Balancing Efficiency and Fairness in On-Demand Ridesourcing

June 19, 2019

Through this supporting information, we present the detailed handling of raw datasets used in generating real-time requests, Manhattan road network, and other dependencies. We also provide the details on our simulation of real pickup and drop-off of passengers based on taxi dispatch algorithm REASSIGN and the corresponding source codes.

1 Data Pre-processing

1.1 Request Generator[1,2]

To generate our requests, we use the publicly available dataset of taxi trips in New York City[3], which contains for each day the dates, times, and coordinates of all the pickups and drop-offs executed by each of the active taxis. Each trip record also includes fields such as the taxi medallion number, fare amount, and tip amount. We focus on the 15,285,049 trips happening in May 2013. For our multi-period experiment, we choose a representative 2-hour horizon, 1700 – 1900, whereas in the single-batch setting, we use the requests coming in its first 30 seconds.

Our data pre-processing[4] includes the checking of trip records validity, such as missing coordinates, negative coordinates, and identical pickup and drop-off coordinates. We exclude trips that are deemed to be too short (travel time of less than 2.5 minutes) or too long (travel time of more than 1 hour). Then, we match each coordinate in our trip data to its closest node (by Euclidean distance) in the road network. To account for our network sparsity, we exclude requests that are potentially unreachable, e.g. not within $\Omega = 120$ seconds from sufficiently many (0.2% of all generated trips) drop-off nodes. This is to avoid vehicles being unable to escape one node after serving request, since we assume no rebalancing of unassigned vehicles in any rounds.

1.2 Road Network[5,6]

To build a road network of Manhattan, we use data from Mapzen[7,8] (similar to OpenStreetMap). We filtered the streets of Manhattan, selecting only the specific road classes: primary, secondary,

[1] `./Data/RequestGenerator/May_2013_HOUR17_singlebatch.pkl`; external link: bit.ly/2IhGSes
[2] `./Data/RequestGenerator/May_2013_HOUR17.pkl`; external link: bit.ly/2KXPB7l
[3] Dan Donovan, Brian; Work. New york city taxi trip data (2010-2013), 2016.
[4] `./Data/RequestGenerator/taxicsv_to_FINAL.py`
[5] `./Data/RoadNetwork/manhat_point.pkl`; external link: bit.ly/2XLDWfh
[6] `./Data/RoadNetwork/manhat_edge.pkl`; external link: bit.ly/2Re7sb8
[7] `./Data/RoadNetwork/RoadPoint.csv`
[8] `./Data/RoadNetwork/RoadEdge.csv`

tertiary, residential, unclassified, road, living street. Several other classes, such as foot, service, trunk, bridleway, pedestrian, were deliberately excluded. Next, we extracted the street intersections to build a network in which nodes are intersections and directed links are roads connecting those intersections. The extracted network of street intersections was then manually cleaned[9] for obvious inconsistencies or redundancies (such as duplicate intersection points). In order to simplify the network, we deleted few edges and points such that the final network is strongly connected and each node's degree is strictly more than two. In the end, our road network contains 3,671 nodes and 7,674 directed edges.

1.3 Travel Time Estimates[10]

To actually emulate real road situations, we need to estimate the travel times between arbitrary origin/destination nodes in the road network, due to the lack information of trajectory, vehicle speed, and traffic conditions in the NYC trip data. We use the estimation heuristic[11] described in the paper "Quantifying the benefits of vehicle pooling with shareability networks"[12] and store the hourly estimates into look-up tables [13].

2 Simulation

2.1 Setups and Background

Single-Batch Setting. We handpick days where there are more than 200 requests arriving in the first 30 seconds of 1700 to 1900 horizon and experimented with several cross-sectional scenarios when vehicles have been on the road for some time and are available to serve new request. To introduce discrepancies in activity history, we define two groups of vehicles with different activity types, V_H and V_L comprised of vehicles with *high* and *low* historical utilities, respectively.

Scenarios are created based on the following:

- Type of requests generated, e.g. trip length ≥ 400 seconds
- Ratio between the number of requests m and vehicles n, e.g. $n = 1.2m$
- Initial vehicle locations, e.g. within $\Omega = 210$ seconds from ≥ 10 pickup nodes.
- Proportion between vehicles with different activity types, $n_H : n_L$, e.g. $n_H = 5n_L$
- Historical utilities sampling distribution of each activity type, e.g. $U(200, 400)$ and $U(50, 100)$

Multi-Period Setting. Simulations are run for 10 different days and fairness weightages $\lambda \in [0, 1]$. We discretize 2-hour horizon into $T = 240$ rounds such that each round is 30-seconds long. At round 0, $n = 2000$ vehicles with capacity $\xi = 4$ are initialized at various locations within reasonable distance from possible pickup nodes. The historical utility of each vehicle is set to 0. Requests are generated

[9] `./Data/RoadNetwork/manhat_preprocess.py`

[10] `./Data/TravelTimePrediction/timecost17.csv`; external link: bit.ly/2WCMsf3

[11] `./Data/TravelTimePrediction/training_edge_traveltime.py`

[12] Paolo Santi, Giovanni Resta, Michael Szell, Stanislav Sobolevsky, Steven H Strogatz, and Carlo Ratti. Quantifying the benefits of vehicle pooling with shareability networks. Proceedings of the National Academy of Sciences, 111(37):13290–13294, 2014.

[13] Accessible through: bit.ly/2F9aR6j

according to the dataset at the start of each round to emulate real-time demand. We assume that requests are only available for assignment in the round it is generated.

Following is the summary of what happens in each round:

- Build a bipartite graph of available vehicles and requests based on specified constraints, such as maximum waiting time $\Omega = 120$s and maximum delay $\Gamma = 300$s.
- Solve a maximum weight matching problem for the given λ using REASSIGN Algorithm
- Move the assigned vehicles according to assignment (output of matching includes routing); unassigned vehicles need to wait until the next round assignment
- Historical utilities are updated to account for the newly assigned trip

For simplification, we assume there is no reassignment of requests in any subsequent rounds. In the ridesharing setting, it is possible to output new routing by "stacking" newly assigned requests to the old ones.

2.2 Code Summary

The simulation codes can be summarized into several parts as follows,

Main Script. To import data, setup parameters, and perform other initialization steps. We have three such scripts for each single-batch[14], and multi-period rideshare setting[15].

Algorithms[16,17]. To take in the bipartite graph built in the main script and output optimal routing for each vehicle. Routing is uniquely determined by allocated request, which is trivial in the no ridesharing setting. In ridesharing setting, a vehicle can be assigned to at most one additional request and how to stack this request to the prior allocated route is determined by the shortest travel time. We use 2 algorithms in these scripts:

- REASSIGN - Compute the desired matching based on λ and the efficient and fair matching. Efficient matching is computed using the original bipartite graph. Fair matching is reached by iterative edge deletion, i.e. removing the edges where the fairness value is below some threshold.
- Hungarian Algorithm[18,19] - Compute maximum weight matching in each iteration of fairness threshold

Routing[20,21]. To convert a list of requests to an optimal routing or to check the feasibility of trips given the waiting time, delay, and other constraints. The latter is used when building the bipartite graph; an edge between vehicle and request shows that the optimal route satisfies the constraints.

[14] `./SimulationCodes/NoRidesharing/main_singlebatch_artificial.py`
[15] `./SimulationCodes/Ridesharing/main_ridesharing.py`
[16] `./SimulationCodes/NoRidesharing/bipartite_algo.py`
[17] `./SimulationCodes/Ridesharing/ridesharing_algo.py`
[18] `./SimulationCodes/NoRidesharing/HungarianAlgorithm.py`
[19] `./SimulationCodes/Ridesharing/HungarianAlgorithm.py`
[20] `./SimulationCodes/NoRidesharing/searching.py`
[21] `./SimulationCodes/Ridesharing/searching.py`

Mapsystem[22,23]**.** A system built based on the road network of Manhattan such that each location (i.e. a *Waypoint* object) is determined by the travel time from the nodes at two endpoints. The main function of this class is to update vehicle location while moving to serve its assigned requests.

Requests[24,25] **and Vehicles**[26,27]**.** To create *request* and *vehicle* objects in the simulation. The functions largely involve updating of status in each round after assignment, pickup, and drop-off.

2.3 Output

Efficiency and *fairness* are recorded for each day and fairness weightage λ in each round[28]. In the multi-period setting, we additionally record the results corresponding to the end of 2-hours assuming that the same λ is used in all 240 rounds[29]. The performance measures used are $\mathcal{E}(M_\lambda)/\mathcal{E}_{\text{opt}}$ and $\mathcal{F}(M_\lambda)/\mathcal{F}_{\text{opt}}$, which compare the *efficiency* and *fairness* of our matching M_λ to the optimal efficiency and fairness. It is important to note that while in the single-batch setting $\mathcal{E}_{opt} = \mathcal{E}(M_{\text{eff}})$ when $\lambda = 0$ and $\mathcal{F}_{opt} = \mathcal{F}(M_{\text{fair}})$ when $\lambda = 1$, this is not necessary true in the multi-period setting due to our myopic assignment and demand randomness. Therefore, in general, we define $\mathcal{E}_{opt} = \max_{\lambda'} \mathcal{E}(M_{\lambda'})$ and $\mathcal{F}_{opt} = \max_{\lambda'} \mathcal{F}(M_{\lambda'})$. Once we have these two measures for all 10 randomly sampled days, we take the mean and variance for each λ. Error bars plotted correspond to 95% confidence interval.

Specific to the single-batch setting, we record the accumulated trip utilities h_v of each vehicle[30] for $\lambda = \{0, 1\}$ and show the fairer redistribution of trips when $\lambda = 1$.

[22] `./SimulationCodes/NoRidesharing/Waypoint.py`
[23] `./SimulationCodes/Ridesharing/Waypoint.py`
[24] `./SimulationCodes/NoRidesharing/request.py`
[25] `./SimulationCodes/Ridesharing/request.py`
[26] `./SimulationCodes/NoRidesharing/vehicle.py`
[27] `./SimulationCodes/Ridesharing/vehicle.py`
[28] `./Result/BipartiteTradeoff.csv`
[29] `./Result/Tradeoff_w_Horizons.csv`
[30] `./Result/ActiveTime.csv`