IMPROVING TRANSPORTATION SERVICES BY CENTRALIZED ROUTING

Richard Perryman [1], Loic Nassif [2], Amjad Mobayed [3]

ABSTRACT

The popular approach to match customers and vehicles in a taxi-like service (e.g. Uber) is to assign the closest available cab to the newly appeared customer. Once assigned, the cab's destination is locked to the customer's location. We decide to look into the idea of recalculating the optimal assignment of cabs and customers in a city map where customers appear following a random distribution. We investigate the efficiency of changing the the routes of already assigned cabs to different customers to minimize the overall average waiting time of customers. We apply our model to two different situations; one where the cabs are human-driven and the other when the cabs are self-driven. We found that for all cases, rerouting or not, human driven or self-driven, the results are very similar. By small margins, the most efficient method was to have self-driving cabs in a rerouting model.

Keywords: Vehicle Routing Problem, Centralized Routing, Probabilistic Model, Heuristic Algorithms, MAT482/1750 Final Project

[1]???
[2]Student ID: 1001309808
[3]???

INTRODUCTION

Background

The problem we are tackling falls under a larger set of problems named Vehicle Routing Problems (VRP). These problems have been studied very extensively in the past 60 years(Laporte 1992)(C. Koulamas 1994), and the heuristic methods developed and analyzed in past research will influence our choices of algorithms. The VRP is the challenge of finding optimal paths for one or more vehicles to one or more customers scattered around a location such as a city. This can be solved with a number of different approaches. These approaches can usually be classified as exact algorithms or heuristic, or approximate, algorithms. The problem with exact algorithms, such as direct search trees, is that the VRP is an NP-hard problem. This makes solving the problem for a realistic number of vehicles and customers impossible. Exact algorithms can solve the problem for a couple dozen customers(N. Christofides 1981). This is simply not useful for our purposes and so we turn to heuristic approaches. The VRP can often times be represented as a programming problem and as such can be solved using relaxation methods such as Lagrangian relaxation, where the Lagrangian multipliers are used as a weighted penalty for violating the inequality constraints, instead of making these constraints completely binding. A popular search algorithm that we will use a variation of in our model is called the A^* search algorithm. The basic idea of A^* is that the algorithms will consider all possible combinations of nodes between the vehicle and the customer and pick the one that minimizes a particular cost, in our case the distance travelled by the vehicle to the customer.

It seems that none of the previous literature looks at the VRP when there is a time-dependent aspect to the system. As more customers appear randomly throughout the city, the optimal path between vehicles and customers may change in such a way that reassigning vehicles-customers pairing may reduce the overall travel and wait time.

Motivation

There are a few reasons why we might suspect this to be a good idea to investigate. In terms of practicality, reducing customer wait time increases customer satisfaction, while reducing travel time reduces maintenance cost for the business, and increases the ability to serve more customers with fewer vehicles. The idea of rerouting may seem to be a very real possibility for the near-future of self-driving cabs, since machines can react instantaneously to immediate and multiple rerouting changes that a human driver may not be able to handle as efficiently.

The Model

A large part of the development of our model is the construction of our model city from which all of our tests are gathered from. The way that a city is modeled is to represent each intersection as a node in a graph. The challenge then is to find a way to link up thousands of nodes to form a city network. It is in our effort to create the network where most of our model assumptions arise. To begin with, we assume each link, which is analogous to an edge on a graph, is connected in a straight path. This is not a bad assumption to make for a large metropolitan city, since most of these cities are arranged in a grid-like configurations and most of their major roads are straight. The distance from one node to its neighbouring node is a value saved within the node. The next assumption is tied to the data that is available to us online. We tie the frequency of customers appearing at a given intersection to the density of pedestrians at the intersection. This naturally means that for an intersection to be useful to us in this model, we need to find data on the pedestrian density for each intersection. The Toronto Open Data Catalogue has a data set of vehicle and pedestrian volumes over the 8-hour traffic peak time at intersections that have traffic signals(City of Toronto's Traffic Safety Unit, Traffic Management Centre, Transportation 2018). This eliminates a lot of Toronto's intersections. Another Toronto data set, one simply registering all intersections within the City of Toronto(Geospatial Competency Centre 2018), has a count of about 50000 intersections. Contrasting that to ~ 2000 intersections with volume data shows that a majority of Toronto cannot be accurately modeled.

We continue our assumptions with the reasonable assumption that every node can be accessed, through some path, from any other node. This is usually true for most cities, although we do not consider things such as one-way streets or closed-off streets that may inhibit the path of the vehicle.

Another important aspect of our model is how customers and vehicles behave. We model the customer appearance rate at an intersection using a random process. The likelihood of customers appearing is scaled by the volume of pedestrians at the given intersection. This is using the assumption that more busy intersections will harbor more customers. If we consider human drivers, then we introduce a reaction delay time. In many cases, the rerouting may occur unexpectedly and the human driver may need a few seconds to react to the change. When using self-driving cars, this constraint is not considered.

The centralized system that assigns the employees driving vehicles to the waiting customers is modeled simply as a computer that has access to the location of each customer and each vehicle. This computer calculates routes for the employees to follow along with how far the vehicle would have to travel using an extension of Dijkstra's Algorithm (Dijkstra 1959) commonly called A^*, which uses the location of the intersections on the Earth to improve performance. The computer then uses these routes to find a close to optimal set of pairings in order to minimize the distance traveled by the vehicles. The results are not exact due to some pruning done to meet time constraints, but there is very rarely a significant difference.

CITY MODELING

There are no real benefits for our goals to model Toronto accurately. The point here is to be able to model an example city accurately. We would simply need a graph that accurately describes the general patterns of a metropolitan city, and so a crude approximation of Toronto suffices. There are a few ways of linking the nodes. The first one is to simply link them randomly. The danger with doing so is that clustering may occur. That is, sub-networks of nodes, detached from all other nodes, may start to form. To prevent this, before randomizing the links, We connect all nodes in a successive chain, ensuring that all nodes are connected.

Odd patterns that do not capture the structure of a city may form if we create a network randomly. A better approach would be to connected nodes that are geographically near-by. We can allow a node to connect to its four closest nodes and do so for all other nodes in the graph. The data retrieved from the City of Toronto saves the locations of the intersections in a MTM 3 Degree Zone 10 NAD27, WGS84 Latitude/Longitude projection. That is the latitude and longitude are in degrees. We first convert these coordinates in radian and apply the Haversine formula to find the distance,

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos\varphi_1 \cos\varphi_2 \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \tag{1}$$

where r is the radius of the earth, φ_2 and φ_1 are the latitudes of point 2 and point 1, respectively, and λ_1 and λ_2 are the longitude of point 1 and point 2, respectively.

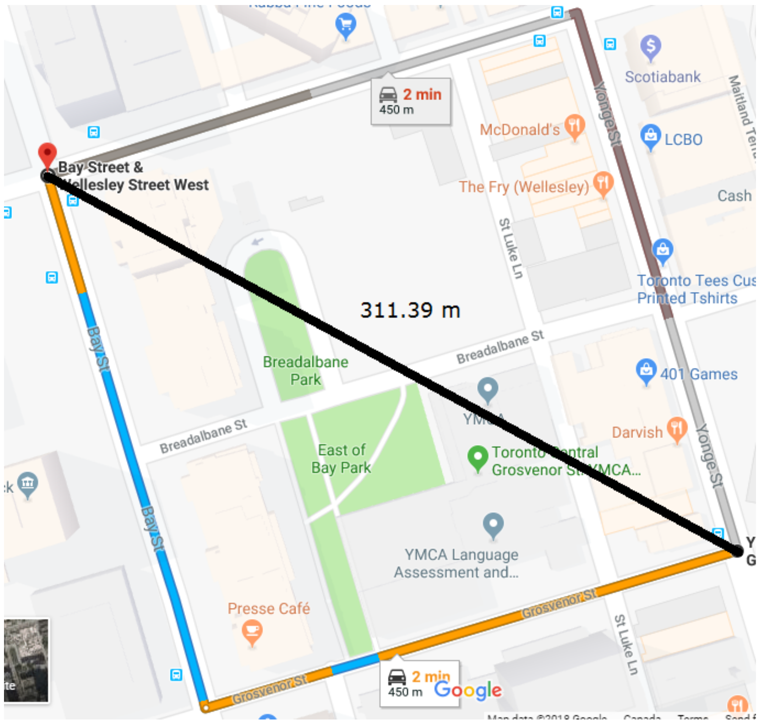


FIG. 1. An issue highlighted by the fact that not all intersections are considered in our model.

Shortcomings to this approach exist. For example, in Figure 1, the black line drawn is how our link of the two nodes is conducted. This is of course not what the real map looks like, and so accuracy in simulating Toronto is diminished. On the other hand, the actually geometry of the graph is not as important as its link density connection. The average intersection in our model is linked to 3.0 other intersections. This is similar to what a city like Toronto would be expected to have.

The area of Toronto that we consider is from the latitude 43.591686475 to latitude 43.85545 and longitude -79.63929 to longitude -79.3896419. This is equivalent to the bottom left corner being at 97 City Centre Dr, Mississauga, ON L5B 1M7, Canada and the top right corner at 35 West Beaver Creek Rd, Richmond Hill, ON L4B 1K4, Canada. Table 2 demonstrates such a range. The number of intersections in that range is about 26862. We captured about 1210 of these intersections. This might seem little, but do note that all intersections that have vehicle and pedestrian volume data are intersections with traffic signals, and these intersections are the ones that tend to be larger, and more important for our purposes, than the intersections without traffic signals.

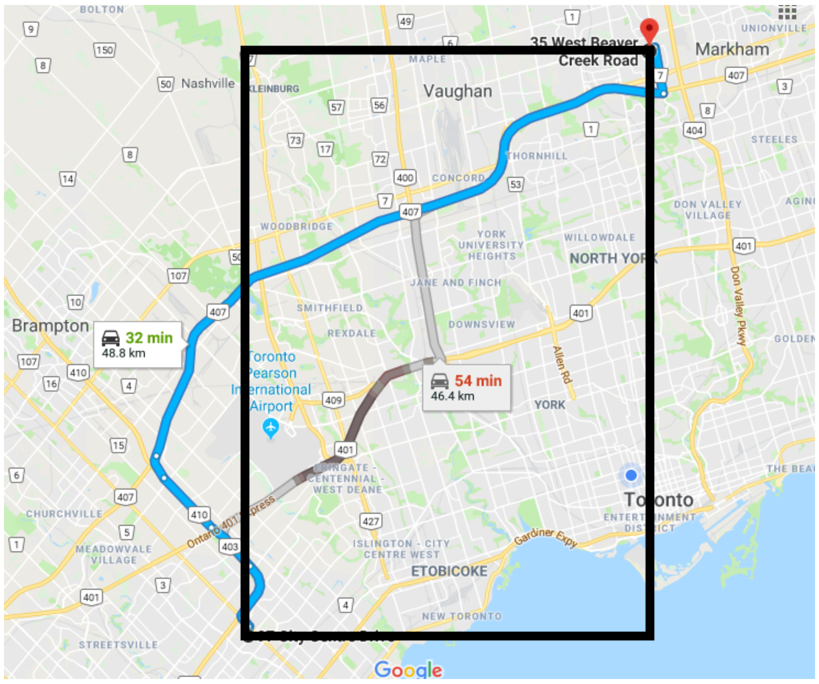


FIG. 2. The extent of our model covering parts of Toronto.

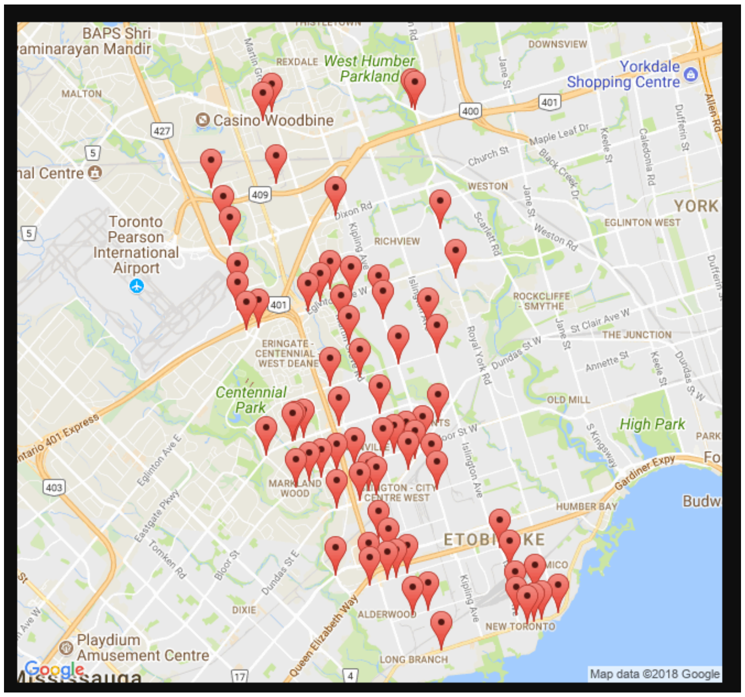


FIG. 3. A small representation of our intersections distribution in a small area of our model's region.

Table 1 shows the contrast between our model's and Toronto's distribution of intersection types. The meaning of each specific type is not very relevant to discuss, but the point is that our model samples a large portion of Minor-Single Level intersections, which is the dominant type of intersection in Toronto. What our model fails to capture are Lesser-Single Level intersections, which our model has none of, but consists of a large portion of Toronto's infrastructure. This is explained by the fact that Lesser-Single Level intersections are intersections that involve one-way roads. One-way roads are usually associated with smaller street intersections which normally do not posses traffic signals, which is a binding criteria for our model.

TABLE 1. Distribution of street types between our model and Toronto in the area considered.

	Our Model (1210)	Toronto (26862)
Minor-Single Level	1068 (88.26%)	18159 (67.60%)
Minor-Multi Level	39 (3.22%)	488 (1.82%)
Major-Single Level	98 (8.10%)	252 (0.94%)
Pseudo Intersection Overpass/ Underpass	2 (0.17%)	1393 (5.19%)
Major-Multi Level	3 (0.25%)	17 (0.06%)
Lesser-Multi Level	0 (0.00%)	108 (0.40%)
Pseudo Intersection-Single Level	0 (0.00%)	670 (2.49%)
Statistical-Single Level	0 (0.00%)	7 (0.03%)
Expressway Interchange-Single Level	0 (0.00%)	312 (1.16%)
Lesser-Single Level	0 (0.00%)	5456 (20.31%)

CUSTOMER MODELING

Denote $X_{p,i}$ as the number of people in the 8-hours interval in the i^{th} region. Assuming every hour has the same number of people, divide the number of pedestrians by 8 for each region i and denote it by $x_{p,i}$. The results are in the Excel sheet.

For the average rate, we've done it in two different ways. The first way is by taking the average (mean) of the number of people per hour,

$$\text{Average of Pedestrians} = \frac{\sum_{i=1}^{n} x_{p,i}}{n} \tag{2}$$

and the second way is by using the weighted sum, which for our model and goals is a more reasonable approach.

First, we've summed the number of pedestrians, then dividing the number of pedestrians x_i of each region by the total number of customers to obtain the weight of the pedestrian in a given region, which we denote as $w_{p,i}$,

$$w_{p,i} = \frac{x_{p,i}}{\sum_{i=1}^{n} x_{p,i}} \tag{3}$$

To get the weighted sum, multiply each weight with its corresponding number of pedestrians. Afterwards, take their sum and divide that sum by the sum of the weights,

$$\text{Weighted Average of Pedestrians } (\alpha) = \frac{\sum_{i=1}^{n} w_{p,i} x_{p,i}}{\sum_{i=1}^{n} w_{p,i}} \tag{4}$$

Next, we fix a percentage of pedestrians whom are the customers that will appear in a given region. The percentage we considered reasonable was 2%. This number comes from the ratio of cab trips in Toronto daily [1] and the total population of Toronto [2]. In other words, 2% of pedestrians will be customers and we denote it by $x_{c,i}$,

$$\text{Number of Customers in a given Region}(x_{c,i}) = 0.02\alpha \tag{5}$$

Next, we find the weight of the customers, $w_{c,i}$,

$$w_{c,i} = \frac{x_{c,i}}{\sum_{i=1}^{n} x_{c,i}} \tag{6}$$

Proceeding, we find the weighted average of customers, λ, by mimicking similar calculations that were performed for the weighted average of pedestrians,

$$\text{Weighted average of customers}(\lambda) = \frac{\sum_{i=1}^{n} w_{c,i} x_{c,i}}{\sum_{i=1}^{n} w_{c,i}} \tag{7}$$

Finally, we use a Poisson distribution to find the probability of customers being picked up in the given hour. As previously stated, y represents the number of customers being picked up at a given region, and Y is the random variable,

$$P(Y = y) = \frac{\lambda^y e^{-\lambda}}{y!} \tag{8}$$

These flat rates were also used to simulate traffic, by replacing each λ with a function: $\lambda(t) = \lambda\psi(t)$. The function ψ was chosen to have a spike shape around when we expect rush hour to occur. Once we had chosen an appropriate ψ we rescaled the random variable to have the same weight by numerically integrating and then rescaling.

VEHICLE BEHAVIOUR

Modelling how the employees of this transportation service drive proved to be difficult, if only because there was almost no data to use as a base for assumptions. The

[1]https://en.wikipedia.org/wiki/Taxicabs_of_Canada

[2]https://en.wikipedia.org/wiki/Toronto

vehicle movement was also the most driving reason behind choosing that most of the model would be agent-based. Determining a function for the time until an employee vehicle picks up a given customer would be extremely difficult due to the large number of dependent variables that it would rely on. Too much information would be lost by grouping the discrete customers and vehicles together into groups large enough to treat as continuous variables, and there would still be a large number of decision variables, such as dispatch, that would further complicate any attempts at deriving an ODE or PDE representation of the time a customer would wait.

A detail that was important for our model to capture was the effect of human error on the benefits of constant rerouting. We assumed that when a human driver was given a new route, they didn't realise they should change their current behaviour with some probability, p. We then assumed that each consecutive time they received a route from the central computer, the probablity of them continuing not to realize that they should change behaviour would be p/N where N is the number of times in a row that this has occurred. If we let M be a random variable equal to the number of errors a given driver makes, then we can say:

$$\begin{aligned} m(k) &= \frac{p^k}{k!}(1-p) \\ E[M] &= \sum_{k=0}^{\infty} k\frac{p^k}{k!}(1-p) \\ &= (1-p)\sum_{k=0}^{\infty}(p/k!)\left(\frac{d}{dp}p^k\right) \\ &= e^p p(1-p) \\ VAR[M] &= E[M^2] + E^2[M] = e^p p^2(1-p) \end{aligned}$$

This somewhat reasonably gives a random variable with a very small mean and variance for small p. This gives roughly what was expected, which is usually no error occurs, and rarely one or two errors occur for p near 0.15. The idea behind such a choice was to make the model robust with respect to the choice of p.

In reality, employess of a service company similar captured by our model would likely tend to drive to particular areas, such as an airport. We attempted to relax this behaviour by having the employees drive in a random walk. This would cause them to usually stay in roughly the area of the customers that they had picked up previously, while still giving enough variation in the distribution of vehicles around the city. Using similar reasoning, the employees drive the vehicles randomly while they are carrying a passenger.

The last assumption made about vehicle behaviour for this model is also one of the most significant. Each vehicle travels from its current intersection to the next intersection in a single time step. This was done to greatly simplify the structure of the model. A way around this that is likely worth considering for extending this model is to add a series of queues between intersections, with cars only appearing at the other end once they have waited long enough in the queue. This could have allowed for a better description of traffic in the model, which was one of the weaker points in the design.

OPTIMAL ROUTER

Modelling the central computer that performed the routing calculations required relatively few assumptions compared to the rest of the model. The most significant asusmption made was that the communication between the vehicles, customers, and the router was absolutely perfect. Beyond that, the computer had an assumed perfect uptime and needed only to meet the time limit of one minute imposed by the assumption for vehicle movement.

To meet the time requirements, the algorithm used to find the optimal dispatch of vehicles to customers needed to be intelligent enough to avoid unnecessary computations. For the purposes of this model, performing repeated path finding routines using a modification of Dijkstra's algorithm (Dijkstra 1959) using a guiding heuristic, which is often called the A^* algorithm (Zeng and Church 2009). Simply put, the approach is to visit the node that the heuristic predicts is closest to the destination. Then the program updates the actual distance to that node and adds the neighbours of that node to the set of nodes it should check. The heuristic being used was the same great circle distance as was used for constructing the connections between intersections.

This heuristic was further used in two other scenarios in order to decrease the amount of time spent finding the optimal routes. When considering a particular vehicle and customer pair, if the distance predicted by the heuristic was larger than an actual distance already found to the customer from some other vehicle, the current vehicle was skipped, as there was no possible way for an improvement to be made. In addition, when all of the distances from each vehicle to each customer was found, finding the subset of all sets of routes that minimises the total distance traveled by the vehicles is difficult. Instead of finding that, the algorithm that the central computer runs greedily chooses the best vehicle for each customer one at a time, and never backtracks to try to improve by swapping previous decisions. Since there are generally far more vehicles than customers, this is usually almost exactly correct.

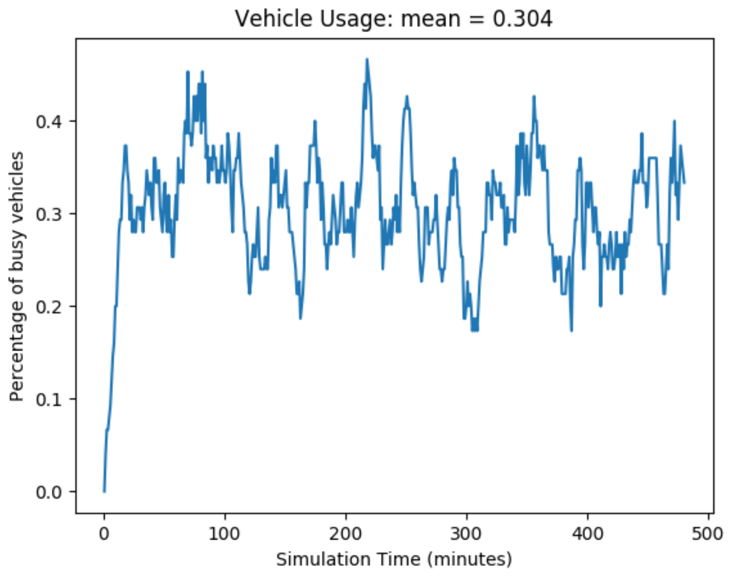
Algorithm 1 A* algorithm used for vehicle routing

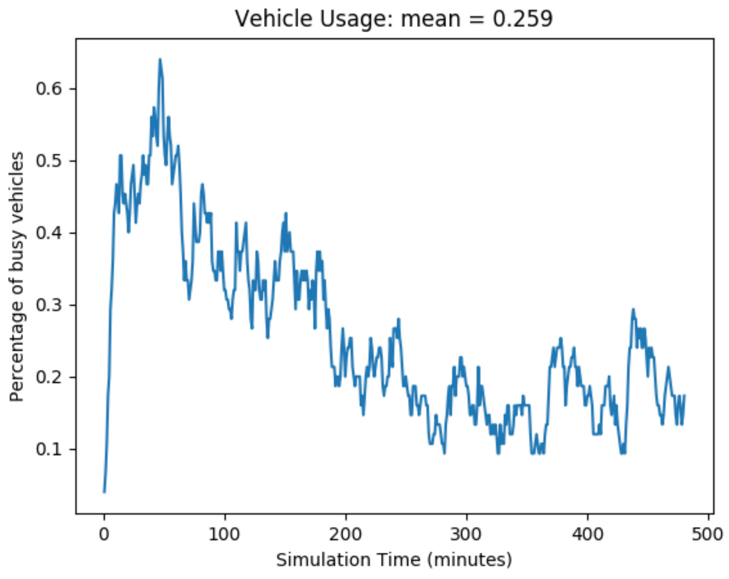
```
1:  function OPTIMAL ROUTING( $customers, vehicles$ )
2:      $shortest \leftarrow \{\{customer, vehicle\} \mapsto route\}$        ▷ The initial routes have infinite length
3:      $overall \leftarrow \{customer \mapsto \infty\}$
4:      for $vehicle \in vehicles$ do
5:          $visited \leftarrow \emptyset$
6:          $toVisit \leftarrow \{vehicle\}$
7:          $startToNode \leftarrow \{vehicle \mapsto 0\}$        ▷ Others are infinite
8:          $prevOptimalNode \leftarrow \{node \mapsto node\}$
9:          for $customer \in customers$ do
10:             $heuristicMap \leftarrow \{vehicle \mapsto heuristic(vehicle, customer)\}$
11:             if $heuristicMap(vehicle) > overall(customer)$ then
12:                 continue
13:             end if
14:             repeat
15:                 $current = \arg\min_{n \in toVisit}(heuristic(n, customer))$
16:                 if $current = customer$ then
17:                     break
18:                 end if
19:                 $toVisit \leftarrow toVisit \setminus current$
20:                 $visited \leftarrow visited \cup current$
21:                 for $neighbour \in neighbours(current)$ do
22:                     if $neighbour \in visited$ then
23:                         continue
24:                     end if
25:                     $toVisit \leftarrow toVisit \cup neighbour$
26:                     $dist \leftarrow startToNode(current) + neighbourDist(neighbour)$
27:                     if $dist < startToNode(neighbour)$ then
28:                         $previousOptimalNode(neighbour) \leftarrow current$
29:                         $startToNode(neighbour) \leftarrow dist$
30:                         $heuristicMap(nbr) \leftarrow dist + heuristic(nbr, customer)$
                            ▷ Name shortened for typesetting
31:                     end if
32:                 end for
33:             until $toVisit = \emptyset$
34:             $route \leftarrow reconstructRoute(prevOptimalNode, dest)$
35:             $shortest(\{vehicle, customer\}) \leftarrow route$
36:             $overall(customer) \leftarrow \min(routeDistance(route), overall(customer))$
37:         end for
38:     end for
39:     return $shortest$
40: end function
```

RESULTS

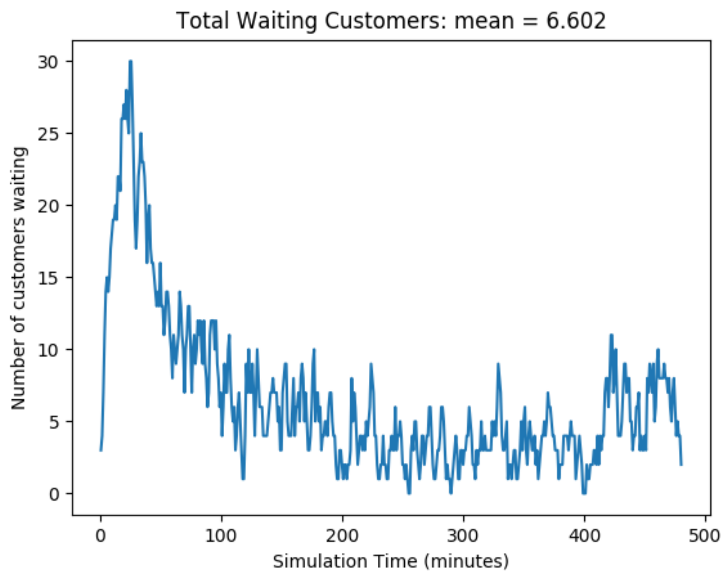
Overall, the results indicate that performing frequent rerouting operations is superior to the more mundane strategy of routing only when new customers appear. However, there are some caveats that suggest the model is not quite accurate to reality.

We observe that the model behaves as expected when exposed to traffic, and that the system adapts very quickly to changes in rates:

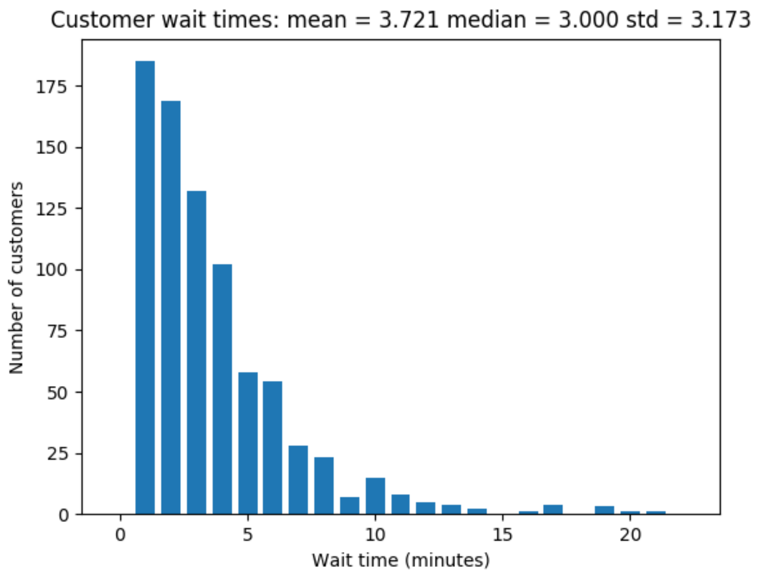


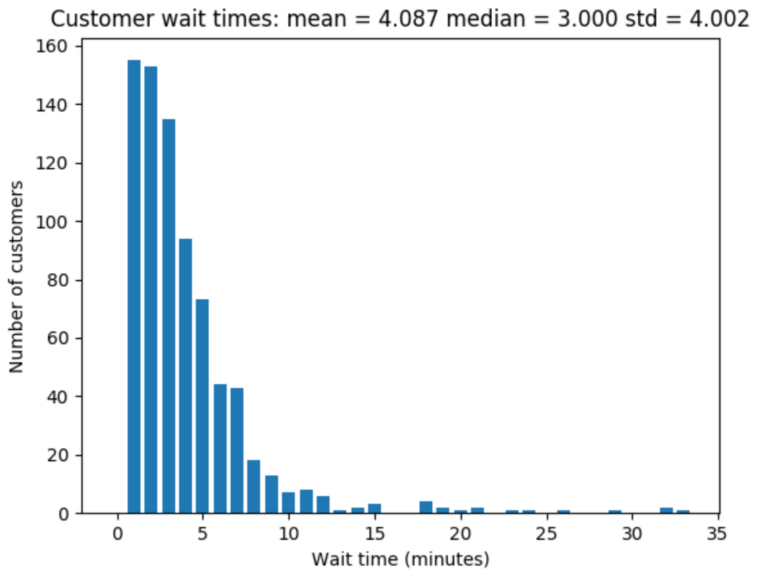


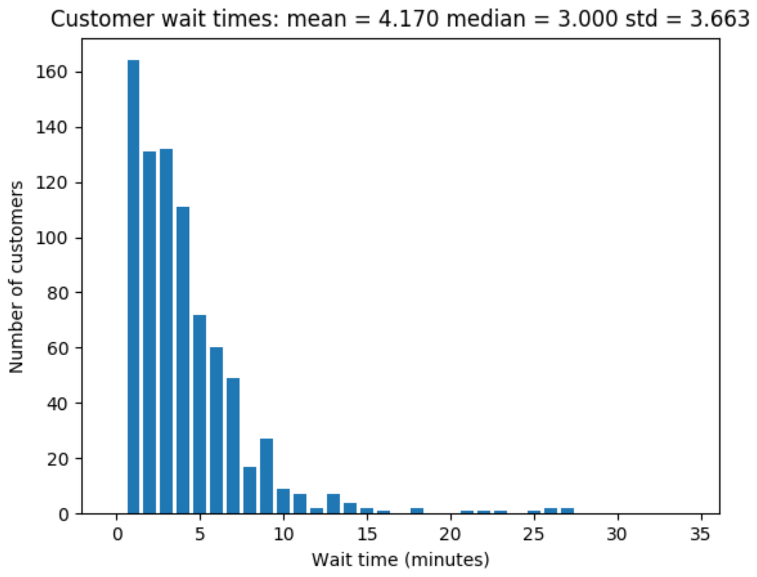
The reason we consider the vehicle usage over the number of pedestrians waiting to be picked up, is that the customers are much noisier, but otherwise tell the same story. The additional noise likely comes from the large number of customers that are picked up almost immediately. The plot below shows results for the same run of the model as the traffic case above:



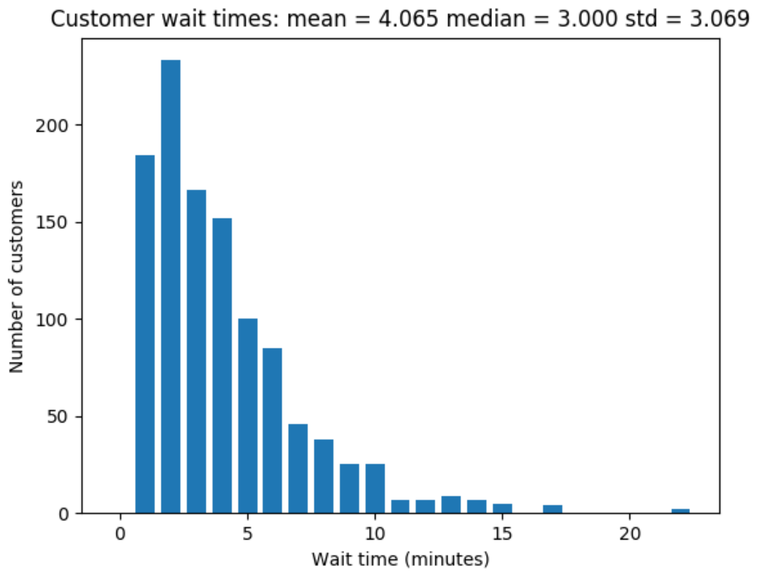
We also observe that our self-driving car model has better performance than the human driver model. We further see that human drivers are superior to self-driving cars when the self driving cars are not subject to constant rerouting:







It was incorrectly stated in the presentation that the model was highly sensitive to the probability that humans made errors. There was likely an error in labelling data, as the results could not be reproduced. The above plot shows an initial error rate of 15%, and the one below shows results for an error rate of 25%. The results are noticeably worse, but still better than without rerouting.



We found that in our cases with about 75 vehicles on 1210 intersections and 900 customers over the eight hours, that around 450 total rerouting events occurred. That means that the number of times a vehicle received a route that was different from its previous route was aroud 450. This helps explain the better results of the rerouting algorithm, as that means there was on average around one reroute every single time step, as eight hours is 480 minutes, and each step is a minute.

REFERENCES

C. Koulamas, SR. Antony, R. J. (1994). “A survey of simulated annealing applications to operations research problems.” *Omega, Int. J. Mgmt Sci*, 22(1), 41 – 56.

City of Toronto’s Traffic Safety Unit, Traffic Management Centre, Transportation (2018). “Traffic signal vehicle and pedestrian volumes, <https://www.toronto.ca/city-government/data-research-maps/open-data/open-data-catalogue/#7c8e7c62-7630-8b0f-43ed-a2dfe24aadc9> (03).

Dijkstra, E. W. (1959). “A note on two problems in connexion with graphs.” *Numerische Mathematik*, 1(1), 269–271.

Geospatial Competency Centre (2018). “Intersection file - city of toronto, <https://www.toronto.ca/city-government/data-research-maps/open-data/open-data-catalogue/#8d8cb501-4b98-3ce6-d584-94b74bca20c8> (03).

Laporte, G. (1992). “The vehicle routing problem: An overview of exact and approximate algorithms.” *European Journal of Operational Research*, 59(3), 345 – 358.

N. Christofides, A. Mingozzi, P. T. (1981). “Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations.” *Mathematical Programming*, 20(1).

Zeng, W. and Church, R. L. (2009). “Finding shortest paths on real road networks: the case for A*, <https://doi.org/10.1080/13658810801949850> (April).