Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)

Joint Pricing and Matching for City-Scale Ride-Pooling

Sanket Shah[1], **Meghna Lowalekar**[2], **Pradeep Varakantham**[2]

[1] Harvard University, Cambridge, MA
[2] Singapore Management University, Singapore
sanketshah@g.harvard.edu, meghna.lowalekar@gmail.com, pradeepv@smu.edu.sg

Abstract

Central to efficient ride-pooling are two challenges: (1) how to 'price' customers' requests for rides, and (2) if the customer agrees to that price, how to best 'match' these requests to drivers. While both of them are interdependent, each challenge's individual complexity has meant that, historically, they have been decoupled and studied individually.

This paper creates a framework for batched pricing and matching in which pricing is seen as a meta-level optimization over different possible matching decisions. Our key contributions are in developing a variant of the revenue-maximizing auction corresponding to the meta-level optimization problem, and then providing a scalable mechanism for computing posted prices. We test our algorithm on real-world data at city-scale and show that our algorithm reliably matches demand to supply across a range of parameters.

1 Introduction

On-demand Ride-pooling allows the ride-pool operator (e.g. Uber) to serve more customers with the same number of vehicles and, hence can increase the earnings for drivers while decreasing the costs for riders. Because of these benefits, ride-pooling services have become increasingly popular in recent years, with more than 20% of all Uber trips coming from their UberPool offering (Heath 2016).

Central to the efficiency of ride-pooling are two sets of choices that an operator has to make – *(1) Pricing:* When a customer requests a ride, what price should be quoted?, and *(2) Matching:* If the customer accepts the quoted price, to what vehicle should they be matched? These choices can be thought of as 'levers' that the operator can use to maximise their revenue (or some other metric) subject to constraints on the quality of service (e.g. customer 'wait time' for) and reliability of service (e.g., the ratio of the number of requests for which customers have accepted the quoted price to the number of customers that are actually served).

It is important to note here that these two problems are interdependent – the price that is quoted influences who accepts the quote and, as a result, which people can be matched. Given this dependence, it is vital that these two metaphorical levers be adjusted together to create the optimal outcome. However, the individual matching or pricing problems are hard to solve even independently[1], and the need for joint optimization makes the problem more challenging. Therefore, even though the importance of doing both pricing and matching together has been acknowledged (Özkan 2020), most existing work focuses on optimizing along only one dimension (Banerjee et al. (2015), Bimpikis et al. (2019), Lowalekar et al. (2019), Shah et al. (2020)).

Existing research on jointly optimizing pricing and matching decisions is either not scalable (Chen et al. 2019b) or rely on an unrealistic assumption of knowing all the (current and future) customer requests in advance (Ma, Fang, and Parkes 2019) for taxi-on-demand (vehicles having unit-capacity) problem. In this paper, we consider the challenging problem of jointly optimizing the pricing and matching decisions for multi-capacity vehicles in a scalable manner without making any unrealistic assumptions.

Contributions:

1. A novel framework for joint pricing and matching for city-scale ride-pooling by considering pricing as a meta-level optimization over matching decisions. (Sec 2).
2. A variant of the revenue maximizing auction (Myerson 1981) to represent the meta-level optimization problem (Sec 4.1) for pricing.
3. A principled and scalable algorithm for computing posted prices using the revenue maximizing auction (Sec 4.2).
4. An open-source simulator for joint pricing and matching based on real-world data (Sec 5.1).
5. Demonstration of the algorithm efficacy on the above simulator using real-world data from New York Yellow Taxi (Sec 5.3). Our proposed approach not only provides high revenue for ride-pooling companies, it also increases customer satisfaction by finding prices such that nearly all the accepted requests can be matched to available vehicles.
6. An analysis to show that there is significant scope for additional gains from algorithmic improvement (Sec 6).

2 Problem Description

We are given a set of vehicles V, each of which can potentially serve multiple customers simultaneously (e.g., UberPool). In parallel, the operator receives a stream of requests

[1] **Matching** is challenging as vehicles have to be matched to groups of requests. **Pricing** is challenging as revenue maximization is constrained by requests and price consciousness of customers.

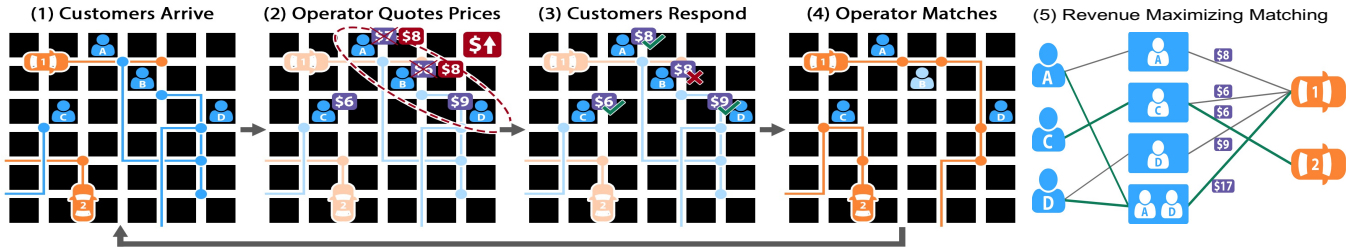

Figure 1: Schematic with Example. We begin in (1) with two vehicles that comprise V (in orange), each of which has existing customers that they must drop off at the positions marked by the orange dots. We also have four requests that comprise R (in blue), each of which has an origin and a destination. Then, in (2), the operator has to quote prices μ for each of the requests. Assume that Vehicle 1 has capacity of 2 and, as a result, cannot serve requests A, B, and D. Consequently, the operator raises the prices such that only two out of the three accept, in expectation. In (3), the prices are shown to customers who decide whether to proceed at these prices. Finally, in (4), the operator matches the requests R_μ of the customers who have accepted the prices to vehicles such that the utility to the operator is maximised. The revenue maximizing matching is shown in (5).

from customers which are batched together over a fixed interval (e.g., 60 seconds) and the resultant set of requests is given by R. Given this set of vehicles and requests, the consequent sequence of events is described in Figure 1. The central challenge of pricing is to price such that an objective (e.g., revenue) is maximised (in expectation) in the consequent matching. We first introduce relevant symbols and then formally define the 'matching' and 'pricing' problems.

2.1 Relevant Symbols

- R The set of customers' *requests* for rides.
- V The set of *vehicles* that can serve these requests.
- μ The set of *prices* quoted by the operator for the requests in R. The price for a request $r \in R$ is denoted by μ_r.
- p The price-sensitivity function maps the set of prices μ to the probability of acceptance $p(\mu)$. It is assumed to be known apriori. For a given request $r \in R$, $p_r(\mu_r)$ is the probability of acceptance of the customer at the price μ_r.
- R_μ The set of requests for which the customers accepted the prices quoted by the operator.
- t A set of requests collectively known as a 'trip'. This is the fundamental unit of matching and is described in more detail in Sec 2.2. The set of all possible trips is denoted by T, the subset of trips that contain the request r is given by T_r, the subset of requests that correspond to a trip t is given by R_t, the set of vehicles that can serve trip t is given by V_t and the subset of prices for the requests in t is given by μ_t.
- X A set of indicator variables that denotes a matching solution. $x_v^t \in X$ denotes whether or not the trip t has been assigned to the vehicle v.
- C A function that maps the set of accepted requests R_μ to the set of feasible matchings, determined by the constraints on the matching requests to vehicles.
- O The objective to be maximised for the matching. It is defined as the sum over the individual values of the objective o_v^t for each trip t assigned to vehicle v in the final matching (see Sec 2.2). Each o_v^t value is, in turn, dependent on the prices of each of the requests $r \in t$.

2.2 Matching

Input: The set of vehicles V, the set of requests for which the customers accepted the prices R_μ, the set of prices μ, the function C that generates feasible matchings based on the constraints, and the objective function O.

Matching in a Tripartite Graph: The aim of matching is to create a mapping from $R_\mu \mapsto V$ such that the objective O is maximised subject to some constraints on matchings. This problem can be seen as a 'maximum weight bipartite matching' in the unshared case but, bipartite matching doesn't capture the structure of the matching problem when multiple requests can be matched to a single vehicle. For example, consider the two requests A and C in Figure 1. While Vehicle 1 can serve the two requests individually, it cannot serve them together as it would lead to an unacceptable amount of detour; these kinds of combinatorial constraints cannot be captured in a bipartite graph. The way to solve this problem is to create an intermediate representation called a 'trip' which is a combination of requests (Alonso-Mora et al. 2017). Then, requests are mapped to vehicles via trips such that both vehicles and requests can be assigned at most one trip. Each trip and vehicle mapping has an associated weight of o_v^t, and the solution to the matching problem can be seen as a maximum weight matching in this resulting tripartite graph – (5) in Figure 1. We write this optimization using the equation:

$$X^* = \arg\max_{X \in C(R_\mu)} \sum_{x_v^t \in X} x_v^t \cdot o_v^t(\mu_t) \tag{1}$$

Solution Method: The matching is performed by solving an Integer Linear Program given below. While there is a combinatorially large number of trips in the worst case, Alonso-Mora et al. (2017) suggests an approximation method for generating feasible trips that works well in practice.

$$\begin{aligned} \max \quad & \sum_{x_v^t} x_v^t * o_v^t \\ s.t. \quad & \sum_{v \in V} x_v^t = 1 && ::: \forall t \in T \\ & \sum_{t \in T_r} \sum_{v \in V_t} x_v^t \leq 1 && ::: \forall r \in R \end{aligned}$$

	Matching	Pricing	Future Demand and Price-Sensitivity	Capacity of Vehicles	Request Processing
(Banerjee, Riquelme, and Johari 2015) (Banerjee, Johari, and Riquelme 2016) (Bimpikis, Candogan, and Saban 2019)	Not optimized	optimized	Known Distribution	Unit Capacity	Region based Pricing Sequential (Matching)
(Ma, Zheng, and Wolfson 2013) (Zheng, Chen, and Ye 2018)	optimized	Not optimized	No Information	Multi-Capacity	Batched (Matching)
(Chen et al. 2019a)	optimized	optimized	Known Distribution	Unit Capacity	Batched(Matching) Sequential (Pricing)
(Ma, Fang, and Parkes 2019)	optimized	optimized	Exact Information	Unit Capacity	Batched
Our Work	optimized	optimized	Known Distribution	Multi-Capacity	Batched

Table 1: Summary of Differences Between Our Work and Related Work

Matching Objective: In our formulation, we abstract away the objective in terms of o_v^t, but concretely it represents the system-level objective that the operator wants to maximise. This could be an obvious goal like profit, revenue, or the number of trips served, but may also be a way to incorporate future information (Shah, Lowalekar, and Varakantham 2020) or even fairness metrics (Lesmana, Zhang, and Bei 2019). All these objectives can be modelled as a linear function of the revenue of the trip[2]

- *Profit:* $o_v^t(\mu_t) = [\sum_{r \in R_t} \mu_r] - cost_v^t$, where $cost_v^t$ is a constant denotes the marginal increase in cost incurred by serving a trip t with vehicle v.
- *Number of requests:* $o_v^t(\mu_t) = 0 \cdot [\sum_{r \in R_t} \mu_r] + |R_t|$, where you ignore the revenue entirely.
- *NeurADP (Shah, Lowalekar, and Varakantham 2020):* $o_v^t(\mu_t) = [\sum_{r \in R_t} \mu_r] + \gamma \cdot \mathbb{V}_v^t$, where the second term $\gamma \cdot \mathbb{V}_v^t$ is a learned constant. While we look at determining the optimal pricing and matching for a given batching interval in this paper, NeurADP attempts to match riders to drivers such that it is optimal across multiple batching intervals. Their solution takes the form of adding a 'future value' $\gamma \cdot \mathbb{V}_v^t$ to the matching objective in every batching interval and fits neatly into our generalisation.
- *Historical Earnings (Lesmana, Zhang, and Bei 2019):* $o_v^t(\mu_t) = [\sum_{r \in R_t} \mu_r] + hist_v$, where the second term is a constant that denotes the historical earnings for a given driver. In the paper, they even out driver earnings by adding the driver's historical earnings to the objective.

As a result, we focus on maximising a set of general objectives that can be written as a linear function of a trip's revenue. This is concretely given by the following equation, where α_v^t and β_v^t are constants:

$$o_v^t(\mu_t) = \alpha_v^t \sum\nolimits_{r \in R_t} \mu_r + \beta_v^t \tag{2}$$

The overall objective is combinatorial in the requests $r \in R_\mu$ because each o_v^t component depends on the source and destination of all the requests in trip $t \subset R_\mu$, as well as the existing trajectory of vehicle v.

2.3 Pricing

Input: The set of vehicles V, the set of requests R, the price-sensitivity function p, a function that generates the set of feasible matchings based on constraint in C, and the objective O. We assume that the price-sensitivity function p is known apriori. This is a reasonable assumption given the amount of data available with ride-pool operators (Korolko et al. 2018).
Objective: The aim of pricing is to ensure that the matching it induces maximises the objective in expectation. In this sense, pricing can be seen as a meta-level optimization over matching. Concretely, it can be written as:

$$\mu^* = \arg\max_{\mu} \mathop{\mathbb{E}}_{R_\mu \sim p(\mu)} \Big[\max_{X \in C(R_\mu)} \sum_{x_v^t \in X} x_v^t \cdot o_v^t(\mu_t) \Big] \tag{3}$$

Challenges: This formulation adds an additional layer of complexity over an already complex matching problem. In fact, even calculating the expected value of a fixed set of prices μ naively is exponential in the number of requests because it requires considering all the different possible combinations of people accepting or rejecting the prices. In Sec 4.2, we propose a nice way to (indirectly) solve this problem by showing a connection between this formulation of pricing and the literature in mechanism design.

3 Related Work

The existing work for optimizing ridesharing systems can be categorized based on different dimensions, as shown in Table 1. These dimensions include how the pricing and matching decisions are made, the capacity of the vehicles considered, sequential (one by one) or batched (considering all active requests together) processing of requests and the amount of future information available to the algorithm.

As seen in the table, most of the existing work for ridesharing systems has studied the decision-making in isolation, i.e., they either perform optimized matching by assuming a heuristic or fixed pricing (Ma, Zheng, and Wolfson 2013; Zheng, Chen, and Ye 2018), or focus on pricing decisions and ignore the optimization for matching (Banerjee, Riquelme, and Johari 2015; Bimpikis, Candogan, and Saban 2019). Most existing work also considers a sequential processing of requests, for at least one of the decision-making components. The sequential solution is faster to compute but is typically of poorer quality than the batched solution, given that it has less information with which to make a decision (Uber 2018).

Banerjee et al. (2015; 2016) provide a threshold-based pricing mechanism where the platform raises the price whenever the available vehicles in the region fall below a

[2]Details in appendix at https://tinyurl.com/52y8446c.

threshold. Their focus is on studying the theoretical properties of such threshold-based policies. Bimpikis, Candogan, and Saban (2019) provide a spatial pricing scheme for ridesharing markets where they set a price for each region. As opposed to our work, these works do not focus on optimizing matching decisions, and heuristically match the customer with unit-capacity vehicles available in the region. While these region-based simplifications may work well in the unit-capacity case, they cannot be approximated in the same way in the multi-capacity case.

Özkan (2020) theoretically prove that optimizing only along one dimension is not optimal in general and joint pricing and matching optimization can significantly increase performance. Unlike this paper, they do not provide an explicit algorithm, but rather focus on deriving conditions in which individually optimizing pricing or matching can help.

There has been some research on providing algorithms for jointly optimizing pricing and matching decisions for these systems. Ma, Fang, and Parkes (2019) provide a spatio-temporal pricing mechanism which provides competitive equilibrium prices and can achieve a higher revenue than a myopic pricing scheme. They make an unrealistic assumption that all the information about future requests is exactly known. On the other hand, the only assumption made about requests in this paper is that we know the price-sensitivity of the customers making the request.

Chen et al. (2019a) provide a learning framework for jointly optimizing pricing and matching decisions. They use a contextual bandit algorithm for pricing each request *separately* and use temporal difference learning to optimize matching decisions for unit capacity vehicles. However, as Figure 1 shows, pricing in the multi-capacity setting requires considering not only how compatible different requests are with each other, but also how compatible they are to the trajectory of the vehicle they're being assigned to. In contrast, this paper prices all the requests in a batch jointly.

Chen et al. (2019b) model the problem as a Markov Decision Process (MDP) with the vehicle distributions on each node as states, the price and dispatch along each edge as actions, and the revenue as the immediate reward. But the formulation does not scale to a large number of locations and time intervals, which makes it unsuitable for use in the real-world ridesharing systems that are the focus of this paper.

4 Approach to Pricing

In this section, we detail our overall approach to pricing in our proposed framework. In Sec 4.1, we relate Equation 3 to auctionsand present an auction formulation for ride-pooling. In Sec 4.2, we use the resulting auction to (indirectly) solve the concrete pricing problem posed by Equation 3.

4.1 Ride-Pooling as an Auction

An auction typically consists of a seller, a set of buyers, a set of items to be sold, and an objective that needs to be maximised. In ride-pooling, the buyers are the customers who want rides, and the seller is the ride-pool operator. Motivated by real-world ride-pool offerings, in which customers have no control over the other customers or driver they're matched to, we consider that a single item is being sold – the 'service of being transported from source to destination'. This assumption makes ours a 'single-parameter' auction, in the language of mechanism design. Finally, the objective that we want to maximise here is the seller's *'utility'* – a combinatorial function of the revenue, as described in Equation 2. Given that we're maximising the seller's *utility* here, we dub this the *Utility-Maximising Auction*.

An auction is then characterised by two rules: *(1) Allocation Rule* – given a set of sealed bids $\mathbf{b}$ from the customers, who gets the item?, and *(2) Payment Rule* – how much do the winners pay for the item. Then, to define the ride-pooling auction, we must come up with both sets of rules. In this paper, we limit our search for the 'right' auction to the space of Dominant Strategy Incentive Compatible (DSIC) auctions *without any loss of generality* (Gibbard 1973). DSIC auctions are those in which the buyers are incentivised to bid their true values, i.e., the set of bids $\mathbf{b}$ is equal to the set of 'intrinsic values' $\boldsymbol{\nu}$ that we discuss a below.

p and Intrinsic Values: The insight that allows us to connect ride-pooling to auctions is that price-sensitivity function p_r is decreasing in the price μ_r and hence can be interpreted in terms of the 'intrinsic value' ν_r that a customer has for a ride. In an auction, a buyer has a fixed intrinsic value for a given item, and they buy the item if and only if the price at which the item is being offered is lower than this value. While this exact value is unknown to the seller, the seller knows the distribution f_r from which ν_r is drawn. As a result, the probability of a buyer accepting a request at some price μ_r, is equal to the probability that the buyer's intrinsic value is higher than μ_r. This can be written down in terms of the CDF F_r of the intrinsic value distribution as follows:

$$p_r(\mu_r) = 1 - F_r(\mu_r) \tag{4}$$

Auction Objective: While there are similarities between an auction and the pricing problem in Equation 3, there is one big difference – the intrinsic values are provided as input to the auction (in the form of sealed bids). In contrast, in our pricing problem, it is assumed that this information is not available. Because the auction has additional information in terms of the 'sealed bids', the *utility* generated by the auction is an upper-bound on the value of Equation 3. Given oracle access to the intrinsic values, the joint pricing and matching problem reduces to just the matching problem. Then, the aim would be to maximise the matching objective for the case in which the price for each customer is substituted by the 'payment' from the 'payment rule' ($\mu_r(\boldsymbol{\nu})$) of the auction. Formally, we are trying to maximise:

$$\mathop{\mathbb{E}}_{\boldsymbol{\nu}\sim\mathbf{f}}\Big[\max_{X\in C}\sum_{x_v^t\in X} x_v^t\cdot(\alpha_v^t\sum\nolimits_{r\in R_t}\mu_r(\boldsymbol{\nu})+\beta_v^t)\Big] \tag{5}$$

Allocation Rule: We want to come up with an allocation rule that will maximise the expression in Equation 5. However, it's not clear from this equation how to perform the maximisation because the payment $\mu_r(\boldsymbol{\nu})$ is a black box. To solve this, we improve on the result from Myerson (1981) that relates the allocation rule described in Equation 5 and the payment rule that determines $\mu_r(\boldsymbol{\nu})$ in single-parameter auctions. The result is summarised below.

Proposition 1 (Utility Maximising Auction) *We can maximise the expression in Eq 5 by allocating according to:*

$$\max_{X \in C} \sum_{x_v^t \in X} x_v^t \cdot (\alpha_v^t \sum_{r \in R_t} \varphi_r(\nu_r) + \beta_v^t) \tag{6}$$

where $\varphi_r(\nu_r) = \nu_r - \frac{1-F_r(\nu_r)}{f_r(\nu_r)}$ *denotes the 'virtual value' for a request.*

Proof Sketch. The eight step proof[3] for utility maximization builds on the proof for revenue maximization from Myerson (1981), which shows that that maximizing expected revenue is equal to maximizing virtual welfare. The major difference is in the form of the final expression. While revenue maximising auctions aim to maximise $\mathbb{E}_{\boldsymbol{\nu}}[\sum_{r \in R} \mu_r(\boldsymbol{\nu})]$ that only includes pricing variables, we have a complicated expression (Equation 5) that includes matching/allocation variables in addition to pricing variables. The main trick is in simplifying the utility maximization expression (in step 3 of the proof in supplementary) using the following property of the matching algorithm:

For a fixed set of intrinsic values for the other buyers $\boldsymbol{\nu}_{\text{-}r}$*, and all possible intrinsic values* $\nu_r \in [0, \nu^{max}]$ *of buyer* r*, the buyer is assigned to a maximum of one trip-vehicle combination (represented by the indicator variable* x_v^t*).*

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\nu_r \sim f_r} [\sum_{x_v^t} x_v^t(\boldsymbol{\nu})[\alpha_v^t \cdot \mu_r(\boldsymbol{\nu})]] \\
&= \int_0^{\nu^{\max}} \mu_r(\boldsymbol{\nu})[\textstyle\sum_{x_v^t} x_v^t(\boldsymbol{\nu})\alpha_v^t] f_r(\nu_r) d\nu_r \\
&= \int_0^{\nu^{\max}} \left[\int_0^{\nu_r} z \cdot \textstyle\sum_{x_v^t} x_v^{t\,\prime}(z, \boldsymbol{\nu}_{\text{-}r}) dz\right] \cdot \\
&\qquad \cdot \left[\textstyle\sum_{x_v^t} x_v^t(\nu_r, \boldsymbol{\nu}_{\text{-}r})\alpha_v^t\right] f_r(\nu_r) d\nu_r \qquad (7) \\
&= \int_0^{\nu^{\max}} \int_0^{\nu_r} z \cdot [\textstyle\sum_{x_v^t} x_v^{t\,\prime}(z, \boldsymbol{\nu}_{\text{-}r})\alpha_v^t]\, dz \cdot f_r(\nu_r)\, d\nu_r \quad \blacksquare
\end{aligned}$$

We then use allocation rule from Eq 6 in the next section to solve Eq 3. Note here that the resultant allocation rule is identical to the matching problem from Equation 2 with the prices μ_r swapped for the virtual values φ_r. As a result, to determine the winners of the auction, we run our matching algorithm with the prices substituted by the virtual values.

4.2 Generating Posted Prices for Ride-Pooling

While auctions do a better job of maximising the utility, they also suffer from serious usability issues – eliciting exact values from customers is often difficult, the description of the auction can be complex, and the results of the auctions are often not interpretable because items may sometimes be awarded to buyers with lower bids than alternate candidates. Additionally, none of the major ride-pool operators use this mechanism. So , in this section, we describe an algorithm for (indirectly) solving the pricing problem posed in Equation 3.

Specifically, we use Algorithm 1 that has been previously described in Chawla et al. (2010). Given that the auction upper bounds the revenue of the posted price mechanism, this

[3]Detailed in appendix at https://tinyurl.com/52y8446c

Algorithm 1: Generate Posted Prices

1: **Input:** Number of samples s
2: Initialise a counter N_r for each request
3: **for** $i \in [1, \ldots, \text{s}]$ **do**
4: Sample a set of intrinsic values $\boldsymbol{\nu} \sim \mathbf{F}$
5: Run the optimal auction from Sec 4.1 with the sampled intrinsic values $\boldsymbol{\nu}$
6: **for all** $r \in R$ **do**
7: Increment N_r if r is assigned a vehicle in the auction.
8: **for all** $r \in R$ **do**
9: Get the probability $prob_r = \frac{N_r}{s}$ of r being served
10: Smooth out this probability by clipping it to the range [MIN_PROB, MAX_PROB]
11: Generate the price: $\mu_r = F_r^{-1}(1 - prob_r)$

algorithm attempts to match the behaviour of the auction. It does this in 2 steps – first, it estimates the probability with which a request r is assigned a vehicle in the auction. Next, it charges a price such that the request in the posted price case will be accepted with the same probability as in the auction.

The intuition here is that, given that the probability of being allocated a ride is correlated with the customer's intrinsic value, the auction helps determine the threshold value above which it makes sense to assign a vehicle to them. Suppose an item is assigned to a given request r with probability $prob_r$, then we can determine that their intrinsic valuation $\nu_r \geq F_r^{-1}(1 - prob_r)$ in a $prob_r$ fraction of the cases (where F_r^{-1} is the inverse-CDF of the intrinsic values associated with request r). The algorithm then charges them a price equal to that threshold valuation.

Theoretical Guarantees: This algorithm is motivated strongly by the fact that it lends itself well to theoretical analysis. Chawla et al. (2010) show that for a revenue-maximising objective, in the unshared case (UberX-type rides), the algorithm achieves at least a third of the revenue of the auction in the worst-case. However, while the algorithm has desirable worst-case guarantees, it is sub-optimal even in simple settings as we show in Sec 6. This suggests that there is great scope for improvement in this area.

Additional Heuristic: In Line 10 of Algorithm 1, we 'smooth' the probability by clipping it to range between MIN_PROB and MAX_PROB. Given a direct relationship between probability and price, this is equivalent to setting a minimum and maximum price for every request. We do this for 2 reasons – (1) to mitigate the error from sampling, and (2) to ensure that the price is in an acceptable range.

5 Experiments

In this section, we empirically quantify the effectiveness of our approach.

5.1 Simulator

We build on the basic matching simulator provided in Shah, Lowalekar, and Varakantham (2020), which is based on the New York Yellow Taxi dataset (NYYellowTaxi 2016) with

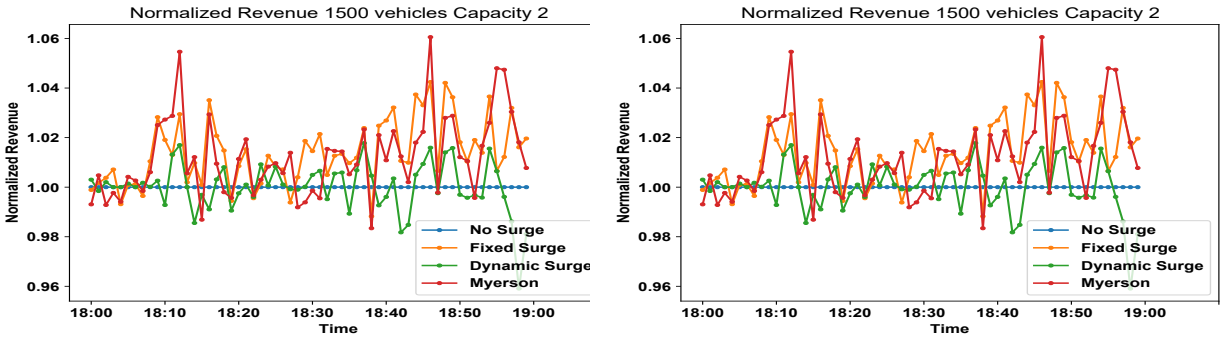

Figure 2: Time vs Revenue for different price sensitivities (Uber and Price Conscious)

over 300,000 requests on a week day. We then use a model of price sensitivity learned on Uber data that can be found in Korolko et al. (2018). Here, the probability of acceptance is a function of a multiplicative surge on top of the base price.

$$p_r(\mu_r) = \frac{1}{1 + e^{\frac{0.67\mu_r}{BP_r} - 1.69}} \tag{8}$$

Here BP_r corresponds to the base price for a given request r. We then combine this with the cost structure of UberX in Manhattan to obtain the final price-sensitivity function.

5.2 Experiment Set-Up

Matching Algorithm: In the experiments, we maximise the *revenue* subject to constraints: (a) *Maximum Delay* – the time taken by a pooled ride should not exceed the time taken by an unshared ride by more than 10 minute; (b) *Maximum Wait Time* – it should not take more than 5 minutes for a vehicle to reach a customer; and (c) *Capacity* – A vehicle can't have more customers in it than its maximum occupancy.

Baselines: We compare against 3 pricing baselines: (1) *No Surge*: We employ 'monopoly price' for a given price-sensitivity function p. For a single seller, it never makes sense to offer an item at a price lower than this; (2) *Fixed Surge*: We run the 'No Surge' baseline for the given run and find the ratio of incoming requests to served requests. Given this, and the price-sensitivity function, we find the price at which the number of accepted requests would equal the number of served requests and charge this surge for the entire run; and (3) *Dynamic Surge*: We update the surge at a given timestep based on the ratio of the number of served to accepted requests. If this number is lower than a fixed threshold (0.9), we increase the surge by a constant value (0.01), otherwise we decrease it by that value.

Our approach (Algorithm 1) is referred to as ***Myerson***. The matching algorithm is the same for all approaches.

Controlling Variance: The main source of variance in our experiments is whether or not customers accept the requests. To address this, we set the random seed value such that the intrinsic values of customers is the same across different pricing strategies. To make sure that this didn't induce a bias in the results, we repeated the experiments for different sets of seed values and found a standard deviation of less than 0.5% in the total revenue.

Price-Sensitivity Function: We find that the surge values produced while using the price-sensitivity function from Sec 5.1 (referred to as 'Uber') are quite high. To make customers more sensitive to changes in the price, we propose two variants of this function by scaling the coefficient of μ_r and the constant term in Equation 8 both by a factor of 10 ('Price Conscious') and 100 ('Very Price Conscious') which makes the transition from 'accept' to 'not accept' more steep.

Computation: We run the experiments on a 'g4dn.2xlarge' AWS instance that has a 2^{nd} Generation Intel Xeon Scalable (Cascade Lake) processor, 64GB memory and 256GB SSD drive. To simulate real settings, all experiments take less than 60 seconds for a 60-second decision interval.

5.3 Results

In this section, we compare the performance of our approach with the baseline algorithms along different dimensions. All the experiments are run on multiple time slots: morning peak hour from 8:00 AM - 9:00 AM, low demand in afternoon from 2:00 PM - 3:00 PM, before peak hour from 4:00 PM - 5:00 PM and evening peak hour from 6:00 PM - 7:00 PM. The results are averaged over 5 runs and were repeated over multiple weekdays. The results are similar and we show results from evening peak hours here and the rest 3 time slots are provided in the appendix. Unless otherwise mentioned, we consider the 'default' setting of 1500 vehicles with capacity 2 and the 'Price Conscious' price-sensitivity function.

Impact on Revenue: We first compare the revenue obtained in each timestep. We normalize the revenue with respect to the revenue obtained by the 'No Surge' baseline at each timestep. From Figure 2, we see that 'Myerson' obtains a better revenue than the 'No Surge' baseline in most of the timesteps. In addition, 'Dynamic Surge', which modifies the price charged based on past request patterns, has the largest fluctuations. Overall, 'Myerson' obtains a 2.28% improvement in revenue over the 'No Surge' baseline, while the other baseline has a lower revenue than 'No Surge'. This may not seem significant at first glance but we'd like to highlight here that a 2% increase in revenue *with no associated increase in cost* is huge for a multi-billion dollar industry.

Impact on Reliability: While it is important to optimize the revenue, it is also important to provide a reliability to customers. We define reliability as the probability of being assigned a ride conditioned on the customer accepting

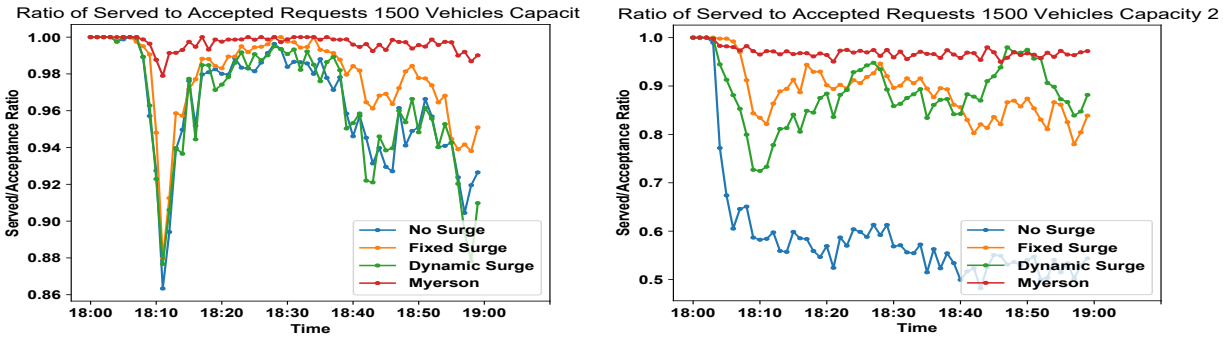

Figure 3: Time vs Served/Accepted Ratio for different price sensitivities (Uber and Price Conscious)

	Variants	No Surge	Fixed Surge	Dynamic Surge	Myerson
Vehicles	2000	1.0000 ± 0.0000	1.0059 ± 0.0032	0.9936 ± 0.0044	$\mathbf{1.0140 \pm 0.0022}$
	1500	1.0000 ± 0.0000	0.9894 ± 0.0044	0.9752 ± 0.0025	$\mathbf{1.0223 \pm 0.0029}$
	1000	1.0000 ± 0.0000	0.9829 ± 0.0035	0.9480 ± 0.0064	$\mathbf{1.0306 \pm 0.0023}$
Capacity	4	1.0000 ± 0.0000	0.9853 ± 0.0031	0.9461 ± 0.0102	$\mathbf{1.0090 \pm 0.0018}$
	2	1.0000 ± 0.0000	0.9894 ± 0.0044	0.9752 ± 0.0025	$\mathbf{1.0223 \pm 0.0029}$
	1	1.0000 ± 0.0000	1.0295 ± 0.0061	1.0349 ± 0.0124	$\mathbf{1.0499 \pm 0.0024}$
Price-Sensitivity	Uber	1.0000 ± 0.0000	$\mathbf{1.0138 \pm 0.0025}$	1.0001 ± 0.0007	1.0118 ± 0.0033
	Price Conscious	1.0000 ± 0.0000	0.9894 ± 0.0044	0.9752 ± 0.0025	$\mathbf{1.0223 \pm 0.0029}$
	Very Price Conscious	1.0000 ± 0.0000	0.8634 ± 0.0037	0.8559 ± 0.0126	$\mathbf{1.0025 \pm 0.0033}$

Table 2: Results of the Ablation Studies: We present the normalised revenue for different hyper-parameter values.

the quote provided by the ride sharing operator. Therefore, to measure the reliability, we compare the ratio of number of served requests to number of accepted requests. Figure 3 highlights the high reliability that 'Myerson' provides to customers, with the ratio remaining close to 1 for all timesteps. On the other hand, No Surge over-promises and is not able to serve most of the accepted requests, resulting in customer dissatisfaction. Even though the surge baselines, especially 'Dynamic Surge', attempt to address this imbalance through pricing, they suffer from fluctuations.

The ineffectiveness of surge baselines here highlight that there is more to effective pricing in ride-pooling than just accounting for temporal variation in demand. Considering the matching component allows 'Myerson' to set a price such that most of the accepted customers can be matched. These results highlight the importance of jointly optimizing pricing and matching in ride-pooling.

Ablation Studies: Finally, we check the robustness of our results by varying different parameters. We vary each parameter while keeping the others same as in our 'default' setting. As shown in Table 2, 'Myerson' earns similar or better revenue than baselines. For a higher capacity (4) and a higher number of vehicles (2000), the gain obtained by our algorithm reduces. When there are enough vehicles available, there is no advantage to increasing the price and reducing the chances of a customer accepting the requests. Therefore, the prices obtained will be close to the no surge case.

6 Analysis of Our Posted-Pricing Approach

While we show that our approach performs well empirically, in this section we analyse its performance from a theoretical perspective. For this, we compare the performance of our approach to strong baselines in a simple setting.

Setting: We look at the case where n identical customers are available to be matched with a single unit-capacity vehicle. We additionally assume that the customers intrinsic values are drawn independently from the uniform distribution between [0,1]. As in the Experiments, we consider a revenue-maximising objective and compare the revenue that different methods have in this simple setting[4].

Optimal Posted Price (Sandholm and Gilpin 2003): The recurrence relation for revenue, ρ_i (when all customers, $1 \leq k \leq i-1$ declined quoted price, μ_k) is:

$$\rho_i = (1 - \mu_i) \cdot \mu_i + \mu_i \cdot \rho_{i+1} \tag{9}$$

where the overall revenue $\rho = \rho_1$, and the base case is $\rho_n = (1 - \mu_n) \cdot \mu_n$. To find the optimal posted prices, we can differentiate ρ_i w.r.t. μ_i and equate to 0 at every point of the recurrence, working backwards from the ρ_n to ρ_1.

Optimal (Non-Discriminatory) Posted Price: The method above assigns different prices to identical customers – for e.g., to customers with the same origin and destination locations. We provide a method that sets same price μ for all n customers and the resultant revenue ρ for this algorithm is:

$$\rho = (1 - \mu^n) \cdot \mu \tag{10}$$

Differentiating and equating to 0 yields μ^* and ρ^*:

$$\mu^* = (n+1)^{-\frac{1}{n}}, \qquad \rho^* = \frac{n}{(n+1)^{\frac{n+1}{n}}}$$

[4]Details are present in appendix.

		$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=10$	$n=100$
Revenue	Optimal Posted Price	0.25	0.391	0.483	0.550	0.601	0.741	0.963
	Optimal ND Posted Price	0.25	0.385	0.473	0.535	0.583	0.715	0.945
	Optimal Auction	0.25	0.417	[0.25, 0.6]	[0.6, 0.667]	[0.667, 0.714]	[0.818, 0.833]	[0.980,0.981]
	Our Algorithm	0.25	0.381	0.457	0.503	0.532	0.586	0.628
Pricing	Optimal ND Posted Price	0.5	0.577	0.630	0.669	0.699	0.787	0.955
	Our Algorithm	0.5	0.625	0.708	0.766	0.806	0.900	0.990

Table 3: Revenue and difference in pricing for different methods

Optimal Auction: The optimal revenue-maximising auction is Myerson's Auction (Myerson 1981). While it is possible to calculate the expected revenue of this auction in closed form for small n, calculating it in the general case is challenging. Instead, we calculate bounds on the value of the expected revenue – (a) It is *lower-bounded* by the revenue of a second-price auction with no reserve price, and (b) It is *upper-bounded* by the revenue of a second-price auction with $n+1$ bidders and no reserve price (Bulow and Klemperer 1994). Here, the expected revenue of a second-price auction with k bidders is the second-order statistic of the k intrinsic values. In our setting:

$$\rho = \frac{k-1}{k+1}$$

Our Algorithm: To calculate the posted price generated by our algorithm, we first calculate the fraction of samples in which none of the customers are assigned a vehicle.

For a reserve price of $\frac{1}{2}$ and n customers, this corresponds to a probability of $\frac{1}{2^n}$. In all other cases, one of the customers is assigned a vehicle. As all the customers are identical, by symmetry, they have an equal chance of being assigned the vehicle. Then, the price μ for a customer is given by:

$$\mu = F^{-1}(1 - \frac{1-\frac{1}{2^n}}{n}) = \frac{n-1}{n} + \frac{1}{n \cdot 2^n} \tag{11}$$

We can plug μ into Equation 10:

$$\rho = (1 - (\frac{n-1}{n} + \frac{1}{n \cdot 2^n})^n) \cdot (\frac{n-1}{n} + \frac{1}{n \cdot 2^n}) \tag{12}$$

Compute the expected revenue as $n \to \infty$

$$\lim_{n\to\infty} \rho = \lim_{n\to\infty} (\underbrace{\frac{n-1}{n}}_{1} + \underbrace{\frac{1}{n \cdot 2^n}}_{0}) - \underbrace{(\frac{n-1}{n} + \frac{1}{n \cdot 2^n})^{n+1}}_{1^\infty \text{ form}}$$
$$= 1 - e^{\lim_{n\to\infty}(\frac{n-1}{n} + \frac{1}{n\cdot 2^n} - 1)(n+1)}$$
$$= 1 - e^{-1} \approx 0.632$$

So, even if there are an infinite number of customers, our algorithm doesn't converge to an expected revenue of 1.

Results: We collate the revenue results for different methods in the 'Revenue' rows in Table 3. The analysis clearly shows that, even in this simple setting, our algorithm does not provide the same revenue as the optimal posted price. In addition, the gap between our algorithm grows with n and even converges to a sub-optimal solution as $n \to \infty$. The reason for this seems to be that our algorithm prices too aggressively ('Pricing' rows in Table 3). Note that this analysis generalises easily to the multi-capacity case, but we describe the simplest setting here to highlight the significant scope for improvement in this problem.

From a practical point of view, this setting can be thought of as a surge scenario in which there's n times as many customers as drivers. In this context, $n = 5$ is a realistic possibility, and the nearly 10% difference between the revenue obtained by our algorithm and the optimal posted price promises significant impact if algorithm is improved.

7 Future Work

Incorporation of Future Information: Pricing has minimal impact on revenue if customers are sensitive to price fluctuations because there isn't much leeway to increase the prices. When the price-sensitivity is low, we find that one can significantly improve the revenue by pricing aggressively. In those cases, however, considering the pricing problem across multiple timesteps becomes essential and has been the focus of significant past work (Banerjee, Riquelme, and Johari 2015; Bimpikis, Candogan, and Saban 2019). In our framework, one way to mitigate the myopic decision-making is by incorporating the future value, as in Shah et al. (2020).

Algorithms for Posted Pricing: As we highlight in Sec 6, there is an opportunity for algorithmic improvement in solving the matching-aware pricing problem. Empirically, it may be fruitful to begin at Eq 3, and make relaxations that perform well in practice. Theoretically, there hasn't been any study of 'posted pricing' in the context of the tripartite-graph matching problem present in ride-pooling. Also, most 'posted pricing' work in the literature considers a sequential model of pricing and matching customers. In our framework, however, the prices have to all be provided together, and the matching is done in a batch. While this means that prices cannot be adapted based on observations, it also means that it is possible to do better matching because of batching. Given that the techniques don't transfer from the sequential to the batched case, this is a promising future direction.

8 Conclusion

In this paper, we have provided a novel auction based approach to jointly handle pricing (of customer requests) and matching (of vehicles to customer requests) in the extremely challenging on-demand ride-pooling problem. We demonstrate that our approach is not only scalable, but is also able to outperform existing approaches with respect to revenue and reliability. In addition, we provide a theoretical basis for the method employed and also an analysis on potential improvement feasible beyond our approach.

Ethics Statement

We highlight possible ethical implications of our algorithm: ***Discriminatory Pricing:*** Our framework determines the optimal surge price such that supply is matched to demand. It is possible to use personalised models of price-sensitivity to price customers differently. This concern is especially relevant as it is something that ride-pool operators have been accused of in the past (Mahdawi 2020).

In response, we want to reiterate that *we do not do this*. We have a fixed price-sensitivity function for all customers. As similar users have similar chances of winning the utility-maximising auction, they will be priced similarly in our proposed algorithm. This is important because it is possible to increase the revenue by pricing similar customers differently (Blumrosen and Holenstein 2008).

Incorporating Fair Matching Algorithms: On a more positive note, the same generality of our framework provides allows us to incorporate previous work on fair matching (Nanda et al. 2020; Lesmana, Zhang, and Bei 2019) that uses a combination of constraints and surrogate objectives to create more equitable outcomes.

Acknowledgements

This research/project is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG2-RP-2020-016). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

Alonso-Mora, J.; Samaranayake, S.; Wallar, A.; Frazzoli, E.; and Rus, D. 2017. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 201611675.

Banerjee, S.; Johari, R.; and Riquelme, C. 2016. Dynamic pricing in ridesharing platforms. *ACM SIGecom Exchanges*, 15(1): 65–70.

Banerjee, S.; Riquelme, C.; and Johari, R. 2015. Pricing in ride-share platforms: A queueing-theoretic approach. *Available at SSRN 2568258*.

Bimpikis, K.; Candogan, O.; and Saban, D. 2019. Spatial pricing in ride-sharing networks. *Operations Research*, 67(3): 744–769.

Blumrosen, L.; and Holenstein, T. 2008. Posted prices vs. negotiations: an asymptotic analysis. *EC*, 10: 1386790–1386801.

Bulow, J.; and Klemperer, P. 1994. Auctions vs. negotiations. Technical report, National Bureau of Economic Research.

Chawla, S.; Hartline, J. D.; Malec, D. L.; and Sivan, B. 2010. Multi-parameter mechanism design and sequential posted pricing. In *Proceedings of the forty-second ACM symposium on Theory of computing*, 311–320.

Chen, H.; Jiao, Y.; Qin, Z.; Tang, X.; Li, H.; An, B.; Zhu, H.; and Ye, J. 2019a. InBEDE: Integrating Contextual Bandit with TD Learning for Joint Pricing and Dispatch of Ride-Hailing Platforms. In *2019 IEEE International Conference on Data Mining (ICDM)*, 61–70. IEEE.

Chen, M.; Shen, W.; Tang, P.; and Zuo, S. 2019b. Dispatching through pricing: modeling ride-sharing and designing dynamic prices. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 165–171.

Gibbard, A. 1973. Manipulation of voting schemes: a general result. *Econometrica: journal of the Econometric Society*, 587–601.

Heath, A. 2016. Inside Uber's quest to get more people in fewer cars. https://www.businessinsider.com/uberpool-ride-sharing-could-be-the-future-of-uber-2016-6/. Accessed: 2022-04-04.

Korolko, N.; Woodard, D.; Yan, C.; and Zhu, H. 2018. Dynamic pricing and matching in ride-hailing platforms. *Available at SSRN*.

Lesmana, N. S.; Zhang, X.; and Bei, X. 2019. Balancing efficiency and fairness in on-demand ridesourcing. In *Advances in Neural Information Processing Systems*, 5309–5319.

Lowalekar, M.; Varakantham, P.; and Jaillet, P. 2019. ZAC: A Zone Path Construction Approach for Effective Real-Time Ridesharing. In *Proceedings of the ICAPS*, 528–538.

Ma, H.; Fang, F.; and Parkes, D. C. 2019. Spatio-temporal pricing for ridesharing platforms. In *Proceedings of the 2019 ACM Conference on Economics and Computation*, 583–583.

Ma, S.; Zheng, Y.; and Wolfson, O. 2013. T-share: A large-scale dynamic taxi ridesharing service. In *IEEE 29th International Conference on Data Engineering*, 410–421.

Mahdawi, A. 2020. Is your friend getting a cheaper Uber fare than you are? https://www.theguardian.com/commentisfree/2018/apr/13/uber-lyft-prices-personalized-data. Accessed: 2022-04-04.

Myerson, R. B. 1981. Optimal auction design. *Mathematics of operations research*, 6(1): 58–73.

Nanda, V.; Xu, P.; Sankararaman, K. A.; Dickerson, J.; and Srinivasan, A. 2020. Balancing the Tradeoff between Profit and Fairness in Rideshare Platforms during High-Demand Hours. *Proceedings of the AAAI*, 34(02): 22102217.

NYYellowTaxi. 2016. New York Yellow Taxi DataSet. http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml. Accessed: 2022-04-04.

Özkan, E. 2020. Joint pricing and matching in ride-sharing systems. *European Journal of Operational Research*.

Sandholm, T.; and Gilpin, A. 2003. Sequences of take-it-or-leave-it offers: Near-optimal auctions without full valuation revelation. In *International Workshop on Agent-Mediated Electronic Commerce*, 73–91. Springer.

Shah, S.; Lowalekar, M.; and Varakantham, P. 2020. Neural Approximate Dynamic Programming for On-Demand Ride-Pooling. In *Proceedings of the AAAI*, volume 34, 507–515.

Uber. 2018. Uber Matching Solution. https://www.uber.com/us/en/marketplace/matching/. Accessed: 2022-04-04.

Zheng, L.; Chen, L.; and Ye, J. 2018. Order dispatch in price-aware ridesharing. *Proceedings of the VLDB Endowment*, 11(8): 853–865.