Fair Vehicle Routing

Remi Duneau, Sebastian Stein

September 2019

1 Introduction

This project aims to reduce vehicle congestion by providing alternate routes for vehicles, with an emphasis on city networks. This paper will be split into two parts; the creation of a traffic simulator, followed by a discussion of different algorithms with results.

2 Simulator

A simple space-discrete, time-discrete traffic simulator was built. The reason an already existing simulator was not used for this project was due to complexity - most simulators require a steep learning curve which, due to time constraints, may have impeded progress in collecting results and findings.
The simulator was therefore created with simplicity as a core principle, and was designed specifically with this project in mind.

2.1 Simulator Components

2.1.1 Road network

Let $G = (V, A)$ be a graph representing a road network, where $V = \{1, ..., n\}$ is the set of vertices representing intersections, and A is the set of arcs, where every arc (i, j) represents a road $R_{(i,j)}$. Each road is associated with a length, a maximum speed, and a density.

2.1.2 Vehicles

Let $m = (v_{start}, v_{end})$ be a vehicle, where v_{start} represents the vehicle's start node and v_{end} the vehicle's end node. Each vehicle travels on roads from v_{start} to v_{end} via a path

$$P = \{R_{(v_{start},a)}, R_{(a,b)}, ..., R_{(y,z)}, R_{(z,v_{end})}\}$$

2.1.3 Simulating movement

Multiple traffic movement methods were considered to model vehicle motion, such as cellular automata (Nagel and Schreckenberg, 1992) and car following techniques (Gazis *et al.*, 1961). These techniques involved non-deterministic vehicle movement, such as random acceleration/deceleration and reaction times. For this simulator, a more simple approach was taken:
A vehicle holds a fixed speed, which is set every time it reaches a new road, and cannot accelerate nor decelerate. The speed is set using an equation that considers the road's current density k, which is calculated by dividing the current number of vehicles on the road by the road's maximum number of vehicles (this is worked out by dividing the road's length by the average vehicle length, which is

always assumed to be $6m$). Therefore each road's density is between 0 and 1 inclusive.
The speed equation is:

$$u = c \ln\left(\frac{k_j}{k}\right) \tag{1}$$

where u is the speed of the vehicle, c is the road's speed limit and k_j is the road's density during a traffic jam, which is 1 (and k, as previously mentioned, is the road's current density) (Greenberg, 1959). This equation can be derived based on car following principles (see Pipes, 1966).

2.1.4 Simulating time

The simulator uses a time-discrete approach, performing a number of ordered actions for every time step:

- Add a certain number vehicles to the given network, based on a predefined flow rate.
- Move every active vehicle a distance equal to its current speed.
- Remove any vehicles that have completed their trip from the network.
- Recalculate road densities.

2.1.5 Simulator drawbacks

The simulator's simplicity leads to some potential drawbacks. Firstly, since the speed of every vehicle m is not directly affected by the vehicle $m-1$ in front of it (m does not slow down at any point if it is going faster than $m-1$), phenomena such as 'ghost traffic jams' do not occur.

3 Discussion and results

This section will define different routing algorithms and discuss their performance in the simulator.

3.1 Routing algorithms

3.1.1 Dijkstra

The first algorithm is Dijkstra's algorithm (which will be called Dijkstra throughout). This well known algorithm finds the least cost path between two vertices in a graph, with cost in our case being time. Dijkstra's algorithm explores the start vertex v_{start}, adding all its neighbours to a list of unvisited vertices. The closest vertex from v_{start} is then explored, and its neighbours added to the unvisited list, until the end vertex v_{end} is reached.
This algorithm will be assumed to be optimal when comparing to other algorithms.

3.1.2 Least Density (LD)

LD uses the same principles as Dijkstra, however instead of finding the quickest path, it finds the least congested one (i.e. taking roads with the smallest density). Note that the densities are summed, which leads to consequences that will be highlighted with this example:
Assume a road network $G = (\{a, b, c\}, \{(a, c), (a, b), (b, c)\})$, $k_{R_{(a,c,)}} = 0.7$, $k_{R_{(a,b,)}} = 0.5$, $k_{R_{(b,c,)}} = 0.5$ and a vehicle $m = (a, c)$ is to be assigned a path. Taking path $P_1 = \{R_{(a,c)}\}$ will be favoured over path $P_2 = \{R_{(a,b)}, R_{(b,c)}\}$ since the cost of P_1 is 0.7 whereas the cost of P_2 is $0.5 + 0.5 = 1$.
The next algorithms attempt to mitigate this problem while still trying to reduce congestion.

3.1.3 Least Density with Road Length (LDRL)

This algorithm incorporates the road's length into the cost function. By multiplying the road's length by the road's density, this algorithm prioritises taking fewer, shorter roads.

3.1.4 Least Density Exponential (LDE)

This algorithm's cost function sums e^k. This means more dense roads are exponentially less favoured, so

3.1.5 Least Density Exponential with Dijkstra (LDED)

This algorithm's cost function multiplies least density exponential with dijkstra.

3.1.6 Future Dijkstra

Some future planning algorithms were looked at, where vehicles use knowledge of other vehicles' positions and paths on the network to determine their fastest path, however this required large amounts of processing power, especially for larger road networks.

3.2 Results

3.2.1 Experimental setup

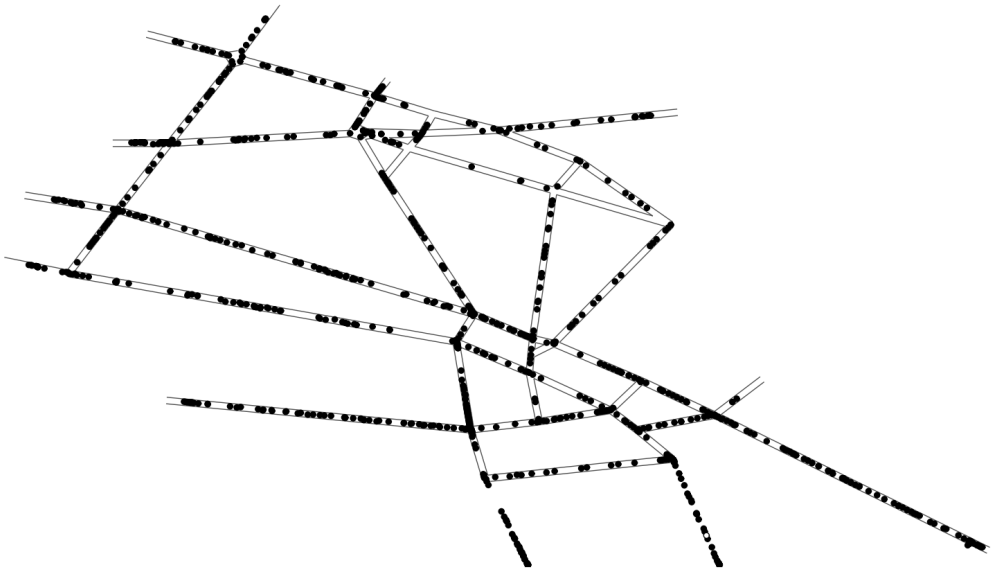

Figure 1: The road network used.

The experimental setup for the following results is as follows:

- Vehicles' start and end vertices are randomly assigned (this is seeded, so every vehicle's start and end vertex remains the same for each simulation).
- Every time step, 0.2% of the total vehicle population is added into the network.
- The network is 'initialised' for 50 time steps (which means 10% of the population).
- The next 20% of vehicles are tracked vehicles. The graphs below will be from information and statistics regarding only these vehicles. They are inserted into the network at the normal rate, therefore it takes approximately 100 time steps for all the tracked vehicles to be added (this can be more due to a tracked vehicle not being able to take its desired road because the road is full, so it waits until the road's density drops below 1).

- The simulation runs until all tracked vehicles have finished their trips.

The network used was a simplified model of part of Berlin, Germany (see Lochert *et al.*, 2003), shown in figure 1.

3.2.2 Static algorithms results

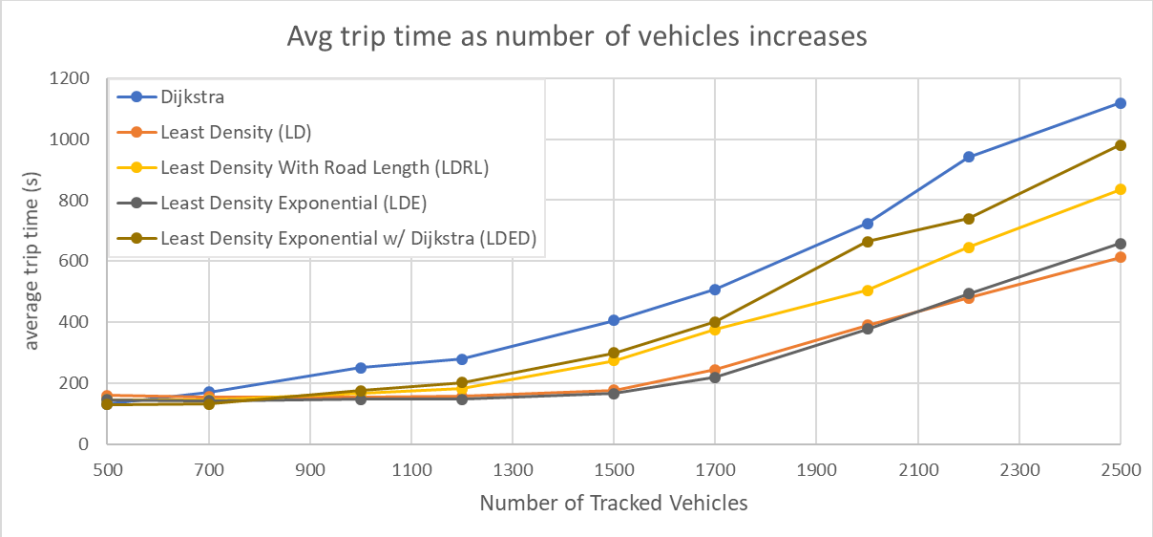

Figure 2: Average vehicle time as number of vehicles increases for different routing algorithms.

From figure 2, we can see Dijkstra performs well when there are few vehicles on the road, however as the number of vehicles increases, the average trip time increases greatly and it starts to perform worse than other algorithms. One explanation for this is that the network has a small number of well connected 'main' roads which allow vehicles to access different parts of the network quickly. When every vehicle is routed by Dijkstra at high traffic flow rates, it is trivial to see that these roads are among the first to become congested. However, during the lag time between vehicles deciding to take a main road and vehicles actually reaching it, more vehicles are choosing main roads, and since vehicles are being routed statically, they are unable to change their paths and will be stuck in traffic, and therefore have longer trip times at higher vehicle flow rates.

LD does not have the above problem at high vehicle counts, since vehicles are being routed away from congested roads. This reduces the number of vehicles in traffic jams, so the average trip time decreases, even though some vehicles are making significant detours from their optimal route. These detours are what causes the average trip time at low vehicle counts to be higher than other algorithms such as Dijkstra - when there are fewer total vehicles, there is less importance in trying to reduce congestion as most roads will not become congested.

LDRL seems to be in between Dijkstra and LD in terms of trip time at all vehicle counts. This is likely due to LDRL favouring fewer, shorter roads, meaning a more direct path than LD will be taken, so some of the behaviour observed when routing with Dijkstra appears.

LDE routes vehicles very similarly to LD, being slightly worse at higher vehicle numbers while performing marginally better otherwise.

LDED is the closest to Dijkstra in terms of trip time, having a slightly lower trip time due to the LDE part of the algorithm. LDED also performs well at low a vehicle count since the Dijkstra part of the algorithm becomes more dominant in calculating the cost of the path.

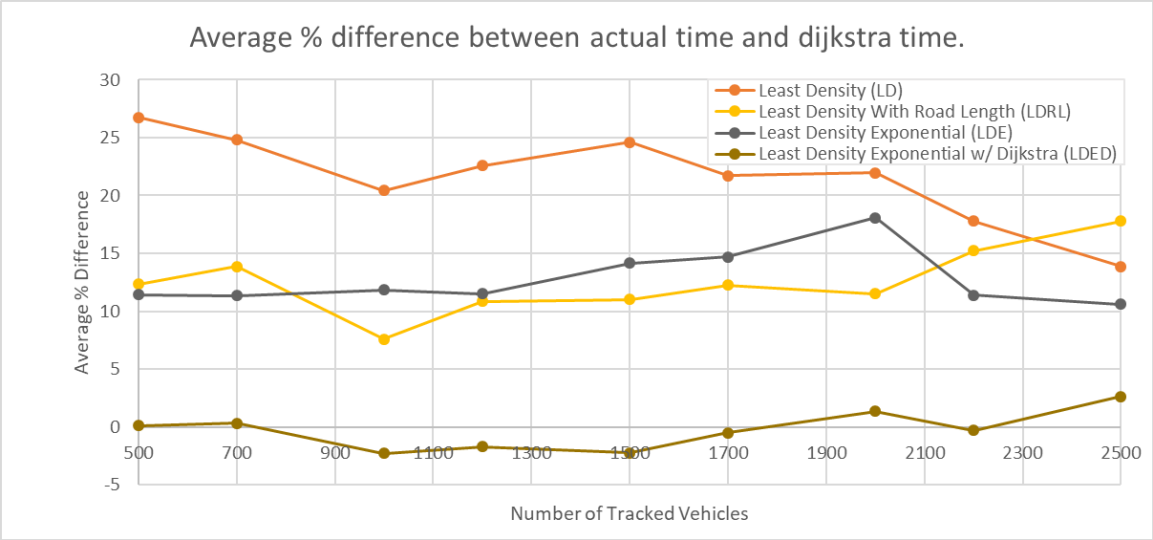

Figure 3: A graph highlighting the difference in trip time between a vehicle taking a Dijkstra path vs a path from another algorithm

From figure 2 we can observe LD and LDE are the best performing algorithms regarding average trip time. However there are some drawbacks to these algorithms. While the overall average trip time decreases drastically, there is little incentive for any vehicle to take a route given by LD or LDE, as each individual vehicle can travel faster by taking their optimal route (Dijkstra).
This was explored by simulating, for every tracked vehicle m, what the trip time of m would be if m were to take its optimal route instead of the route assigned to it, while all other vehicles keep their original routes.

Figure 3 shows, for the average vehicle, how much slower their assigned route was compared to if they had taken their optimal route. This is displayed as a percentage.
We can see from this that an LD vehicle, on average, will be between 14% - 27% slower than if it had taken a Dijkstra path.
LDED performs better than Dijkstra in some cases, perhaps due to the same reasons Dijkstra struggles at higher flow rates. This difference becomes even worse when we look at the worst 10% of vehicles for each algorithm, as shown in figure 4.

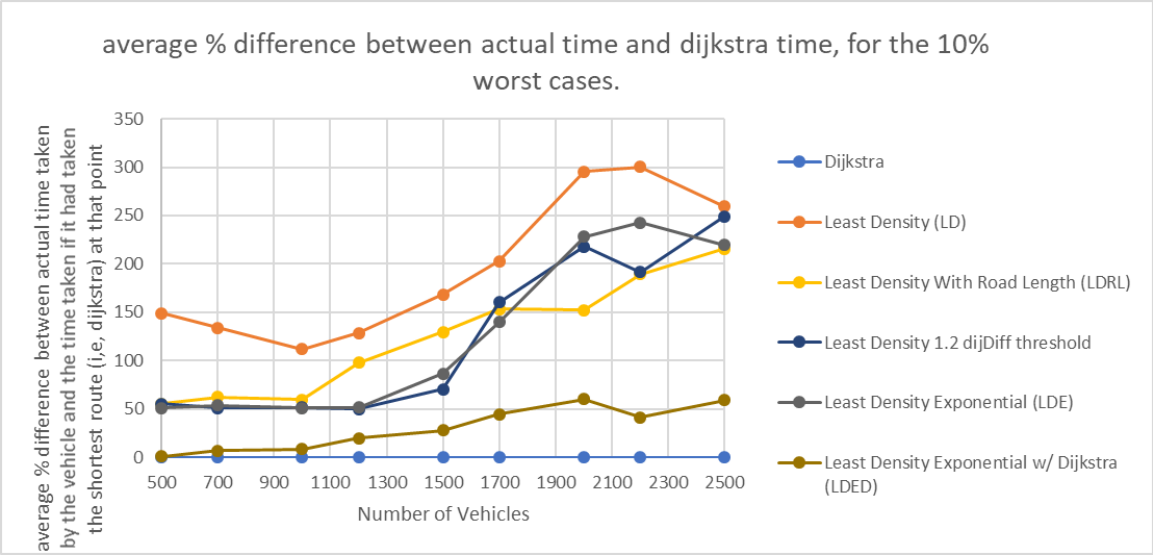

Figure 4: A graph highlighting the difference in trip time between a vehicle taking a Dijkstra path vs a path from another algorithm, for the worst 10% of vehicles

3.2.3 Reducing the Dijkstra difference

A few ways of trying to reduce this aforementioned percentage difference were investigated, one of which is the 'safety parameter'. This is a preset parameter between 0 and 1 which defines how the vehicle will be routed. It attempts to find an LD route using only roads with a density lower than the safety parameter. If this is not possible, the vehicle is routed by Dijkstra instead. This allows freedom to decide the proportion of vehicles being routed by Dijkstra, allowing for a compromise between minimising average trip time and minimising the percentage difference between taking Dijkstra's route over the given route. However, this only minimises the average, as for the worst cases of LD vehicles the percentage difference is still intolerably high.
The safety parameter was therefore abandoned, and results will not be shown to keep conciseness, however this idea led to another method of reducing the difference between actual and optimal times.

The Dijkstra difference threshold follows on from the saftey parameter idea. Instead of having a cutoff threshold to decide if a vehicle should be routed by LD or by Dijkstra, the Dijkstra difference threshold takes a less polar approach.
The threshold is a number above 1 which determines how far from the optimal Dijkstra a vehicle's estimated trip time can be. This is done by multiplying the Dijkstra trip time by the threshold, and then only allowing paths with an estimated trip time less than this result to be taken. This ensures that, unlike with the safety parameter, all vehicles have a lower Dijkstra difference instead of a split between some vehicles having no Dijkstra difference and others having a large difference.

3.2.4 Dynamic Routing

All previous results have been using static routing algorithms - once a vehicle is given a path, its route does not change. This section will briefly show our findings of routing vehicles with the same algorithms in a dynamic context, such that every time each vehicle reaches a vertex v_i, the vehicle is given a new route with start vertex v_i and end vertex v_{end}.

From figure 5 we can clearly see that the dynamic algorithms have lower average trip times than their static counterparts. This is likely due to vehicles being able to respond to changes in traffic by taking diversions if their original path becomes jammed or overly congested. However figure 6 shows some drawbacks to dynamic routing. At high traffic densities, all dynamic algorithms including

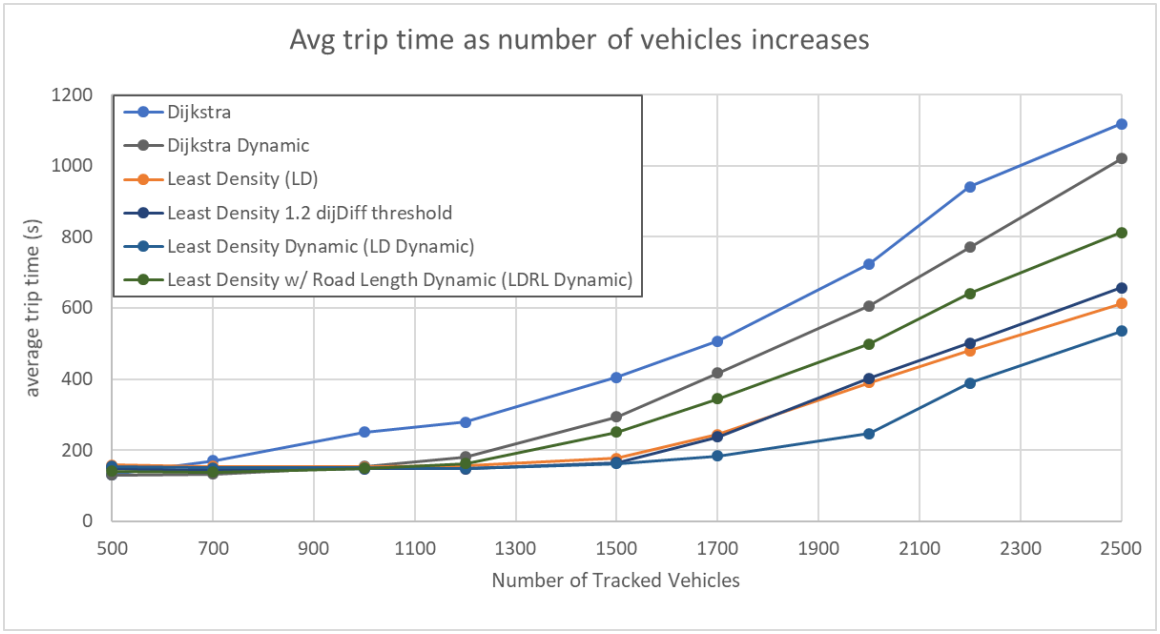

Figure 5: A graph showing average vehicle time as number of vehicles increases for different dynamic routing algorithms.

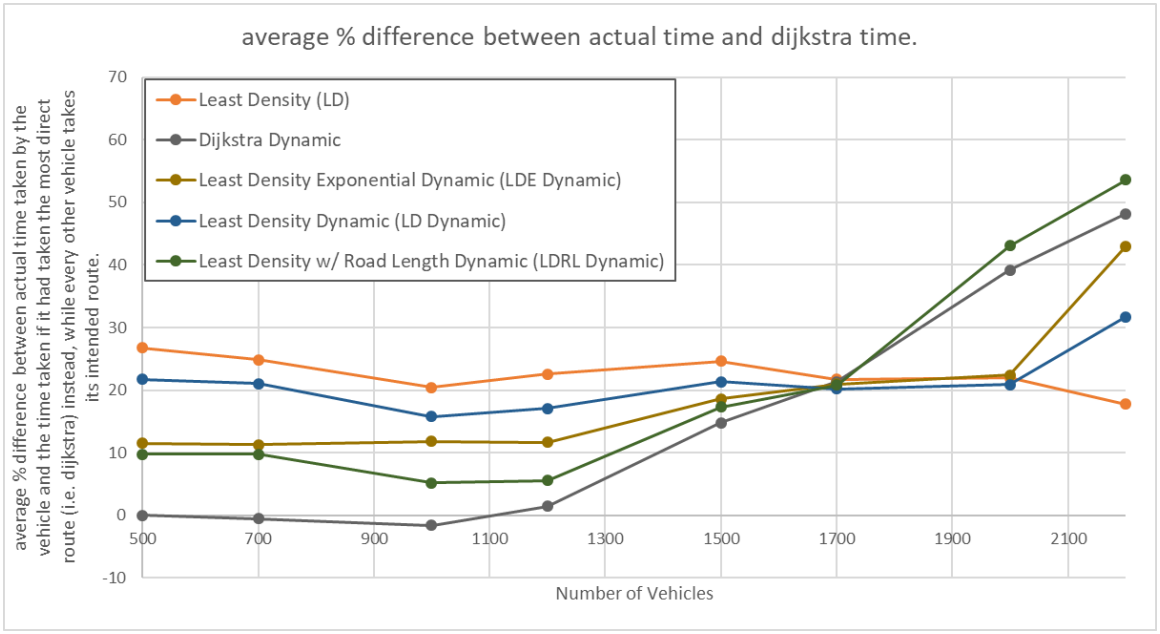

Figure 6: A graph highlighting the difference in trip time between a vehicle taking a Dijkstra path vs a path from another (dynamic) algorithm.

Dijkstra dynamic have an interestingly high Dijkstra difference. For LD-based algorithms this could be explained due to the fact that vehicles are able to dynamically adapt their route to reduce congestion, hence any vehicle taking an optimal route will be able to move at a faster speed due to lower congestion. For Dijkstra-based algorithms, the above reasoning also applies, however more investigation should be done on whether this reason is sufficient to explain why Dijkstra dynamic has a higher difference than many dynamic LD-based algorithms.

3.2.5 Long term routing

To explore how fairly vehicles are routed with regards to each other over the long term, the simulation was run 14 times with the same vehicle start and end points.
Keeping the aim of this project to route vehicles fairly in mind, each vehicle was given an 'unfairness' score. A greater ratio between the vehicle's estimated trip time and estimated Dijkstra trip time results in a greater unfairness score. Estimates are used over actual times as computing the actual Dijkstra time for all vehicles 14 times was deemed too costly in terms of processing time.
The unfairness score is used to determine how well a vehicle will be routed. This is done by calculating the ratio between the vehicle's unfairness score and the average unfairness score of all other vehicles. The ratio is then used to set a Dijkstra difference threshold τ on the routing algorithm, with a higher ratio leading to a lower threshold. Once again, the threshold is set based on the estimated Dijkstra time and not the actual Dijkstra time.

Different equations were tested to set the Dijkstra difference threshold based on the difference in unfairness. Two of these equations which will be shown are:

$$\tau = 1 + e^{-0.7(U)} \tag{2}$$

$$\tau = 1 + 0.5e^{-0.9(U)} \tag{3}$$

where U is the current vehicle's unfairness divided by the average unfairness over all vehicles.
Note (3) imposes stricter τ values than (2) (i.e. vehicles being routed over the long term using (3) will have a lower Dijkstra difference than using (2)). The results for (2) are shown in figures 7, 8 and 9, while the results for (3) are shown in figures 10, 11 and 12.

From these graphs we can see that the average trip time for each algorithm (apart from Dijkstra) is slightly higher than the single trip times in figure 2. This is due to *tau* increasing congestion by forcing vehicles to take paths closer to Dijkstra.
A look at the average unfairness shows as one would expect a lower unfairness overall when using (3) compared to (2). Looking at the worst 10% of cases, we can see that the unfairness is still fairly high for algorithms such as LD. There is a trade-off between lowering average unfairness and lowering average trip time, where reducing one increases the other.

The effects of the proportion of vehicles following an alternative routing algorithm compared to following the fastest route was also investigated, as shown in figures 13, 14, 15, and 16. The Dijkstra line indicate the ratio between the Dijkstra-only vehicles (which were the same for each algorithm) and the other vehicles being routed by Dijkstra.
The vehicle population was randomly split into two groups: one group was routed by Dijkstra only, while the other group was routed by whichever algorithm was specified. At low proportions of Dijkstra-only vehicles with a high number of total vehicles on the road, the average trip time of non-Dijkstra only vehicles was, in some cases, lower than the average trip time for Dijkstra only vehicles. This is surprising as one would expect a small proportion of vehicles being routed only by Dijkstra
An explanation for this is that Dijkstra-only vehicles are heavily affected by themselves, perhaps since they take main roads more than non-Dijkstra vehicles. However, looking at the most common

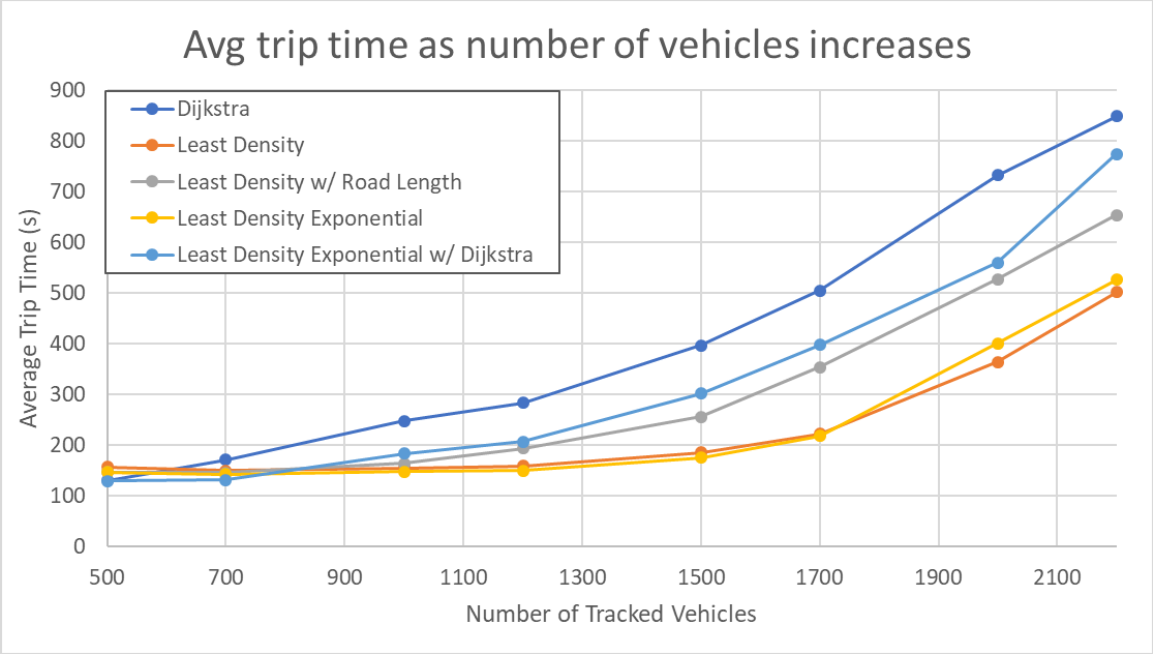

Figure 7: A graph highlighting the average trip times of different static routing algorithms over the long term (14 days), using (2) to determine τ.

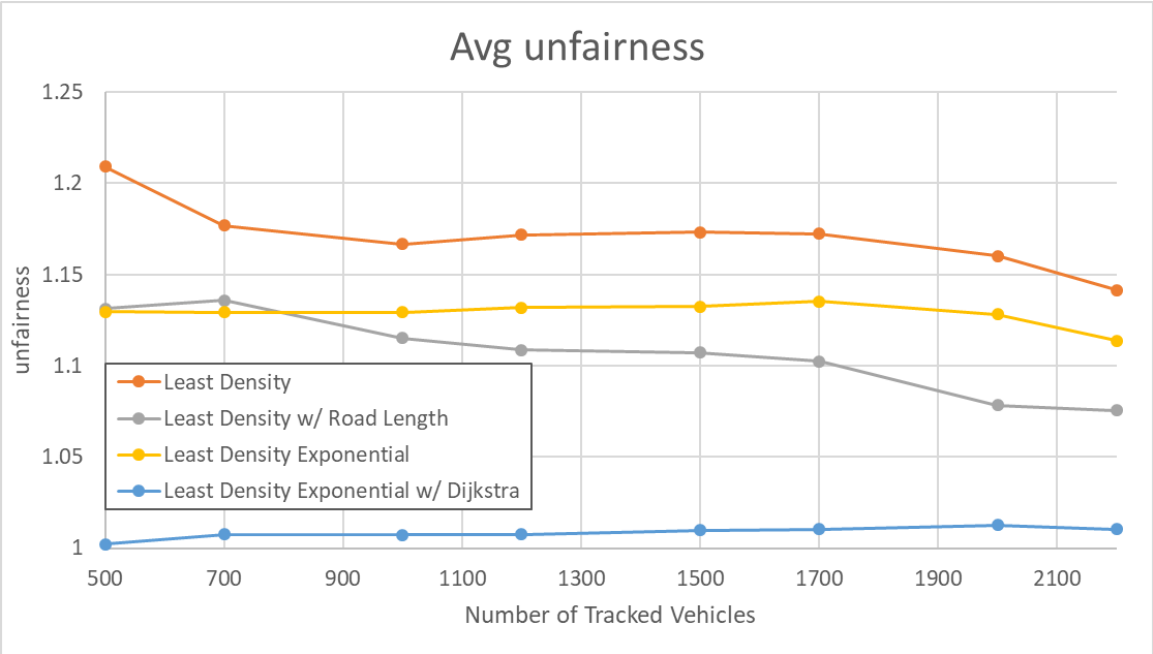

Figure 8: A graph highlighting the average unfairness of vehicles for different algorithms, using (2) to determine τ.

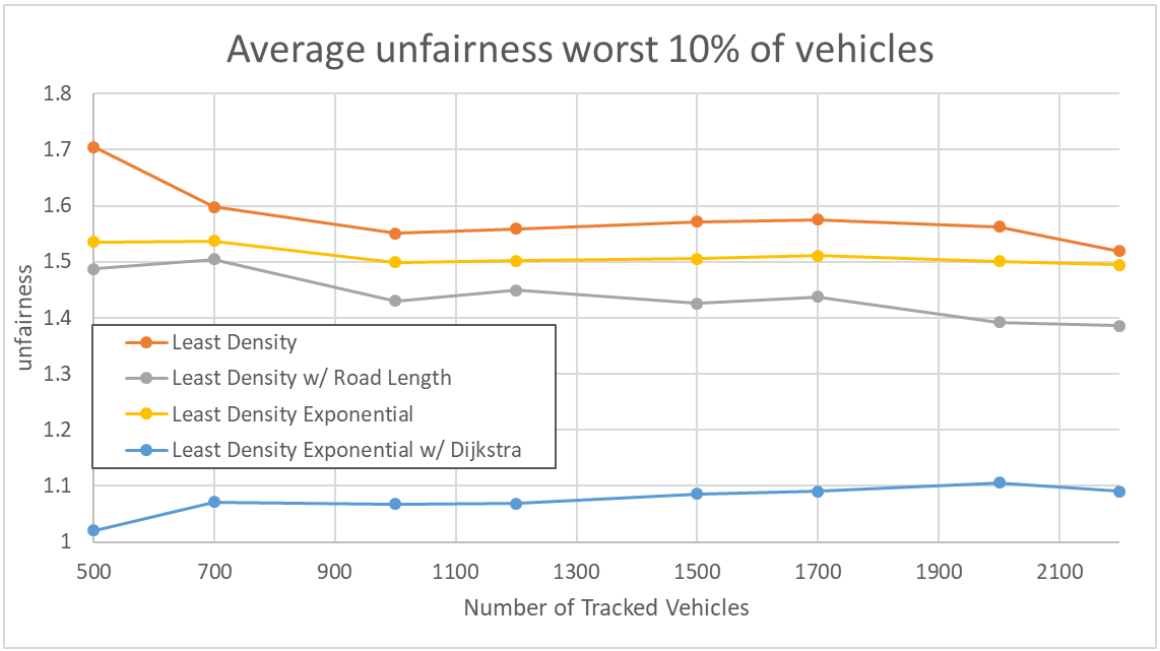

Figure 9: A graph highlighting the average unfairness for the worst 10% of vehicles for different algorithms, using (2) to determine τ.

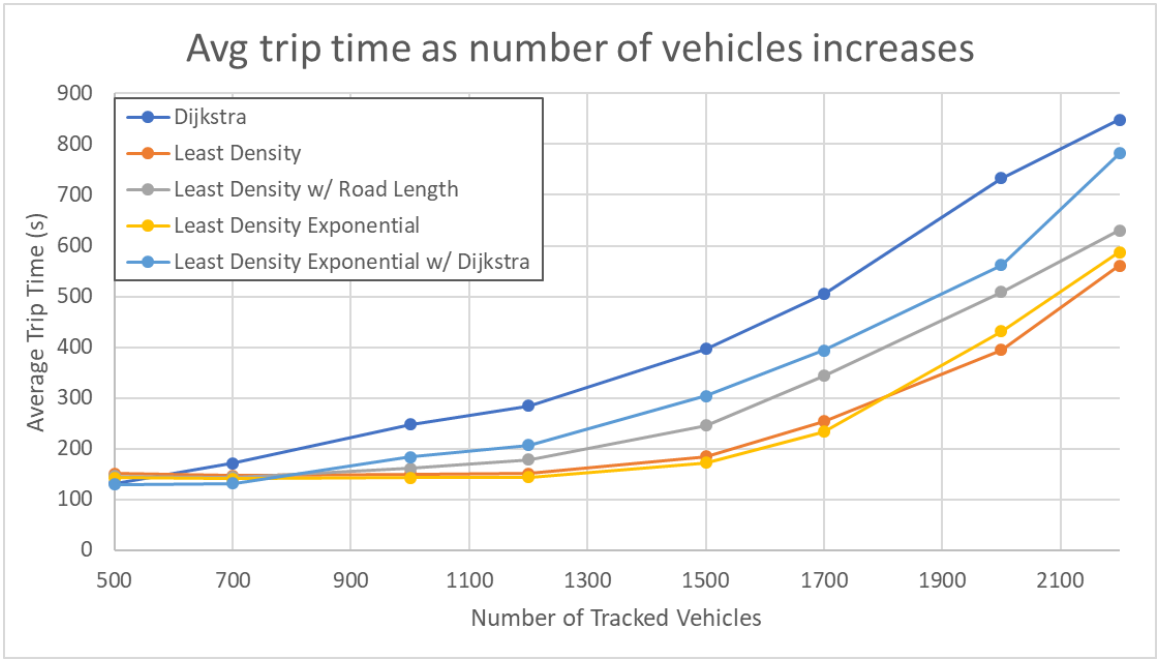

Figure 10: A graph highlighting the average trip times of different static routing algorithms over the long term (14 days), using (3) to determine τ.

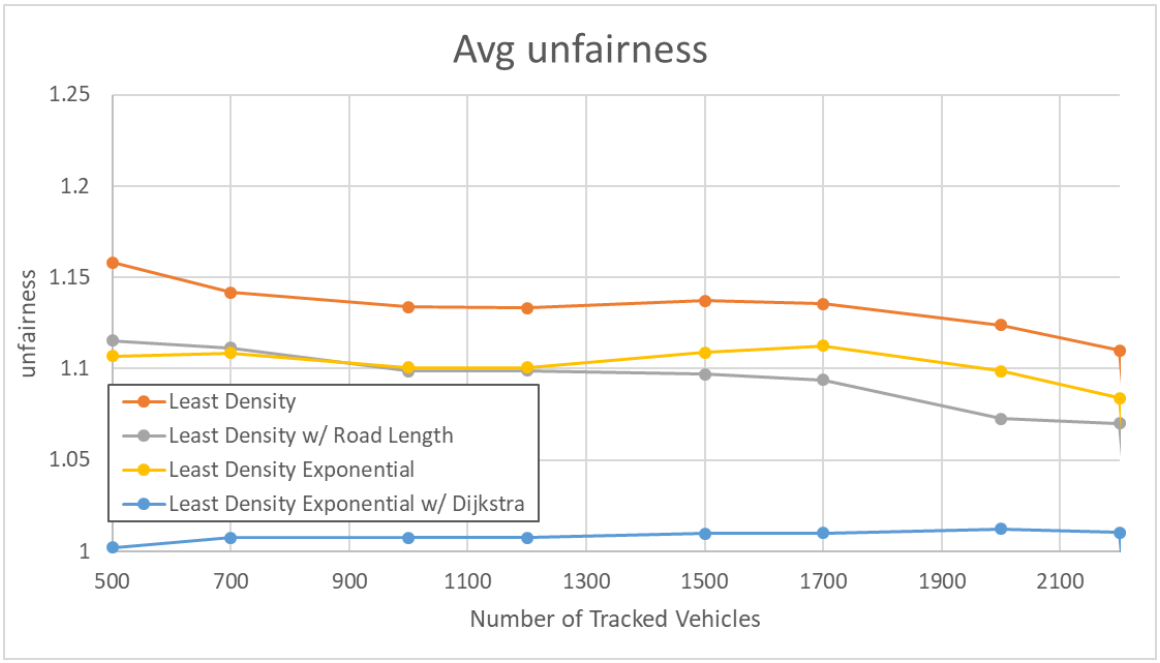

Figure 11: A graph highlighting the average unfairness of vehicles for different algorithms, using (3) to determine τ.

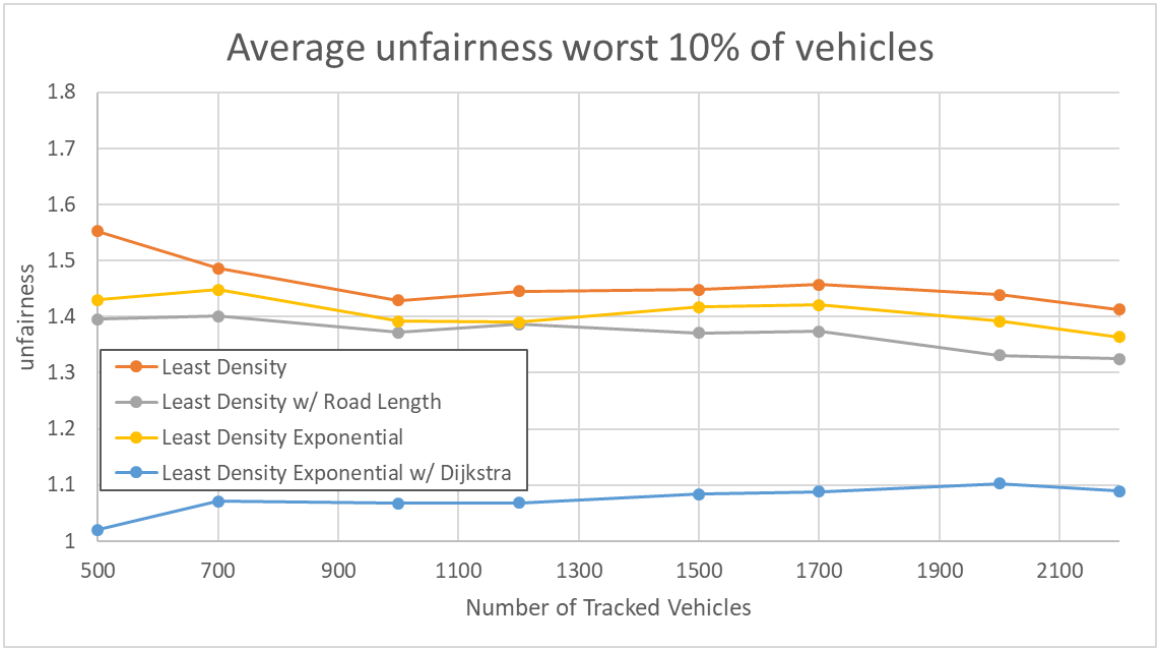

Figure 12: A graph highlighting the average unfairness for the worst 10% of vehicles for different algorithms, using (3) to determine τ.

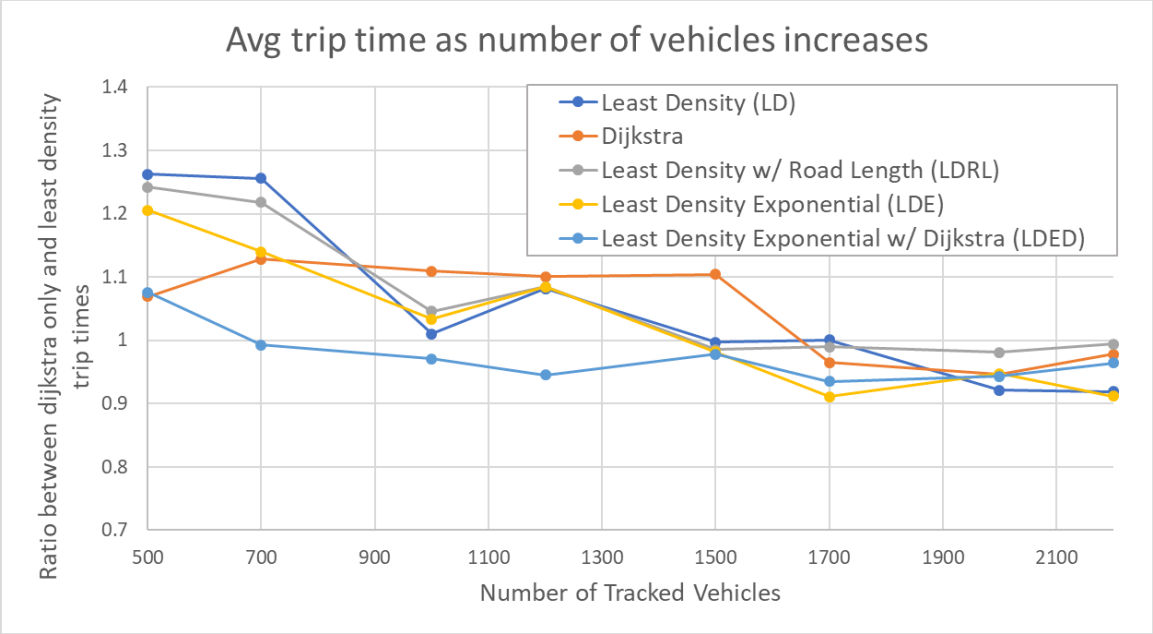

Figure 13: A graph showing the ratio between trip times of Dijkstra-only vehicles compared to other algorithms, with a probability of a given vehicle being Dijkstra-only of 10%.

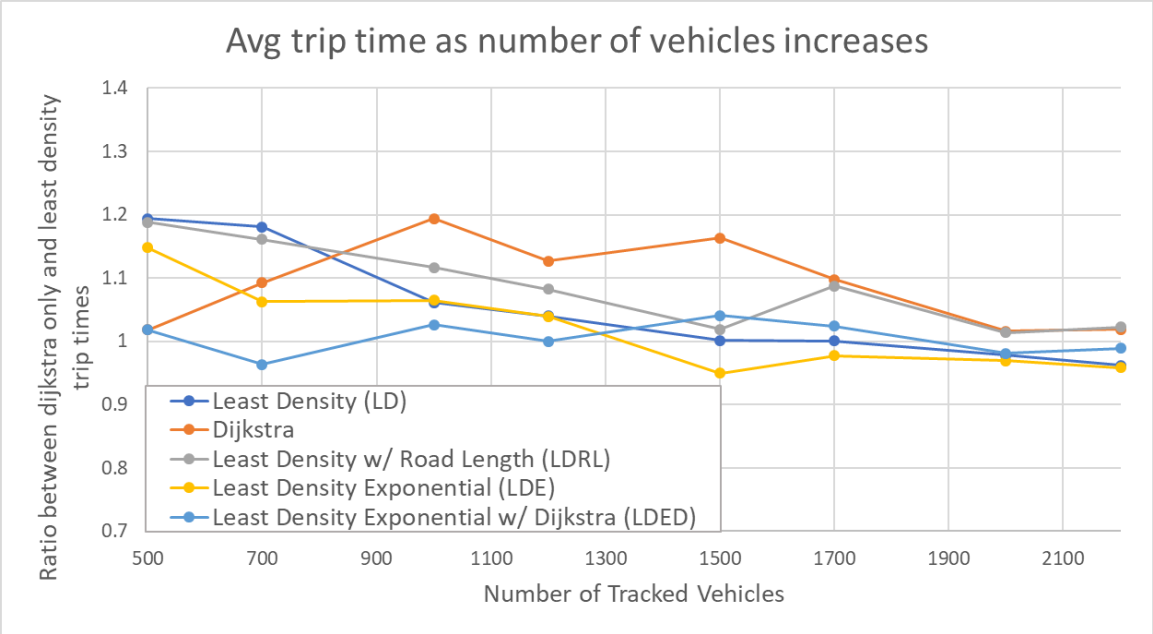

Figure 14: A graph showing the ratio between trip times of Dijkstra-only vehicles compared to other algorithms, with a probability of a given vehicle being Dijkstra-only of 10%.

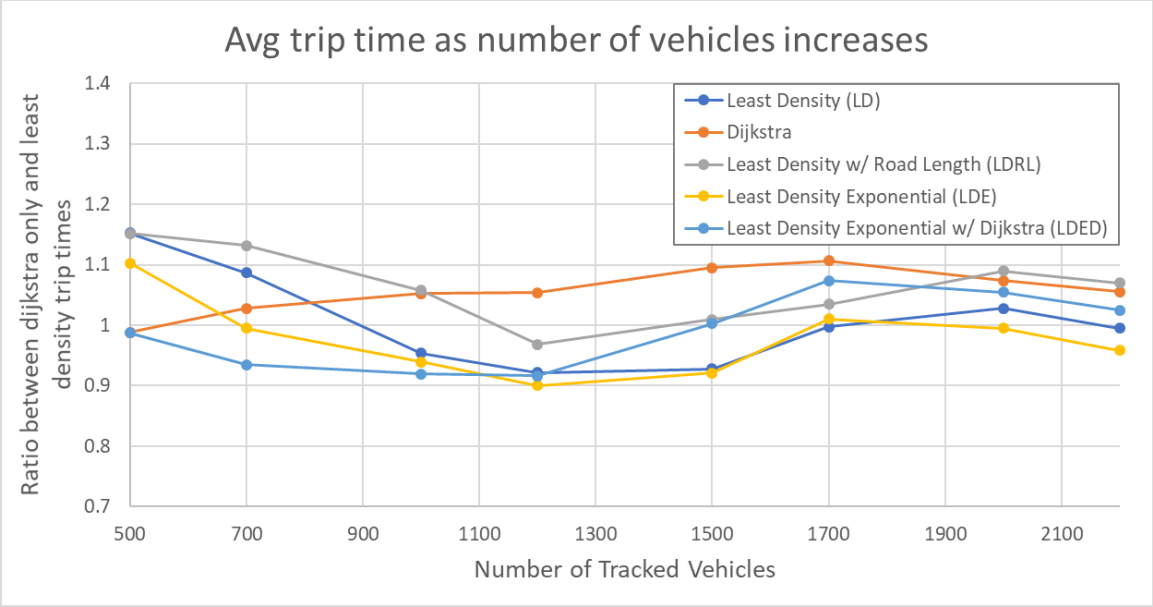

Figure 15: A graph showing the ratio between trip times of Dijkstra-only vehicles compared to other algorithms, with a probability of a given vehicle being Dijkstra-only of 10%.

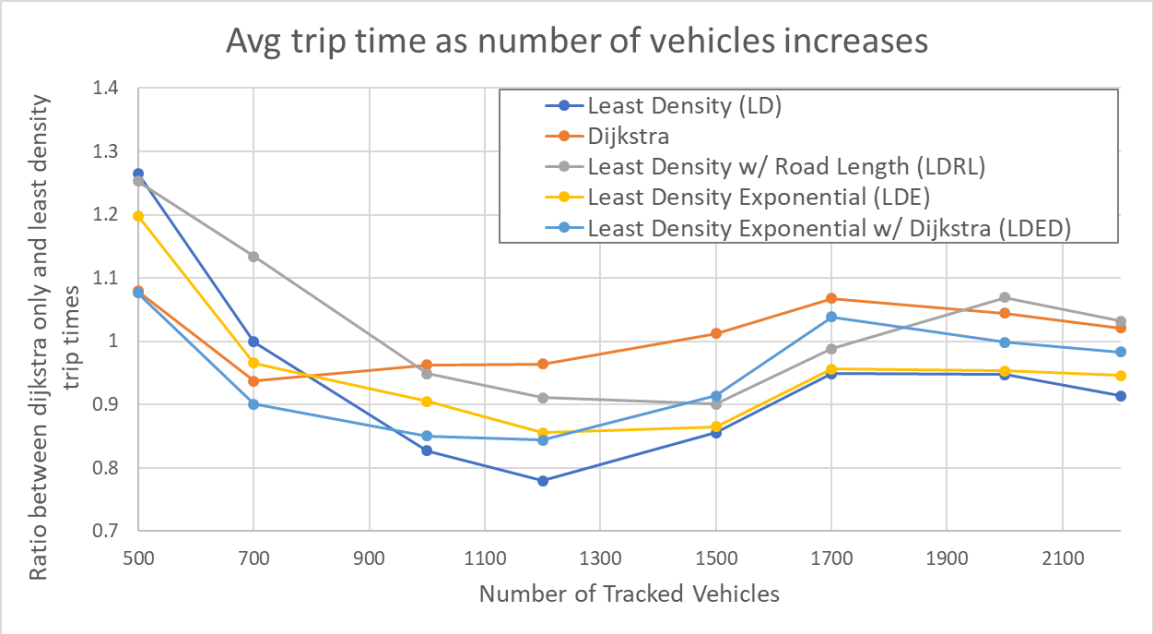

Figure 16: A graph showing the ratio between trip times of Dijkstra-only vehicles compared to other algorithms, with a probability of a given vehicle being Dijkstra-only of 10%.

roads taken by Dijkstra-only and non-Dijkstra vehicles shows no evidence to suggest that Dijkstra-only vehicles take more congested roads. It is unclear why Dijkstra-only vehicles behave this way, and whether this behaviour is intended by the simulator, and perhaps more work observing this should be done in future work.

4 Conclusion

In conclusion, it seems that alternative routing algorithms can benefit vehicles in the long term regarding trip times. In particular, LDED (Least Density Exponential with Dijsktra) decreases vehicles' average trip times considerably compared to Dijkstra (see figure 10), while only increasing the average vehicle's trip time by an estimated 1% (see figure 11).

5 References

1. Nagel, K. and Schreckenberg, M., 1992. A cellular automaton model for freeway traffic. Journal de physique I, 2(12), pp.2221-2229.
2. Gazis, D.C., Herman, R. and Rothery, R.W., 1961. Nonlinear follow-the-leader models of traffic flow. Operations research, 9(4), pp.545-567.
3. Greenberg, H., 1959. An analysis of traffic flow. Operations research, 7(1), pp.79-85.
4. Pipes, L.A., 1966. Car following models and the fundamental diagram of road traffic. Transportation Research/UK/.
5. Lochert, C., Hartenstein, H., Tian, J., Fussler, H., Hermann, D. and Mauve, M., 2003, June. A routing strategy for vehicular ad hoc networks in city environments. In IEEE IV2003 Intelligent Vehicles Symposium. Proceedings (Cat. No. 03TH8683) (pp. 156-161). IEEE.