Autonomous Taxi Routing Algorithms

Kyle Christianson
chri3310@umn.edu

Peter Wall
wall0660@umn.edu

December 16, 2015

Abstract

Driverless vehicles will soon be on the road, bringing many improvements to our transportation systems. For example, the cost of taxi rides will fall with driverless taxi fleets, especially if they have intelligent routing and task assignment algorithms. In this project, we simulate a taxi routing system with different dispatching algorithms. Realistic passenger requests are generated using New York City taxi data [13]. Our results show that distance based auctions perform the best, however we believe there is still room for improvement.

1 Introduction

Driverless vehicles have always been a vision of the future, but now they are closer to reality than ever before. Google has been developing a driverless car since 2009 and has over one million miles driven on public roads [5]. Some projections show that driverless cars may be available to purchase as early as the year 2018. This technology opens the doors for a vast amount of innovation. For example, driverless taxi fleets may provide an efficient and cost effective mode of transportation, so much that you may never choose to own a driverless car [1]!

There are many appealing aspects to a driverless taxi fleet. First, the taxi company will save money by not needing to hire human drivers. This savings is especially pronounced on a slow day where a taxi might sit idle waiting for a request. An idling driverless taxi does not waste any resources by paying a driver to sit idle. The taxi system will also benefit from efficient routing algorithms which will reduce the total miles driven and thus the cost of operation. The cost benefits may lead to a consumer subscription model, similar to Netflix. Instead of purchasing a car or even individual rides, the passengers may pay a monthly subscription fee to ride a commuter taxi. Furthermore, the cost benefits will grow exponentially if the passengers agree to share rides with other passengers on similar routes.

In order for a driverless taxi fleet to be successful, it must have efficient and dependable dispatch and routing algorithms. We define the taxi dispatch algorithms as the algorithms which assign taxis to particular passenger trip requests. Routing algorithms include the turn-by-turn routing directions between two locations, which has been previously implemented [10]. There are many unique challenges involved in the dispatch algorithms. First, the environment is complex and taxi requests come in real time so the algorithm must constantly adjust. In AI terminology, the environment is multi-agent, dynamic, continuous, partially observable, and non-deterministic. Second, requests come in very frequently. In New York City, there are an average of 330 new taxi trip requests every minute. The algorithms must be efficient enough to handle this workload and

not fall behind. Third, different parties my wish to minimize different attributes of the system. For example, the taxi company wants to minimize operating cost which likely means drive the fewest miles. However the passengers want to minimize the time it takes to reach their destination. These solutions could be drastically different especially in a ride sharing situation. Fourth, there are many real-world obstacles to consider. Traffic, crashes, and vehicle breakdowns are difficult to predict.

In this paper, we propose and analyze several taxi dispatch algorithms. However, we barely scratch the surface of possibilities. Due to time and resource limitations, we are not able to address all of the challenges outlined above. Specifically, we assume passengers enter trip requests into the system as a pre-planned route. This follows a model where a passenger calls or uses a mobile app to request a taxi trip for a particular pickup location and dropoff location. While this doesn't capture the demand to "hail a cab", we feel our model is realistic enough to provide a valid comparison between the different algorithms.

In addition to the analysis, we implemented three of the proposed algorithms and tested them in a custom built simulation environment. Again we have limitations to our simulation which prevented us from modeling real-world obstacles and measuring the run-time performance of the algorithms. But the algorithms were tested in the same environment for equal comparison.

The rest of the paper is organized as follows. Section 2 describes the related work in the field of taxi routing and task assignment. Section 3 describes our simulation environment and the algorithms we implemented. Section 4 shows the results of our simulations and analyzes the performance of the algorithms. Section 5 contains our future work and conclusions.

2 Related Work

Finding efficient ways to route taxis is as old as taxis themselves. There are two categories of prior work that we considered. The first involved algorithms designed for taxi systems. The work by Jung, et al. shows a taxi system that includes request chaining and ridesharing [7]. Also, work by Nunes, et al.[11] and Lin, et al.[9] use genetic algorithms and simulated annealing respectively, but we found their simplifications to their environments to be too restrictive for our simulation.

The second involved agent assignment algorithms. We considered many of the algorithms introduced in-class. The Sequential Single-Item Auctions introduced by Koenig, et al. is one we implemented in our system [8].

3 Approach

We approached this project from two angles, analytical and empirical. Analytically we proposed multiple taxi routing algorithms and hypothesised their performance. Then we created a Java simulation environment to empirically test the algorithms.

3.1 Algorithms

Our simulation environment was made flexible to make changing the dispatch algorithm simple. This allowed us to quickly compare empirical results from the different algorithms. Each one used different approaches for handling the set of unfulfilled requests, R, and the set of vehicles, V. For convenience, we also define U in Equation 1 as a special subset of R: the set of unassigned requests.

$$U \subset R \wedge \forall r \in R, \neg assigned(r) \Rightarrow r \in U \tag{1}$$

3.1.1 Naive Assignment

The simplest algorithm puts incoming requests into a queue and then assigns them to any available vehicle. Without any intelligent methods used for picking the vehicles or requests, the assignments are effectively random. Assignments continue until all requests are assigned or all vehicles are busy. Equation 2 shows the method of assignment for this algorithm.

$$\forall r \in U, \exists v \in V, assignments(v) = \emptyset \Rightarrow assign(v, r) \tag{2}$$

3.1.2 Distance-based Assignment

Like the previous algorithm, the Distance-based Assignment algorithm assign requests by their order of availability. However, it assigns the request to the vehicle nearest to the pickup location. Assignments also continue until all requests are assigned or all vehicles are busy. Equation 3 shows the method of assignment for this algorithm.

$$\begin{aligned} &\forall r \in U, \exists v \in V, \forall v' \in V, \\ v \neq v' \wedge assignments(v) = \emptyset \wedge assignments(v') = \emptyset \\ \wedge dist(v, r) \leq dist(v', r) \Rightarrow assign(v, r) \end{aligned} \tag{3}$$

3.1.3 Distance-based Auction Assignment

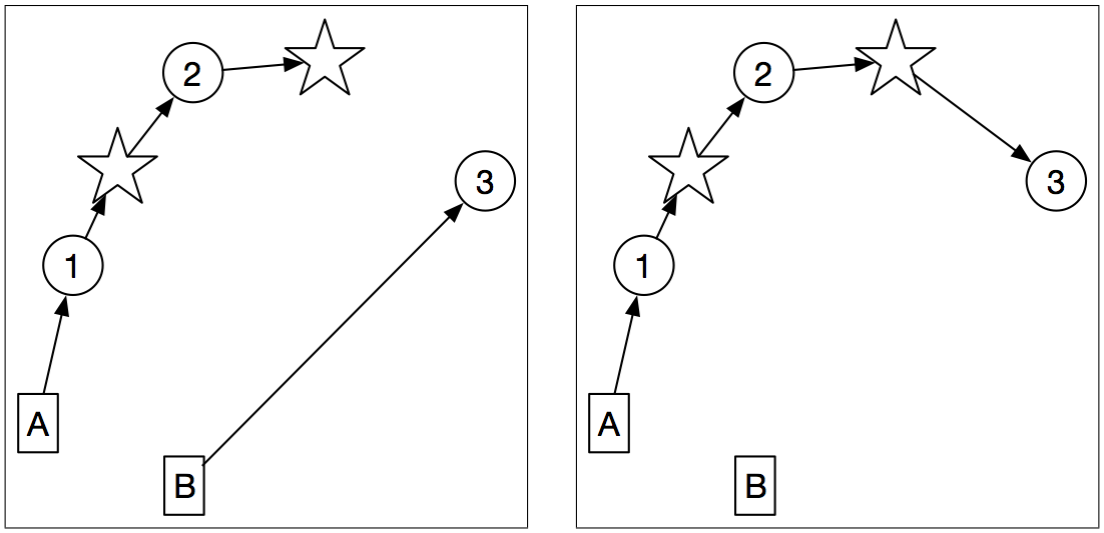
The next four algorithms use auctions to assign the request, which allow the vehicles themselves to participate as independent agents. The distance-based algorithm present all requests to each agent which respond with a bid for each. Agents which already have assignments respond with an "abstain" bid, removing them from consideration. The other agents bid their distance to the pickup location of the request. The auctioneer then assigns the request with the lowest bid to the bidder, then removes any other bids from that bidder or for that request. Assignments continue until no more bids remained. Equation 4 shows the bidding strategy for this algorithm.

$$\forall r \in U, bid(v, r) := \begin{cases} dist(v, r), & assignments(v) = \emptyset \\ abstain, & otherwise \end{cases} \tag{4}$$

3.1.4 Distance-based Auction Assignment with Reassignment

Each algorithm to this point disregarded any agents that already had an assignment. The algorithm with reassignment also includes agents that have been assigned a request, but has not yet picked up the passengers. This time agents that are already carrying passengers submit "abstain" bids, but otherwise they bid their distance to the pickup location of the request. When determining the assignments, a request could potentially be taken away from one agent and given to another. This adds work as many more requests are considered during every auction cycle. Equation 5 shows the bidding strategy for this algorithm.

$$\forall r \in R, bid(v, r) := \begin{cases} dist(v, r), & \neg pickedup(r) \wedge \neg haspassengers(v) \\ abstain, & otherwise \end{cases} \tag{5}$$



(a) Chaining algorithm (b) Sequential Single-Item Auctions

Figure 1: The difference between the Chaining and the Sequential Single-Item Auction algorithms

3.1.5 Distance-based Auction Assignment with Chaining

The Chaining modification allows agents to accept multiple requests, with the goal of finding synergies among requests. An agent without a request bids the distance to the pickup location as normal. An agent with one or more requests already assigned bids the distance of the total path of its current requests and the additional distance from the final drop-off location to the pickup location of the potential request. By bidding the total path, the winning bid is from a vehicle that can reach that request faster than any other vehicle, regardless of if the others has assignments or not. Equation 6 shows the bidding strategy for this algorithm.

$$\forall r \in U, bid(v,r) := \begin{cases} dist(v,r), & \neg haspassengers(v) \wedge assignments(v) = \emptyset \\ path(v) + dist(endpoint(v), r), & \neg haspassengers(v) \wedge |assignments(v)| > 0 \\ abstain, & otherwise \end{cases} \tag{6}$$

Figure 1a shows how one car can chain multiple requests with pickup and drop-off synergies. In the figure, vehicle A is assigned request 1 because it is the closest. Then it is assigned request 2 because the path distance from its current location to request 1's pickup location to its drop-off location to the pickup of request 2 is still shorter than the distance to vehicle B. Request 3 is assigned to vehicle B because it is closer than the full path travelled by vehicle A through the end of request 2.

3.1.6 Sequential Single-Item Auction Assignment

The Sequential Single-Item Auction algorithm is derived from work by Koenig, et al. to efficiently assign multiple requests to agents. This method ends up being very similar to our chaining algorithm, except the agent does not include the existing path distance in its bid. By bidding only the incremental addition to the path cost, the winning bid is from the vehicle that is closest to that

request, either now or in the future after it completes the assignments it already has[8]. Equation 7 shows the bidding strategy for this algorithm.

$$\forall r \in U, bid(v,r) := \begin{cases} dist(v,r), & \neg haspassengers(v) \wedge assignments(v) = \emptyset \\ dist(endpoint(v),r), & \neg haspassengers(v) \wedge |assignments(v)| > 0 \\ abstain & otherwise \end{cases} \quad (7)$$

In contrast to the Chaining algorithm, Figure 1b shows how using the incremental path distance will assign request 3 to vehicle A instead of vehicle B. This succeeds in minimizing the total path length for each vehicle. The big downside is that it increases the amount of time a request will be waiting for pickup, since vehicles will be assigned much longer chains of requests.

3.2 Simulation

To perform empirical experiments with our algorithms, we needed a simulation environment to run. As a whole, a taxi system is very large and complex to simulate. Our simulation needed to model a city map, vehicles located on the map, and passenger trip requests all following physical and time constraints. Furthermore, we needed to easily plug-in and swap different routing algorithms while maintaining the rest of the environment consistent.

We first researched existing open source simulation software only to find there are no good solutions that fit our needs. However we did find many open source projects that would assist us in implementing our own simulation environment.

3.2.1 Open Source Maps and Routing

Fundamental to our simulation environment is a street map of the target city. We used OpenStreetMap [4] which was well suited for this project. With the availability of the New York City taxi trip data, we targeted New York City.

Many vehicle routing algorithms (i.e. calculate directions from point A to point B) already exist which we can reuse in our simulator. We looked into Taxisim [3] as a possible routing framework which was developed specifically for research on the New York City data. However, we instead chose Open Source Routing Machine (OSRM) [10] because it allowed us to run routing queries without relying on online commercial systems (e.g. Google Maps). Such systems often induce restrictions on the number of free requests per day. Additionally, we were able to build and install OSRM locally which greatly improved performance over online solutions.

3.2.2 New York City Taxi Data

The requests we generate for our simulation are based on actual requests that took place in New York City. In 2014, Chris Whong submitted a FOIL request to the New York City Taxi & Limousine Commission, which had been gathering data on all of its taxis from 2010 until 2013 [12]. This data is now freely available to download online [6] and contains about 700,000,000 taxi trips in comma separated value (CSV) format. This data contains information including the pickup and drop-off times and GPS coordinates for every request during those four years.

We hit some challenges processing this large set of data. First, about 10% of the taxi trips are invalid. For example, several lines included bad CSV formatting, non-normalized units of measure, and impossible travel distances. Second, there are so many trip requests that we could not easily

consume them in CSV format. To help overcome these challenges, we loaded just one month (January 2013) into a local PostgreSQL database. We used a python script to sanitize the data skipping all invalid trips and loading the valid ones into the database. The python script was based off some code in the open source project gpsresilience [2].

With the database in place, we could easily and efficiently query taxi trips. In just the one month of data we imported, there were 13,255,835 valid taxi trips, 13,210 unique taxi medallions, and 31,766 unique human drivers. This provided us a realistic set of New York City taxi trip requests to feed our simulation.

3.2.3 Java Runtime

Finally, we implemented the core runtime framework of the simulation environment in Java. We developed this portion in Java mainly due to coding familiarity. We wrote specialized interfaces to both OSRM and the PostgreSQL database. The biggest challenge in developing the runtime was representing time. Time is important because it determines when requests are made available and when vehicles reach their destination. Due to our limited compute power and development effort, we made some sacrifices on modeling time. We kept the entire system in sync with a common clock, but essentially paused the clock at every time unit to process all of the taxi requests and routing directions. The consequences of this is that we will not be able to analyze the real run-time of our algorithms. However the metrics to evaluate the algorithms are maintained with this method.

A request service generates simulated taxi requests from the New York City taxi trip database. When the clock hits the time of the request, it will add it to the current request pool. To run our simulation in a reasonable amount of time, we limited our requests to only use 20% of the New York City taxi trips.

To easily swap in and out different dispatch algorithms, we implemented a strategy pattern with a common interface into the simulation environment. The dispatch algorithms are also written in Java so we could easily code them to the environment's interfaces. Each simulation run can change the dispatch algorithm strategy with a simple property file change. Each dispatch strategy is then responsible for processing the taxi requests and sending request assignments to the vehicles.

4 Results

4.1 Analytical Results

4.1.1 Naive Assignment

We never expected the naive algorithm to perform well. Since there were no considerations of the location of the vehicles when making assignments, many vehicles were assigned requests that were no where near their current location. This is reflected in the experimental results with a drastic increase in driving distance and request idle times.

4.1.2 Distance-based Assignment

Just adding the basic improvement of picking the nearest car to the request dramatically improved the efficiency. Every request would be fulfilled by a nearby agent, though not guaranteed to be the nearest. For example, the first request will always get the nearest vehicle, despite that there might be a nearer request to that vehicle later in the request queue.

4.1.3 Distance-based Auction Assignment

Changing the system to use auctions allowed the agents themselves, the taxis, to determine which requests to take. This allows for assignments to be made regardless of the order of requests. When there is only one request in the queue, this will result in the same assignment as with the Distance-based Assignment algorithm. However, when there are multiple unassigned requests, the auctioneer picks the lowest bid first, which will always be the closest vehicle to any unassigned request. Since this assignment is greedy, the assignments may be sub-optimal if picking the shortest distance for one car forces another car to pick a remote distance, preventing both cars from using two medium distance requests that may be cumulatively shorter.

The effectiveness of this algorithm depends on the frequency of requests that come in. If there are typically multiple unassigned requests when the auctions run, this may find better assignments. However, if requests come in slowly, there is no benefit to the additional complexity.

4.1.4 Distance-based Auction Assignment with Reassignment

With reassignment all requests, even those that have been assigned, but not picked up, are able to be re-auctioned. This greatly increases the computational load, since a request is only removed from future auctions when a taxi arrives at its pickup location. This means that a request could be auctioned hundreds or thousands of times. The trade-off for this cost is the possibility of finding an assignment that is better than the current assignment. However, limits should be introduced to prevent a distant request from being ignored permanently.

4.1.5 Distance-based Auction Assignment with Chaining

Chaining allows vehicles to pick up multiple requests and use the request synergies to their advantage. This allows for requests to be assigned quickly, potentially reducing their idle time. However, requests are chained solely on their pickup and drop-off locations, meaning that a distant request may be waiting longer than if a car was dispatched to it directly. This is limited because the bids for chained requests are based on the total path. Once a chain gets too long, it will be assigned to another car that can reach it directly.

This algorithm will reduce driving distance since using multiple assignments allows the vehicles to exploit request synergies.

4.1.6 Sequential Single-Item Auction Assignment

Finally, the Sequential Single-Item Auction algorithm by Koenig, et al. will minimize the driving distance. However, the cost to the request idle time will likely be too large for actual use in a real taxi system.

4.2 Experimental Results

To test our taxi dispatch algorithms, we implemented each algorithm in Java and plugged it into our custom simulation environment. Unfortunately due to time constraints we were only able to implement and test the first three algorithms. We hope to implement and test the remaining algorithms in future projects. However, the algorithms we tested performed as we expected.

4.2.1 Metrics

To measure the performance of each algorithm, we collected several metrics during the simulation. The most meaningful metrics include:

- Request idle time
- Vehicle drive distance
- Vehicle park time

Specifically, request idle time refers to the elapsed simulation time between when a new passenger trip request is available to the time a taxi arrives at the pickup-location. Intuitively this is how long a passenger waits for a taxi. These taxi customers will be happier if the request idle time is minimized. For our experiments, we total the idle time for all requests and compare average idle time per request. We realize that average may not be the best metric because outliers can easily skew the results. A better measurement may have been to use the median which is not as sensitive to outliers.

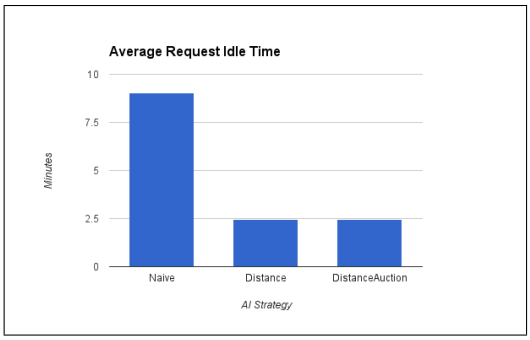
The vehicle drive distance metric is intuitively how far the taxi vehicles drove during the simulation. This metric is interesting because taxi operation cost is correlated to driving distance. For example, the number of miles you drive per gallon of gas, or the number of miles before vehicle maintenance is needed. Thus, the taxi company wants to minimize the driving distance. Commonly, driving distance and request idle time go up and down together, however there are situations in which this is not true. For example, in simple routing, one route may be longer but takes a shorter amount of time because it is on the highway. Another example in dispatch assignment we see with the sequential single-item auction strategy. Many trip request may line up nicely for one taxi with minimal distance traveled, but dividing the requests among a second taxi will fulfill the requests faster at the cost of higher driving distances.

The final metric is vehicle park time. We calculated the total time a taxi is parked without a request. This metric shows how well each strategy utilizes the available vehicles. It also can show if the taxi system has the appropriate number of taxis in its fleet (see Figure 3). We do not necessarily want to minimize or maximize this time. On one hand, parked taxis are good because they are not spending gas and miles driving around. On the other hand, parked taxis mean we have unused resources that my be better utilized. An interesting point arises with respect to the cost of parked taxis with or without a driver. With a human driver, a parked taxi is costing the taxi company money to pay the driver. However in a driverless taxi, a parked taxi is essentially free as long as it has a free place to park. This distinction can make a big difference in cost and how a company adjusts their dispatch strategies.

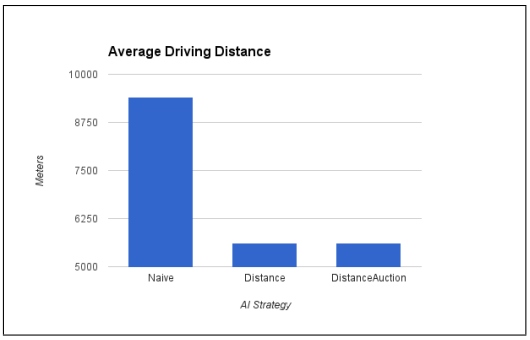
4.2.2 Simulation Results

First, we ran the simulation of each algorithm with 1000 taxis and 20% of one day's New York City taxi trips. Consuming only 20% of the New York City taxi trips over a day took about 20 minutes to run the simulation. These results can be found in Figure 2. Then we used the distance strategy algorithm to run the simulation for varying taxi fleet sizes. These results can be found in Figure 3.

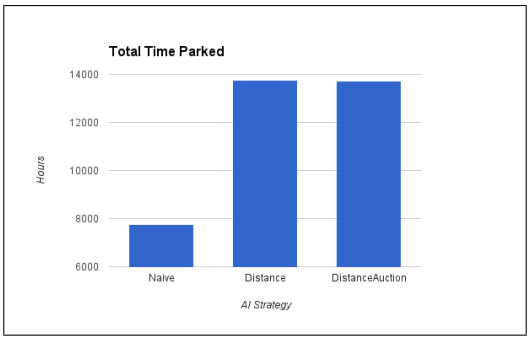
As you can see in Figure 2a and 2b, the distance-based strategy algorithms performed much better than the naive algorithm. We expected the centralized distance algorithm to perform similarly to the distance auction algorithm because the difference between these is mostly structural



(a) Average request idle time



(b) Average driving distance



(c) Total time parked

Figure 2: Comparing dispatch strategy algorithms

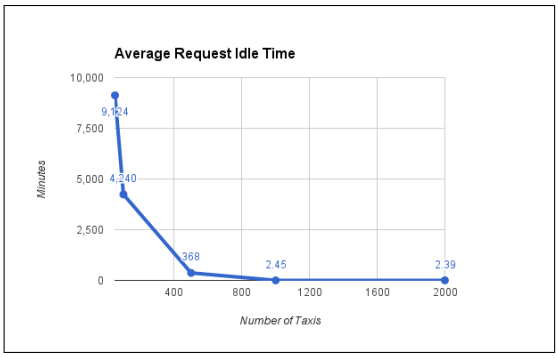
(centralized vs decentralized) and not functional. The park time in Figure 2c shows that the naive algorithm is using all of its taxis more often than the distance algorithms. This is bad however because of the higher driving distance. We can conclude that the naive algorithm is driving too far for its requests. The distance strategies have a much higher parked time because they are more efficient in assigning nearby requests.

Figure 3 shows the effects of taxi fleet size on the metrics. The trend lines confirm intuition that as more taxis are added, the request idle time drops, driving distance drops, and parked time increases. We can use these graphs to fine tune the taxi fleet size. The ideal number of taxis will depend on the demand of requests and may vary from day-to-day and even hour-to-hour.

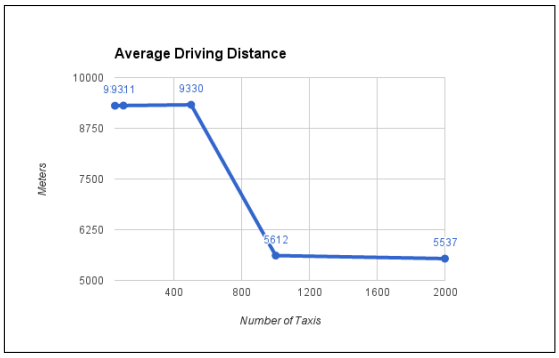
5 Conclusion

In this project, we researched driverless taxi dispatch algorithms. We propose several algorithms and analyze them theoretically and empirically through simulations. We believe the distance-based auction with reassignment is the best algorithm we propose, however there is more research that needs to be done to prove this. Empirically we only tested three algorithms and confirmed our hypothesis that distance based algorithms performed the best.

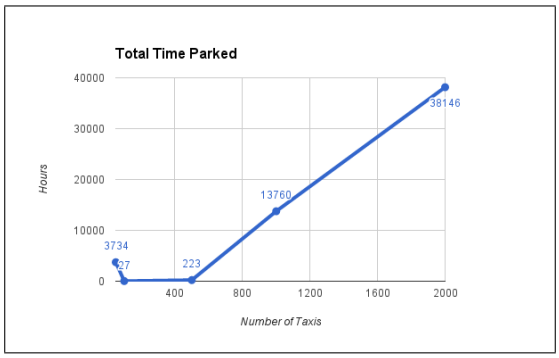
As stated throughout the paper, we have several limitations and assumptions in our simulation environment. First, we were limited on compute power so we could only run a small time period. Compute power also prevented us from having a real distributed system. We assumed an infinite amount of computation per time unit and thus we could not measure the actual run-time perfor-



(a) Average request idle time



(b) Average driving distance



(c) Total time parked

Figure 3: Effects of varying taxi fleet size

mance of the algorithms. Second, we ignored many real-world obstacles of taxi systems. We did not account for vehicle refueling, passenger load and unload time, traffic, breakdowns and crashes, and communication problems. Although these assumptions my affect the outcome of the results, we feel that all algorithms were compared with the same assumptions so it was still a fair comparison.

For future work, we would like to better analyze the algorithms as well as implement more of them to run in the simulation. We could theoretically find the optimal solution to the problem and see how far from optimal our algorithms fall. There are also other dispatch strategies in which we think would improve our algorithms. Improvements may include evenly distributing taxis while they are not busy, learning patterns of requests, and using multiple dispatch centers. Our simulation could use improvements to better model real-world obstacles and run in real-time to measure run-time performance. Ultimately, we would like to compare our driverless taxi system with real human drivers and measure the cost benefits.

References

[1] David Cardinal. *Autonomous taxis: Why you may never own a self-driving car*. Mar. 4, 2014. URL: `http://www.extremetech.com/extreme/176672-autonomous-taxis-why-you-may-never-own-a-self-driving-car` (visited on 10/18/2015).

[2] Brian Donovan. *Gpsresilience*. GitHub. July 1, 2015. URL: `https://github.com/Lab-Work/gpsresilience` (visited on 10/18/2015).

[3] Brian Donovan. *Taxisim*. GitHub. June 13, 2015. URL: https://github.com/Lab-Work/taxisim (visited on 10/18/2015).

[4] Geofabrik GmbH. *OpenStreetMap Data Extracts*. 2015. URL: http://download.geofabrik.de/ (visited on 10/18/2015).

[5] Google. *Google Self-Driving Car Project*. 2015. URL: https://www.google.com/selfdrivingcar/ (visited on 12/07/2015).

[6] University of Illinois. *NYCT&L Taxi Data*. 2015. URL: https://uofi.app.box.com/NYCtaxidata (visited on 10/18/2015).

[7] R. Jayakrishnan J. Jung and J. Park. "Design and Modeling of Real-time Shared-Taxi Dispatch Algorithms". In: *TRB 92nd Annual Meeting Compendium of Papers*. Transportation Research Board. 500 Fifth Street, NW, Washington, DC 20001 USA, 2013.

[8] S. Koenig et al. "The power of sequential single-item auctions for agent coordination". In: *In Proceedings of the National Conference on Artificial Intelligence*. 2006.

[9] Yeqian Lin et al. "Research on Optimization of Vehicle Routing Problem for Ride-sharing Taxi". In: *Procedia - Social and Behavioral Sciences* 43 (2012). 8th International Conference on Traffic and Transportation Studies (ICTTS 2012), pp. 494–502. ISSN: 1877-0428. DOI: http://dx.doi.org/10.1016/j.sbspro.2012.04.122. URL: http://www.sciencedirect.com/science/article/pii/S1877042812010038.

[10] Dennis Luxen and Christian Vetter. "Real-time routing with OpenStreetMap data". In: *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. GIS '11. Chicago, Illinois: ACM, 2011, pp. 513–516. ISBN: 978-1-4503-1031-4. DOI: 10.1145/2093973.2094062. URL: http://doi.acm.org/10.1145/2093973.2094062.

[11] Jorge Nunes, Luís Matos, and António Trigo. "Taxi Pick-Ups Route Optimization Using Genetic Algorithms". English. In: *Adaptive and Natural Computing Algorithms*. Ed. by Andrej Dobnikar, Uroš Lotrič, and Branko Šter. Vol. 6593. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 410–419. ISBN: 978-3-642-20281-0. DOI: 10.1007/978-3-642-20282-7_42. URL: http://dx.doi.org/10.1007/978-3-642-20282-7_42.

[12] Chris Whong. *FOILing NYC's Taxi Trip Data*. 2014. URL: http://chriswhong.com/open-data/foil_nyc_taxi/ (visited on 10/18/2015).

[13] Dan Work. *Open data*. 2015. URL: https://publish.illinois.edu/dbwork/open-data/ (visited on 10/18/2015).