Profit Optimization in Commercial Ridesharing (Extended Abstract)

Arpita Biswas
Indian Institute of Science
arpita.biswas@csa.iisc.ernet.in

Ragavendran Gopalakrishnan
Conduent Labs India
Ragavendran.Gopalakrishnan@conduent.com

Theja Tulabandhula
University of Illinois Chicago
tt@theja.org

Koyel Mukherjee
IBM Research-India
kmukherj@in.ibm.com

Asmita Metrewar
Apple India
asmita_metrewar@apple.com

Raja Subramaniam Thangaraj
Conduent Labs India
Rajasubramaniam.T@conduent.com

ABSTRACT

Ridesharing is undoubtedly appealing from a sustainability perspective; however, profit optimization is very important for commercial providers. As such, ridesharing offers a valuable opportunity to garner profit by reducing the number of driver-miles that need to be paid for in order to serve the same set of passengers. In this paper, we study profit optimization for commercial ridesharing service providers such as UberPool and LyftLine.

CCS Concepts

•**Applied computing** → **Transportation;** *Economics; Multi-criterion optimization and decision-making;*

Keywords

Profit optimization; ridesharing; computational sustainability

1. INTRODUCTION

Ridesharing reduces harmful emission of greenhouse gases, and hence, is an important means to move into a sustainable future. In this work, we focus on commercial ridesharing service providers such as UberPool and LyftLine, that hold a significant share of the ridesharing population, and whose continued existence is important to further enhance the adoption of ridesharing. While ridesharing is undoubtedly appealing from a sustainability perspective, profit maximization is key to commercial providers. Thus, it is crucial to model the real-time ridesharing problem and match streaming trip requests into shared rides efficiently, while maximizing profit.

2. RELATED WORK

A rich body of literature has surveyed and studied several optimization problems related to ride sharing [1, 3, 8]; however, the problem of profit optimization for commercial ridesharing providers has been rarely discussed. Banerjee et al. [2] study pricing in ride sharing platforms; however, their work is focused on characterizing the pros and cons of static versus dynamic pricing, with respect to (a) profit earned, and (b) robustness to fluctuating system parameters. Jung et al. [4] provide methods and heuristics for (a) minimizing passenger wait times and detours, and (b) maximizing profit, while ensuring that passenger wait times and detours are bounded. While our work resembles theirs, unlike them, we do not explicitly bound detours. Instead, we handle detours by proposing a detour-based discount scheme for passengers, and then match passengers for ridesharing to maximize profit for the service provider. Nguyen [7] proposes a dynamic, demand-responsive ride sharing system through an auction mechanism, where customers are offered a discount that is dependent on the detour incurred; however, the discount is additive in contrast to the multiplicative detour-based discount suggested in our work.

Appears in: *Proc. of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, São Paulo, Brazil.*

3. RIDESHARING SYSTEM MODEL

We assume that all passengers requesting rides are willing to share their ride against a given discount function. Our model handles both kinds of ride requests: (a) "Ride Now" requests (passengers who are ready to leave as soon as possible from the time of the request and (b) "Ride Later" requests (passengers opting to ride at a later point of time). For "Ride Later", we delay processing their requests until a few minutes before the requested departure time. Requests from multiple passengers arrive at the system dynamically and continuously. Request i arrives into the system at time a_i, and is associated with a source-destination pair (S_i, D_i), and a time duration t_i that denotes the maximum amount of time that the passenger is willing to wait for a cab to pick them up at source S_i. We assume that a matching algorithm runs frequently and on a predetermined schedule.

At any time t, given the geographical distribution of cabs with at least one vacant seat that is available for ridesharing, an expected time duration for passenger i to be picked up, $\tau_i(t)$, can be estimated as

$$\tau_i(t) = \alpha_i(t) + \beta_i(t)$$

where $\alpha_i(t)$ denotes the expected time duration from t until the next time the matching algorithm will be run, and $\beta_i(t)$ denotes the expected time duration from $t + \alpha_i(t)$ for a cab to reach S_i if all the requests waiting to be processed by the next run of the matching algorithm were to be assigned dedicated cabs with no ridesharing, on a first-come-first-served basis. $\tau_i(t)$ is continuously updated on the user interface so that it ensures $t_i > \tau_i(a_i)$ when request i is submitted and arrives at the system at time a_i.

At a high level, whenever the matching algorithm is scheduled to be run (say at time T), we do the following:

(1) We run the matching algorithm with all the waiting requests up to time T, subject to the constraints that each request i gets picked up no later than time $a_i + t_i$. Since we ensure that $t_i > \tau_i(a_i)$, the trivial matching (where there is no ridesharing and every request is assigned a separate cab) is always feasible.

(2) At the end of the matching algorithm, after the matched requests have been assigned cabs, if any unmatched requests have a large enough t_i that they can afford to wait until the next run of the matching algorithm, i.e., if $t_i > T + \tau_i(T) - a_i$, then we keep such requests in the system, since they may be matched with the next batch of requests that arrive. Otherwise, we assign each of them to a dedicated cab.

The points above describe, at a high level, how a standard commercial ridesharing platform (such as UberPool, LyftLine) may be expected to work. The core aspect of these platforms involves running a matching algorithm at regular time intervals whose input is a pool of waiting requests, and whose output is a set of matched requests. The matching algorithm should provide the set of matched requests in real time (within fraction of a second).

4. HEURISTICS FOR REAL-TIME RIDE MATCHING

In this section, we describe a class of real-time greedy heuristics that run frequently and on pre-determined schedules. The algorithm takes as input n cabs, each with a dedicated request assigned to it initially, and returns a set of at most n matched cabs, each with one or more requests assigned to it. We define the capacity constraint of a cab to be the maximum number (say, C) of passengers it can serve at a time. We also define "positive gain constraint" to be the restriction that the incremental profit (the difference between the profit gained on merging two cabs over the sum of individual profits of the cabs) is always positive.

The matching algorithm starts by creating a list of the n cabs according to a certain order (e.g., increasing or decreasing order of total distance traveled, total time taken, net profit earned, etc.). The method initializes a set of "fully allocated" cabs to the empty set and repeatedly performs the following steps:

(a) Remove the cab at the top of the list, designating it as the cab that could potentially be merged with another cab in the list.

(b) Traverse the new list (examining the remaining cabs) from the top and select the first cab that can be merged with the designated cab in Step (a), subject to the capacity constraint, positive gain constraint, and any routing constraints, i.e., all the requests must be served within their respective waiting time thresholds according to the best merged route.

  (i) If such a cab is found, remove it from the list, and merge the requests in the two cabs into a single new cab. If there is still space for one or more additional requests that could potentially be served in this new cab, insert the new cab into the list of cabs at the appropriate location according to the order, and if not, the new cab is added to the set of "fully allocated" cabs.

  (ii) If such a cab is not found, then the designated cab in Step (a) cannot be merged with any other cab, so, add it to the set of "fully allocated" cabs.

Since every iteration of the above procedure decreases the size of the list of cabs by at least one, it will eventually become empty, at which point the method terminates and returns the set of "fully allocated" cabs as the output.

5. EXPERIMENTAL EVALUATION

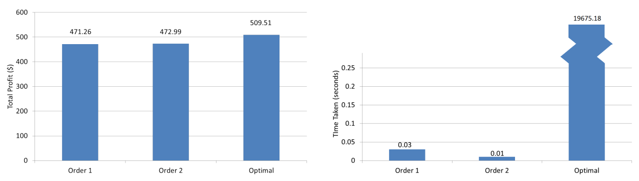
We perform an extensive evaluation of the greedy heuristics on publicly available New York taxi trip data [9]. The dataset has logs for trips in New York for the entire year of 2013. Each trip in the dataset specifies pickup and dropoff coordinates, pickup time, dropoff time, trip distance and travel time, and the trip can be considered as a taxi request in the simulation. We use this data to represent a real-time ridesharing platform with ride requests arriving and getting matched to cabs. We assume that each cab has a capacity $C = 3$. The values for base cost, cost per unit distance, cost per unit time, and driver's payment are taken from publicly available LyftLine prices [5, 6].

Spatial considerations: We map the pickup and dropoff locations of each trip to the closest location point among a predefined set of approximately 2500 location points. For this purpose, we have considered only those trips which both originate and end in New York City (for example, some trips in the original dataset originate in New York, and end in New Jersey; we exclude such requests).

Temporal considerations: We consider a time window of 10 minutes. In a given time window, we match requests regardless of their requested pickup times. Thus, in the time window 10:00 - 10:10 am, a passenger requesting a ride at 10:00 am can be matched with a passenger requesting a ride at 10:08 am. In online scenarios, this window could be as small as desired and our methods will still exhibit similar properties.

For comparison of our heuristics, we consider an optimal algorithm that exhaustively checks all possible ride-matches and groups the requests into shared rides that yield the highest net profit.

We consider 10 different sets of 20 trip requests, each between 10:00 - 10:10 am (rush hours) from the New York City taxi trip data. We then run various matching methods on each data set and compare the total profit and total time taken by each method averaged over 10 sets, against the optimal solution. Figures 1a and 1b show the total profit and time taken by "Order 1" (non-increasing order of distance travelled by each cab), "Order 2" (non-decreasing order of profit earned by each cab), and the optimal algorithm.



(a) Total profit averaged over 10 different datasets. **(b)** Time taken averaged over 10 different datasets.

Figure 1: Comparison of greedy heuristics against the optimal

We observe that the total profit obtained by using our heuristic methods is up to 92% optimal and runs 10^6 times faster than the optimal algorithm on a machine with a quad Intel Core i7 processor with 32GB RAM.

REFERENCES

[1] N. Agatz, A. Erera, M. Savelsbergh, and X. Wang. Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, 223(2):295–303, 2012.

[2] S. Banerjee, R. Johari, and C. Riquelme. Pricing in ride-sharing platforms: A queueing-theoretic approach. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, pages 639–639, 2015.

[3] M. Furuhata, M. Dessouky, F. Ordóñez, M.-E. Brunet, X. Wang, and S. Koenig. Ridesharing: The state-of-the-art and future directions. *Transportation Research Part B: Methodological*, 57:28–46, 2013.

[4] J. Jung, R. Jayakrishnan, and J. Y. Park. Design and modeling of real-time shared-taxi dispatch algorithms. In *Proc. Transportation Research Board 92nd Annual Meeting*, 2013.

[5] LyftLine. Lyft Line Driver Pay. https://help.lyft.com/hc/en-us/articles/213582268-Lyft-Line-Driver-Pay, 2016.

[6] LyftLine. Lyft Line Pricing. https://help.lyft.com/hc/en-us/articles/213815178-Lyft-Line-Pricing, 2016.

[7] D. T. Nguyen. *Fair cost sharing auction mechanisms in last mile ridesharing*. PhD thesis, Singapore Management University, 2013.

[8] D. Pelzer, J. Xiao, D. Zehe, M. H. Lees, A. C. Knoll, and H. Aydt. A partition-based match making algorithm for dynamic ridesharing. *IEEE Transactions on Intelligent Transportation Systems*, 16(5):2587–2598, 2015.

[9] C. Whong and A. Monroy-Hernández. New York City taxi trips. http://www.andresmh.com/nyctaxitrips.
Last accessed: 2016-06-18.