Price-aware Real-time Ride-sharing at Scale - An Auction-based Approach

Mohammad Asghari, Dingxiong Deng, Cyrus Shahabi, Ugur Demiryurek, Yaguang Li
University of Southern California, Los Angeles, CA, USA
{masghari,dingxiod,shahabi,demiryur,yaguang}@usc.edu

ABSTRACT

Real-time ride-sharing, which enables on-the-fly matching between riders and drivers (even en-route), is an important problem due to its environmental and societal benefits. With the emergence of many ride-sharing platforms (e.g., Uber and Lyft), the design of a scalable framework to match riders and drivers based on their various constraints while maximizing the overall profit of the platform becomes a distinguishing business strategy.
A key challenge of such framework is to satisfy both types of the users in the system, e.g., reducing both riders' and drivers' travel distances. However, the majority of the existing approaches focus only on minimizing the total travel distance of drivers which is not always equivalent to shorter trips for riders. Hence, we propose a fair pricing model that simultaneously satisfies both the riders' and drivers' constraints and desires (formulated as their profiles). In particular, we introduce a distributed auction-based framework where each driver's mobile app automatically bids on every nearby request taking into account many factors such as both the driver's and the riders' profiles, their itineraries, the pricing model, and the current number of riders in the vehicle. Subsequently, the server determines the highest bidder and assigns the rider to that driver. We show that this framework is scalable and efficient, processing hundreds of tasks per second in the presence of thousands of drivers. We compare our framework with the state-of-the-art approaches in both industry and academia through experiments on New York City's taxi dataset. Our results show that our framework can simultaneously match more riders to drivers (i.e., higher service rate) by engaging the drivers more effectively. Moreover, our framework schedules shorter trips for riders (i.e., better service quality). Finally, as a consequence of higher service rate and shorter trips, our framework increases the overall profit of the ride-sharing platforms.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services-*Commercial services*

Keywords

Ride-sharing, Revenue Maximization, Spatial Crowdsourcing

SIGSPATIAL'16, October 31-November 03, 2016, Burlingame, CA, USA
© 2016 ACM. ISBN 978-1-4503-4589-7/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2996913.2996974

1. INTRODUCTION

Real-time ride-sharing, as an alternative transportation service, alleviates traffic congestion and decreases auto emissions. With the emergence of many commercial platforms (e.g., Uber and Lyft), which automatically match drivers and riders on-the-fly, real-time ride-sharing becomes more and more popular. According to [1], millions of trips have been taken on UberPool since its launch at August 2014, and thousands of passengers take it five times a week during commuting hours. Enabled by the development and technology advances of smart phones and location-based services, ride-sharing platforms typically operate as follows: (1) Riders and drivers can join the platform via their smart phones, (2) a rider can submit a request, which consists of the pick-up and drop-off points, to the platform, (3) once a new request is received, the platform determines a driver (even en-route) to pick up the rider, (4) when the trip is completed, the platform calculates the rider's fare and the driver's income. With these platforms, riders can share a vehicle with reduced trip cost while enjoying fast and convenient transportation.

Many challenges exist to enable such real-time ride-sharing platforms. From a business point of view, the platform provider (e.g., Uber) seeks to maximize its own profit. However, higher profits should not be at the cost of either charging passengers more or paying drivers less than what would compromise participation and retention due to no monetarily incentive for either parties. Consequently, the design of a fair pricing model becomes an essential business strategy. This is particularly important in cases of carpooling where riders share their ride with other riders. Even though carpooling reduces the riders' cost, it incurs extra distance (i.e., detour) for riders. While each rider's fare should be discounted as a function of the length of the detour, the driver should be rewarded more as the total travel distance is increased due to all detours. Furthermore, different users (i.e., riders and drivers) might value their time differently. Therefore, a fair pricing model should be available to both riders and drivers to express, for a certain amount of detour, how much discount or compensation they expect. Finally, in addition to fair pricing scheme for riders and drivers, the model should account for the provider's revenue as well.

The second challenge of a ride-sharing platform is to process incoming requests in real-time. This involves two different tasks: i) checking which drivers can add the new requests (new pick-up and drop-off locations) to their current trip without violating the constraints of that trip (i.e., scheduling) and ii) selecting the best driver among those who can serve the new request (i.e., matching). Processing the schedule of potentially thousands of drivers to check if they can accommodate a new request, with the additional task of matching their pricing profile with the incoming rider's profile, becomes computationally intensive. Therefore, the design of an efficient and scalable algorithm that assigns riders to drivers with a fair pricing model while maximizing the provider's revenue, is very challenging.

The majority of previous studies [2, 3, 4, 5] focus on improving the efficiency of on-the-fly assignment with the objective of minimizing the total travel distance of drivers. In particular, in existing studies a new request is assigned to a driver who can fit the request in his schedule with the least amount of increase in the total traveled distance. However, minimizing drivers' total travel distance is not always equivalent to overall shorter trips for riders. Consequently, when assigning a new request, the driver who would incur the minimum increase in total travel distance is not necessarily the most cost effective option. To illustrate, suppose driver a has two passengers on board and driver b has only one. To serve an incoming request, a's incurred detour is 2 miles while for b the detour is 3 miles. Even though a's detour is shorter, the platform owner has to compensate both passengers of a for 2 miles (a total of 4 miles) while in the case of b it has to compensate only one passenger for a total of 3 miles. In addition, from the riders' perspectives, in the first scenario two riders incur extra detour while in the second only one rider incurs an extra detour. Few studies [6, 7] consider a pricing model by defining monetary incentives for riders and drivers. In [6], a pricing model is introduced where instead of being compensated, a rider can potentially end up being penalized for longer detours by paying a higher fare. Ma et. al. [7] overcomes the unfairness issue in [6] to some extent. Even though, a new rider can incur detour in his trip, their model only compensates riders that are already on board. In addition, since this model is targeted for a different application, the notion of revenue fails to provide any incentive for the platform provider. Furthermore, in all previous studies [6, 8, 7], a centralized server is responsible for matching and scheduling incoming requests. Most of these studies utilize a spatiotemporal index to enable matching, i.e., narrowing down the number of potential drivers who can serve an incoming request. With thousands of drivers in the system, even after applying the spatial index, the centralized server still needs to perform scheduling and profile matching for all the candidate drivers. We show that with large number of drivers, these frameworks fail to process new requests in real-time (Section 5.3.4) and hence not scalable.

To address aforementioned challenges, in this paper, we introduce an Auction-based Price-Aware Real-time (*APART*) ride-sharing framework. We propose a general and versatile pricing model that allows both riders and drivers to set their monetary expectations for participating in ride-sharing based on their predefined profiles. Specifically, each rider's profile defines the expected discount ratio for the detours incurred by ride-sharing. For example, one rider can express that he is willing to accept a 10 mile detour for 30% discount. On the other hand, each driver's profile defines the expected cost in terms of his total travel distance and time. The model also accounts for the revenue of the platform provider. Consequently, our objective is to maximize the revenue of the ride-sharing framework while satisfying various temporal and monetary constraints of all users. APART is price-aware because a new request is assigned to a driver which generates the highest profit. Since our pricing model is designed to compensate riders for detours, the most profitable choice is also the one where *riders* incur the least amount of detour, hence better service quality. Finally, APART also maximizes the revenue of the provider by increasing the service rate (throughput) in the system through engaging available drivers more effectively to serve more requests.

To efficiently assign riders to the candidate drivers, we introduce a distributed auction-based framework. With our framework, APART, the server broadcasts a new request to a set of candidate drivers and the mobile app of each candidate driver[1] computes and submits a bid based on the driver's current schedule, his and his other passengers' profiles and other spatiotemporal constraints. Subsequently, the server collects all the bids from candidate drivers and assigns the rider to the highest bidding driver. To guarantee high service quality, each driver runs a branch-and-bound algorithm that performs an exhaustive search to find out whether it can fit a new request into its current scheduling. Each driver carries a small number of riders so even an exhaustive search can be performed in real-time. Due to the distributed nature of APART, all candidate drivers perform the search in parallel. Once each driver finds its own best schedule, the server simply selects the driver that generates the highest profit. Consequently, APART is able to find the most profitable drivers in real-time.

We conducted extensive experiments on a large scale New York City taxi dataset and show that APART is scalable and efficient, capable of processing hundreds of tasks per second in the presence of thousands of drivers. By comparing our framework with the state-of-the-art approaches [8], we show that our framework can simultaneously match up to 10% more riders to drivers (i.e. higher service rate), while the total travel distance of riders are 20% less (i.e., better service quality), hence our framework can generate more profit than other approaches with an even better service quality. On the other hand, we show that in a framework were riders are assigned to drivers with the least increase in the driver's travel distance, up to 25% of the requests are not assigned to the most profitable driver.

The remainder of this paper is organized as follows. We define our problem in Section 2, and explain our pricing model in Section 3. We present our APART framework, and discuss its auction-based approach in Section 4. In Section 5, we report the experiment results. We discuss the related work in Section 6 and conclude the paper in Section 7.

2. PROBLEM DEFINITION

In this section, we define the terminologies, and formally define the problem under consideration.

2.1 Basic Concepts

The road network is represented as a graph $G(I, E)$, where each node represents intersections, and each edge represents a road segment. Each edge $(i, j) \in E$ $(i, j \in I)$ is associated with a weight $c(i, j)$ which is a travel cost (can be either time or distance) from i to j. The shortest path cost $d(s, t)$ is defined as a minimal cost path connecting s and t. With our approach, APART, time and distance can be converted from one to the other.

DEFINITION 1 (RIDE REQUEST). *A ride request r can be represented as $\langle s, e, w, \epsilon, f \rangle$ consisting of a starting point $s \in I$ and an end point $e \in I$. Each request also specifies w as the maximum time the rider can wait after making a request and the maximal detour $\epsilon \cdot d(s, e)$ the rider can afford . In addition, a rider's profile $f : \mathbb{R}_+ \rightarrow [0, 1]$, specifies the relative discount in exchange for an incurred detour of $\Delta d \in \mathbb{R}_+$.*

Upon the acceptance of a request, APART assigns it to a driver.

DEFINITION 2 (DRIVER). *A driver v is represented as $\langle L, n, g \rangle$ where L is the list of ride requests assigned to v, and n is the maximum number of requests v can accept at any point in time. A driver also has a profile $g : \mathbb{R}_+ \rightarrow \$$[2] which specifies the monetary cost of v driving a distance $d \in \mathbb{R}_+$ while servicing its assigned requests.*

[1]Hereafter we use the term "driver" to refer to both the human driver and the software running on his mobile device.

[2]In this paper, we show monetary values with $

DEFINITION 3 (SCHEDULE). *Given a set L with n requests, a schedule $S = \langle x_1, \cdots, x_{2n} \rangle$ is an ordered sequence of pick-up and drop-off points for these request, where for each $r_i \in L$, $r_i.s$ precedes $r_i.e$ in S.*

A schedule is *valid* for a driver *v*, if it satisfies the following conditions:

- The riders' waiting time constraint: for any request r_i, the waiting time from the time the request is made until v arrives at $r_i.s$ should be less than $r_i.w$.
- The driver's capacity constraint: the number of riders in the vechile cannot exceed the total capacity n.
- Detour constraint: the maximum distance of every rider's trip should be less than $(1 + \epsilon) \cdot d(r_i.s, r_i.e)$.
- The driver's and all riders' (the new rider and the those already in the vehicle) monetary constraints (See Section 3)

The driver follows the sequence of picking up and dropping off riders. The schedule changes over time as riders are serviced (picked-up/dropped-off) and new requests are added to the schedule. In fact, adding a new request to a schedule can re-order some requests that already exist in the schedule. For example, when a new request r_3 arrives, the initial schedule of $\langle s_1, s_2, e_1, e_2 \rangle$ can be reordered to $\langle s_1, s_2, s_3, e_2, e_3, e_1 \rangle$, where rider 1 is dropped off after rider 2.

DEFINITION 4 (MATCHING). *Assuming we have a set of drivers V and a set of requests R, we call $M \subset V \times R$ a matching if for each $r \in R$ there is at most one $v \in V$ such that $(v, r) \in M$. We call $(v, r) \in M$* a match.

In a matching M, for every driver v, there exists a *valid* schedule S_v, such that $(v, r_i) \in M \implies r_i.s \in S_v \wedge r_i.e \in S_v$ (or simply $r_i \in S_v$).

In Section 3, we define a generic *pricing model* where given a driver and its schedule, the pricing model computes the final fare each rider has to pay, the income of the driver and the ride-sharing platform's profit. Subsequently, we can define the *ride-sharing* problem as follows:

DEFINITION 5 (RIDE-SHARING PROBLEM). *Given a set of ride requests R and a set of drivers V, the goal of the ride-sharing problem is to find a matching M between R and V such that the revenue of M is maximized.*

The ride-sharing problem is NP-Hard since the Vehicle Routing Problem (VRP) [9] is reducible to the ride-sharing problem in polynomial time. A globally optimal solution to the ride-sharing problem can be achieved when a Clairvoyant exists which knows what requests are going to be submitted to the framework at what time and also has the knowledge of which drivers are going to be available, in advance. However, in this paper we study the *online* version of the problem, i.e., the framework has no knowledge regarding future requests and incoming requests have to be matched with drivers as soon as they are submitted to the framework. The optimality of online algorithms are usually analyzed using *competitive ratio* [10], i.e., an algorithm $\mathcal{A}$ is called *c-competitive* for a constant $c > 0$, if and only if, for any input $\mathcal{I}$ the result of $\mathcal{A}(\mathcal{I})$ is at most c times worst than the globally optimal solution. In the following we show no online algorithm can achieve a good competitive ratio for ride-sharing problem.

THEOREM 2.1. *There does not exist a deterministic online algorithm for the ride-sharing problem that is c-competitive ($c > 0$).*

PROOF. Suppose there exists an algorithm $\mathcal{A}$ that is *c-competitive*. For $\mathcal{A}$ to be *c-competitive*, it should be at most c times worst than the optimal solution for *every* input. Consequently, to show no *c-competitive* algorithm exists, we only need to show one input for which where $\mathcal{A}$ does not have a competitive ratio of *c*.
We assume there exist an adversary which knows every decision $\mathcal{A}$ makes and consider the input generated by this adversary. For simplicity, we assume there is only one driver at point $(0, 0)$. The input starts with r_1 with a pick-up location at $(w, 0)$ and r_2 with pick-up location at $(-w, 0)$ (we assume all requests have a maximum wait time of w). The algorithm can make three choices for the driver. (1) move toward r_1, (2) move towards r_2 and (3) stay still. If choice 1 is selected, the adversary can generate the input such that at time $t = 1$, n more request are submitted with pick-up location at $(-w - 1, 0)$ and drop-off locations similar to r_2. Similar arguments can be made if choice 2 or 3 are selected by the algorithm. A globally optimal solution can complete $n+1$ requests while $\mathcal{A}$ can at most complete one request. By adding more drivers far away in a similar situation, the adversary can make $\mathcal{A}$'s solution unboundedly worse than the optimal solution. Therefore, we contradicted the assumption that $\mathcal{A}$ is *c-competitive*. □

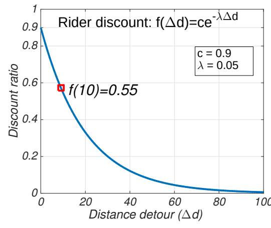
3. PRICING MODEL

In a ride-sharing platform where the objective is to maximize the monetary profit, it is important to utilize a pricing model which is *fair* to both riders and drivers. For example, the pricing model introduced in [6] compensates drivers based on the distance they travel and this is the total fare all riders have to pay (split). Therefore, for the portions of the trip where there are more than one passenger, the fare gets divided by the number of passengers on the vehicle. It is true that on average, riders end up paying less as compared to when they are the only passenger on the vehicle. However, the problem with this model is that the riders are still paying for the detours incurred in their trip, even though they split the cost. Therefore, if long detours are incurred in a rider's trip, the rider may end up paying even more than when he is the only passenger. For example, with a simple experiment on New York City's taxi dateset, we observed that up to 10% of riders pay more than what they would have paid if they did not participate in carpooling (see Section 5.4).

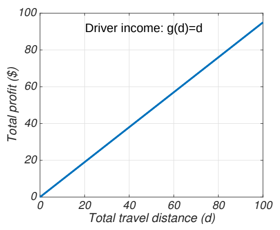
In this section, we define a generic pricing model which aims to satisfy the monetary constraints of the users of the system. Before we continue, it is important to note that one of the building blocks of any real-world ride-sharing system is to compute the shortest path between any two points in the road network. Without loss of generality, we define our pricing model based on a static road network where edge weights remain stationary during computation. However, our algorithms can be extended to incorporate time-dependent networks where cost of edges are time varying. For example, the fare of a ride is dependent on the *distance* between the pick-up and the drop-off points in a static network where in a time-dependent network it can be dependent on the *travel time* between those two points. A *fair* pricing model has to satisfy the following rules:

- For *every single rider*, if the rider's trip is longer than the shortest trip between his pick-up and drop-off location, the rider should receive a discount proportional to his detour (i.e., the difference between the actual and the shortest trip).
- For *a driver*, if the driver's trip is increased by serving more riders, the driver's compensation should increase proportional to the distance of the driver's trip.

Consequently, the pricing model should answer three key questions: (1) *"How much should the riders pay for a trip?"* (2) *"How*



(a) Example Rider's Profile



(b) Example Driver's Profile

Figure 1: Example User Profiles

much should the drivers be compensated for serving riders?" and (3) *"What is the revenue of the ride-sharing platform?"*. We define our generic pricing model by answering these three questions.

Every request r has a default fare based on the shortest distance, $d_r \in \mathbb{R}_+$, from s_r to e_r. In other words, every pricing model should have an arbitrary function $F : \mathbb{R}_+ \rightarrow \$$ such that $F(d_r)$ is the default fare of a ride. In a ride-sharing system, the actual route between the pick-up and drop-off locations of a ride is not necessarily the shortest route between the two points. We represent the actual route between the two end points of a ride with d'_r and define the detour of a ride as $\Delta d_r = d'_r - d_r$. As explained in Definition 1, each request is associated with a profile. We introduce the concept of a *rider's profile* as a tool for the rider to specify how much discount he expects to receive in return for a certain amount of detour on his trip. A rider's profile can have different formats: e.g., linear decay, exponential decay, etc., which represents that rider is not willing to take a service after the decay point. Fig. 1(a) shows an example of a rider's profile, where the rider will have 55% of discount for 10 miles of detour.

Subsequently, for a request r with shortest distance d_r, detour Δd_r and a profile f_r, the final fare is represented as:

$$fare(r) = F(d_r) f_r(\Delta d_r) \tag{1}$$

This guarantees that no rider pays more than what he would have paid if he took a solo ride. In fact, the rider will get compensated for longer trips due to the detour. This satisfies the first rule of our fair pricing model.

Every driver has a unique profile which allows him to specify the cost of his service. Similar to riders, drivers can have different expectations for participating in ride-sharing platforms. The drivers' profile allows them to set their expectations with respect to how much they expect to be paid for participating in the platform. The driver's profile can be any function. In fact, it can take any arbitrary input in addition to the distance. For example, it is possible to define the driver's profile as $g : \mathbb{N} \times \mathbb{R}_+ \rightarrow \$$ where for $\langle n, d \rangle \in \mathbb{N} \times \mathbb{R}_+$, n and d are the number of passengers being serviced by the driver and the driver's total traveled distance, respectively. Without loss of generality, we assume distance is the only input of the driver's profile. Intuitively, the profile is a monotonically increasing function. For example, the profile in Fig. 1(b) is one where the driver charges \$1 per mile.

At any point in time, each driver has a schedule. A driver will be compensated during the time its schedule is not empty. Therefore, for every driver v, the income is:

$$income_v = \int_{start_s}^{end_s} I\left(S_v(t) \neq \langle\rangle\right).g(d(t))dt \tag{2}$$

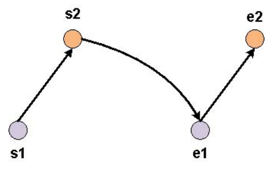
Where $I()$ is the indicator function, $S_v(t)$ and $d(t)$ are the driver's schedule and the distance he travels at time t, respectively. In addition, $start_s$ and end_s are the first pick-up time and last drop-off time of S_v. Consequently, regardless of the serviced requests, each driver receives an income only based on his total travel distance. This satisfies the second rule of our fair pricing model.

The amount of a driver's compensation does not necessarily have to be the same as what the riders pay for the same distance. It is the framework's responsibility to assign riders with drivers where their profiles are compatible. The profit APART makes from driver v is the difference between the fares collected from all riders serviced by v and the income v receives for himself. Subsequently, the total profit (revenue) of APART is the sum of the profits received from all drivers:

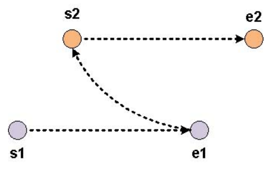
$$profit_v = \sum_{r_i \in s_v} fare(r_i) - cost_v \tag{3}$$

$$revenue = \sum_{v \in V} profit_v \tag{4}$$

A price-aware framework can utilize pricing models and profiles in order to provide a better service quality. Consider the example in Fig. 2 where a driver is en-route to pick-up a rider from s_1 and drop-off at e_1. Before the driver reaches s_1 a new request arrives with s_2 and e_2 as the pick-up and drop-off locations, respectively. We also assume both $\{s_1, s_2, e_1, e_2\}$ (Fig. 2(a)) and $\{s_1, e_1, s_2, e_2\}$ (Fig. 2(b)) are valid schedules. Any algorithm with the objective of minimizing the total travel distance will select the route in Fig. 2(a). With APART, the riders are able to set their profiles such that the framework might end up selecting either routes. For example, if riders want to get to their destination with the least amount of detour, they can set their expected compensation for small detours to a high value and the framework will select the route in Fig. 2(b) as the schedule for the driver. On the other hand, if riders are willing to share a ride and reduce the cost of their ride, they can do so by configuring their profiles differently. Depending on what the riders and drivers accept as a higher quality service, by setting their respected profiles they can adjust APART to provide them with a service that better suits them. Here we only considered two riders and a single driver in order to make the example in Fig. 2 simple. Even though in this example, a smaller detour is achieved through avoiding carpooling, in the experiments in Section 5.4 we show that at least 80-90% of the riders do engage in carpooling and yet, end up with a detour of only 6-7% of their original trip.



(a) Minimize Driver's Distance



(b) Minimize Rider's Detour

Figure 2: Examply of Price-aware Scheduling

We end this section with a note on the versatility of our pricing model. We explained our pricing model based on the objective of our framework which is maximizing the provider's revenue. However, using the same definitions and by configuring the riders' and drivers' profiles differently, we can adjust the framework to achieve other objectives as well. For example, consider minimizing total travel distance of drivers. Many studies [6, 8, 7] achieve this goal by assigning a new request to the driver with the least increase in travel distance. Theorem 3.1 shows, with only configuring the riders' and drivers' profiles appropriately, APART can make exactly the same assignments and achieve the same objective.

THEOREM 3.1. *If all the riders' profiles are set to $f'_r = 1$ and every drivers' profiles to $g'_v = 1$, by selecting the most profitable driver, APART will select the driver with the minimum increase in travel distance.*

PROOF. Upon arrival of a new request r, each driver in APART finds a schedule which generates the most profit. Since $f'_r(\Delta d_r) = 1$, regardless of Δd_r, the final fare for r is equal to $F(d_r)$ where d_r is the length of the shortest path between r's pick-up and drop-off points. Therefore, for an incoming request r, every driver v can compute the maximum profit it can generate for the system upon accepting r as: $profit_v = F(d_r) - g'(\Delta d_v)$ where Δd_v is the increase in v's traveled distance if r is assigned to v. Since $F(d_r)$ is the same regardless of the driver, the most profitable driver is the one with the smallest Δd_v. □

4. APART FRAMEWORK

In a real-time ride-sharing application, once the server receives a request, it needs to determine the driver who can best accommodate the new request with respect to his current schedule. With a large number of candidate drivers, the scheduling phase becomes the bottleneck in centralized frameworks where one single server processes the requests. Therefore, we introduce APART which overcomes this shortcoming by distributing the scheduling task to the drivers themselves. We first explain the auction framework in which the server broadcasts new arriving requests to the drivers. Subsequently, we discuss how each driver generates a bid based on its current schedule.

4.1 Dispatch Requests

Auction frameworks have been effectively used for assignment problems [11, 12]. Following the terminology used in auction frameworks, APART considers drivers as bidders and ride requests as goods. Note that the actual human driver does not engage in bidding, instead, his mobile app software does the bidding based on various constraints and goals. The server plays the role of a central auctioneer in APART. With APART, once a new request is received by the server (auctioneer), it presents the request to the drivers (bidders). Each driver computes a new schedule which incorporates the incoming request, and generates a bid based on the driver's and riders' profile. Subsequently the bid is submitted to the server. The bidding process is performed as a *sealed-bid auction* where drivers simultaneously submit bids and no other driver knows how much the other drivers have bid. In the end the server selects the driver with the highest bid as the winner and matches the request with the driver.

With APART, drivers that are far away from the pick-up location of an incoming request, are not asked to bid on the request. The server only sends an incoming request to *eligible drivers* that are defined as:

DEFINITION 6 (ELIGIBLE DRIVERS). *An available driver v is said to be eligible for servicing a newly submitted request r, if and only if:*

$$distance(v, r.s) \le r.w \times avg\_speed$$

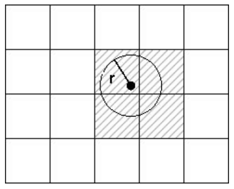


Figure 3: Spatiotemporal Grid Index

In other words, an available driver *d* is eligible for serving request *r*, if he has enough time to reach the pick-up location of *r* within *r*'s waiting time. In order to find eligible workers for each request, the server maintains a spatial index on the location of the drivers. With APART, the server does not need to know the exact location of the drivers to filter out non-eligible drivers. We use a grid index where the server only keeps track of which cell a driver is located in. For example, in Fig. 3, assuming the black dot is the pick-up location of a new request and r is the maximum wait time for the new request, any driver in the shaded cells will receive the new request. In our grid index, we use the filter and refine process in [13] in order to enable continuous query processing on the underlying road network using Euclidean distances.

Algorithm 1 Dispatch($V_r, r, startTime$)

```
Input: $V_r$ is the set of currently available drivers, $r$ is a new request and $startTime$ is the current time
Output: $v \in V_r$ as the driver that request $r$ is assigned to
1: $v_{selected} \leftarrow$ null
2: $Bids \leftarrow \emptyset$
3: for $v \in V_r$ do
4:     $b_v \leftarrow$ ComputeBid($v, v.schedule, r, startTime$)
5:     $Bids \leftarrow Bids \cup \{b_v\}$
6: end for
7: $v_{selected} \leftarrow \arg\max_x \{b_x \in Bids\}$
8: return $v_{selected}$
```

Algorithm 1 outlines the process of assigning an incoming request r, where V_r is the set of eligible drivers for request r (line 3). For each candidate driver v, the *ComputeBid* method (line 4) is executed to perform scheduling and compute v's bid (Section 4.2). Subsequently, the platform chooses the driver with the highest bid. In case of a tie in line 7, the algorithm randomly selects one driver among the ones with the highest bid. Notice that all the iterations of the **for** loop in Algorithm 1 (lines 3-6) run in parallel.

4.2 Bid Computation & Payments

Once a driver is notified of a new request, it has to compute a bid. The bid each driver generates reflects the profit the system can gain if the request is assigned to that driver. Once the ride-sharing application receives the request, it generates a bid and submits the bid to the server. When a new request is assigned to a diver, he will be notified with an updated schedule. This means that the human driver's interaction with APART is limited to configuring his profile on the ride-sharing application.

Algorithm 2 ComputeBid($v, v.schedule, r, startTime$)

```
Input: $v$ is a driver with schedule $v.schedule$, $r$ is a new request and $start\_time$ is the current time.
Output: additional $profit$ that $v$ can generate by accepting $r$
1: $src \leftarrow \{r'.s | r' \in v.schedule\}$
2: $src \leftarrow src \cup \{r.s\}$
3: $newProfit, newSchedule \leftarrow$FindBestSchedule($v, \emptyset, src, -\infty, \emptyset, startTime$)
4: $oldProfit \leftarrow$ GetProfit($v, v.schedule, startTime$)
5: $additionProfit \leftarrow newProfit - oldProfit$
6: return $additionProfit$
```

Algorithm 2 outlines the bid computation process. First, it inserts the pick-up locations (including the new request's pick-up point) in the list named src (lines 1-2). Subsequently, the algorithm calls *FindBestSchedule* which finds the best valid schedule and its corresponding profit using Algorithm 3. Because each driver's bid is the *additional* profit that the new request can generate for the platform, the algorithm calculates $oldProfit$ for v's original schedule using Algorithm 4 (line 4). Hence, the additional profit that v can generate by accepting r is the difference between $newProfit$

and $oldProfit$. The reason *FindBestSchedule* is initially called with only the pick-up locations is to guarantee, for every request its pick-up location is scheduled before its drop-off location.

Algorithm 3 FindBestSchedule($v, curList, remList, bestProfit, bestSchedule, startTime$)

```
Input: curList and remList are lists of pick-up/drop-off points that
    have been added and to be added to a valid schedule, respectively.
    $bestProfit$ and $bestSchedule$ are the best profit and corresponding
    schedule observed so far, and startTime is the current time.
Output: $bestProfit, bestSchedule$ as the best profit and corresponding
    schedule for input points if a valid schedule exists. Otherwise $-\infty, \emptyset$
1: if $curList$.size + $remList$.size > $v$.n ×2 then
2:     return $bestProfit, bestSchedule$
3: end if
4: for $p$ in $remList$ do
5:     $f' \leftarrow curList$
6:     $f'.add(p)$
7:     $profit \leftarrow$ GetProfit($v, f', startTime$)
8:     if $profit \neq -\infty$ then
9:         $r' \leftarrow remList$
10:        $r'.remove(p)$
11:        if $p$.type == pick-up point then
12:            $r'.add(p.req.e)$ // add the drop-off point into $r'$
13:        end if
14:        if $r'$.size == 0 then
15:            return $profit, f'$
16:        end if
17:        $profit, schedule \leftarrow$ FindBestSchedule($f', r', bestProfit,$
           $bestSchedule, startTime$)
18:        if $profit > bestProfit$ then
19:            $bestProfit \leftarrow profit$
20:            $bestSchedule \leftarrow schedule$
21:        end if
22:    end if
23: end for
24: return $bestProfit, bestSchedule$
```

We now explain how to find the most profitable schedule in Algorithm 3. The idea is to enumerate every valid schedule, calculate its profit and choose the most profitable one. Therefore, the algorithm recursively performs an exhaustive search to find the best valid schedule. Given the set of nodes that have already been added to the schedule(i.e., $curList$), and the remaining nodes(i.e., $remList$), at each iteration of Algorithm 3 (lines 4-23), one node from $remList$ is added to $curList$ (lines 5-6). Each time a new node is added to $curList$, the algorithm checks whether this partial schedule is valid. If the partial schedule is invalid, *GetProfit* in line 7 returns $-\infty$ and the search continues to the next branch. Otherwise, the variable $profit$ contains the profit of the partial schedule $curList$. Once the pick-up node of a request is added to $curList$, the corresponding drop-off node of the same request is added to $remList$ (line 11-13). If the remaining nodes get empty, the search on the current branch stops and the current branch's profit is returned (lines 14-16); otherwise, it recursively checks the new branch r' (line 17). The best profit is updated once the search finds a profit higher than $bestProfit$ (lines 18-21).

Figure 4 shows an example of using Algorithm 3 to find a best schedule with two requests r_1 and r_2. Each rectangle represents a node in the search tree, where the left and right sections contain the scheduled points and the remaining points, respectively (sets $curList$ and $remList$ in Algorithm 3). Initially $remList$ is started with the two pick-up locations. Each time a new pick-up point (e.g., s_1) is scheduled and moved from the right section to the left, the corresponding drop-off point (e.g., e_1) will be added to the right section. The shaded rectangles contain an invalid partial schedule in the left section and hence, the tree does not expand under them. The search continues until all the branches have been visited and the complete schedule with the highest profit is returned.

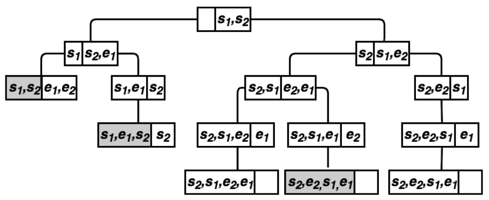


Figure 4: Illustration of Algorithm 3

Algorithm 4 GetProfit($v, schedule, startTime$)

```
Input: schedule is an ordered list of pick-up/drop-off points of
    driver $v$ and startTime is the current time.
Output: the $profit$ of performing the input $schedule$ at time
    $startTime$. If $schedule$ is not valid it returns $-\infty$
1: $time \leftarrow startTime$
2: $loc \leftarrow v$.loc
3: $distance \leftarrow 0$
4: for $p$ in schedule do
5:     $trip \leftarrow$ ShortestPath($loc, p$.loc)
6:     $distance$ += Distance($trip$)
7:     $time$ += TravelTime($trip$)
8:     if $p$.type == pick-up point then
9:         if $time > p$.req.req_time + p.req.w then
10:            return $-\infty$
11:        end if
12:        $pickUp$[$p$.req] $\leftarrow distance$
13:    end if
14:    if $p$.type == drop-off point then
15:        $\Delta d \leftarrow distance - pickUp[p$.req]
16:        if $\Delta d > p$.req.$\epsilon \times p$.req.sp then
17:            return $-\infty$
18:        end if
19:        $fare$ += $p$.req.f($\Delta d$) × F($p$.req.sp)
20:        $cost \leftarrow v$.g($distance$)
21:    end if
22:    $loc \leftarrow p.loc$
23: end for
24: $profit \leftarrow fare - cost$
25: return $profit$
```

Algorithm 4 computes the profit driver v can generate by completing $schedule$ at $startTime$. Each driver v runs this algorithm locally, and hence, $v.loc$ (line 2) and $v.g$ (line 20) refer to v's current location and profile, respectively. Also, for any node p in the schedule, p.req and p.req.sp refer to the corresponding request of node p and the shortest path for that request, respectively. Algorithm 4 iterates through the nodes in *schedule* one by one keeping track of the added $time$ and $distance$. Each node is either a pick-up node or a drop-off node. For pick-up nodes, the algorithm checks if the maximum wait time constraint is violated (line 9). For every drop-off node, the detour constraint is checked (line 16). If the check is successful, the algorithm computes the actual travel distance for the request and determines the incurred detour (line 19). After computing the detour, the algorithm computes the added fare and cost using Eqs. (1) to (3). If the input *schedule* is not valid, the algorithm returns $-\infty$ as the profit.

Once drivers submit their bids, the server selects the driver with the highest bid and assigns the new request to that driver.

5. EXPERIMENTS

5.1 Dataset

We evaluate our algorithms using one month (May, 2013) of New York City's taxi dataset [14], which contains 39437 drivers and around 500,000 trips per day. Each ride in the dataset has a pick-up latitude/longitude, a drop-off latitude/longitude and request time. We extracted the road network of New York City from Open Street Map (OSM), which is represented as an undirected graph with 55,957 vertices and 78,597 edges. Subsequently, we mapped the source and destination of each trip to the road network. Similar to [8], we maintain a cache for shortest paths between vertices, which means that the shortest path can be found in constant time. Initially, each driver is randomly located on one vertex of the road network. When the vehicle is serving rider requests, we assume it is following the schedule and moving constantly towards the destination.

5.2 Experiment Setup

5.2.1 Algorithms

We compared the results of our framework (**APART**) with two other approaches: **TREE** (i.e., Kinetic tree [8]) from academia and **NN** (i.e., Nearest Neighbor) from industry.

Our implementation of TREE is based on the algorithms in [8]. Since TREE [8] does not provide any pricing model, once a ride is completed, we compute its incurred detour and use Eqs. (1) to (3) to compute the platform's revenue. Also, to make the comparison fair, before assigning a request to a driver we perform profile matching to insure the provider does not end up loosing money. If the profiles were not compatible, we select the next driver with the shortest increase in travel distance.

The NN algorithm is implemented based on the current approach adopted by major ride-sharing platforms such as Uber. To the best of knowledge, these platforms find the first nearest driver to the pick-up location of a new request. If the driver is able to fit the new request in its schedule without violating any constraints, he accepts the request. Otherwise the request is rejected and the algorithm tries to assign the request to the next nearest driver. This continues until a driver accepts the request, or every driver rejects it in which case the request is dropped.

Parameter	Values
Max Wait Time (w)	3min, **6min**, 9min, 12min, 15min, 20min
# of Drivers	1000, 2000, **5000**, 10000, 20000
Max Passengers (n)	2, 3, **4**, 5, 6
Max Allowed Detour (ϵ)	25%, **50%**, 75%, 100%

Table 1: Parameters for Algorithm Comparison

In the first set of experiments, we use the pricing model explained in Section 3 and compare the three approaches. In Section 5.4 we utilize the pricing model introduced in [7] for both APART and TREE. Because there is no concept of *revenue* (similar to what we introduced in Section 3) in the pricing model of [7], in order to compare the generated revenue, we assume each driver has to pay 20% of their income as the platform's share.

5.2.2 Configurations and Measures

In our experiments we measure the following metrics by varying different parameters of the system: (1) service rate as the percentage of requests that were completed, (2) the revenue of the system and (3) the response time for matching a request with a driver. Table 1 shows the different values we used for various parameters to evaluate our framework (default values are shown in **bold**).

For the pricing model by default we configure it as:

$$\begin{aligned} F(d) &= 2 \times d \\ \forall r, f_r(\Delta d_r) &= 1 - (0.25 \times \Delta d_r^2) \\ \forall v, g_v(d) &= 1.5 \times d \end{aligned} \tag{5}$$

5.3 Algorithm Comparison

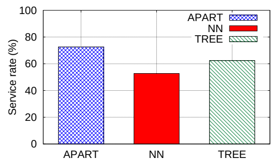
5.3.1 Overall comparison

In this section, we compare and analyze the performance of the three approaches using the default parameters in Table 1 with respect to service rate, generated revenue and response time.

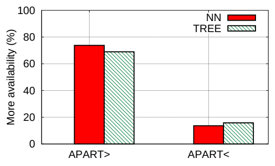
As shown in Fig. 5(a), APART is able to serve more requests (i.e. higher service rate) compared to the other two approaches. For all three approaches we use the same set of requests and drivers for each iteration, which means all approaches start with the same configuration in the road network. However, each approach assigns riders to drivers differently and hence, after a while the dynamism of the network (i.e., location of the drivers on the road network) will be different in each algorithms. To understand the reason for higher service rate, for each incoming request, we also count the number of eligible workers (Section 4.1). We say, the driver availability in algorithm $\mathcal{A}$ is higher than $\mathcal{B}$ with regard to request r, if during the simulation, more eligible drivers in algorithm $\mathcal{A}$ are available for r compared to that of algorithm $\mathcal{B}$. Fig. 5(b) shows the percentage of the requests for which APART has higher(/lower) driver availability compared to the other two approaches. The left two bars in Fig. 5(b) show that for more than 60% of the requests, APART has a higher driver availability compared to both NN and TREE. This means that APART engages drivers more effectively compared to NN and TREE and hence, is able to serve more requests.

Next, we compare the algorithms with regard to how much revenue they generate. Fig. 5(c) shows APART generates almost 20% and 50% more revenue compared to TREE and NN, respectively. In Section 1 we mentioned that the driver with minimum increase in his travel distance is not necessarily the most profitable driver. To verify this theory, when running TREE, for each incoming request (and after the algorithm chose the driver with least increase in traveled distance), we also check all eligible drivers to see which one generates the highest profit by serving the new request. We performed a similar check when running NN. Based on our observations, for 23% of the requests, the driver chosen by TREE is different from the most profitable driver. This number for NN is 70%. In Section 5.2.1 we explain, with implementation of TREE and NN the request is not assigned to a driver that cannot satisfy the monetary constraint. In other words, both approaches make sure by assigning a request to a driver, the platform provider does not loose money. If we relax this check for both approaches, in TREE, 5% of the requests are assigned to drivers that loose money. This number is 40% for NN.

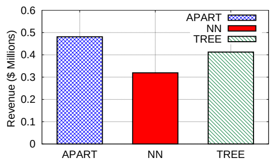
The final metric we compare, is the average response time for processing a single request. As shown in Fig. 5(d), with the default setting, the processing time in TREE is almost twice the response time in APART. The reason is that although TREE utilizes a "Kinetic Tree" data structure which maintains the current available schedules to expedite the scheduling process, the server has to perform scheduling for eligible drivers sequentially while APART



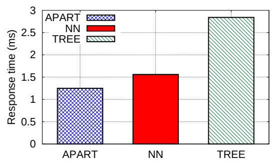
(a) Service Rate



(b) Driver Availability



(c) Revenue



(d) Response Time

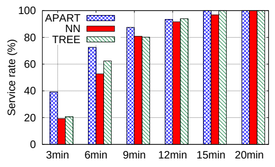
Figure 5: Comparing Algorithms with Default Values

distributes the scheduling to the drivers. Nevertheless, with the default settings, all three approaches process the requests under 3ms which is acceptable for a real-time framework.

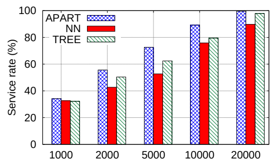
Following we vary different parameters based on Table 1 and evaluate the effect of each parameter on the same metrics.

5.3.2 *Service Rate*

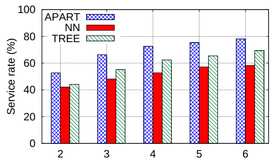
In this set of experiments we compare the service rate of the three approaches. As shown in Fig. 6, all algorithms generate high service rates when the constraints are relaxed or there is high resource availability. However, under tight constraints or limited resources, APART outperforms the other two approaches by up to 20%. In the previous section we showed how APART copes with the dynamism in the system better than the other two approaches.



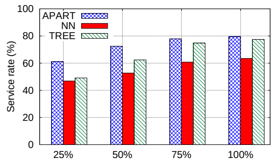
(a) Maximum Wait Time



(b) Number of Drivers



(c) Maximum Passengers



(d) Maximum Allowed Detour

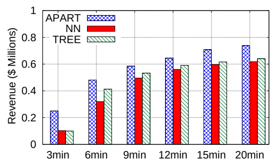
Figure 6: Comparing Service Rate of the Algorithms

5.3.3 *Revenue*

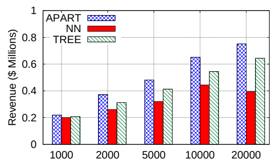
As mentioned, the main objective of APART is to maximize the ride-sharing platform's revenue. In this experiment, we compare the generated revenue of each algorithm. Towards that end, we apply the pricing model explained in Section 3. Here, we want to evaluate the effect of varying the parameters in Table 1 on the revenue and compare different algorithms. In Section 5.4 we apply different pricing models to the algorithms and compare revenue under different pricing models.

Fig. 7 shows that regardless of the values of different parameters, APART generates more revenue than any other approaches. When we compare the results in Fig. 7 with Fig. 6, even under configurations where all algorithms have the same service rate, APART manages to generate at least 10% more revenue. The main reason for higher revenue is that APART is designed to make a *price-aware* assignment, i.e., assign the request to a driver that generates the most profit. On the other hand, the TREE and NN algorithms were not designed to maximize revenue. As explained in Section 3, the pricing models that are used in APART are designed such that the higher profits are not gained by scamming the riders.

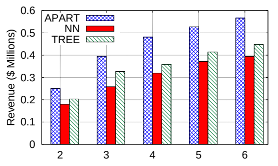
5.3.4 *Response Time*



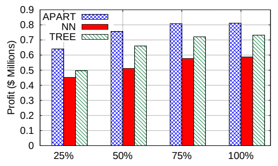
(a) Maximum Wait Time



(b) Number of Drivers



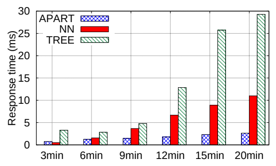
(c) Maximum Passengers



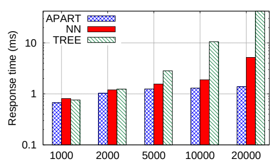
(d) Maximum Allowed Detour

Figure 7: Comparing Revenue of the Algorithms

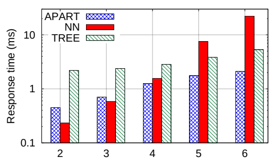
Similar to [6, 8], APART instantly processes a request once it is submitted. In order to evaluate the scalability of our framework, our next set of experiments evaluate the response time of processing a single request.



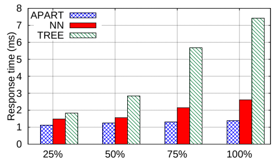
(a) Maximum Wait Time



(b) Number of Drivers



(c) Maximum Passengers



(d) Maximum Allowed Detour

Figure 8: Comparing Response Time of the Algorithms

Fig. 8(b) shows that when more drivers are added, the scalability of TREE suffers as it has to perform scheduling for a larger number of vehicles. On the other hand, due to the distributed nature of APART's auction-based approach, each driver does scheduling for itself and adding drivers does not affect the overall response time of APART as much. In Fig. 8(c) we observe that although APART's response time does not go beyond 5ms, TREE handles the increase in maximum passengers better due to the Kinetic Tree structure implementation [8]. The reason for NN's poor performance is that it has to perform scheduling computation sequentially, for possibly multiple drivers. Finally, in Fig. 8(a) and Fig. 8(d) we conclude that for relaxed constraints, the response time of TREE increases up to 4 times higher than that of APART. The main reason is that the Kinetic Tree structure keeps track of all valid orders of requests that are assigned to a driver. As we relax the constraints, the number of feasible permutations of the requests increases which makes the size of the Kinetic Tree larger and updates become more expensive. This in turn increases the response time. Fig. 8 shows unlike the other two approaches, APART's scalability does not suffer by varying different parameters of the framework.

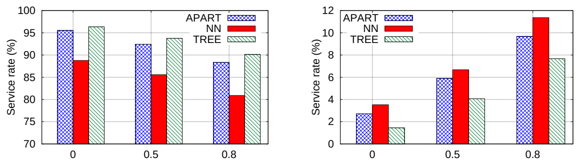
5.4 Comparing Different Pricing Models

In this section, we evaluate the effect of the pricing model. First we show the importance of designing a fair pricing model. We utilize the three approaches with the model in [6] and show how some riders may suffer by participating in ride-sharing. Subsequently, we perform some experiments utilizing the pricing model in [7] and show that as a result of price-aware assignments, regardless of the model, APART generates more revenue for the platform provider. Finally, we show the flexibility that profiles provide for the users.

Fig. 9 shows the result of utilizing the pricing model in [6]. Based on this pricing model, the driver's income is:

$$c.d_1 + (1+\alpha).c.d_2$$

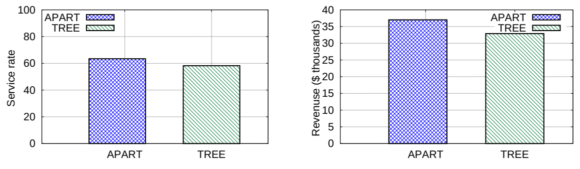
where d_1 is the distance the driver had only one rider on-board, d_2 is the total distance the driver had more than one rider on-board and c is some predefined constant. α takes a value between 0 and 1 which determines the increase in the driver's income for serving more than one rider. As we show in Fig. 9(a), by participating in ride-sharing, the majority of riders save money (pay less as compared to riding alone). Fig. 9(a) supports the claim in [6] that on average riders will save money. However, Fig. 9(b) shows that regardless of what algorithm is used, up to 10% of riders pay more by participating in ride-sharing which is not acceptable. The reason is that, riders have to pay even for detours. Even though riders split the fare on detours, if detours are sufficiently long, even carpooling riders loose money.



(a) Rider's that Saved Money (b) Rider's that Lost Money

Figure 9: Fairness of Pricing Models

In the next set of experiments, we apply the model in [7] and evaluate the performance of APART and TREE. In this model, riders get compensated for any detour incurred in their trip. The amount of compensation is based on the new rider's fare and the length of a rider's detour compared with the detour of other riders on the vehicle. Because the algorithm in [7] is similar to TREE, we only compared APART with TREE. Fig. 10(a) shows that APART provides a slightly higher service rate than TREE. However, due to assigning the riders to the most profitable drivers, APART ends up generating 10% more revenue.



(a) Service Rate (b) Revenue

Figure 10: Effect of Applying an Arbitrary Pricing Model

In Section 3, we mentioned by setting their profiles, users can configure APART to make assignments the way they find desirable. In the last set of our experiments, we use two different configurations to represent the riders' profiles. First, we set the riders' profiles to $f_T(\Delta d_r) = \frac{1}{(\Delta d_r+1)}$. Such profile is suitable for a rider who wants to minimize his detour and is willing to share a ride only if the detour is short. Since the rider sets **Tight** constraints we show this profile by f_T. In the second iteration, we set the profile of the riders to $f_R(\Delta d_r) = 1 - (\frac{\Delta d_r}{max\delta})$. This profile is more **Relaxed** (hence, f_R) and it is expected that more riders share a trip. Fig. 11 shows the result of utilizing APART and TREE with f_T and f_R. Since TREE does not make price-aware assignments, the results in both iterations were the same. However, as we observe with APART_T, almost 10% fewer riders ended up sharing a ride while on average they only observed 6-7% increase in their trips. On the other hand, with APART_R, almost every rider shares a ride and the average increase in their trip was almost 20%. An interesting observation in Fig. 11 is that with APART_R, more riders share a ride compared to TREE while their average detour was still less.

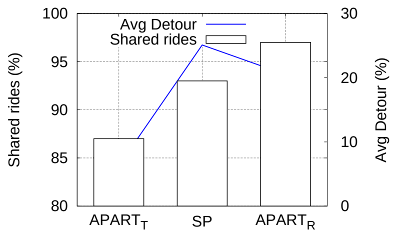


Figure 11: Effect of Profiles

In conclusion, APART is agnostic of the price model and is able to generate more profit. In addition, APART supports different types of riders' expectations by adjusting the profiles.

6. RELATED WORK

There are mainly two categories of ride-sharing, i.e., static and dyanmic ride-sharing. Most existing studies [15, 16, 17] focus on static ride-sharing, where all riders and drivers are known a priori and thus, trips are prearranged. Furuhata et al. [15] provoides a comprehensive suvery of the different types of ride-sharing regarding their formulations, optimizations and key computation challenges. Santi et al. [16] proposes a graph-based approach to quantify the potential of ride-sharing using New York's taxi data, and Cici et al. [17] evaluated the potential of carpooling using four cities' mobile dataset. In addition, ride-sharing problem can be treated as a special class of the dial-a-ride problem (DARP) [18], or dynamic vehicle routing problem (VRP) [9, 19] in operational research, which is proven to be NP-hard. All these studies assume that the riders' and drivers' statuses are know in advance, and hence can afford high computation cost, which is not the case in real-time ride-sharing.

With the emergence of many ridesahring mobile applications (e.g., Uber and Lyft), real-time ride-sharing [6, 7, 8, 2, 3, 4, 5] has recently attracted more research interest. Ma et al. [6, 7] proposed a ride-sharing dispatch system named "T-share" to serve the rider request on-the-fly with the objective of reducing drivers' total travel distance. Their work focuses on maintaining a spatial-temporal index to retrieve the candidate drivers. On the other hand, Huang [8] proposed a kinect tree scheduling algorithm to dynamically match trip request to drivers with minimum incurred travel distance. Ota et al. [2] introduced a data-driven simuation framework that enables the analysis of ride-sharing by using New York's taxi dataset. Santos et. al [20] propose a ride-sharing system to maximize the number of matched request. The majority of these studies aim to minimize the total travel distance of drivers, however, we show that this does not necessarily mean shorter travel distance for the riders. Compared with these work, we discuss the conflicting interest between riders, drivers and platform providers. We propose a general and versatile pricing model and our objective is to maximize the total profit of the platform provider. We show that by maximizing the overall profit, our framework achieves higher service rate and quality. Finally, we introduced a decentralized auction-based framework to support scalable and real-time scheduling, which differs from existing centralized scheduling framework.

Pricing mechanisms in realtime ride-sharing have also been studied in [21, 22, 23]. Their main focus is to adapt the well-known VCG mechanism [24] for truthful bidding, while addressing the specific challenges such as computational issues, incentive compatibility [21, 22] and deficit control [23]. These studies are orthogonal to our paper, and can be integrated into APART when we compute the bid of each candidate driver.

Our work is also related to the task assignment and scheduling problem in spatial crowdsourcing [25, 26, 27]. Spatial crowdsourcing is a platform, which enables a requester to comission workers to physically travel to some specified locations to perform a set of spatial tasks. For example, Kazemi et. al [25] formuated task assignment in spatial crowdsourcing as a min-cost max-flow problem, Deng et. al studied both task scheduling problem for one single worker [26] and multiple workers [27]. Unlike spatial crowdsourcing, in real-time ride-sharing, each rider request consists of both pickup and dropoff locations. In addition, their assignment and scheduling are processed in a batch fashion (e.g., batching tasks and workers every 10 minutes), whereas each rider request in real-time ride-sharing must be processed in a short amount of time.

7. CONCLUSION AND FUTURE WORK

In this paper, we studied the problem of real-time ride-sharing. Unlike existing studies, our objective is to maximize the revenue of the platform provider without compromising service quality, hence we introduced APART with a fair pricing model so that higher profits cannot be gained by scamming the riders. With APART, riders were able to set their monetary preferences through their profiles, and consequently the trips were generated according to our price-aware assignment mechanism. We validated our framework with a large scale New York City's taxi trips data. The experimental results demonstrated the effectiveness and efficiency of our framework in terms of both service rate and quality. Also, the distributed nature of APART allowed for real-time processing of requests.

In future, we plan to expand the framework by considering a time-dependent road network, where pre-computing the shortest paths between different nodes of the road network becomes impossible. Another interesting direction is to batch the requests arriving within a short period of time (e.g., 5 seconds).

Acknowledgement

This research has been funded in in part by NSF grants IIS-1320149 and CNS-1461963, METRANS Transportation Center under grants from Caltrans (DTRT12-G-UTC57), the USC Integrated Media Systems Center, and unrestricted cash gifts from Google, Northrop Grumman and Oracle. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any of the sponsors.

8. REFERENCES

[1] "Uberpool," https://newsroom.uber.com/us-california/its-a-beautiful-pool-day-in-the-neighborhood/.

[2] M. Ota, H. Vo, C. Silva, and J. Freire, "A scalable approach for data-driven taxi ride-sharing simulation," in *Big Data (Big Data), 2015 IEEE International Conference on*, Oct 2015, pp. 888–897.

[3] B. Cici, A. Markopoulou, and N. Laoutaris, "Designing an on-line ride-sharing system," in *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. GIS '15, 2015, pp. 60:1–60:4.

[4] B. Cao, L. Alarabi, M. F. Mokbel, and A. Basalamah, "Sharek: A scalable dynamic ride sharing system," in *2015 16th IEEE International Conference on Mobile Data Management*, vol. 1, June 2015, pp. 4–13.

[5] 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*, vol. 16, no. 5, pp. 2587–2598, 2015.

[6] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE, 2013, pp. 410–421.

[7] ——, "Real-time city-scale taxi ridesharing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 7, pp. 1782–1795, 2015.

[8] Y. Huang, F. Bastani, R. Jin, and X. S. Wang, "Large scale real-time ridesharing with service guarantee on road networks," *Proceedings of the VLDB Endowment*, vol. 7, no. 14, pp. 2017–2028, 2014.

[9] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Manage. Sci.*, vol. 6, no. 1, pp. 80–91, Oct. 1959.

[10] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Commun. ACM*, vol. 28, no. 2, pp. 202–208, Feb.

[11] M. Lagoudakis, M. Berhault, S. Koenig, P. Keskinocak, and A. Kleywegt, "Simple auctions with performance guarantees for multi-robot task allocation," in *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 1, Sept 2004, pp. 698–705 vol.1.

[12] A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani, "Adwords and generalized on-line matching," in *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, Oct 2005, pp. 264–273.

[13] U. Demiryurek, F. Banaei-Kashani, and C. Shahabi, "Efficient continuous nearest neighbor query in spatial networks using euclidean restriction," in *Proceedings of the 11th International Symposium on Advances in Spatial and Temporal Databases*, ser. SSTD '09, 2009, pp. 25–43.

[14] "Nyc taxi trips," http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.

[15] 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*, vol. 57, pp. 28–46, 2013.

[16] P. Santi, G. Resta, M. Szell, S. Sobolevsky, S. H. Strogatz, and C. Ratti, "Quantifying the benefits of vehicle pooling with shareability networks," *Proceedings of the National Academy of Sciences*, vol. 111, no. 37, pp. 13 290–13 294, 2014.

[17] B. Cici, A. Markopoulou, E. Frias-Martinez, and N. Laoutaris, "Assessing the potential of ride-sharing using mobile and social data: a tale of four cities," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 2014, pp. 201–211.

[18] J.-F. Cordeau and G. Laporte, "The dial-a-ride problem: models and algorithms," *Annals of Operations Research*, vol. 153, no. 1, pp. 29–46, 2007.

[19] Y. Li, D. Deng, U. Demiryurek, C. Shahabi, and S. Ravada, "Towards fast and accurate solutions to vehicle routing in a large-scale and dynamic environment," in *International Symposium on Spatial and Temporal Databases*. Springer, 2015, pp. 119–136.

[20] D. O. Santos and E. C. Xavier, "Dynamic taxi and ridesharing: A framework and heuristics for the optimization problem."

[21] E. Kamar and E. Horvitz, "Collaboration and shared plans in the open world: Studies of ridesharing," in *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, ser. IJCAI'09. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 187–194. [Online]. Available: http://dl.acm.org/citation.cfm?id=1661445.1661476

[22] A. Kleiner, B. Nebel, and V. A. Ziparo, "A mechanism for dynamic ride sharing based on parallel auctions," ser. IJCAI'11. AAAI Press, 2011, pp. 266–272.

[23] D. Zhao, D. Zhang, E. H. Gerding, Y. Sakurai, and M. Yokoo, "Incentives in ridesharing with deficit control," in *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems (AAMAS)*, 2014, pp. 1021–1028.

[24] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic Game Theory*. New York, NY, USA: Cambridge University Press, 2007.

[25] L. Kazemi and C. Shahabi, "Geocrowd: enabling query answering with spatial crowdsourcing," in *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*. ACM, 2012, pp. 189–198.

[26] D. Deng, C. Shahabi, and U. Demiryurek, "Maximizing the number of worker's self-selected tasks in spatial crowdsourcing," in *Proceedings of the 21st acm sigspatial international conference on advances in geographic information systems*. ACM, 2013, pp. 324–333.

[27] D. Deng, C. Shahabi, and L. Zhu, "Task matching and scheduling for multiple workers in spatial crowdsourcing," in *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2015, p. 21.