Rideshare Matching Algorithms Case Study

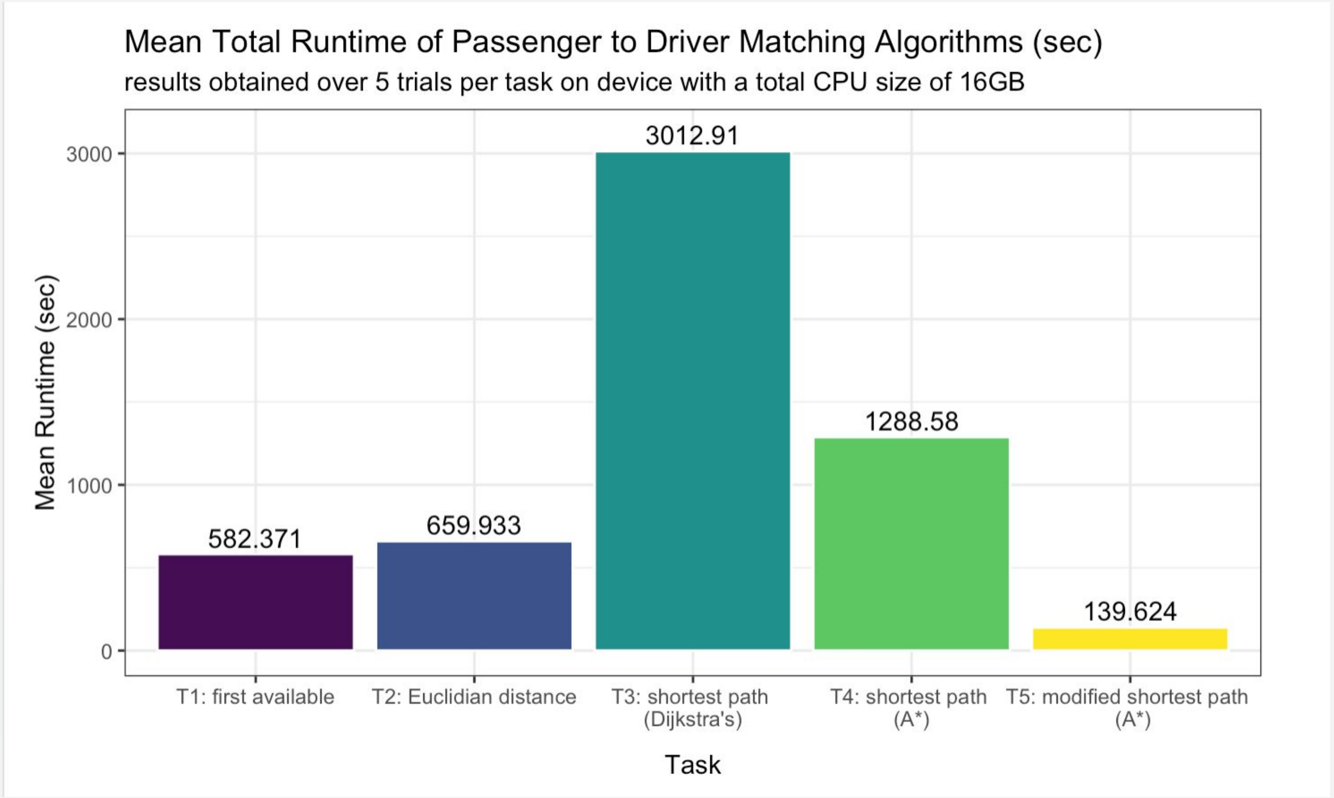
Executive Summary

Project Description We leverage NYC road network data and ride-share data to create and analyze algorithms that match drivers with passengers in order to get passengers to their destination as efficiently as possible while maximizing profits for drivers.

Findings In terms of the three fundamental desiderata, we found that our approach to Task 5 yielded the most balanced results:

> **D1 Passengers want to be dropped off as soon as possible.** In T5, passengers got to their destinations far faster because of a reduction in wait times resulting from our probability model for driver re-entry into the queue.
>
> **D2 Drivers want to maximize ride profit.** In T4, the mean ride profit was greatest, since we will always select the closest driver to the current passenger.
>
> **D3 Algorithms should be empirically efficient and scalable.** By far, T5 was the most efficient algorithm, running about 4x as fast as the next-fastest algorithm (T1).

Runtime Overview



General Modeling

For all tasks, we modeled the structure as follows:

Drivers and passengers: we use the Python standard library module `heapq` to order the passengers and drivers by their timestamp (for passengers, the ride request time; for drivers, the time they (re) enter the list of available drivers).

Driver burnout (T1-T4): a driver will rejoin the queue at a probability of 95% after each ride they complete, or they will reach a maximum shift length of 8 hours and log out. That is, the driver will either complete an 8-hour shift or, using a geometric distribution, complete an average of 1 / (1 - 0.95) = 20 rides.

Time to destination: for any path from one node to another, we calculate the length of the path in terms of the sum of the time to traverse each edge in the path given the *current* date/time of the passenger. That is, a ride requested at 2014-04-25 01:16:00 will use the graph calculated from the road speeds on a weekday (Friday) at 1:00 AM, unless the passenger has to wait until 2014-04-25 02:00:00 to be matched, in which case we use the graph reflecting speeds on a weekday at 2:00 AM.

Key:
P = passengers
D = drivers
E = edges in the map
V = vertices in the map

General Runtime Analysis

Graph Generation: O(E). We generate a total of 48 graphs that map the roads provided in the data. Each graph represents either a weekday or a weekend and an hour, such that we use the length of the edge and the speed at the graph's time/day to calculate an edge weight representing the time it takes to traverse the edge. Each graph takes O(E) to generate since we perform a constant time operation for each edge in the graph, so the total runtime of the graph generation is 48 * O(E) = O(E).

Brute force search for closest node to a query point (T1-T3): O(|V|). We obtain the latitude and longitude for a passenger or driver and then iterate through all vertices to find the one with the shortest Euclidean distance. Calculating the Euclidean distance is O(1) for each vertex, so the total runtime is O(|V|). To do this for all passenger and drivers and put them in a queue, this is O((P+D) |V|).

These general use functions do not dominate the runtime for any of the algorithms.

T1 Modeling & Algorithm

We use a basic first-come-first-serve approach of a queue of passengers and a queue of drivers. That is, in chronological order, ride requests are fulfilled by matching the first available driver with the current passenger.

A ride begins when *both* the driver and the passenger are available (the current date/time is after both individuals' date/time).

Using Dijkstra's algorithm, we then identify the shortest path between the driver and the passenger, and between the passenger and their destination. We update the following driver attributes according to the shortest path lengths and passenger destination: `node`, `date_time`, `profit`, `source_lat`, `source_lon`, and `num_rides`.

Finally, we re-insert the driver into the queue with probability 95% if they have not yet completed the maximum shift length of 8 hours.

Key:
P = passengers
D = drivers
E = edges in the map
V = vertices in the map

T1 Runtime

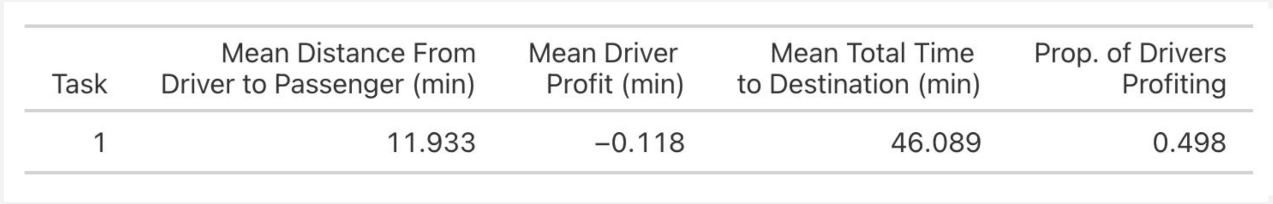
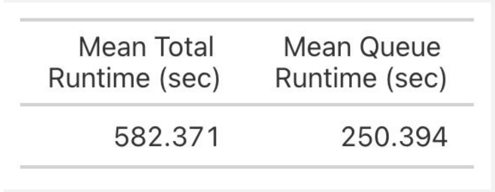
$$O((P + D)\,V) + O(E) + O(P\,(V + E)\log V)$$

$$= O(P\,(V + E)\log V)$$

Before running the matching algorithm, we create the passenger and driver queues in O((P + D) V) time and build the graph in O(E) time. Neither of these terms dominate the total runtime of T1.

Within the matching algorithm, we iterate through every passenger, identifying the first available driver in O(1) time. Then, for each passenger, we run Dijkstra's shortest path algorithm twice in O((V + E) log V) time to find the shortest path between the driver and the passenger, and then the passenger and their destination.

T1 Experimental Results

Task	Mean Distance From Driver to Passenger (min)	Mean Driver Profit (min)	Mean Total Time to Destination (min)	Prop. of Drivers Profiting	Mean Total Runtime (sec)	Mean Queue Runtime (sec)
1	11.933	−0.118	46.089	0.498	582.371	250.394

Our experimental results for T1 serve as a baseline for which we can do further analysis for additional algorithms.

- Desiderata 1: The mean distance between a matched driver and passenger is 11.933 min, with a total of 46.089 min from the time of ride request to time of arrival at destination.
- Desiderata 2: The average driver profit is -0.118, with 49.8% of drivers profiting – this is impractical and should be improved upon.
- Desiderata 3: The runtime is 582.371 seconds.

T2 Modeling & Algorithm

We interpret the statement "whenever there is a choice" to mean that for any given passenger, if there is more than one driver available at the time of their request, we will select the best one based on the given criteria. In the event that there are no drivers currently available, the passenger will be matched to the next available driver and the ride will begin when the driver becomes available.

We improve upon T1 by maintaining most of the structure of the baseline algorithm, but minimizing the Euclidean distance between a given passenger and the driver they are matched to. That is, for each passenger, we iterate through all *available* drivers, calculating the Euclidean distance and using it to select the closest driver.

Once we have identified the closest driver by Euclidean distance, we identify the shortest path between the driver and passenger then passenger source and destination by running Dijkstra's shortest paths algorithm. We update the following driver attributes according to the shortest path lengths and passenger destination: `node`, `date_time`, `profit`, `source_lat`, `source_lon`, and `num_rides`.

T2 Runtime

Key:
P = passengers
D = drivers
E = edges in the map
V = vertices in the map

$$O((P + D)\,V) + O(E) + O(P\,D + P\,(V + E)\log V)$$

$$= O(P\,D + P\,(V + E)\log V)$$
$$= O(P\,(V + E)\log V) \text{ on average}$$

Before running the matching algorithm, we create the passenger and driver queues in O((P + D) V) time and build the graph in O(E) time. Neither of these terms dominate the total runtime of T1.

Within the matching algorithm, we iterate through all passengers, calculating the Euclidean distance of each available driver in O(P D) time. Then for each passenger, we run Dijkstra's shortest path algorithm twice in O((V + E) log V) time to find the shortest path between the driver, the passenger, and their destination.

Despite the above being the worst-case runtime, Dijkstra's will dominate in the average case, since there will generally be a small number of available drivers to iterate through compared to the size of the graph, i.e. **$D < (V + E)\log V$**.

T2 Experimental Results

Task	Mean Distance From Driver to Passenger (min)	Mean Driver Profit (min)	Mean Total Time to Destination (min)	Prop. of Drivers Profiting
1	11.933	−0.118	46.089	0.498
2	9.886	1.931	41.684	0.590

Mean Total Runtime (sec)	Mean Queue Runtime (sec)
582.371	250.394
659.933	268.868

- Desiderata 1: Compared to T1, passengers do not have to wait as long before their driver arrives, such that on average, passengers arrive at their destination around 5 minutes earlier than in T1.
- Desiderata 2: Compared to T1, passengers are closer to their matched drivers by about 2 minutes, thus increasing the driver's ride profit by the same amount. The proportion of drivers who profit also increases from 49.8% to 59%.
- Desiderata 3: Compared to T1, the algorithm runs slightly slower (659.933 vs. 582.371 seconds), but the overall time complexity remains the same in the average case.

T3 Modeling & Algorithm

We interpret the statement "whenever there is a choice" to mean that for any given passenger, if there is more than one driver available at the time of their request, we will select the best one based on the given criteria. In the event that there are no drivers currently available, the passenger will be matched to the next available driver and the ride will begin when the driver becomes available.

We improve upon T2 by modifying the criteria to match passengers to drivers. We will still iterate through all available drivers for each passenger, but instead of calculating the Euclidean distance between each pair, we will use Dijkstra's algorithm to find the closest driver by identifying the shortest path in terms of time to traverse from driver to passenger.

Once we have identified the closest driver by Dijkstra's algorithm, the remainder of the algorithm is the same as T2: we identify the shortest path between the passenger source and destination by running Dijkstra's shortest paths algorithm. We update the following driver attributes according to the shortest path lengths and passenger destination: `node`, `date_time`, `profit`, `source_lat`, `source_lon`, and `num_rides`.

Key:
P = passengers
D = drivers
E = edges in the map
V = vertices in the map

T3 Runtime

$$O((P + D)\,V) + O(E) + O(P\,D\,(V + E)\log V)$$

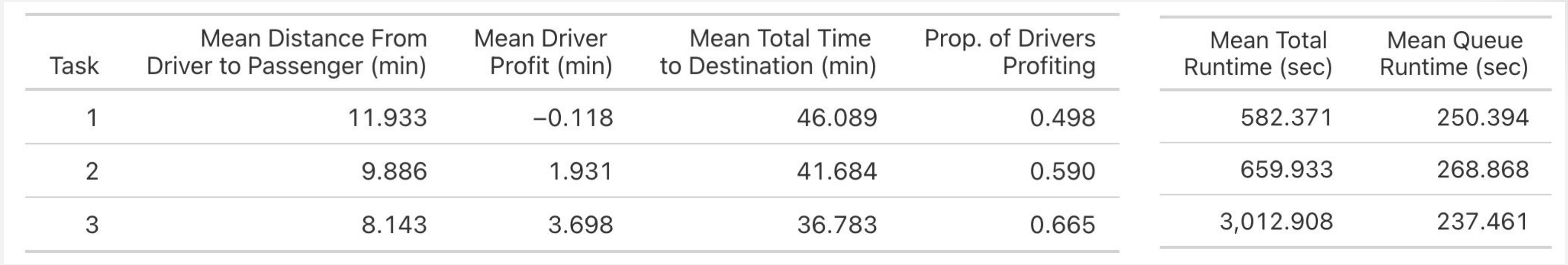
$$= O(P\,D\,(V + E)\log V)$$

We create the passenger and driver queues in O((P + D) V) time and build the graph in O(E) time. Neither of these terms dominate the total runtime of T1.

Within the matching algorithm, we iterate through all passengers, running Dijkstra's algorithm for each available driver in O(P D (V + E) log V) time. Then we store the minimum time from the previous matching step and we run Dijkstra's once to get the time from the passenger’s source to their destination, which in total takes O(P (V + E) log V).

The total runtime is dominated by the time it takes to run Dijkstra's on each potential driver for each passenger, which is the worst case.

T3 Experimental Results



Task	Mean Distance From Driver to Passenger (min)	Mean Driver Profit (min)	Mean Total Time to Destination (min)	Prop. of Drivers Profiting
1	11.933	−0.118	46.089	0.498
2	9.886	1.931	41.684	0.590
3	8.143	3.698	36.783	0.665

Mean Total Runtime (sec)	Mean Queue Runtime (sec)
582.371	250.394
659.933	268.868
3,012.908	237.461

- Desiderata 1:
  Compared to T1 and T2, passengers do not have to wait as long before their driver arrives, such that on average, passengers arrive at their destination around 5 minutes earlier than in T2 and 10 minutes earlier than in T1.
- Desiderata 2:
  Compared to T1 and T2, the distance between passengers and matched drivers is lower by about 1.7 minutes, thus increasing the driver's ride profit by the same amount. The proportion of drivers who profit also increases from 59% to 66.5%.
- Desiderata 3:
  Compared to T1 and T2, T3 is far less efficient and scalable with an empirical runtime about 5 times as long either of the previous algorithms.

T4 Modeling & Algorithms

We improve upon T3 in the following ways:

(i) Preprocessing nodes: To build the passenger and driver queues, we want to find the closest node in the graph to each individual. Before doing so, we build a *k*-dimensional tree (*k = 2*) of the nodes in the graph. This will optimize the process of identifying the closest node to a given query point by performing a nearest neighbor search on the *k*-d tree.

(ii) Shortest paths: To compute the shortest path between two points, we use A* to improve upon Dijkstra's. Our algorithm efficiently traverses the graph by calculating a time-based heuristic for each potential path using the ***Manhattan distance*** between the points and the travel speed of the current edge:

```
time = (abs(end.lat - start.lat) + abs(end.lon - start.lon)) / speed
```

T4 Runtime

Key:
P = passengers
D = drivers
E = edges in the map
V = vertices in the map

$$O((P + D) \log V) + O(E) + O(P\,D\,(V + E) \log V)$$

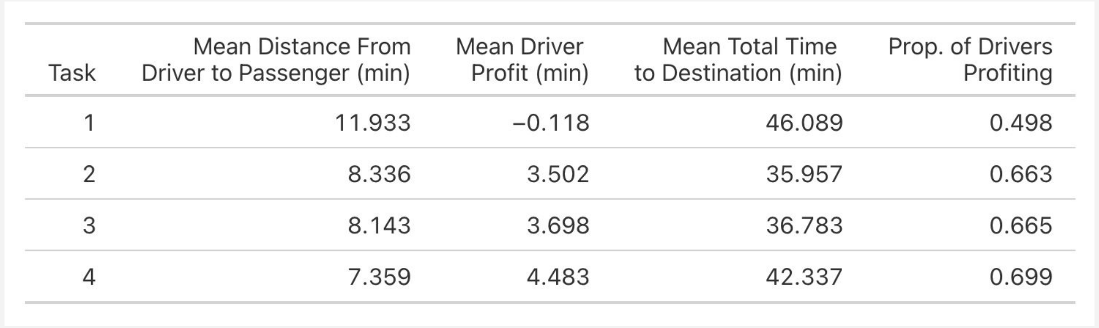
$$= O(P\,D\,(V + E) \log V)$$

Our improved algorithm for finding the closest node to a given query point lets us create the passenger and driver queues in O((P + D) log V) time and build the graph in O(E) time. Neither of these terms dominate the total runtime of T4.

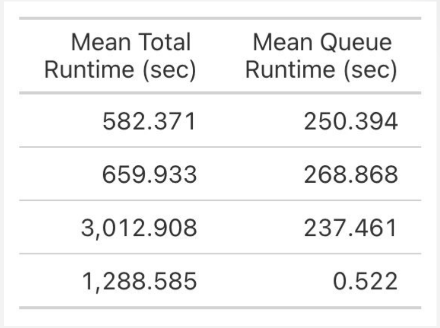
Within the matching algorithm, we iterate through all passengers, running A* for each available driver. In the worst case, our heuristic is no better than random, in which case A* has the same time complexity as Dijkstra's. However, our empirical runtimes indicate that our A* heuristic improves upon the runtime of Dijkstra's such that T4 runs in about 25% of the time that T3 runs in.

The total runtime is dominated by the time it takes to run A* on each potential driver for each passenger, which in the worst-case scenario is O(P D (V + E) log V), the runtime of Dijkstra's.

T4 Experimental Results



Task	Mean Distance From Driver to Passenger (min)	Mean Driver Profit (min)	Mean Total Time to Destination (min)	Prop. of Drivers Profiting
1	11.933	−0.118	46.089	0.498
2	8.336	3.502	35.957	0.663
3	8.143	3.698	36.783	0.665
4	7.359	4.483	42.337	0.699



Mean Total Runtime (sec)	Mean Queue Runtime (sec)
582.371	250.394
659.933	268.868
3,012.908	237.461
1,288.585	0.522

- Desiderata 1: Passengers have a shorter total time to their destination than in T1, but longer than in T2 and T3. This is likely due to natural fluctuations in the size of the driver queue between trials leading to longer wait times for passengers.
- Desiderata 2: Compared to all previous tasks, the distance between passengers and matched drivers is lower by about 1.7 minutes, thus increasing the driver's ride profit by the same amount. The proportion of drivers who profit also increases from 59% to 66.5%.
- Desiderata 3: Performed significantly better than T3 in terms of overall runtime. The *k*-d tree data structure allowed for assignment of nodes to passengers and drivers in O(log V) time and the A* algorithm on average performs significantly faster than Dijkstra by only choosing nodes that move toward the destination endpoint.

T5 Modeling & Algorithm

We improve upon the previous algorithms in the following ways:

1. For each passenger, we will identify the five closest available drivers based on Manhattan distance. From these, we will run A* to determine who is closest by travel time. This should significantly improve algorithmic efficiency from T4 since A* will be run a constant, rather than linear number of times for each passenger.
2. The probability-based sampling model that takes into account the number of rides a driver has given (*r*), how long the driver has been logged in (*t*), and their current total profit (*p*) to calculate the probability of them leaving the driver queue.

A probability for *X* = *r, t, p* was calculated as:

$$P(X) = 1 - e^{-0.693X/k}$$

Where *k* is a randomized value with an estimated mean equal to the expected average value for each of *r*, *t*, and *p* obtained from T4. If the calculated probability exceeds the threshold, then the driver will not rejoin the queue.

$$P(r) \times P(t) \times P(p) \geq threshold$$

Key:
P = passengers
D = drivers
E = edges in the map
V = vertices in the map

T5 Runtime

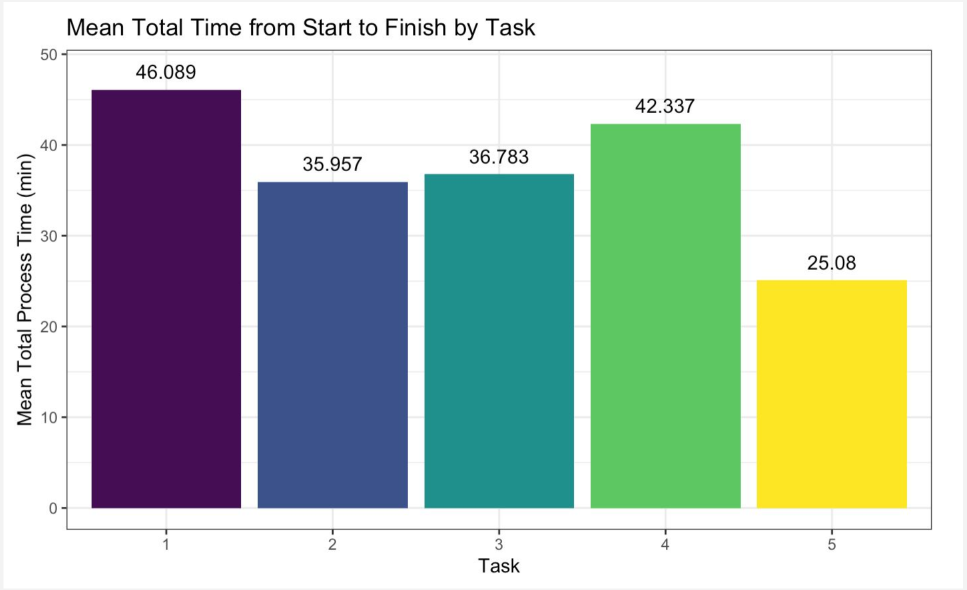
$$O((P + D) \log V) + O(E) + O(P (V + E) \log V)$$

$$= O(P (V + E) \log V)$$

Like T4, we create the passenger and driver queues in O((P + D) log V) time using a *k*-d tree approach and build the graph in O(E) time. Neither of these terms dominate the total runtime of T5.

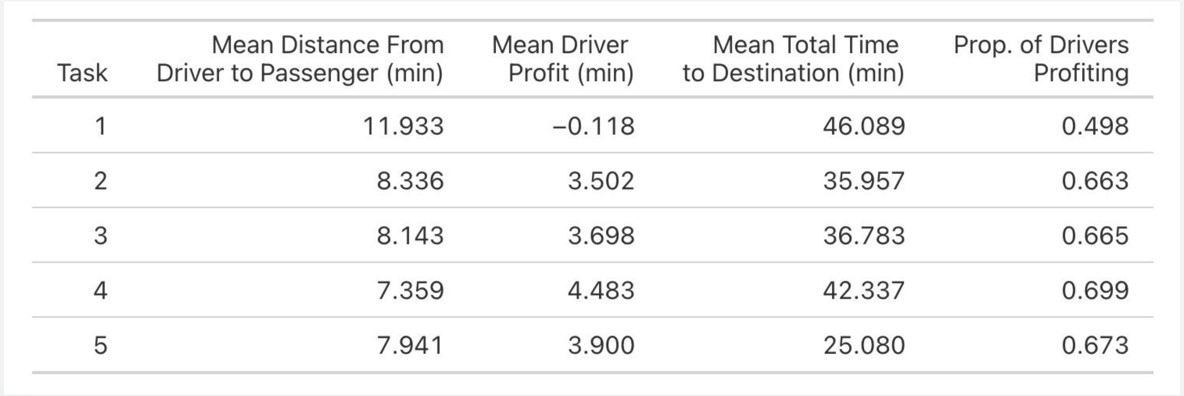
Within the matching algorithm, we iterate through all passengers, identifying the 5 closest *available* drivers to the current passenger. We run A* for each of the closest 5 drivers to select the closest driver to the passenger based on path travel time. Like T4, the worst-case runtime of A* is no better than Dijkstra's at O((V + E) log V), but our heuristic makes A* more efficient than Dijkstra's in the average case.

T5 Experimental Results

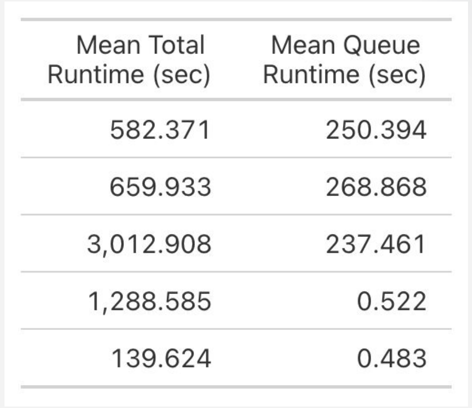


- Desiderata 1: Given the total time between ride request and arrival at destination for each passenger, passengers reach their destination in the shortest amount of time in T5. This is likely due to our changes to the model of driver re-entry into the queue, increasing the number of available drivers and thus reducing the amount of time a passenger must wait.

T5 Experimental Results



Task	Mean Distance From Driver to Passenger (min)	Mean Driver Profit (min)	Mean Total Time to Destination (min)	Prop. of Drivers Profiting
1	11.933	−0.118	46.089	0.498
2	8.336	3.502	35.957	0.663
3	8.143	3.698	36.783	0.665
4	7.359	4.483	42.337	0.699
5	7.941	3.900	25.080	0.673



Mean Total Runtime (sec)	Mean Queue Runtime (sec)
582.371	250.394
659.933	268.868
3,012.908	237.461
1,288.585	0.522
139.624	0.483

- Desiderata 1: The above confirms our results from the previous plot – since the mean distance between drivers and passengers is not significantly different from T4, the reduction in total time to reach the destination is a result of reduced wait times in T5.
- Desiderata 2: Profits of drivers are slightly lower than they were in T4 since our runtime optimization means we are not guaranteed to get the closest driver to a passenger.
- Desiderata 3: Total runtime is significantly improved over *all* other algorithms.

Thanks!