the same OD pair having the least travel-time. Therefore, the Hybrid Search always gives an optimal solution.

$$\textbf{Variance(Exhaustive, Hybrid)} = \textbf{0\%}$$

(b) **Number of queries:** We tabulate the number of queries required to give an optimal output by both Hybrid Search and Exhaustive Search for 40 trips.

i. The exhaustive search iterates through all N OD pairs and finds an optimal solution in $O(N)$.

$$\textbf{Average no. of queries (Exhaustive)} = \textbf{2488.5}$$

ii. The hybrid search stops its iterations if the current OD pair distance is lesser than its next base distance. Hence, sometimes it takes $< N$ iterations to find the optimal solution.

$$\textbf{Average no. of queries (Hybrid)} = \textbf{1799.275}$$

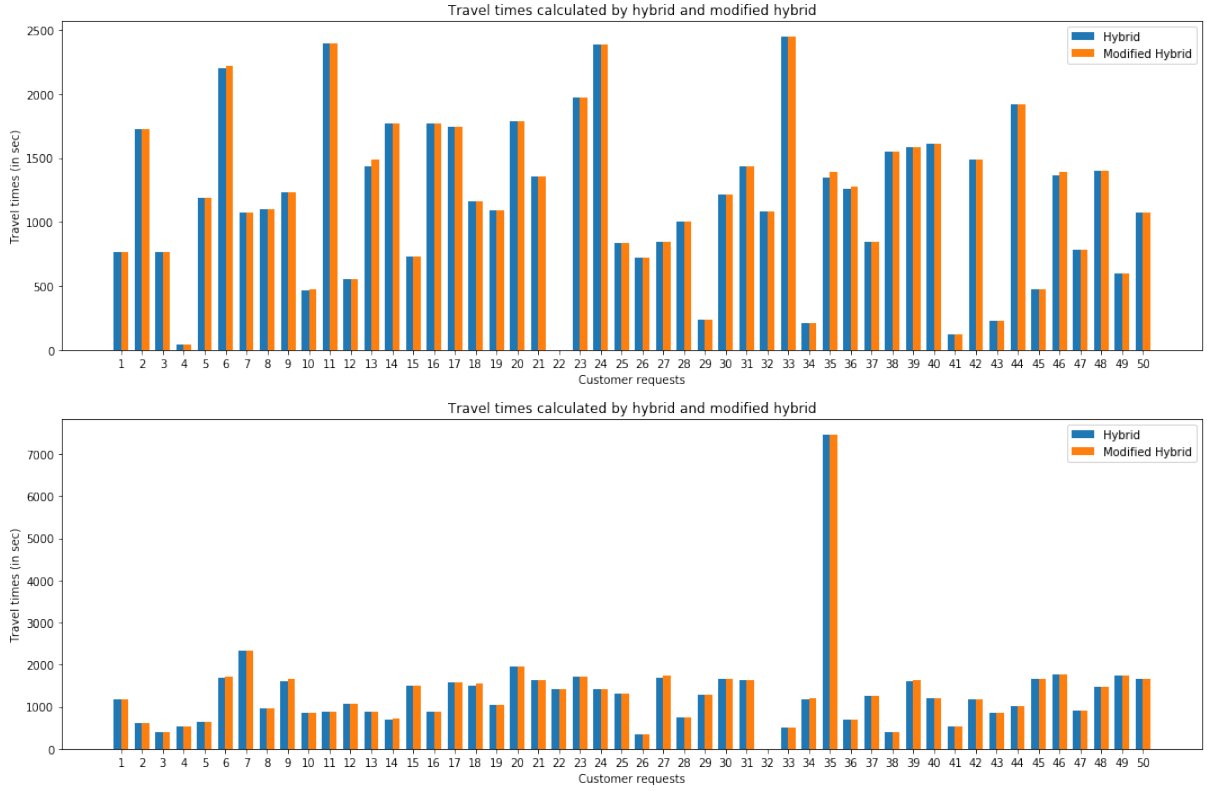
iii. The experiment shows that Hybrid Search finds the optimal value in lesser number of queries than the Exhaustive search. Hence, its more efficient.

$$\textbf{\% queries saved (40 trips)} = \textbf{31.93\%}$$

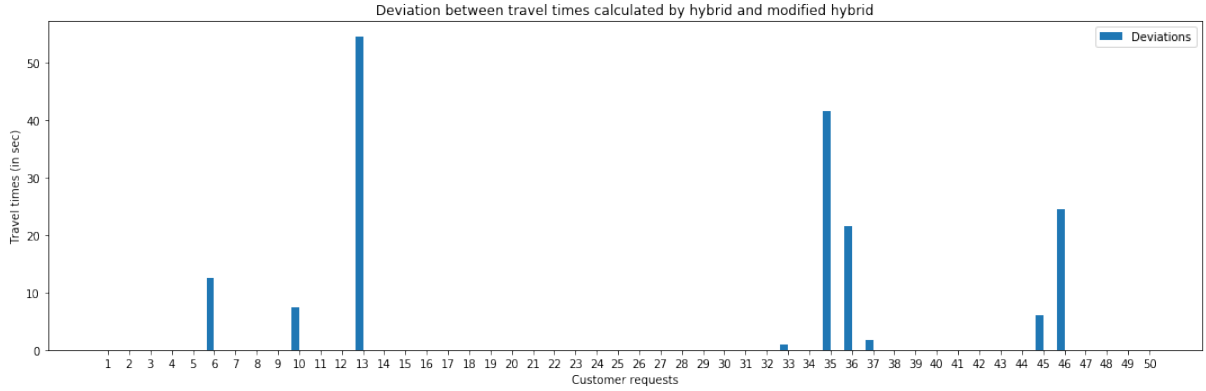
No.	Exhaustive	Hybrid	Saved%
1	2166	2166	0.0
2	1425	190	86.67
3	3150	3150	0.0
4	340	0	100.0
5	2144	2144	0.0
6	1587	225	85.82
7	1575	1575	0.0
8	2686	236	91.21
9	1778	1778	0.0
10	7072	7072	0.0
11	5600	172	96.93
12	3096	3096	0.0
13	455	165	63.74
14	5967	1415	76.29
15	1116	1116	0.0
16	1056	1056	0.0
17	2380	0	100.0
18	812	10	98.77
19	3672	373	89.84
20	200	200	0.0
21	2277	2277	0.0
22	5396	5396	0.0
23	442	2	99.55
24	1652	1652	0.0
25	1705	1705	0.0
26	3740	3740	0.0
27	1836	1836	0.0
28	2310	2310	0.0
29	5452	5452	0.0
30	6004	5924	1.33
31	465	465	0.0
32	864	864	0.0
33	2352	19	99.2
34	3009	3009	0.0
35	7316	7316	0.0
36	2656	2656	0.0
37	1920	171	91.09375
38	729	729	0.0
39	280	280	0.0
40	858	29	96.62

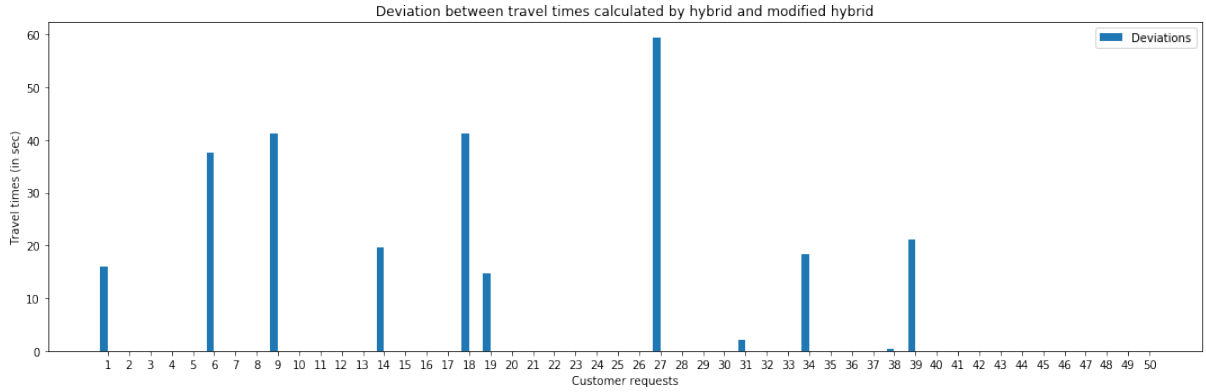
3. **Hybrid Search V.S. Modified Hybrid Search:** The Modified Hybrid Search uses an extension of the Hiring Secretary Problem on top of the classical Hybrid Search and stops after (N/e) iterations.

   (a) **Deviation between the travel times:** The travel times outputted by both the searches are plotted for 100 trip.



From the above figures, we see that the travel times outputted by the Hybrid Search act as a lower bound to the travel times outputted by the Modified-Hybrid Search. Hence, Modified-Hybrid Search is not always optimal. To see the deviation in optimality from the two approaches, we plot the difference in travel times below.





Its seen that the Modified-Hybrid Search is near optimal for 80% of the demand. Since the average deviation is < 10 seconds, this approach still results in very little delay.

$$\textbf{Average Deviation(Hybrid, Modified-Hybrid)} = \textbf{(3.42+5.44)/2} = \textbf{4.43 seconds}$$

(b) **Number of queries:** The number of queries required to find the optimal solution for each search is recorded for 100 trips. The results are as follows:

$$\textbf{\% Avg. queries saved(Hybrid)} = \textbf{(11.17+5.4)/2} = \textbf{8.285}$$
$$\textbf{\% Avg. queries saved(Modified-Hybrid)} = \textbf{(65.28+64.12)/2} = \textbf{64.7}$$

Therefore, Modified-Hybrid search finds near-optimal solutions 8x faster than Hybrid search.

4. **Congestion evaluation:** To evaluate congestion, we create demands of 3 different sizes: off-peak, moderately-peak, high-peak. All demands originate and end in Downtown Singapore, which is the busiest region of the city for majority of the day. After decomposing the system optimal flows into routes using shortest successive path, we record if all demands are satisfied

or not. Since, all the decomposed routes follow system-optimal flow constraints, they are bound to be within capacity. This means they all travel in a congestion-free network.

(a) **Off-peak (70 requests):** All demands are satisfied in a congestion-free manner. Thus, all vehicles travel at a free-flow rate.

(b) **Moderately-peak (275 requests):** All demands are satisfied in a congestion-free manner. Thus, all vehicles travel at a free-flow rate.

(c) **High-peak (2884 requests):** 97.92% demands are satisfied in a congestion-free manner. Thus, all these vehicles travel at a free-flow rate. The rest 2.08% trips could not be routed since the capacity of the network was filled. Hence, these are added to a waiting queue. These trips would be served in the next batch-process.

Demand Size	**Congestion Rate%**	**Waiting Queue Size**	**Cost of System (secs)**
Off-peak: 70	0	0	253,263.78
Moderately-peak: 275	0	0	1,109,868.73
High-peak: 2884	0	60	73,576,342.89

7 Conclusion

8 Future Scope

...

References

[1] R. W. Caves, "Encyclopedia of the city," *Routledge p.141*, 2004.

[2] B. Templeton, "Traffic congestion and capacity," *http://www.templetons.com/brad/robocars/congestion.html*, 2010.

[3] W. Mitchell, B. Borroni, and L. Burns, "Reinventing the automobile: Personal urban mobility for the 21st century," *MIT Press*, 2010.

[4] K. Spieser, K. Treleaven, R. Zhang, E. Frazzoli, D. Morton, and M. Pavone, "Toward a systematic approach to the design and evaluation of autonomous mobility-on-demand systems: A case study in singapore. in: Road vehicle automation," *Springer International Publishing, Cham, Switzerland*, 2014.

[5] M. Pavone, S. Smith, E. Frazzoli, and D. Rus, "Robotic load balancing for mobility-on-demand systems," *Int Journal of Robotics Research*, 2012.

[6] M. Barnard, "Autonomous cars likely to increase congestion," *http://cleantechnica.com/2016/01/17/autonomous-cars-likely-increase-congestion*, 2016.

[7] H. Ergo, "Implementation of odd-even rule in new delhi," *https://www.hdfcergo.com/blogs/car-insurance/implementation-of-odd-even-rule-in-new-delhi*, 2019.

[8] I. UK, "A car-free future? how uk cities are moving towards a pedestrian age," *https://www.independent.co.uk/environment/car-free-cities-pedestrianisation-cycling-driverless-vehicles-york-oslo-birmingham-a9299856.html*, 2020.

[9] J. Pérez, F. Seco, V. Milanés, A. Jiménez, J. Diaz, and T. De Pedro, "An rfid-based intelligent vehicle speed controller using active traffic signals.," *Sensors 10(6):5872–5887*, 2010.

[10] R. Zhang and M. Pavone, "Control of robotic mobility-on-demand systems: a queuing-theoretical perspective," *Proceedings of Robotics: Science and Systems Con- ference, Berkeley, CA*, 2014.

[11] P. Santi, "Quantifying the benefits of vehicle pooling with shareable networks," *Proc Natl Acad Sci USA 111(37):13290–13294*, 2014.

[12] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment," *Proceedings of the National Academy of Sciences of the United States of America*, 2017.

[13] J. Alonso-Mora, A. Wallar, and D. Rus, "Predictive routing for autonomous mobility-on-demand systems with ride-sharing," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[14] A. Wallar, J. Alonso-Mora, D. Rus, and M. van der Zee, "Vehicle rebalancing for mobility-on-demand systems with ride-sharing," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[15] K. Spieser, S. Samaranayake, W. Gruel, and E. Frazzoli, "Shared-vehicle mobility-on-demand systems: A fleet operator's guide to re-balancing empty vehicles," *Transportation Research Board 95th Annual Meeting, January Washington, DC*, 2016.

[16] M. Meghjani, S. D. Pendleton, K. Anna, Marczuk, Y. Hong Eng, X. Shen, M. H, J. Ang, and D. Rus, "Multi-class fleet sizing and mobility on demand service," *Comples Systems Design Management Asia*, 2018.

[17] F. Rossi, R. Zhang, Y. Hindy, and M. Pavone, "Routing autonomous vehicles in congested transportation networks: structural properties and coordination algorithms," *Springer Link: Autonomous Robots (May)*, 2018.

[18] J. G. Wardrop, "Some theoretical aspects of road traffic research," *Proceedings of the Institution of Civil Engineers*, 1952.

[19] V. Pillac, M. Gendreau, C. Guéret, and A. Medaglia, "A review of dynamic vehicle routing problems," *Eur J Oper Res*, 2013.

[20] S. Wollenstein-Betech, A. Houshmand, M. Salazar, M. Pavone, C. Cassandras, and I. Paschalidis, "Congestion-aware routing and rebalancing of autonomous mobility-on-demand systems in mixed traffic," *https://arxiv.org/abs/2003.04335*, 2020.

[21] M. Frank and P. Wolfe, "Frank-wolfe algorithm," *Frank-Wolfe algorithm Wikipedia*, 1956.

[22] J. Dibbelt, B. Strasser, and D. Wagner, "Customizable contraction hierarchies," *Journal of Experimental Algorithmics*, 2016.

[23] K. Solovey, M. Salazar, and M. Pavone, "Scalable and congestion-aware routing for autonomous mobility-on-demand via frank-wolfe optimization," *https://arxiv.org/abs/1903.03697*, 2019.

[24] M. Meghjani, S. Manjanna, and G. Dudek, "Fast and efficient rendezvous in street networks," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[25] A. Wallar, J. Alonso-Mora, and D. Rus, "Optimizing vehicle distributions and fleet sizes for mobility-on-demand," *International Conference on Robotics and Automation (ICRA)*, 2019.

[26] OpenStreetmap and Wiki, "Openstreetmaps," *www.openstreetmap.org/*, 2004.

[27] Government, of, and Singapore, "Lta datamall," *mytransport.sg/content/mytransport/home/dataMall/*, 2020.

[28] TopCoder, "Data science algorithms: Successive shortest path," *topcoder.com/community/competitive-programming/tutorials/minimum-cost-flow-part-two-algorithms/*, 2016.

[29] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems, Science, and Cybernetics*, 1968.

[30] P. Freeman, "The secretary problem and its extensions: A review," *International Statistical Review / Revue Internationale de Statistique*, 1983.

[31] P. Freeman, "Pytrans/urban network analysis," *https://github.com/PyTrans/Urban-Network-Analysis*, 2018.

[32] T. N. for Research Core Team, "Transportation networks for research," *https://github.com/bstabler/TransportationNetworks*, 2016.