Path Optimization for Destination-Oriented Ridesharing Drivers

Adiki Madhu Babu Shinya Hirata Vyas Srinivasan Tarun Bhatia

Amelia Glaese Abdul Lateef Haamid

I. Main Idea

Rideshare driving is a highly popular occupation all over the world. Uber, one of the leading rideshare service providers, has over 1.5 million Uber drivers worldwide, 45,000 in NYC alone [1]. Uber drivers in the US have an average monthly income of $360, suggesting that many of them drive part-time with a destination in mind [1].

Unfortunately, drivers don’t know a potential passenger’s destination before accepting a ride request. Thus, they are faced with a series of bad choices: 1. Use Uber’s rate-limited and overly-restrictive “destination-mode” to try to get trips along the shortest path to their target, 2. Blindly accept rides that may lead them far from their intended ending point, or 3. Entirely forego pickups along the way. We propose to create a data-driven tool to solve this challenge.

II. Problem Definition

Given a start and end point, we must calculate multiple paths that drivers can follow to maximize the probability of finding rides along the way. These paths should be chosen with some consideration given to distance travelled. Additionally, all computation and resultant paths should be wrapped in a web application that drivers can interface with.

III. Literature Review and Current Practice

Substantial research has been conducted on visualizing travel and start/destination taxi data and improving ride-sharing apps. For instance, [2] analyzed techniques to recommend personalized routes to individual drivers based on personal preferences such as time efficiency and safety. [3] proposed a new method for matching NYC riders to shared trips. Other research is dedicated to inferring hourly travel times between two destinations [4] and modeling data in a way that allows users to visually query taxi trips [5]. A lot of existing research is focused on methods and tools to increase hourly driver wages by identifying specific locations having a high probability of picking up passengers (i.e. “hot spots”). For example, [6] used spectral analysis of the New York Taxi data set to identify highly frequented locations depending on time and day. Meanwhile, others used spatio-temporal clustering to quantify the “hotness” of areas, which they used to make recommendations based on the driver's location [7], find types of trips that tend to stay in hot areas [8], and find areas of improvement in transportation system resilience [9]. Similarly, tools have been developed that suggest parking locations on road segment level for taxi drivers with high probability of picking up passengers and that reduce the costs of cruising without a customer [10]–[12]; using density peaks clustering and probability based statistical learning, respectively. In contrast, other tools recommend cruising routes for taxi drivers based on profitability of road segments and the attractiveness of the drop-off location with respect to the next pick-up [13], [14]. [15] additionally takes into account drivers’ personal preferences and past driving behaviors when recommending desirable routes.

However, the research presented above is limited by a lack of consideration of the driver’s preferred destination when suggesting places or cruising routes with high probability of picking up customers.

While the taxi-specific literature is silent on optimal s-t path planning, several algorithms have been created to solve this problem in more general contexts. [16] proposed first identifying the shortest path tree via Dijkstra’s algorithm, and then running a breadth-first search on a graph consisting of “side-track” edges. The MSA algorithm follows a similar process, but its “secondary graph” is a node-expansion of the initial graph [17]. The K* algorithm does not suffer from the expensive initial computation of the shortest path tree. Instead, it iteratively runs A* until a source-destination path is found or updated, then runs Dijkstra's on a "side-track" edge graph [18]. Note that K* can be made faster by running A* and Dijkstra's in parallel. As far as graph exploration is concerned, the Forest Fire algorithm randomizes the neighbors that a traditional breadth-first search algorithm includes in its queue [19]. The Random Walk traversal relaxes the requirement that only unseen nodes are added to the queue [20]. By leveraging process level parallelism, these randomized exploration techniques can be used to quickly find paths between any pair of points.

As for current practice, ride-sharing apps offer a "destination-mode" [21] that matches drivers with rides that are on the shortest path to their destination. Unfortunately, drivers can use this mode only two to six times a day. Additionally, many drivers complain that the mode is overly restrictive and filters out potential rides that are not precisely on the shortest path. Contrastingly, Taxi drivers rely solely on intuition and experience to plan routes to a given destination [22].

IV. Proposed Method

The following sections describe how we build transition graphs, calculate routes between locations, and display paths.

A. *Transition Graph Generation*

We created transition graphs with nodes that represent Manhattan intersections and edges that represent trips between these intersections (edge weights are trip frequencies – these can be converted to probabilities on-demand).

1) Node Generation

We queried every Manhattan street-avenue pair against Google's geocoding API to determine the GPS coordinates of every intersection. Mapping these as nodes resulted in a granular transition graph.

2) Transition Edge Generation

We also created a transition graph with the same nodes as above, but edges corresponding to transition probabilities. To do this, we downloaded the New York City taxi dataset and examined the relevant fields for each trip: *Datetime*, *Pickup_Longitude*, *Pickup_Latitude*, *Dropoff_Longitude*, and *Dropoff_Latitude*.

Based on these trips, we generated transition graphs as follows:

1. Find pickup and drop-off nodes closest to the pickup and drop-off locations respectively.
2. Filter out trip if pickup/dropoff locations were far from assigned nearest node.
3. Create edge between assigned pickup and dropoff nodes.
4. Count total trips mapped to each edge and assign that number as the edge's weight.

We created multiple transition graphs, where each one corresponded to all the trips in a given month at an hour of day. Since we used taxi data from January 2014 to June 2016, we created 30*24 = 720 transition graphs. To map each taxi trip to its corresponding transition graph and calculates the edge weights, we ran a MapReduce cluster on Amazon Web Services. The resulting transition graphs were stored in a SQL database.

3) Transition Graph Aggregation

In this step, we took the 720 transition graphs from above and grouped them by hour of day. For each group, we generated a new aggregated transition graph by taking the weighted sum (that weighs recent months more heavily) of each edge across the individual graphs. This resulted in 24 aggregated transition graphs, where each one corresponds to a certain hour of the day.

B. *Path Computation*

In order to find the paths with highest likelihood of pickups, we created a variation of the K* algorithm [18]. The K* pipeline consists of three steps: A*, Path Graph Creation, and Dijkstra's.

1) Initial Graph Exploration (A)*

The edges were transformed by taking the negative log of every edge's probability. Additionally, we used a distance-to-target heuristic that maps the distance to the target vertex into a range compatible with our negative log probability space. A* stops once the target vertex has been explored.

2) Sidetrack Edge Uncovering

During the execution of A*, all "sidetrack" edges, which go from a vertex being explored to a previously explored vertex, are identified and stored in a heap assigned to each vertex. They are ordered based on the length of the detour introduced to the path.

3) Path Graph Creation

From these sidetrack edges, we create a new graph for finding next-best paths. The edges are based on the relationship between each sidetrack edge and A*'s shortest path.

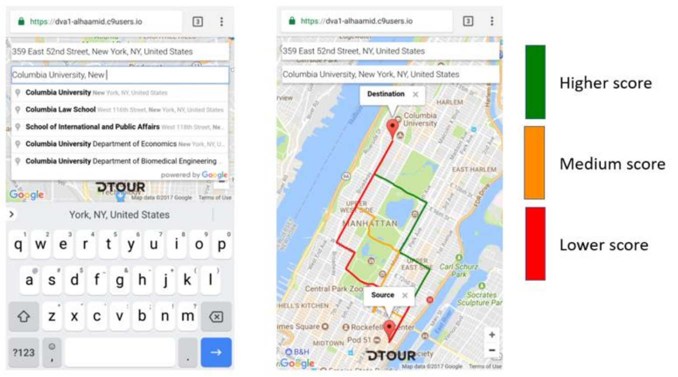
4) Path Finding

We run Dijkstra's algorithm on the Path Graph. The shortest tree path to each vertex in this Path Graph corresponds to an actual detour path in the map (the original graph). Additionally, using a heap results in Dijkstra's returning paths in decreasing order of pickup likelihood.. From here, we combine the rank of each path's pickup likelihood with its rank based on distance travelled, and return the top three paths.

C. Frontend and Visualization

We utilize Google Map's Directions and Services APIs to build the frontend of our web application. Our application starts as a road map centered in Manhattan and incorporates buttons and search boxes.

Our application provides the standard Google Maps buttons to control zoom and view. On start, we attempt to retrieve the user's current location in the backend. Because Google Maps location sharing requires HTTPS, we host our server at Cloud9. If location sharing is disabled or not supported, we alert the user and suggest that they either enter their origin manually or change their settings. If location sharing is enabled, the map automatically centers on the current location and shows the user's location while driving along the path[1].


https://dva1-alhaamid.c9users.io
359 East 52nd Street, New York, NY, United States
Columbia University, New|
Columbia University
Columbia Law School
School of International and Public Affairs
Columbia University Department of Economics
Columbia University Department of Biomedical Engineering
DTOUR
Destination
Source
Higher score
Medium score
Lower score

Our application provides a destination and an optional origin search box. While the user is entering a location, we query Google Maps for location suggestions and display them in a drop-down menu. Once both locations are chosen, we query Google Maps for the coordinates and send them to the backend for processing.

The frontend receives coordinates for each path computed by the backend. For each path, we query Google Maps to find routes between all intermediate nodes. These routes are color-coded based on their distance/pickup likelihood trade-off.

D. Linking Frontend and Backend

The backend is coupled in a jar file and stored at the server for the frontend. Whenever a driver requests paths, the coordinates of the source and destination are sent to the server. The server runs the jar file on the backend which contacts the database hosted on Azure for calculations. The jar file returns the sorted paths to the server which sends the paths to the driver's device for visualization.

V. List of Innovations

1. Application for rideshare drivers that finds passengers on the way to a given destination, rather than just finding hotspots.
2. K-d tree is used to speed up the lookup of the closest node to pickup/drop-off locations. Transition graphs are stored in a modular way so that they can be combined to form new transition graphs.
3. Rather than store the entire graph in memory during A*'s execution, we query neighbor edges on demand from the database. Additionally, during Path Graph construction, we implement a sophisticated reference linking scheme to avoid allocating memory for every vertex.
4. The original K* algorithm used all sidetrack edges when creating alternate paths. We recognized that sidetrack edges that don't touch A*'s shortest path aren't worth considering. This drastically improves runtime without impacting correctness.

[1] We disabled the current location feature in the demo web app for testing purposes only, since one can not test the application for New York city while the map is centered in Atlanta.

VI. Evaluation

We evaluated our approach using the NYC taxi dataset by simulating trips as follows:

1. Generated 500 trips between locations in Manhattan (start and end point at least 1 mile apart).
2. Ran our algorithm to calculate the three best paths for these trips.
3. Queried Google Maps to find the default shortest trip path, and mapped this path to our probability graph to serve as the benchmark path (proxy for Destination Mode).
4. Compared DTOUR's best path to the benchmark path in terms of:
    a. Pickup Likelihood
    b. Total Path Distance
5. For each simulation, we recorded the runtime of our algorithm's computation

VII. Results

The results in Figure 1 describe the distributions of each of the three experiments we ran:

- DTOUR v. Destination Mode in terms of overall path likelihood of having a passenger,
- DTOUR v. Destination Mode in terms of overall path length
- DTOUR overall computation time

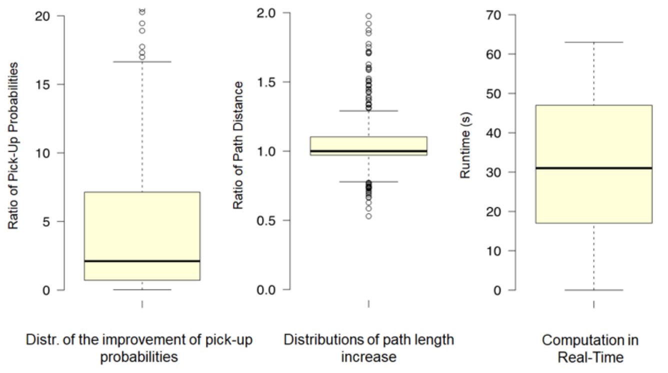


Figure 1: Evaluation Results

On average, the probability that a driver has a passenger in the vehicle with them at any given point on the path is 25 times higher with DTOUR's path than with Destination Mode's path. This average, however, is skewed by outliers (visible in 'Distr. Of the improvement of pick-up probabilities' plot). Thus, the median probability improvement of DTOUR is around 2.5 times.

In terms of path length, our simulations revealed that on average DTOUR's paths are only 5% longer than those of the Destination Mode proxy. Moreover, the variance of these relative path lengths is fairly low as indicated by the 'Distributions of path length increase' box-plot (note how close together both whiskers are). This suggests that, most often, the higher probability paths are only slight deviations off of the shortest path.

Finally, in terms of computation time, we find that DTOUR's algorithm takes only 30 seconds on average to return the best paths. In fact, the third box and whisker plot ('Computation in Real-Time'), indicates that 80% of the time, the computation takes less than a minute.

VIII. Conclusion

In this project, we created a web application that suggests profitable paths for rideshare drivers to take on their way to a specific destination. We implemented a variation of the K* algorithm to find paths with high probability of picking up a passenger at any particular time of the day. According to the probability-distance trade-off of K*'s results, we output the three best paths. The algorithm is embedded in a web application that takes a user's current location and destination based on the Google Maps API and visualizes the three best paths on the map in a color-coded fashion. The algorithm and the front end were evaluated using the NYC taxi data set, which was processed and transformed into transition graphs. By simulating 500 trips through Manhattan and comparing the path lengths and pick up probabilities of our paths against Google Map's shortest path (proxy for Destination Mode), we found that our paths increase the pick-up probability significantly while only introducing a relatively small detour of 5%.

However, we found that the time to compute the three best paths for a requested trip is currently ~30 seconds on average. This is slightly longer than most users would desire. We think that path caching would be a good way to improve our performance. In addition, we would like to do further testing on the time discretization of the path probabilities. Currently we compute the transition graph based on the current hour of the day. However, we do not know if discretizing in different time intervals and for different week days might yield better results. Because creating different transition graphs is very computationally expensive, we leave this task for further projects.

IX. Work Division

All members have done an equal share of work.

TABLE I. DIVISION OF WORK

Member	**Part**
Shinya Hirata & Adiki Madhu Babu	Data Cleaning & Transition Graph Generation
Vyas Srinivasan & Tarun Bhatia	Path Computation / Algorithm Implementation
Abdul Lateef Haamid & Amelia Glaese	Frontend Design Implementation & Application Integration

References

[1] A. Dogtiev, "Uber Revenue and Usage Statistics 2017," 2017. [Online]. Available: http://www.businessofapps.com/data/uber-statistics/. [Accessed: 11-Aug-2017].

[2] J. Dai, B. Yang, C. Guo, and Z. Ding, "Personalized route recommendation using big trajectory data," *IEEE 31st Int. Conf. Data Eng.*, pp. 543–554, 2015.

[3] B. Barann, D. Beverungen, and O. Müller, "An open-data approach for quantifying the potential of taxi ridesharing," *Decis. Support Syst.*, 2017.

[4] C. N. Kamga, "Urban link travel time estimation using large- scale taxi data with partial information," *Transp. Res. Part C Emerg. Technol.*, vol. 33, no. August, pp. 37–49, 2013.

[5] N. Ferreira, J. Poco, H. T. Vo, J. Freire, and T. Silva, "Visual Exploration of Big Spatio-Temporal Urban Data : A Study of New York City Taxi Trips," *IEEE Trans. Vis. Comput. Graph.*, vol. 19, no. 12, pp. 2149–2158, 2013.

[6] J. Deri, "Taxi data in New York city : A network perspective," in *IEEE 49th Asilomar Conference on Signals, Systems and Computers*, 2016, pp. 1829–1833.

[7] K. Zhang, Z. Feng, S. Chen, K. Huang, and G. Wang, "A Framework for Passengers Demand Prediction and Recommendation," *2016 IEEE Int. Conf. Serv. Comput.*, 2016.

[8] E. Kourti, C. Christodoulou, L. Dimitriou, S. Christodoulou, and C. Antoniou, "Quantifying Demand Dynamics for Supporting Optimal Taxi Services Strategies," *Transp. Res. Procedia*, vol. 22, pp. 675–684, 2017.

[9] B. Donovan and D. B. Work, "Using coarse GPS data to quantify city-scale transportation system resilience to extreme events," *arXiv Prepr. arXiv1507.06011*, 2015.

[10] D. Liu, "Density Peaks Clustering Approach for Discovering Demand Hot Spots in City-scale Taxi Fleet Dataset," 2015.

[11] N. J. Yuan, Y. Zheng, L. Zhang, and X. Xie, "T-Finder : A Recommender System for Finding Passengers and Vacant Taxis," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 10, pp. 2390–2403, 2013.

[12] J. Yuan, Y. Zheng, L. Zhang, X. Xie, and G. Sun, "Where to Find My Next Passenger ?," *Proc. 13th Int. Conf. Ubiquitous Comput.*, pp. 1–10, 2011.

[13] H. Dong, X. Zhang, Y. Dong, C. Chen, and F. Rao, "Recommend a Profitable Cruising Route for Taxi Drivers," in *IEEE 17th International Conference on Intelligent Transportation Systems (ITSC)*, 2014, pp. 2003–2008.

[14] M. Qu, H. Zhu, J. Liu, G. Liu, and H. Xiong, "A Cost-Effective Recommender System for Taxi Drivers," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 45–54.

[15] M. Veloso, "Taxi Driver Assistant – A Proposal for a Recommendation System," 2015.

[16] D. Eppstein, "Finding the k shortest paths," *SIAM J. Comput.*, vol. 28, no. 2, pp. 652–673, 1998.

[17] E. de Queiros Vieira Martins and J. L. E. dos Santos, "A New Shortest Paths Ranking Algorithm." 1999.

[18] H. Aljazzar and S. Leue, "K∗: A heuristic search algorithm for finding the k shortest paths," *Artif. Intell.*, vol. 175, no. 18, pp. 2129–2154, 2011.

[19] M. Kurant, A. Markopoulou, and P. Thiran, "Towards unbiased BFS sampling," *IEEE J.*

Sel. Areas Commun., vol. 29, no. 9, pp. 1799–1809, 2011.

[20] L. Lovasz, “Random walks on graphs: A survey,” *Comb. Paul Erdos Eighty*, vol. 2, 1993.

[21] Uber, “Uber Destination Mode.” [Online]. Available: https://d1a3f4spazzrp4.cloudfront.net/chameleon-assets/v1.0.0/e39a73d8-0c92-4e3f-82df-23e39a9a6884/tablet.gif. [Accessed: 13-Oct-2017].

[22] U. People, “Is it just me, or is the Lyft destination filter useless for everyone?,” 2016. [Online]. Available: https://uberpeople.net/threads/it-just-me-or-is-the-lyft-destination-filter-usless-for-everyone.78031/. [Accessed: 13-Oct-2017].