Preprint version:

Citation:
Horner, Hannah, Pazour, Jennifer, & Mitchell, John E. (2021) Optimizing Driver Menus Under Stochastic Selection Behavior for Ridesharing and Crowdsourced Delivery. Transportation Research Part E: Logistics and Transportation Review.

# Optimizing Driver Menus Under Stochastic Selection Behavior for Ridesharing and Crowdsourced Delivery

Hannah Horner[1a], Jennifer Pazour[2a,*], and John E. Mitchell[3a]

[a]Rensselaer Polytechnic Institute , 110 8th St, Troy, NY 12180, USA
[*]Corresponding Author
Declarations of interest: none

May 2021

[1]horneh@rpi.edu, https://orcid.org/0000-0002-8921-6726
[2]pazouj@rpi.edu, https://orcid.org/0000-0002-0463-8744
[3]mitchj@rpi.edu, https://orcid.org/0000-0001-5087-4679

**Optimizing Driver Menus Under Stochastic Selection Behavior for Ridesharing and Crowdsourced Delivery**

**Abstract:** Peer-to-peer logistics platforms coordinate independent drivers to fulfill requests for last mile delivery and ridesharing. To balance demand-side performance with driver autonomy, a new stochastic methodology provides drivers with a small but personalized menu of requests to choose from. This creates a Stackelberg game, in which the platform leads by deciding what menu of requests to send to drivers, and the drivers follow by selecting which request(s) they are willing to fulfill from their received menus. Determining optimal menus, menu size, and request overlaps in menus is complex as the platform has limited knowledge of drivers' request preferences. Exploiting the problem structure when drivers signal willingness to participate, we reformulate our problem as an equivalent single-level Mixed Integer Linear Program (MILP) and apply the Sample Average Approximation (SAA) method. Computational tests recommend a training sample size for inputted SAA scenarios and a test sample size for completing performance analysis. Our stochastic optimization approach performs better than current approaches, as well as deterministic optimization alternatives. A simplified formulation ignoring 'unhappy drivers' who accept requests but are not matched is shown to produce similar objective values with a fraction of the runtime. A ridesharing case study of the Chicago Regional Transportation network provides insights for a platform wanting to provide driver autonomy via menu creation. The proposed methods achieved high demand performance as long as the drivers are well compensated (e.g., even when drivers are allowed to reject requests, on average over 90% of requests are fulfilled when 80% of the fare goes to drivers; this drops to below 60% when only 40% of the fare goes to drivers). Thus, neither the platform nor the drivers benefit from low driver compensation due to its resulting low driver participation and thus low request fulfillment. Finally, for the cases tested, a maximum menu size of 5 is recommended as it produces good quality platform solutions without requiring much driver selection time.

## 1 Introduction

A plethora of ridesharing and crowdsourced logistics companies, such as Uber and Grubhub, apply sharing economy-based business models to transportation services requested on-demand from a mobile smartphone (Gesing, 2017). In this paper a *request* refers to a ride request or a delivery request sent by a customer, and a *driver* refers to a person that communicates to the platform they are available to fulfill transportation requests. Peer-to-peer logistics platforms tend to follow one of two basic dispatching (or matching) approaches. The most common is a centralized platform design, where the platform sends out automatic driver-request assignments after each discrete dispatching window that collects requests and currently available drivers in a given geographical region (Qin *et al.*, 2020). Once a request is sent to a driver, they must promptly accept or reject their assignment (Delfino, 2017; Postmates, 2019). Grubhub, Amazon Flex, Postmates, and Instacart use this model for crowdsourced delivery, while Uber and Lyft use it for ridesharing. Though these companies all send assignments to drivers, the details of the platform-driver interactions are varied. For example, the information a driver has about a request varies between

platforms. To avoid drivers cherry-picking trips based on the location and trip length, platforms such as Uber, Lyft, Amazon Flex, and Postmates, and Instacart do not show the trip's destination until after the driver accepts the request (Cook, 2019; Godil, 2020; Lyft, 2018; Rideguru, 2018). Additionally, drivers are not truly free to reject assignments, as Uber drivers with low request acceptance rates stop receiving requests, effectively being sidelined (JC, 2019). An exception to these policies was implemented with California Uber drivers starting in December 2019, as these drivers can now see the destination of their rider and reject requests without losing their Uber Pro status (Ride Share Guy, 2019). Other variations between companies include how the driver's wage is calculated and how the driver communicates when they are available to work (Cook, 2019; Delfino, 2017). While good at request fulfillment, these approaches are restrictive of driver autonomy. For this reason, other companies use a decentralized platform, where drivers can browse the full set of available requests and select which request(s) they would like to fulfill. Door dash and Roadie are two delivery platforms using this approach. Decentralized platforms have a lower percentage of fulfilled requests while some requests receive multiple offers (Einav *et al.*, 2016; Newton, 2014). What's more, drivers can spend substantial time scrolling through potential requests to find a good match (Newton, 2014). To mitigate this problem, companies allow drivers to filter requests by specifying their travel plans and only viewing requests whose delivery routes are in the same direction they are heading (DoorDash, 2019; Roadie, 2018).

Prioritizing both high request fulfillment and driver autonomy, we offer a third option. As depicted in Figure 1, the platform collects open requests and available drivers over a given batching window for a given geographic region. Then for every decision epoch, there is a three stage process. In Stage I, or the *dispatch window*, the platform determines the personalized menu to send to each driver that will maximize the platform's expected utility. Stage II, the *selection window*, is then comprised of the drivers' feedback on their menus received in Stage I. In this stage, each driver will select one or more requests from their menu to fulfill or can opt to not fulfill any of the requests. Stage III, the *assignment window*, is the platform's processing of this feedback and determining the optimal driver-request recourse assignment given driver responses from Stage II. Then the list of available drivers and requests is updated based on any unmatched drivers/requests and the drivers/requests that joined during the previous decision epoch, and the platform reoptimizes the menu set to send out in the next decision epoch. This method is a form of Stackelberg game, with the platform being the leader and the drivers followers. The use of discrete decision epochs is common in practice, with their length being a tunable parameter (Qin *et al.*, 2020). What is different about our work is the ability for drivers to make selections and rejections. This is something drivers want, as evidence by the fact that drivers face penalties for rejecting too many requests (with a notable exception being in California where drivers are rejecting more requests because they are no longer penalized for doing so (Marshall, 2020)). Our model aims to mitigate the effects of rejection by both offering multiple requests to drivers so that there are more matching options despite the inevitable rejections of some offers, and waiting until all drivers have responded (or the response time limit elapses) to find the most beneficial assignment. While this may increase the length of the selection window time, we envision the length of each epoch to be one minute or less, with much of this time allotted to drivers making their selections. If more time is needed, crowdsourced delivery settings are less

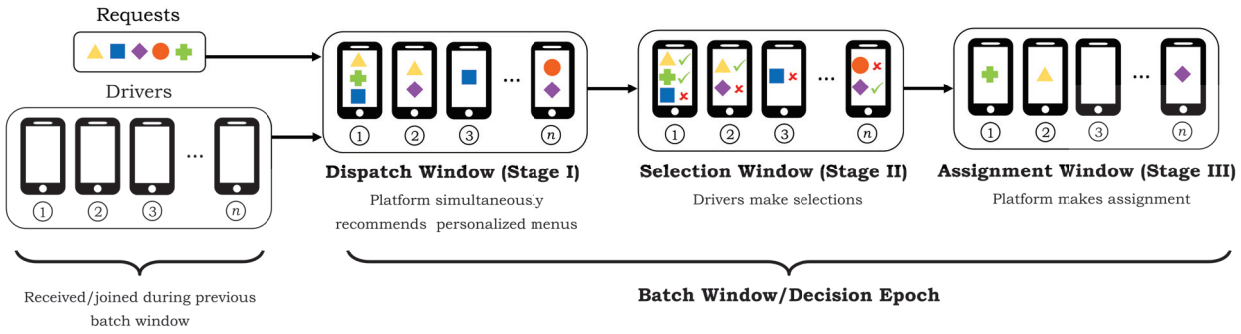



Figure 1: Illustration of the three decision stages, with a set of $n$ drivers logging into a mobile app on their smartphones to fulfill the requests, which are represented by the colored shapes.

time-sensitive than ride sharing applications; for example, Instacart currently allows drivers 4 minutes to accept or reject requests (Captain (2019)), and so this paper's approach would be especially suitable in such crowdsourced delivery applications.

In many ridesharing and crowdsourced delivery settings, in general, but especially with the use of occasional drivers (Punel & Stathopoulos, 2017), it is highly unlikely the platform would have perfect knowledge of the decisions drivers will make. Thus, this work focuses on settings in which selection behavior of each driver is unpredictable. Because uncertainty in driver behavior exists, offering multiple options to each driver can increase the probability a driver accepts at least one request. But offering too many options can also have negative effects, as less popular requests may be skipped in favor of other requests in the drivers' menus. This can result in poor service where some requests having multiple offers from drivers while others receive none. This is problematic as a request with multiple willing drivers can only be fulfilled by one driver while the other drivers wanting to participate are instead not assigned any requests, and requests with no offers go unfulfilled in that time epoch. Providing too many options can also bog down the driver, who does not want to scroll through a long list of requests. Therefore, a goal of this research is to develop a stochastic method able to determine optimal menus, menu size, and request overlaps in menus and to use this method to provide insights into how a platform wanting to provide driver autonomy should think about these decisions.

With few exceptions, current research assumes drivers will accept any request, or at least any request that meets a given criteria such as extra driving time being below a given threshold. The feasibility of each match is thus assumed known a priori; in contrast, our research tackles the challenge of determining request recommendations for drivers that behave stochastically. While a few models allow drivers to reject their matches, none optimize personal menus of multiple requests for each driver. Therefore, our main contributions are as follows. To our knowledge we are the first to explicitly optimize request assignments under stochastic driver behavior and menu sizes larger than one. We employ the Sample Average Approximation method to provide optimal menu sets over an approximated model of driver behaviors. To develop a computationally viable model, we capitalize on a special case that occurs when drivers are allowed in Stage II to signal their willingness to accept each request from their menu rather than making an explicit selection of the request(s). For this behavior model we reformulate our bilevel formulation to

an equivalent single level formulation. We provide a computational analysis of our model in a ridesharing setting using data from the Chicago Regional transportation network, including examining menu and assignment properties, comparing our model's performance against deterministic approaches and current benchmarks, determining recommended training and test sample size, and observing the effects of various parameter values. Our proposed approach, in which the platform recommends multiple requests to each driver, provides a framework that increases drivers' autonomy while still prioritizing high request fulfillment and overall platform utility.

The remainder of this paper is structured as follows. In Section 2 we provide a review of the literature in ridesharing and crowdsourced delivery in relation to our approach. In Section 3 we introduce our bilevel model and in Section 4 we propose its single-level reduction. In Section 5 we present and discuss computational results and in Section 6 we present our conclusions.

## 2 Literature Review

Peer-to-peer logistics research has surged in recent years. We focus our review on ridesharing and crowdsourced delivery methodologies with common elements to our formulation, including stochastic models and bilevel programs. We discuss the existing policies of driver-platform interactions defining the drivers' level of autonomy, and review existing approaches that either offer multiple request options to drivers or offer multiple ride/delivery options to customers. For a more general overview, see reviews for ridesharing (Agatz *et al.*, 2012; Furuhata *et al.*, 2013; Wang & Yang, 2019), crowdsourcing (Alnaggar *et al.*, 2019; Le *et al.*, 2019), and shared mobility (Mourad *et al.*, 2019).

A wide variety of models match drivers to riders or delivery parcels and many include stochastic elements. For example, capturing stochastic travel times (Long *et al.*, 2018), stochastic delays in picking up or dropping off a parcel (McInerney *et al.*, 2013) or in future demand (Lin *et al.*, 2019; Lowalekar *et al.*, 2018; Özkan & Ward, 2020). A few approaches optimize for stochastic driver selection behavior (Barbosa, 2019; Di Febbraro *et al.*, 2013; Gdowska *et al.*, 2018; Li *et al.*, 2019a), but none of them optimize menus with multiple requests per driver.

Because the primary goal of our formulation is to increase the autonomy of drivers participating with the platform, we investigate the variety of matching assumptions representing driver-platform interactions. Many models assume any request can be matched with any available driver, meaning the driver is willing to accept any request(s) assigned to them (Lei *et al.*, 2019; Wang *et al.*, 2016). Another straightforward policy is to a priori capture driver restrictions, usually in the form of an upper limit on the extra driving time/distance (Masoud & Jayakrishnan, 2017; Stiglic *et al.*, 2015), an acceptable time window during which they are available (Hou *et al.*, 2018; Najmi *et al.*, 2017), or a time/distance restriction combined with a maximum number of pickup/drop off stops and/or a maximum number of passengers/parcels (Arslan *et al.*, 2018; Schreieck *et al.*, 2016). None of these models explicitly model driver selection behavior, but instead assume any request meeting a given requirement will be accepted by the driver.

Existing literature uses bilevel programs to determine optimal surge pricing and driver/customer

populations when supply and/or demand are elastic (Zha *et al.*, 2018; Lei *et al.*, 2019). While these do capture some form of driver willingness to participate, they assume drivers will accept the assignments given to them if they remain in the platform. Most are equilibrium models, which create assignment and compensation schemes preferable to the driver under deterministic utilities. The Stackelberg game interactions between the platform and the customers/drivers can also be interpreted as an auction in which the requests are presented and drivers offer bids to complete them (Cheng *et al.*, 2019; Hong *et al.*, 2019). An auction space allows drivers to explicitly state their utility for each request, but requires drivers to spend time deciding whether to bid on every incoming request. Thus, the existing work using bilevel programs is not applicable to our hierarchical decision structure, where the Stackelberg game is due to drivers having autonomy to select requests from recommended menus.

Only a handful of models explicitly model an individual driver's autonomy, with most limited to allowing drivers to reject a request assigned to them. In Soto Setzke *et al.* (2017) drivers have a fixed rejection probability based on their extra driving time for the request compared to their total availability. In Gdowska *et al.* (2018), each delivery request has a fixed probability of being rejected by a crowdsourced driver, in which case it is delivered by the fleet. Barbosa (2019) builds upon Gdowska *et al.* (2018)'s model by using a golden search method to determine the optimal compensation offer when each request's probability of rejection depends on compensation. Di Febbraro *et al.* (2013) sends proposed matches to the current set of drivers and riders, and both the rider and the driver can reject the request at a fixed probability. They then reoptimize the matching if *any* of the rides are rejected by either party, meaning the assignment happens only after all drivers and riders respond positively, which can require multiple iterations. Similarly, Lei *et al.* (2019) make their decisions after all riders have responded to their menus and are assumed to make selections simultaneously. While our approach also waits for all responses before making an assignment, we don't need to reoptimize the menus when drivers reject requests, as our menus of multiple requests are designed to have good recourse assignments even in the face of rejection. Some models like Banerjee *et al.* (2015), Bimpikis *et al.* (2019), Luo & Saigal (2017), and Nourinejad & Ramezani (2019) assume a driver will consecutively fulfill some number of requests with the platform. Drivers are assumed to accept any request assigned by the platform, but will leave the platform if a certain level of satisfaction is not met.

Research exists on demand-side choice, in which a menu of options are offered to the customer, who makes a selection based on factors such as route, wait time, and price in a ridesharing setting, and time of delivery in a crowdsourced delivery setting (Lei *et al.*, 2019; Li *et al.*, 2019b; Chen *et al.*, 2018). In these papers, drivers are assumed to comply with their assignment after riders make selections. Many of the menu-producing models (Chen *et al.*, 2018; Li *et al.*, 2019b) create the menu sets by filtering through options and offering all feasible options or the set of most appealing options. Such methods do not consider the efficiency of their resource allocation, combinatorial trade-offs associated with multiple menus, nor the platform's total utility.

To our knowledge only two formulations offer multiple request options to drivers: Li *et al.* (2019a) and Mofidi & Pazour (2019). In Li *et al.* (2019a) the menus are simply the set of requests with origins within a certain radius of the driver's current location. Such an approach does not require combinatorial

optimization to produce driver menus and instead just filters requests meeting a certain criteria, which ignores interactions between multiple drivers. The menu selections and locations of currently participating drivers follow a Partially Observable Markov Decision Process (POMDP). Because the drivers have the autonomy to make selections and decide how many times to participate with the platform, each driver tries to learn their optimal policy to maximize their total discounted reward. In Mofidi & Pazour (2019)'s model, the platform is assumed to know each driver's expected utility value of each customer request. A hierarchical deterministic approach is implemented to create personalized menus of requests such that the drivers' selections will maximize expected platform utility. However, Mofidi & Pazour (2019) prove that because their formulation has deterministic driver behavior, there exists a menu set with at most one request recommended to each driver that is an optimal menu set and thus their work is unable to provide a method to optimally generate personalized menus. Instead, they rely on a simulation approach that solves multiple integer programs for a fixed menu size and evaluates whether there is value in offering menus to a set of drivers. We therefore are the first to present an optimization method to create hierarchically-motivated menus to drivers that explicitly captures the drivers' stochastic selection behavior.

## 3 Bilevel Stochastic Formulation (BLSF)

We focus on one decision epoch in one geographical region. The platform simultaneously recommends available requests to available drivers, and then based on the drivers' selection responses, assigns them. Table 1 provides a list of notation used for our paper. For driver $j \in N$, the preference of request $i \in M$ is quantified as the net personal utility to the driver, i.e., the driver's utility $u_{ij}$ minus the *no-choice* utility $\phi_j$. Driver $j$'s net utility for request $i$, $u_{ij} - \phi_j$, is assumed to be independent of the contents of their or any other driver's menu. We make this assumption because allowing the preferences of requests to be affected by the contents of a driver's menu would result in nonlinearity as $u_{ij}$ and $\phi_j$ would be variables, influenced by menu composition decisions. Each driver is assumed to select the $\alpha$ requests with the highest personal net utility out of their menu, unless fewer than $\alpha$ requests have a positive personal net utility. In this case they select all requests with a positive personal net utility.

The three stages of our problem are modelled with four sets of binary variables. Our Stage I decision, the menu set that is recommended to drivers, is captured by $\boldsymbol{x}$. Once the menus have been sent out, Stage II begins and each driver accepts or rejects requests in their menu ($\boldsymbol{y}$ variables). Based on the drivers' responses, Stage III entails assigning the requests that are accepted by one or more drivers ($\boldsymbol{v}$ variables). Lastly, we capture drivers who accept one or more requests but are not matched to any requests. These drivers are referred to as *unhappy drivers*, as they would not enjoy being excluded from fulfilling a request and getting paid after they willingly participated in the matching process. Requests accepted by unhappy drivers are recorded using $\boldsymbol{z}$.

Using these variables, our problem can be modeled as a mixed integer bilevel problem. This initial formulation is based on the deterministic formulation presented in Mofidi & Pazour (2019), but is adjusted by considering stochastic selection behaviors, adding the variable $\boldsymbol{v}$ to capture the recourse

Table 1: Overview of sets, indices, parameters, variables, and values.

| Notation | Description |
|---|---|
| **Sets** | |
| $M$ | Set of all requests, $M = 1, 2, ..., m$ |
| $N$ | Set of all available drivers, $N = 1, 2, ..., n$ |
| $S/S^*$ | Set of all training/test scenarios |
| $\Omega/\Omega^*$ | Set of sampled training/test scenarios, $\Omega \subseteq S/\Omega^* \subseteq S^*$ |
| **Indices** | |
| $i$ | Index of request, $i \in M$ |
| $j$ | Index of driver, $j \in N$ |
| $s/s^*$ | Index of training/test scenario, $s \in S/s^* \in S^*$ |
| **Parameters** | |
| $m, n$ | Number of requests and drivers, respectively |
| $c_{ij}$ | Platform's net utility for driver $j$ fulfilling request $i$ |
| $d_{ij}$ | Penalty if driv. $j$ accepts req. $i$ in scen. $s$ and is an unhappy driver in scen. $s$ |
| $\ell, \theta$ | Minimum and maximum menu size, respectively |
| $\alpha$ | Maximum number of requests a driver may accept |
| $q_j$ | Number of requests that can be assigned to driver $j$ |
| $u_{ij}$ | Utility driver $j$ receives from fulfilling request $i$ |
| $u_{ij}^s$ | Realization of $u_{ij}$ in scenario $s$ |
| $\phi_j$ | The no-choice utility the driver receives from electing to not fulfill any request |
| $\phi_j^s$ | Realization of $\phi_j$ in scenario $s$ |
| $p_{ij}$ | Probability that driver $j$ is willing to accept request $i$ |
| $\hat{y}_{ij}^s / \hat{y}_{ij}^{s^*}$ | Willingness: is 1 if driver $j$ is willing to accept request $i$ in scenario $s/s^*$; otherwise is 0 |
| $fare_i$ | Fare paid for request $i$ to the platform |
| $extra_{ij}$ | Extra driving time in hours for driver $j$ if they fulfill request $i$ |
| $miles_i$ | Travel distance in miles of request $i$'s trip |
| $mins_i$ | Travel time in minutes of request $i$'s trip |
| $mins_{ij}$ | Travel time in mins from driv. $j$ to req. $i$ (req. $i$'s wait time if matched with driv. $j$) |
| $wage$ | Wage level, i.e. portion of fare that goes to the driver |
| **Variables** | |
| $x_{ij}$ | Menu: $x_{ij} = 1$ if request $i$ is recommended to driver $j$; otherwise $x_{ij} = 0$ |
| $y_{ij}^s$ | Driver selection: $y_{ij}^s = 1$ if $x_{ij} = 1$ and driv. $j$ accepts req. $i$ in scen. $s$; otherwise $y_{ij}^s = 0$ |
| $v_{ij}^s / v_{ij}^{s^*}$ | Assignment: is 1 if request $i$ is assigned to driver $j$ in scenario $s/s^*$; otherwise is 0 |
| $z_{ij}^s / z_{ij}^{s^*}$ | Unhappy Drivers: is 1 if driv. $j$ accepts req. $i$ but receives no assignment in scen. $s/s^*$; otherwise is 0 |
| **Values** | |
| $inc_{driv}^{s^*}$ | Income of all drivers from assignments in scenario $s^*$ |
| $inc_{plat}^{s^*}$ | Income of platform from assignments in scenario $s^*$ |

assignment, expanding $\boldsymbol{z}$ to include values for every driver-request pair, allowing for multiple driver selections, and allowing a flexible menu size for each driver. A bilevel integer program is already computationally expensive, and our model is further complicated by the fact that the net driver utilites $u_{ij} - \phi_j$ are unknown to the platform. Because the true values of $u_{ij}$ and $\phi_j$ are uncertain, we consider them

random variables that follow some distribution instead of individual values.

We use the popular method of Sample Average Approximation (SAA) to optimize the expected value of the objective function over a sample of possible scenarios of driver behavior (Kleywegt *et al.*, 2002). We define an *SAA* (i.e. *training*) *scenario* to be a set of information about the values of some realization of $u_{ij}$ and $\phi_j$ (for each $i \in M$ and $j \in N$) that is sufficient to determine any driver's decisions given any menu. For example, for our initial formulation, a scenario $s$ consists of a $u^s_{ij}$ and $\phi^s_j$ value for all $i \in M$ and $j \in N$, where $u^s_{ij}$ and $\phi^s_j$ are realizations of $u_{ij}$ and $\phi_j$, respectively. The Stage I variables (the menu set) must have the same values across all scenarios, as the menu recommendations occur before the Stage II stochastic event, the driver selections. Meanwhile, the driver selections and recourse assignment (Stage II and III) are both represented by a different set of variables for each scenario. These menu sets and expected recourse assignments are determined by maximizing the weighted average of the objective values produced by the variable values in each scenario, with the weights being the likelihood of each scenario. As the name suggests, rather than using the complete or *exhaustive* scenario set, we use a randomly sampled subset of the scenarios, as using the exhaustive set is not computationally viable in most cases.

Applying the SAA method to our three-stage problem yields a Bilevel Stochastic Formulation (BLSF) given by (1)-(11), where $\mathbb{P}(s)$ is the probability that scenario $s$ occurs. The objective function (1) optimizes the menus and recourse assignments based on the expected value of two components across the sampled set $\Omega$ of SAA scenarios. The first component, the $c_{ij}v^s_{ij}$ summation, maximizes the platform's expected utility for the sets of matched driver-request pairs in the recourse assignments. The $c_{ij}$ parameter can include the suitability of matching request $i$ to driver $j$, the fare collected for fulfilling request $i$, etc. The second component, the $-d_{ij}z^s_{ij}$ summation, minimizes drivers accepting request(s) but not receiving an assignment. As this could cause reluctance to participate in the future, the platform incurs a penalty for any unhappy drivers. The penalty $d_{ij}$ can depend on the driver $j$'s history of being an unhappy driver in previous epochs, the compensation for request $i$ that driver $j$ would be missing, etc. A penalty is incurred for each request the unhappy driver accepted, as an unhappy driver who is flexible and accepted multiple requests would be even more displeased that they aren't assigned a request compared to if they had just accepted one request.

Constraint (2) specifies a range for the menu size, where $\ell$ can be equal to $\theta$ if the platform prefers all drivers have the same menu size. The platform can also choose to have $\ell = 0$ if they want drivers to not receive a menu if it doesn't benefit the platform. Constraints (3)-(4) ensure a valid assignment of requests to drivers in each scenario, namely requiring an assigned request is both offered to and accepted by the driver it is assigned to, and that each request is only assigned once. Assignment constraint (5) limits the number of requests that can be assigned to each driver in each scenario. This constraint is only necessary when the platform allows drivers to accept more than one request, i.e. when $\alpha > 1$. The $q_j$ values can either be decided by the platform beforehand, or can be decided by driver $j$ when they signal their availability to fulfill requests. This constraint is included because driver $j$ could find multiple requests on their menu appealing enough to accept, but do not want to or do not have time to fulfill more than $q_j$ of them. Constraints (6) and (7) influence the $z_{ij}$ value so it properly represents the meaning of

$$\max_{x,v,z,y} \quad \frac{1}{\sum_s \mathbb{P}(s)} \sum_{s\in\Omega} \mathbb{P}(s) \sum_{i\in M} \sum_{j\in N} \left( c_{ij} v_{ij}^s - d_{ij} z_{ij}^s \right) \tag{1}$$

$$\text{s.t.} \quad \ell \le \sum_{i\in M} x_{ij} \le \theta \qquad \forall j \in N \tag{2}$$

$$v_{ij}^s \le y_{ij}^s \qquad \forall s \in \Omega, \quad \forall i \in M, \quad \forall j \in N \tag{3}$$

$$\sum_{j\in N} v_{ij}^s \le 1 \qquad \forall s \in \Omega, \quad \forall i \in M \tag{4}$$

$$\sum_{i\in M} v_{ij}^s \le q_j \qquad \forall s \in \Omega, \quad \forall j \in N \tag{5}$$

$$z_{ij}^s \ge -1 + y_{ij}^s + (1 - \sum_{k\in M} v_{kj}^s) \qquad \forall s \in \Omega, \quad \forall i \in M, \quad \forall j \in N \tag{6}$$

$$z_{ij}^s \ge 0 \qquad \forall s \in \Omega, \quad \forall i \in M, \quad \forall j \in N \tag{7}$$

$$x_{ij}, v_{ij}^s \in \{0,1\} \qquad \forall s \in \Omega, \quad \forall i \in M, \quad \forall j \in N \tag{8}$$

$$y \in \underset{y}{\operatorname{argmax}} \quad \sum_{s\in\Omega} \sum_{i\in M} \sum_{j\in N} (u_{ij}^s - \phi_j^s) y_{ij}^s \tag{9}$$

$$\text{s.t.} \quad 0 \le y_{ij}^s \le x_{ij} \qquad \forall s \in \Omega, \quad \forall i \in M, \quad \forall j \in N \tag{10}$$

$$\sum_{i\in M} y_{ij}^s \le \alpha \qquad \forall s \in \Omega, \quad \forall j \in N \tag{11}$$

requests accepted by unhappy drivers defined previously. This is done using the logical statement that $z_{ij}^s = 1$ if $y_{ij}^s = 1$ and $1 - \sum_{k\in M} v_{kj}^s = 1$ and $z_{ij}^s = 0$ otherwise. The binary constraints (8) enforce a menu set and assignment that does not partially offer or partially assign requests. The binary nature of the $z_{ij}^s$ variables is naturally enforced by the integrality of the $v_{ij}^s$ and the objective function, so the binary constraint for $z_{ij}^s$ is dropped. The lower level problem is defined by (9)-(11), where each driver selects their $\alpha$ most preferred request(s) to fulfill in each scenario (unless fewer than $\alpha$ requests have a positive net utility for the driver). Each driver has their own lower level problem, but the lower level variables for each driver in a given scenario are not influenced by the lower level variables of any other driver or scenario and their decisions are simultaneous. This allows us to aggregate the lower level problems into a single problem containing all of the drivers' variables and constraints across the set of SAA scenarios. The objective for each driver $j$ is to maximize the total personal net utility of their accepted requests, $\sum_{i\in N}(u_{ij}^s - \phi_j^s) y_{ij}^s$, in each scenario, so the aggregated lower objective function (9) is the summation of the objectives across the set of drivers and scenarios. The constraint (10) enforces that $y_{ij}^s$ is nonnegative and that driver $j$ cannot accept request $i$ unless it is offered to them in their menu. Lastly, constraint (11) dictates how many requests the drivers may accept from their menu. The $y_{ij}^s$ variables must take binary values, but the standard form of the lower level matrix is unimodular and the right-hand side is integral. This guarantees an integer solution to the lower level problem so we do not include binary constraints (Mofidi & Pazour, 2019).

In BLSF, the optimal menus heavily depend on how many requests drivers can accept; if the platform limits the number of requests each driver is allowed to accept, the solver has to be mindful about the likelihood of requests being accepted by multiple drivers. This is especially important when a similar

number of drivers and requests exist, as each additional acceptance of one request can mean another request is not accepted by any driver and therefore can't be matched. In environments where drivers select their top $\alpha$ requests, a driver's behavior under any offered menu can be determined if we know the full ranking order of each request including the no-choice option, i.e. if we can arrange $u^s_{1j}$, $u^s_{2j}$,...,$u^s_{mj}$ and $\phi^s_j$ from lowest value to highest value. Given perfect knowledge of this ranking for every driver, the platform would know what each driver's top $\alpha$ choices are within any menu and whether or not those requests have a positive net utility, and therefore what their selection behavior is in that scenario. Yet, because driver selections depend on the exact preference ranking between all requests, for each driver, the number of unique ways they could behave is bounded by $(m+1)!$, the number of possible ranking orders of the $m$ requests plus the no-choice option. Each of the $n$ drivers' behaviors must fall into one of these $(m+1)!$ ranking orders, so the number of selection scenarios is $((m+1)!)^n$. Further, the calculation of $\mathbb{P}(s)$ is a difficult challenge. Looking at a single driver, we must calculate the probability that $k_1 < k_2 < ... < k_{m+1}$ where $k_1...k_{m+1}$ represent the ranking order of $\phi^s_j$ and the $u^s_{ij}$ in scenario $s$ for $i = 1...m$ and for some $j \in N$. Even if $\phi_j$ and the $u_{ij}$ are normally distributed, there is no closed form solution to the likelihood of a given ranking (Sheffi, 1985). If all variables are discretely distributed, the likelihood can be calculated but it would involve an extensive amount of combinatorial calculations. An approximation of $\mathbb{P}(s)$ can be obtained in a Monte Carlo simulation but this would still be prohibitively expensive in most cases. Hence, in Section 4 we discuss the approaches we take to transform our formulation into a similar but less temperamental model.

# 4 Methodologies to Address Computational Challenges

While BLSF captures the platform's uncertainty in driver behavior, it also presents multiple computational difficulties. First, open questions exist on what information to use to represent a scenario and the calculation of $\mathbb{P}(s)$. Second, bilevel programs are expensive to solve, and having a large number of scenarios quickly increases the computational cost. Next, as we are using only a sampled set of scenarios, a method is needed to obtain an unbiased estimate of our objective value. Finally, due to the exponential nature of our SAA and test scenarios, certain sampled scenario probability values can dominate other scenarios. These challenges are addressed in the Sections 4.1-4.4, respectively.

## 4.1 Scenario Construction

To overcome the challenges associated with scenario generation and probability estimation described in Section 3, in this work we identify a special case of our problem by relaxing constraint (11) to allow drivers to notify the platform of any requests they are willing to fulfill, which is achieved by setting $\alpha$ equal to $\theta$. When selection is unlimited, the driver is willing to accept all requests in their menu with a positive net utility as they no longer have to narrow selection down to their top $\alpha$ choices. Relaxing the selection constraint allows us to characterize our scenarios in a new way. That is because, for a given realization of $u^s_{ij}$ and $\phi^s_j$ values, a ranking order for all drivers, or other sufficient information about $u^s_{ij}$ and $\phi_j$, the driver's actions are known before the first stage decisions are made. Now we can identify the

exact set of requests that each driver is willing to fulfill in scenario $s$ before the menu is determined, i.e. the set of driver-request pairs with $u^s_{ij} - \phi^s_j > 0$.

To efficiently describe a scenario, we introduce the constant $\hat{y}^s_{ij}$, which has value 1 if request $i$ will be accepted by driver $j$ in scenario $s$ if it is offered in their menu, i.e. if $u^s_{ij} - \phi^s_j > 0$, and has value 0 otherwise. Using these constants, a scenario $s$ can be characterized by a set of $\hat{y}^s_{ij}$ values for all $i \in M$ and $j \in N$. This means that we can determine driver selections for a scenario without specifying a distribution for $u_{ij}$ and $\phi_j$ or generating explicit $u^s_{ij}$ and $\phi^s_j$ values. Instead we need only estimate the probability that $u_{ij} - \phi_j > 0$, denoted as $p_{ij}$, which can then be used to generate $\hat{y}^s_{ij}$ values. The total number of scenarios is bounded by $2^{mn}$, as $\hat{y}^s_{ij}$ is binary and a scenario is a set of $\hat{y}^s_{ij}$ values for each $i \in M$ and $j \in N$.

In this paper we assume all $u_{ij}$ and $\phi_j$ distributions are independent of each other so that we can provide a generalized, closed-form formula for $\mathbb{P}(s)$. This assumption is not strictly necessary, so the platform could specify more inter-related distributions if desired, as long as the distributions can be calculated before the first stage decisions. Because the $u_{ij}$ and $\phi_j$ are assumed to have independent distributions, there is a simple formula for the scenario probability $\mathbb{P}(s)$,

$$\mathbb{P}(s) = \left( \prod_{i,j:\hat{y}^s_{ij}=1} p_{ij} \right) \left( \prod_{i,j:\hat{y}^s_{ij}=0} (1 - p_{ij}) \right), \tag{12}$$

where $p_{ij} = \mathbb{P}(u^s_{ij} > \phi^s_j)$. Using (12) accurately represents the scenario probability for these $\hat{y}^s_{ij}$-defined scenarios, that is, there are no functionally equivalent scenarios to $s$ that should be included in the scenario $s$ probability. This is because the information used to represent a scenario ($\hat{y}^s_{ij}$ values) is the minimum possible information needed to determine driver selections, so a unique set of $\hat{y}^s_{ij}$ values corresponds to a unique set of driver selections. The $p_{ij}$ values can either be estimated directly or calculated from the estimated distributions of $u_{ij}$ and $\phi_j$. The $\hat{y}^s_{ij}$ follow a Bernoulli random distribution such that $\hat{y}^s_{ij} = 1$ with probability $p_{ij}$. The total number of scenarios, i.e. $|S|$, can be found by counting the number of $p_{ij}$ values that are in the interval $(0,1)$. This is because if $p_{ij} = 0$, then $\hat{y}^s_{ij}$ is never equal to 1 so we do not need to include any scenarios where $\hat{y}^s_{ij} = 1$. Similarly, if $p_{ij} = 1$ we do not need to include any scenarios where $\hat{y}^s_{ij} = 0$. Therefore the total number of scenarios is

$$|S| = 2^{mn - \left(\sum_{\{i,j:p_{ij}=0\}} 1\right) - \left(\sum_{\{i,j:p_{ij}=1\}} 1\right)}. \tag{13}$$

### 4.2 Reduction to a Single Level Stochastic Formulation (SLSF)

Our new single level stochastic formulation (SLSF) in (14)-(22) is equivalent to BLSF when constraint (11) is dropped. That is, any feasible point in BLSF has an equivalent feasible point in SLSF with the same objective value, and likewise any feasible point in SLSF has an equivalent feasible point in BLSF with the same objective value. Using $p_{ij}$ values (estimated by the platform beforehand), $\hat{y}^s_{ij}$ values can be generated for each driver $j$-request $i$ pair for each scenario $s$, instead of generating $u^s_{ij}$ and $\phi^s_j$ values. With these $\hat{y}^s_{ij}$ values, for each scenario $s$ we have full knowledge of each driver's willingness to accept each request. Using this information, rather than obtaining $y^s_{ij}$ values from solving the lower level problem in BLSF, we can calculate the values using $\hat{y}^s_{ij}$ and $x_{ij}$. The value of $y^s_{ij}$ is one if and only if request $i$ is

accepted by driver $j$, and this will happen if and only if driver $j$'s menu contains request $i$ and driver $j$ is willing to accept that request. In short, we have the relationship that $y^s_{ij} = 1$ if $x_{ij} = 1$ and $\hat{y}^s_{ij} = 1$ and $y^s_{ij} = 0$ otherwise. We could create a series of linear constraints that enforce these values of $y^s_{ij}$; we instead eliminate $y^s_{ij}$ entirely and just use $x_{ij} = 1$ and $\hat{y}^s_{ij} = 1$ directly. The BLSF upper objective does not contain any $y^s_{ij}$ terms, so we only need to adjust constraints (3) and (6) as they are the only constraints containing $y^s_{ij}$. Our SLSF formulation, given by (14)-(22), is the result after a combination of logical constraints replace $y^s_{ij}$, therefore forgoing the lower level problem.

$$\max_{x,v,z} \quad \frac{1}{\sum_{s\in\Omega}\mathbb{P}(s)} \sum_{s\in\Omega} \mathbb{P}(s) \sum_{i\in M}\sum_{j\in N} \left(c_{ij}v^s_{ij} - d_{ij}z^s_{ij}\right) \tag{14}$$

$$\text{s.t.} \quad \ell \le \sum_{i\in M} x_{ij} \le \theta \qquad \forall j \in N \tag{15}$$

$$v^s_{ij} \le x_{ij} \qquad \forall s \in \Omega, \quad \forall i \in M, \quad \forall j \in N \tag{16}$$

$$v^s_{ij} \le \hat{y}^s_{ij} \qquad \forall s \in \Omega, \quad \forall i \in M, \quad \forall j \in N \tag{17}$$

$$\sum_{j\in N} v^s_{ij} \le 1 \qquad \forall s \in \Omega, \quad \forall i \in M \tag{18}$$

$$\sum_{i\in M} v^s_{ij} \le q_j \qquad \forall s \in \Omega, \quad \forall j \in N \tag{19}$$

$$z^s_{ij} \ge -2 + \hat{y}^s_{ij} + x_{ij} + (1 - \sum_{k\in M} v^s_{kj}) \qquad \forall s \in \Omega, \quad \forall i \in M, \quad \forall j \in N \tag{20}$$

$$z^s_{ij} \ge 0 \qquad \forall s \in \Omega, \quad \forall i \in M, \quad \forall j \in N \tag{21}$$

$$x_{ij}, v^s_{ij} \in \{0,1\} \qquad \forall s \in \Omega, \quad \forall i \in M, \quad \forall j \in N \tag{22}$$

**Theorem 4.1.** *Our SLSF formulation is equivalent to BLSF when* (11) *is relaxed and scenarios are categorized using sets of $\hat{y}^s_{ij}$ values.*

The proof of Theorem 4.1 is deferred to Appendix A. In comparison to BLSF, SLSF has no lower level problem, fewer variables, and its scenarios with corresponding probabilities are easy to define and easy to compute.

### 4.3 Menu Performance

The solution to SLSF provides an objective value in (14) that measures the expected performance of the solution menu $\boldsymbol{x}^*$ across the scenarios generated for the problem inputs, i.e. the SAA scenarios. However, this objective value does not include the menu's performance across all possible scenarios, and has an on average positive bias compared to the true objective value across all scenarios (Kleywegt *et al.*, 2002). To obtain an unbiased estimate of our objective value, we generate either the complete set $S^*$ of all possible scenarios to use as *test scenarios* or a random smaller set $\Omega^* \subset S^*$, and calculate the average objective across these test scenarios. To calculate the optimal objective value for a given test scenario $s^* \in \Omega^*$ or $s^* \in S^*$, we need to use the driver behavior in that scenario to determine which requests are accepted from the menu, then determine the assignment that maximizes the objective value. This can be done by

solving the linear integer program (23)-(29), which determines the optimal assignment set and unhappy driver penalties $[\boldsymbol{v}^{s^*}, \boldsymbol{z}^{s^*}]$ given the menu and test scenarios' driver behavior $[\boldsymbol{x}^*, \hat{\boldsymbol{y}}^{s^*}]$.

$$\max_{v^{s^*},\, z^{s^*}} \quad \sum_{i\in M}\sum_{j\in N}\left(c_{ij}v_{ij}^{s^*} - d_{ij}z_{ij}^{s^*}\right) \tag{23}$$

$$\text{s.t.} \quad v_{ij}^{s^*} \le \hat{y}_{ij}^{s^*} \qquad \forall i \in M, \quad \forall j \in N \tag{24}$$

$$\sum_{i\in M} v_{ij}^{s^*} \le q_j \qquad \forall j \in N \tag{25}$$

$$\sum_{j\in N} v_{ij}^{s^*} \le 1 \qquad \forall i \in M \tag{26}$$

$$z_{ij}^{s^*} \ge -2 + \hat{y}_{ij}^{s^*} + x_{ij}^* + (1 - \sum_{k\in M} v_{kj}^{s^*}) \qquad \forall i \in M, \quad \forall j \in N \tag{27}$$

$$z_{ij}^{s^*} \ge 0 \qquad \forall i \in M, \quad \forall j \in N \tag{28}$$

$$v_{ij}^{s^*} \in \{0,1\} \qquad \forall i \in M, \quad \forall j \in N \tag{29}$$

As is the case for SLSF training scenarios, problems quickly become too large to calculate the true performance of a menu by finding the optimal objective value for every scenario if we let $S^* = S$. However, we are able to significantly reduce the number of test scenarios by consolidating the scenarios that are guaranteed to have the same optimal solution and objective value. This also increases the total weight of the scenarios in the test set if a non-exhaustive subset of $S^*$ is used. We can do this because once a menu has been decided, we no longer need information on a driver's behavior for requests not offered in their menu. For a request $i$ not in driver $j$'s menu, whether they are willing or not to accept that request in any given scenario does not affect the value of (23) as they cannot actually accept the request nor be assigned to it, making $v_{ij}^{s^*}$ and $z_{ij}^{s^*}$ zero for any $s^* \in S^*$. Therefore, we define distinct test scenarios to be two scenarios $\hat{s}^*$ and $\bar{s}^*$ such that for some $i \in N$ and $j \in M$ we have $x_{ij} = 1$ and $\hat{y}_{ij}^{\hat{s}^*} \neq \hat{y}_{ij}^{\bar{s}^*}$. The total number of distinct test scenarios is defined by the total number of menu recommendations across all drivers, so the total number is bounded by $2^{\theta n}$, a drastically lower number than $2^{mn}$ for most $\theta$ values. The formula to calculate a consolidated test scenario's probability is given by (30).

$$\mathbb{P}(s^*) = \left(\prod_{i,j:x_{ij}=1,\hat{y}_{ij}^{s^*}=1} p_{ij}\right)\left(\prod_{i,j:x_{ij}=1,\hat{y}_{ij}^{s^*}=0} (1-p_{ij})\right) \tag{30}$$

In (31), a formula for the total number of distinct scenarios, i.e. $|S^*|$, is given. It is similar to (13), the formula for $|S|$. The only difference is that we ignore all driver-request pairs not in the menu set.

$$|S^*| = 2^{\sum_{i,j} x_{ij} - \left(\sum_{\{i,j:p_{ij}=0, x_{ij}=1\}} 1\right) - \left(\sum_{\{i,j:p_{ij}=1, x_{ij}=1\}} 1\right)} \tag{31}$$

That being said, problems still can easily grow too large to calculate the performance across all scenarios. Thus, in Appendix B we run computational experiments to find a recommended test sample size that balances accuracy and computational needs.

### 4.4 Generating Training and Test Scenarios

Due to the exponential nature of our SAA and test scenario probability formulas (12) and (30) respectively, the scenario probability values can vary by several orders of magnitude. For problems with 20 or more drivers and requests, the relative probability of SAA sampled scenarios in early testing are dominated by a small percentage of scenarios, in that a small handful of the scenarios are relevant to the solver while the other scenarios are essentially ignored. This effect is confirmed by observing the $z^s_{ij}$ values in solutions. For most scenarios there are one or more $i \in M$ and $j \in N$ such that $z^s_{ij}$ have a value of 1 while $x_{ij}$ have a value of 0, which would represent the penalty of driver $j$ being unhappy about request $i$ even though request $i$ is not a part of their menu. This is an unnecessarily incurred penalty, so it should not be part of an optimal solution (or near optimal solution), but occurs because that scenario has such a small relative probability of occurring that including the penalty does not noticeably affect the objective.

To create SAA scenarios that all contribute to the objective value, we propose an alternate method of scenario generation. Instead of generating each scenario randomly, we create a set of *mutated* scenarios by initializing a new scenario as a clone of the scenario that has the highest possible probability and then changing or *mutating* the values of some entries. To decide what entries to mutate, we start by generating a random scenario and in a random order compare its entries with those of the most likely scenario. If the entries for a driver-request pair match, we proceed to the next random entry. If the entries do not match, we mutate that entry of the new scenario to match the random scenario and update the new scenario's likelihood of occurring, which is monotonically nonincreasing with each mutation. This process is terminated when the next mutation would make the new scenario's likelihood fall below $1/10^6$ of the likelihood of the most likely scenario, or if all entries of the new scenario and the random scenario match so there are no more entries to mutate. The factor of $10^6$ is chosen so that scenarios have the flexibility to mutate a decent number of entries to create diversity between the scenario sets. If a newly-generated mutated scenario is a duplicate of a previously-generated mutated scenario in the sample set, it is discarded and a new scenario is generated until a unique mutated scenario is found and added to the sample set. The most likely scenario is easy to calculate, as each driver-request pair in a scenario has a binary outcome that either the driver is willing to accept the request or they are not. We know the probability of either outcome, and the outcomes are independent of other driver-request outcomes. Therefore, for each driver-request pair, we can simply pick the binary outcome that is more likely. Doing this for each pair forms the most likely scenario, i.e. the driver is willing to accept any request with $p_{ij} \in [0.5, 1]$ and will reject any request with $p_{ij} \in [0, 0.5)$. Because we terminate once a predetermined probability deviation is met, the probability of the outputted scenario is approximately equal to the probability of the most likely scenario divided by the deviation cap of $10^6$. This is true for each scenario generated with this method, enabling us to generate a set of scenarios that all hold nontrivial weight in the SAA objective value and is the method we use to generate SAA scenarios for SLSF.

These mutated SAA scenarios have bias that may affect the results of our recommended decision variables. However, using mutated scenarios allows us to have all scenarios in a training sample have a high enough probability to be relevant to the solver, which allows us to analyze the impact of training sample size. For our test scenarios, we use randomly sampled scenarios in order to obtain an unbiased

performance estimate.

# 5 Computational Results

## 5.1 Generating Parameter Values for a Ridesharing Problem

We consider a ridesharing application with the following setup. When a driver wishes to participate with the platform, they specify the origin and destination of their planned trip. The requests are from customers requesting a specific ride with an origin and destination. Because drivers are assumed to have an original planned trip, our generated data is suited for a fleet of drivers participating in a single round of assignments (e.g., occasional drivers) as opposed to drivers working for a block of time, signing up for a new request once their current one is completed. The pool of unmatched drivers and requests is updated and the reoptimization of driver menus is completed every epoch, for example every minute. Driver and request trips are assumed to take place within Chicago, which is approximated using a rectangular subgraph of the Chicago Regional transportation network (Chicago Area Transportation Study (CATS), 2016). This subgraph has 350 *zone centroid* nodes that represent origin and destination locations, along with additional regular nodes and single-directional links connecting nodes to represent roads. The network data includes the travel demand between any pair of zone centroids, and the travel time and distance needed to traverse any link when the traffic flow is equal to the equilibrium flow. We consider a ridesharing application for our experiments because this demand represents people traveling in cars, which may be different from the demand for parcels to be transported. Yet, our model can also be applied to crowdsourced delivery, which is less time-sensitive than ridesharing applications. To randomly draw driver and request origins and destinations, we use the demand between each OD (origin-destination) pair in the reduced network as weights and draw OD pairs with probability based on these weights until we have $m$ request OD pairs and $n$ driver OD pairs. The travel times and distances are calculated based on the link route with the lowest travel time. We then calculate our input parameters, summarized in Table 2.

Table 2: Input parameter formulas.

| Parameter | Formula |
|---|---|
| $fare_i$ | $\max\left\{\left(1.79+0.28\cdot mins_i+0.81\cdot miles_i\right),\ 3\right\}$ |
| $c_{ij}$ | $1.85+0.20\cdot fare_i-0.056\cdot mins_{ij}+r_i,$ where $r_i\sim\mathcal{U}([1,15])$ |
| $d_{ij}$ | $fare_i+r_j,$ where $r_j\sim\mathcal{U}([0,3])$ |
| $p_{ij}$ | $\begin{cases} 0 & \text{if } \frac{0.80\cdot fare_i}{extra_{ij}}<10 \\ \frac{\frac{0.80\cdot fare_i}{extra_{ij}}-10}{25-10} & \text{if } 10\le\frac{0.80\cdot fare_i}{extra_{ij}}<25 \\ 1 & \text{if } 25\le\frac{0.80\cdot fare_i}{extra_{ij}} \end{cases}$ |

We assume the platform's utility $c_{ij}$ is the platform's portion of the amount paid by the passenger for

fulfilling request $i$, minus a penalty based on the the wait time for driver $j$ to reach passenger $i$, plus a *match bonus* incentive to fulfill request $i$. According to Lyft, a booking fee of $1.85 goes directly to the platform and then a fare is calculated based on the rider's trip travel time and distance (EstimateFares.com, 2019). Lyft advertises that drivers make 80% of this fare, so we give the platform 20% of the fare (Helling, 2019). For the wait time penalty we have a per-minute penalty with the multiplier being the platform's cut of their per minute fee used in the fare calculation, $0.28 \cdot 0.20 = 0.056$. The fulfillment incentive (match bonus) is included so that the platform can prioritize fulfilling more urgent or potentially less popular requests, for example requests that went unfulfilled in the previous epoch and need to be matched quickly to maintain service quality. The bonus is assumed to be a constant generated uniformly from $[0,15]$ for each request, to represent the variety of priority/urgency levels among requests. The unhappy driver penalty $d_{ij}$ depends on driver $j$'s history with the app (e.g. how many times they've been an unhappy driver) and is captured by a uniformly generated number in $[0,3]$. This penalty is then added to the fare of request $i$, as this is revenue the unhappy driver is missing out on if they had been matched to request $i$. The willingness to accept, $p_{ij}$, is calculated from the driver $j$'s wage per hour on their extra driving time if they fulfill request $i$. It is assumed to be linear with increasing wage per hour within the window of $10 to $25 per hour (see Table 2). We assume any wage per hour below $10 has $p_{ij} = 0$ and any wage per hour above $25 has $p_{ij} = 1$. The minimum menu size $\ell$ is set to be zero for all experiments, and we set $q_j = 1$ to restrict each driver to fulfill at most one request.

Our computational analysis is outlined as follows. In Phase 0 we run tests to determine a recommended test scenario sample size, and conclude that using 5000 test scenarios is sufficient to produce accurate but cost-efficient performance estimates of an outputted menu. This phase is in Appendix B. In Phase 1 we run tests to determine a recommended SAA scenario sample size that produces menus with high performance in a reasonable amount of time. In Phase 2 we confirm SLSF's performance quality against current benchmarks and deterministic optimization approaches. In Phase 3 we make general observations about the properties of menus and the sets of accepted and assigned requests. Finally, in Phase 4 we run tests to analyze the affects of select factors and parameter values. All experiments in this paper are completed in MATLAB R2019a using CPLEX 12.9 on an i7-core 1.9GHz CPU with 32GB RAM. For all experiments, the SLSF solver terminates upon finding a solution with an objective value within 1% of the optimal or by reaching the 500 second time limit.

### 5.2 Phase 1: SAA Sample Size

We first set out to determine a sufficient training sample size, using experiment results from a series of small and large problems. For each problem size, three different sets of parameter instances (e.g., different driver and rider trips) are generated. We test three different small problem sizes, $4 \times 4$, $5 \times 5$, and $6 \times 6$, where the first number in each pair is the number of requests and the second number is the number of drivers. These problems are small enough that we can compute the optimal menu by using SLSF with all distinct training scenarios, that is, $\Omega = S$. This exhaustive menu is used to compare performance against menus in the same problem instance made using fewer training scenarios. We also test a number of large problem sizes, $20 \times 20$, $20 \times 40$, $40 \times 20$, and $40 \times 40$. For the large problems, it is unreasonable

to calculate the optimal menus. Instead, a generated menu is compared against the menu found with the highest performance estimate among the menus generated for the same parameter instance. For each small problem instance, we require the input parameters to yield a complete training scenario set $S$ with cardinality between $2^{10}$ and $2^{13}$ distinct scenarios. This is to ensure our problems have at least some degree of complexity while also not having so many distinct training scenarios that finding the optimal menu is too computationally expensive. For large problems, we only require there to be at least $2^{mn/3}$ distinct training scenarios. For the $4 \times 4$ and $5 \times 5$ problems $\theta = 2$, for the $6 \times 6$ problems $\theta = 3$, and for all large problems $\theta = 5$. The SAA sample sizes tested are 10, 50, 100, 500, and 1,000 samples, and we conduct three trials of menus for each problem instance for each training sample size. One trial consists of generating a set of mutated SAA scenarios and using SLSF ((14)-(22)) to produce a menu. The set of distinct test scenarios depends on the menu being tested; to compare menus from the same instance on a level field, we use the union set of all menus generated under the same problem instance to define $S^*$. A set of 5,000 test scenarios is generated and the same set is used for all problems in the same instance. To analyze performance, we evaluate each menu using (23)-(29) with each of the 5,000 generated test scenarios, and calculate the weighted objective value average of (23) for each menu across the test scenarios.

The performance analysis results are displayed in Figure 2. For SAA sample sizes over 100, the performance of the generated menus plateaus for most of the small problems and actually decreases for most of the large problems. The lower objective values for the larger SAA sample sizes in the large problems is caused by the time limit of the SLSF solver. For most of the large problems, having 500 or 1,000 SAA samples creates an integer program too large for the solver to find menus with a high objective value in 500 seconds. Thus, having fewer SAA samples allows the solver to run to completion and is more valuable than partially solving the SLSF with more SAA samples. The objective value error rates have some correlation to problem size for the small problems, as the error for each SAA sample size tends to decrease as problem size increases. However, this trend is not observed for the large problems. There is a large amount of variability in the error, even between trials in the same problem instance with the same SAA sample size. This is observable in the clear differences between the maximum error graphs (Figure 2a, 2c, and 2e) and the average error graphs (Figure 2b, 2d, and 2f). Due to this variability and the error variability between different problem instances and problem sizes, there is no SAA sample size that always yields the best performance. That being said, for large problems, menus made using 100 SAA scenarios usually are the best or nearly the best out of the menus made for the given problem instance, with the average error rate (when compared to the best-found menu) being 1.7%. The average runtimes of the problem instances for each problem size and SAA scenario sample size are displayed in Table 3. While the runtime monotonically increases for problems of the same size with increasing SAA scenario sample size, the correlation between problem size and runtime is much less consistent. The differences between the runtimes of these problems seems to have as much to do with the qualities of individual problem instances as they do with the problem size. Because of the improved performance and reasonable runtime, we select 100 as our SAA sample size for experiments in Phases 2 through 4. We note that while the $20 \times 20$ problems with 100 SAA scenarios have a much higher runtime than the other large

problems with 100 scenarios, in practice we would expect the runtime average to be closer to those of the other large problems. One of the three $20 \times 20$ instances required an average of 356.04 seconds while the average of the other two instances is only 4.30 seconds, so the overall average is skewed because of our small sample size. Additional experiments confirmed this, as in Sections 5.3 and 5.5.1, using SLSF with 100 SAA scenarios on ten different $20 \times 20$ problems (for each section) required an average of 22.29 seconds in Section 5.3 and 84.02 seconds with a median of only 13.67 seconds in Section 5.3.

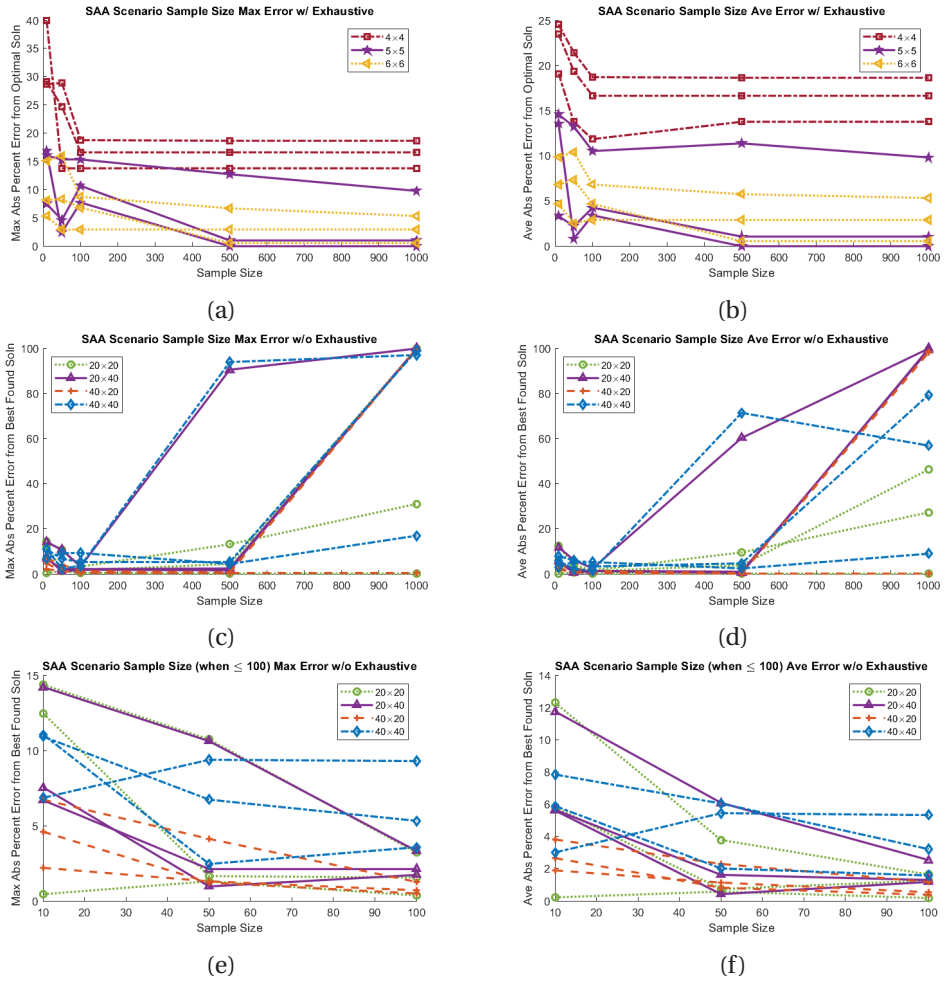



Figure 2: Absolute values of error rates of menus. Graphs (a) and (b) are the small problems where error is calculated by comparing the performance estimate to the performance estimate of the optimal menu; graphs (c)-(f) are the large problems where error is calculated by comparing the performance estimate to the highest performance estimate out of the menus in the given parameter instance. Each line on a graph represents data from the same parameter instance. For each SAA sample size, the error of the three trials is either averaged ((b), (d), and (f)) or the maximum value is recorded ((a), (c), and (e)). Graphs (c) and (e) and graphs (d) and (f) display the same data, but because the detail of lower sample sizes is lost in (c) and (d), (e) and (f) display the results from the data with 100 or fewer samples.

Table 3: Runtimes for Phase 1 for small and large problems. Column 1 separates the problems by size, with the three small problems listed first. Column 2 displays the average log base 2 value of the number of distinct SAA scenarios for the given problem size's problem instances. Column 3 displays the average runtime of either the exhaustive performance calculation for small problems or the runtime of the trial that found the best-performing solution for large problems. Columns 4-8 display, for each of the five training scenario sample sizes, the average runtime of the three trials averaged across the three problem instances of the given size.

| | | Average Time (seconds) | | | | | |
|---|---|---|---|---|---|---|---|
| Size | Ave $\log_2(\|S\|)$ | Exh./Best | 10 scens | 50 scens | 100 scens | 500 scens | 1,000 scens |
| 4×4 | 11.0 | 118.18 | 0.10 | 0.26 | 0.44 | 3.22 | 9.64 |
| 5×5 | 11.7 | 262.17 | 0.09 | 0.40 | 0.76 | 5.04 | 11.63 |
| 6×6 | 12.3 | 175.15 | 0.09 | 0.39 | 0.82 | 3.27 | 6.45 |
| 20×20 | 244.3 | 75.21 | 0.19 | 10.25 | 121.55 | 348.11 | 396.95 |
| 20×40 | 487.7 | 144.95 | 0.28 | 3.13 | 9.42 | 357.15 | 504.52 |
| 40×20 | 438.7 | 312.48 | 0.26 | 1.99 | 4.93 | 254.56 | 389.24 |
| 40×40 | 889.7 | 46.12 | 0.61 | 4.07 | 11.06 | 292.40 | 497.94 |

### 5.3 Phase 2: Comparison Against Current Benchmarks and Deterministic Approaches

With the recommended number of training and test scenarios established in Phases 0 and 1, we test the performance quality of our formulation against a series of alternative approaches. The first two methods, CLOS-1 and CLOS-5, are designed to mimic the current practice used by some platforms that recommends requests to drivers based on proximity. CLOS-1 recommends a menu of exactly one request to each driver, and each request is recommended once. CLOS-5 creates a menu of exactly five requests for each driver, and each request appears in five different menus. With these constraints, CLOS-1 and CLOS-5 select the menu set to minimize the total rider wait time i.e. the travel time between the driver origin and the rider origin. The other two approaches, DET-1 and DET-5, represent a deterministic optimization version of SLSF, in which SLSF is run using only one training scenario of driver selections. The single scenario used is the most likely scenario, that is, driver $j$ will accept request $i$ if $p_{ij} \geq 0.5$ and will not accept if $p_{ij} < 0.5$. DET-1 creates menus with size exactly one, and DET-5 creates menus with size exactly five. The menu size is constrained to be exactly one/five instead of less than or equal to one/five because using SLSF with one training scenario and a minimum menu size of zero yields an assignment instead of a menu set, as adding additional recommendations does not affect the formulation's objective value. Table 4 displays the performance results of SLSF using 100 training scenarios and the four approaches. Ten different problem instances, each with 20 requests and 20 drivers are performed. Performance analysis is done for each menu using 5,000 test scenarios, unless the menu has fewer than 5,000 distinct test scenarios in which case all test scenarios are used. In addition to the average objective value from the performance analysis and average runtime, we include three main components that contribute to the objective function: the number of unmatched requests, the number of requests accepted by an unhappy driver (i.e. the average number of positive $z^s_{ij}$ values), and the average profit, that is, the booking fees and 20% of the fares of the matched requests. The results show that SLSF has the best

performance not only in objective function but also in the three displayed components of the objective except in that CLOS-1 does not have any unhappy driver requests as there is no overlap in driver menus. This illustrates the importance of capturing stochastic driver selections directly into the optimization model. The runtime for SLSF, while much higher than that of the other methods, is still small enough for practical implementations.

Table 4: Performance averages of SLSF and comparative approaches. Column 1 displays the method, while column 2 displays the average objective value (Obj) across the ten problem instances. Columns 3, 4 5, and 6 display the average number of unmatched requests (Unmat), expected number of requests accepted by an unhappy driver (Unhap), the expected profit (Prof), the average runtime, respectively.

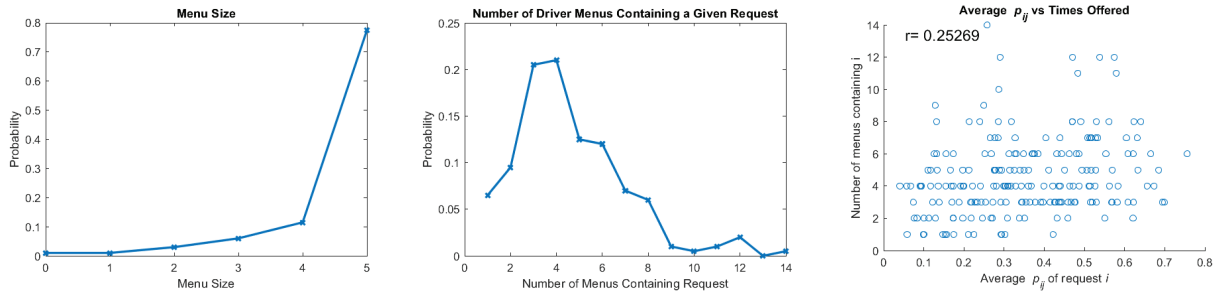
| | Performance | | | | |
|---|---|---|---|---|---|
| | Obj | Unmat | Unhap | Prof ($) | Time (s) |
| SLSF | 212.51 | 1.582 | 0.187 | 81.77 | 22.294 |
| CLOS-1 | 150.06 | 6.928 | 0 | 56.08 | 0.018 |
| CLOS-5 | 191.36 | 2.176 | 1.504 | 80.19 | 0.016 |
| DET-1 | 162.70 | 5.661 | 0.244 | 59.76 | 0.040 |
| DET-5 | 173.35 | 3.336 | 1.448 | 77.54 | 0.035 |

Both of the approaches with menu size one do not perform as well as their comparative approach with menu size five and are much dwarfed by SLSF performance. Thus, the platform can benefit by offering multiple options to drivers. There are also clear limits to the performance of a deterministic version of SLSF. Specifically, DET-5, which ignores stochastic driver behavior, has an average objective function that performs 18.4% worse than SLSF. We conclude that our formulation SLSF benefits from offering multiple requests and by considering stochastic driver behavior, and that methods missing one or both of these elements are unable to achieve as good of performance.

### 5.4 Phase 3: Properties and Performance of SLSF Decision Variables

In this section we explore the properties and performance of SLSF decisions, especially menus generated. For this experiment we generate ten $20 \times 20$ parameter instances with a maximum menu size of five. We use SLSF with 100 SAA samples to generate a menu, and the optimal assignment and objective value is recorded for 5,000 random test scenarios. The category of data from Stages I-III is referred to as *Offered*, *Accepted*, and *Assigned*, respectively. This data is also compared to the driver/request trips and initial parameter data, referred to as the *Original* data. For readability, we create and graph a *discretized* probability density function (pdf) of the data. Such graphs are identified with *Probability* as the $y$-axis label. These discretized pdf graphs convey similar information as histograms, but are plotted as a single line rather than a bar graph and the measurements are normalized to add up to 100%. An example is Figure 3b: when the *Number of Menus Containing Request* is 4, the plotted line has value 0.21. Because the data is from Stage I, 21% of requests are offered to 4 drivers. However, the same cannot be said for graphs using Stage II or Stage III data, as this data is based on multiple test scenarios that have different likelihoods of occurring. For this reason, data from each scenario in Stage II and Stage III are given weights based on the scenario's likelihood that impact the calculated probability.

Figure 3 displays the graphs associated with Stage I: Offered. From Figure 3a, 78% of drivers' menus contain the maximum five requests, and only 1% of drivers are given an empty menu. Thus, the SLSF takes advantage of the stochastic nature of the problem and almost always finds value in offering multiple options to drivers. Figure 3b displays the frequency of offering a given request, with each request being recommended to an average of 4.6 drivers. The results of 3b indicate that the unhappy driver risk is outweighed by the benefit of being able to recommend multiple requests to drivers, which mitigates the request rejections that come with giving drivers the autonomy to decline requests. Offering multiple requests to drivers also likely reduces the negative effect of multiple drivers accepting the same request, as these drivers may also accept other requests that result in alternate assignment option(s) such that all or most of these drivers still receive an assignment. Figure 3c investigates the affect of $p_{ij}$ on the resulting menus. A similar spread of recommendation frequency exists between requests of different *popularity* levels in Figure 3c, and the low correlation coefficient of $r = 0.253$ confirms that the number of drivers a request is recommended to is not correlated with its average $p_{ij}$ value.

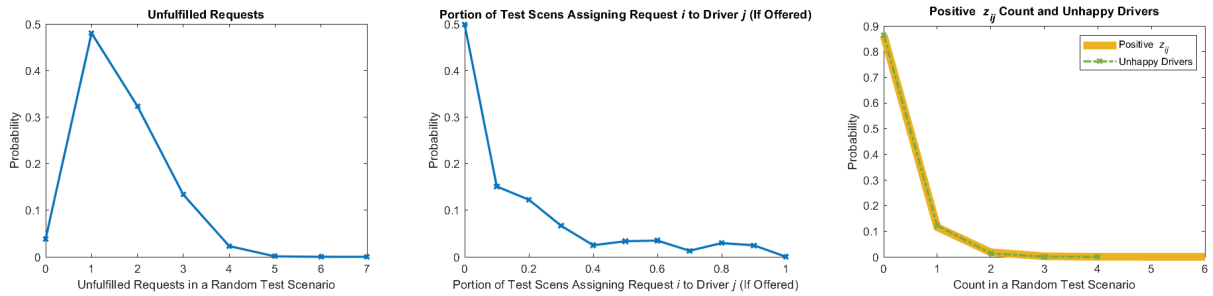



(a) (b) (c)

Figure 3: Graphs representing data from Stage I: Offered. The distribution of the menu sizes across the drivers is displayed in (a). The distribution of the number of times a given request $i$ is offered, i.e. $\sum_{j \in N} x_{ij}$, is displayed in (b). In (c), for each pair in a menu, the average $p_{ij}$ value i.e. $\frac{1}{n}\sum_{j \in N} p_{ij}$ of the offered request $i$ is compared to the number of times request $i$ is offered.

Figure 4 includes four graphs from the driver assignments (Stage III). Figure 4a displays the number of requests not assigned to a driver in the test scenarios for the corresponding menu. It is rare all twenty requests are fulfilled, but it is even more rare to have more than three of the requests be unfulfilled. Across the instances and test scenarios, the probability that a given request is successfully matched is 91.9%. Figure 4b lends some insight to the variability between the assignments of different test scenarios. Half of driver-request pairs in menus are assigned less than 10% of the time, so these pairs are more of a fringe option only assigned on rare occasions. Of the remaining half, many pairs are consistently assigned for most test scenarios, but an even larger portion are assigned neither rarely nor consistently: these pairs play to the uncertain behavior of the drivers and are assigned when the occasional opportunity arises. Figure 4c displays the distribution of unhappy drivers and of requests accepted by unhappy drivers in the test scenarios. In most scenarios, none of the participating drivers become unhappy drivers, and 95% of unhappy drivers have only accepted one request from their menu. A participating driver becomes an unhappy driver only 0.75% of the time, thus, using the negative binomial distribution formula, a driver

would be expected to participate 132 times before becoming an unhappy driver.




(a) (b) (c)

Figure 4: Graphs representing data from Stage III: Assignment. Graph (a) displays the distribution of the number of unfulfilled requests observed. While a given driver-request pair in a menu is offered 100% of the time, during performance analysis, it is actually assigned to a driver only some portion of the time. This portion is recorded and the distribution of portions among all pairs in a menu is plotted in (b). In (c), *Positive* $z_{ij}$ refers to the number of unhappy driver penalties activated by a given test scenario $s$, i.e. the number of nonzero $z^s_{ij}$ values. The distribution of positive $z^s_{ij}$ penalties and the distribution of the number of unhappy drivers are shown in (c).

The remaining analysis, displayed in Figure 5, includes data from Stages I-III and the Original parameter values. We examine six different metrics to measure the data from each stage. Displayed in 5a, the $p_{ij}$ metric is simply the $p_{ij}$ value of a given driver-request pair. *Request Trip Duration*, shown in 5b, is the time required to drive from the request origin to the request destination, and is the only metric used that is independent of the driver. *Extra Driving Time*, displayed in 5c, of a driver-request pair is defined as the duration of the driver's original trip subtracted from the duration of the trip if the driver fulfills the request. *Wait Time*, displayed in 5d, refers to the amount of time required for the driver to arrive at the request origin from their origin, that is, the wait time experienced by the rider. *Paid/Unpaid*, displayed in 5e, is calculated by splitting the driver's trip into *unpaid* time when they are driving alone and *paid* time when they have the given rider in their car. The amount of paid time is then divided by the amount of unpaid time. Shown in Figures 5f, the *Old Path/New Path* value of a pair is calculated by dividing the duration of the driver's original trip by the duration of the driver's trip if they accept the request.

With each sequential decision stage and for each metric besides request trip duration, the Figure 5 graphs reveal a shift in the metric distributions to further increase the density of driver-request pairs with more *desirable* values. That is, $p_{ij}$, the paid/unpaid ratio, and the old path/new path ratio shift to be higher, while the extra driving time and wait time shift to be lower. If the original set of requests is ranked for each driver for each of these five metrics, where a *high* ranking is more preferable, we observe that most of the offered driver-request pairs rank in the top ten ranks with respect to the driver. The ranking distribution of accepted driver-request pairs is even higher, and the assigned driver-request ranking distributions have the highest average with most assignments being within the top few ranks with respect to the driver. All five of these metrics shift to be more desirable likely because $p_{ij}$ is related to the other four metrics as each involves the driver's trip length. A higher $p_{ij}$ means that the driver will accept the request more often which means more matches and a higher objective value, explaining each stage's

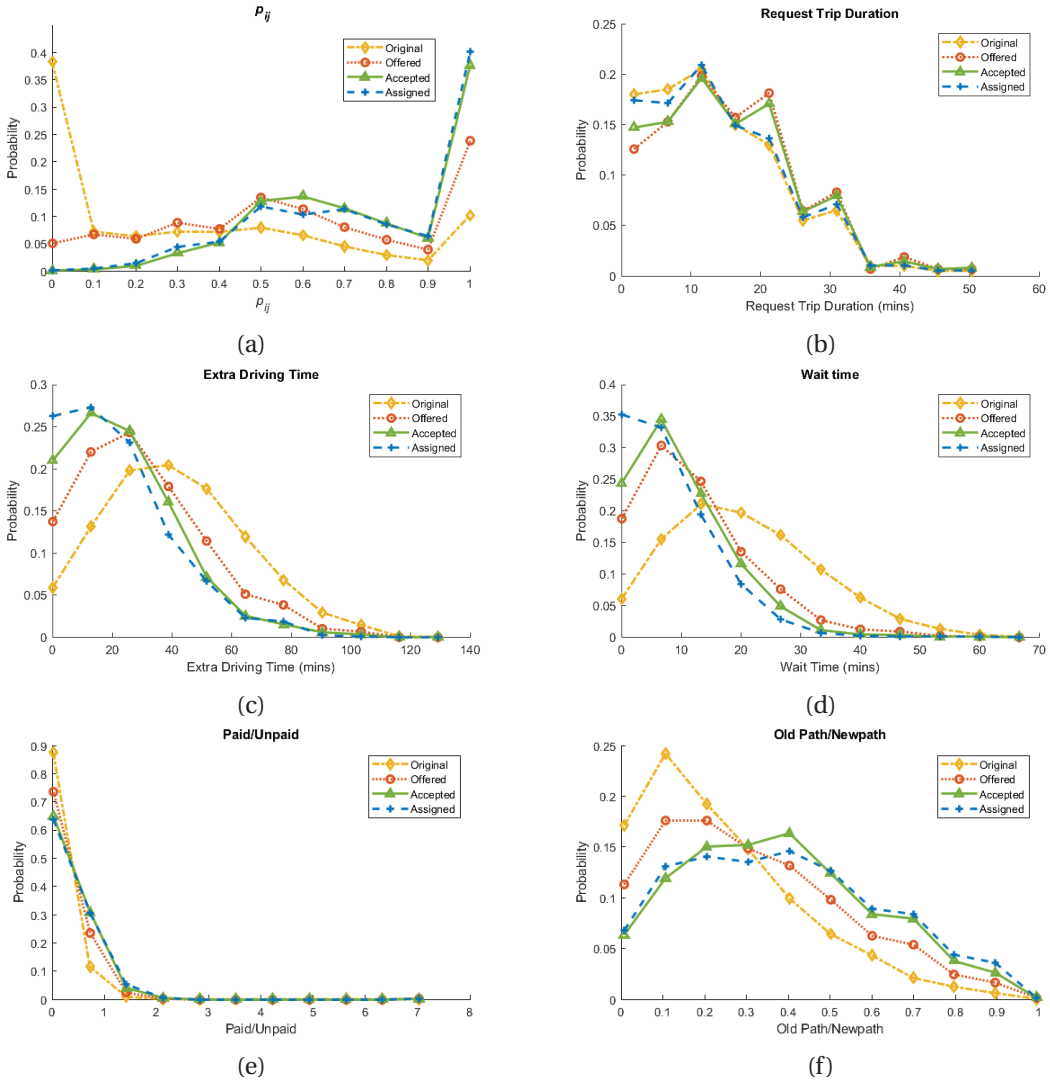



Figure 5: Distributions of the data from Stages I-III and the Original parameters. Graphs (a), (b), (c), (d), (e), and (f) display the distributions of $p_{ij}$ values, request trip duration, extra driving times, wait times, paid/unpaid values, and old path/new path values, respectively.

distribution sequential shift in this direction. The request trip duration has a slightly higher average for the offered and accepted stages, but the assigned distribution is very similar to the original distribution which has a slightly lower average. The assigned distribution is similar to the original distribution because a request's trip duration is independent of the driver it is matched to, and in most test scenarios almost all of the requests get matched, which means the set of matched requests is very similar to the original set of requests.

## 5.5 Phase 4: Impact of Unhappy Driver Penalty, Maximum Menu Sizes, and Driver Compensation

This final phase is composed of the three experiments in Sections 5.5.1-5.5.3: investigating the impact of the SLSF's penalty for unhappy drivers, comparing menus resulting from solving the SLSF with different maximum menu sizes, and analyzing the effects of different driver compensation levels.

### 5.5.1 Unhappy Driver Effect

Many of the constraints ((20),(21)) and almost half of the variables (the $\{z_{ij}^s\}$) in SLSF are dedicated to modeling unhappy drivers, which occur when driver(s) accept at least one request but are not assigned a request. We define an alternate solver, SLSFnoZ, given by the objective function (32) and constraints (15)-(19) and (22), that does not include the unhappy driver variables, constraints, or penalties.

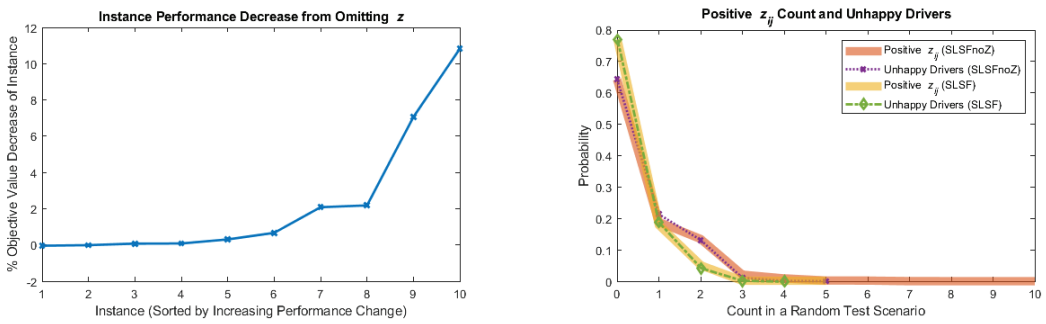
$$\max_{x,v} \quad \frac{1}{\sum_s \mathbb{P}(s)} \sum_{s\in\Omega} \mathbb{P}(s) \sum_{i\in M} \sum_{j\in N} c_{ij} v_{ij}^s \tag{32}$$

By omitting the variables and constraints that model unhappy drivers, the resulting formulation has a simpler structure than the original SLSF. For example, in SLSFnoZ there is no longer an incentive for the solver to only offer larger menus when necessary. The proof of Theorem 5.1 is given in Appendix C.

**Theorem 5.1.** *There always exists a menu with menu size $\theta$ for all drivers that is in the optimal solution set of SLSFnoZ.*

We generate ten sets of parameter instances, each with 20 drivers, 20 requests and 100 SAA scenarios. Then for each instance, we generate one menu using SLSF and one menu using SLSFnoZ with $\theta = 5$. Performance analysis is done for both menus across 5,000 test scenarios, generated using the union set of the SLSF menu and SLSFnoZ menu. For both methods, we use (23)-(29), namely, the unhappy driver penalties are calculated for SLSFnoZ menus as well as for the SLSF menus. Despite the property in Theorem 5.1, the distribution of menu sizes for SLSFnoZ menus has approximately the same menu size distribution as the SLSF menus. Drivers with menu size smaller than five all have well above five requests with a positive $p_{ij}$, indicating that the smaller menu size is not caused by a lack of options. Though the sizes of the outputted menus are of similar sizes, the SLSF and SLSFnoZ menus' compositions vary despite having the same input data. Out of the 1,116 distinct driver-request pairs offered between the 10 SLSF menus and the 10 SLSFnoZ menus, 70% of the pairs are offered in both the SLSF and SLSFnoZ menu corresponding to that pair's instance. This suggests the unhappy driver penalties do have a strong influence on menu recommendations.

The objective value comparison in Figure 6a shows that for six of the ten instances, the objective value estimate is essentially the same for the SLSF and SLSFnoZ menus, while the other four instances experience a decrease in objective estimate by not modeling unhappy drivers, with the biggest difference being over 10% lower for the SLSFnoZ objective. The SLSFnoZ does achieve a higher objective value than the SLSF menu in some scenarios, but the frequency and magnitude of the differences are slightly




(a) (b)

Figure 6: Graphs comparing the performance of the SLSF and SLSFnoZ menus. Graph (a) displays the percent difference in each instance's weighted average objective value, calculated by subtracting the SLSFnoZ instance objective from the SLSF objective and dividing the result by the SLSF objective value. Therefore, a positive value means that the SLSF objective is higher than the corresponding SLSFnoZ objective. Graph (b) displays the distribution of the number of unhappy drivers and the number of activated $z^s_{ij}$ penalties for SLSF and SLSFnoZ menus.

lower than the frequency and magnitude of scenarios where the SLSF menu performs better than the corresponding SLSFnoZ menu.

Another performance measure of interest is the amount of unhappy drivers and penalties incurred when the menu is not specifically designed to reduce them. Figure 6b demonstrates that there are indeed more unhappy drivers and more unhappy driver penalties: on average for the SLSFnoZ menus, the probability that a driver becomes an unhappy driver is approximately 2.6%. This probability, while low, is almost twice as likely as the estimated 1.4% chance a driver is unhappy under the generated SLSF menus. As for the number of unfulfilled requests, the distribution is similar between the two sets of menus, with the SLSF menus having an expected 5.5% of requests go unfulfilled, with the SLSFnoZ menus doing slightly better with an expected 4.8% of requests left unfulfilled.

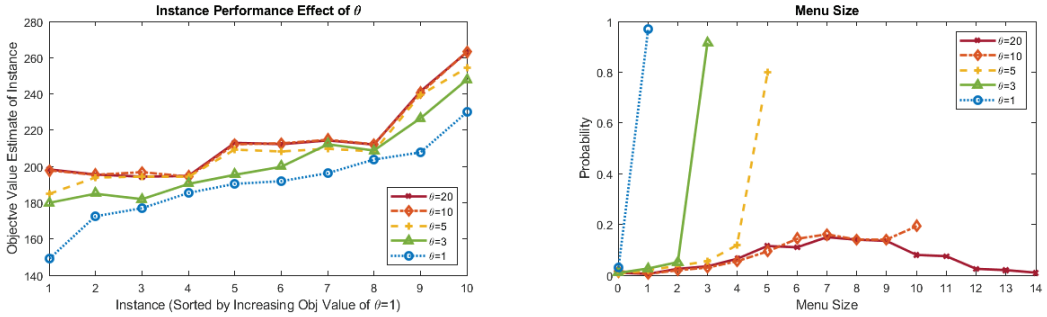
While the objective value estimates of SLSFnoZ menus are on average lower than SLSF menus and the expected number of unhappy drivers is higher, there is one crucial advantage to using SLSFnoZ to produce menus: runtime. Between the ten problem instances, the runtime of using SLSF has a median time of 13.67 seconds, but the average time is 84.02 seconds. Meanwhile, the runtime of using SLSFnoZ has a median time of 1.54 seconds with an average time of only 2.72 seconds.

### 5.5.2 Maximum Menu Size Effect

Our next experiment investigates the impact of the maximum menu size, $\theta$. In previous experiments we use a default $\theta$ value of 5 to give some options to the drivers while keeping menus brief enough for drivers to read and make their selections quickly. But how do the results change if we allow larger or smaller menus? To find out, we create ten sets of $20 \times 20$ parameter instances and use SLSF to generate a menu with $\theta = 1$, 3, 5, 10, and 20 for each instance. Menus from the same instance are evaluated over the same 5,000 test scenarios generated based on the union set of menus from that instance.

The objective performance of the different $\theta$ values is illustrated in Figure 7a. Increasing $\theta$ decreases

the tightness of constraint (15), so if enough test samples are used, a menu's performance estimate is guaranteed to be at least as high as a menu produced with the same input parameters and a lower $\theta$. We use 5,000 test samples, so this almost always occurs but is not always the case, as shown in Figure 7a. Objective performance improves as $\theta$ increases, but there is a diminishing return on value. In fact, the $\theta = 10$ and $\theta = 20$ curves are essentially indistinguishable, implying the freedom to have menus larger than 10 is minimally beneficial. The objective value of the $\theta = 5$ menus average only 2% lower than the corresponding $\theta = 10$ menu in the same instance, meaning performance is not degraded much by enforcing reasonably small menus.




(a) (b)

Figure 7: Graphs comparing the objective values and menu sizes of the menus produced with different $\theta$ values. In (a), the instance performance estimate of the five menus in the same instance are plotted on the same $x$-axis value, so direct comparison between the objective values of menus in the same instance is possible. Rather than display the instances in a random order, they are plotted by increasing objective estimate value of the $\theta = 1$ menu of each instance. Graph (b) displays the probability distribution of driver menu size for the five $\theta$ values.

Data on the menu sizes is shown in Figure 7b. When $\theta = 20$, constraint (15) is no longer an active constraint as the SLSF program recommends menus with fewer than 20 requests for all drivers, with the largest menu containing 14 requests. Given 98% of drivers are not given menus containing all requests that are accepted by them in at least one SAA scenario, i.e. the largest possible menu for that driver, our experiment shows that even when unrestricted, menus can be narrowed down to an expected size of 7.4 requests. When restricted to a maximum of 10 menu items, the distribution of menu sizes is very similar to the $\theta = 20$ menus, with the exception of menu size 10: the probability that a $\theta = 10$ driver's menu size is exactly 10 is approximately the probability that a $\theta = 20$ driver's menu size is at least 10. For menus where $\theta \leq 5$, the maximum menu size constraint is more constricting, with 97%, 91%, and 80% of menus being the maximum size for $\theta = 1$, 3, and 5 respectively.

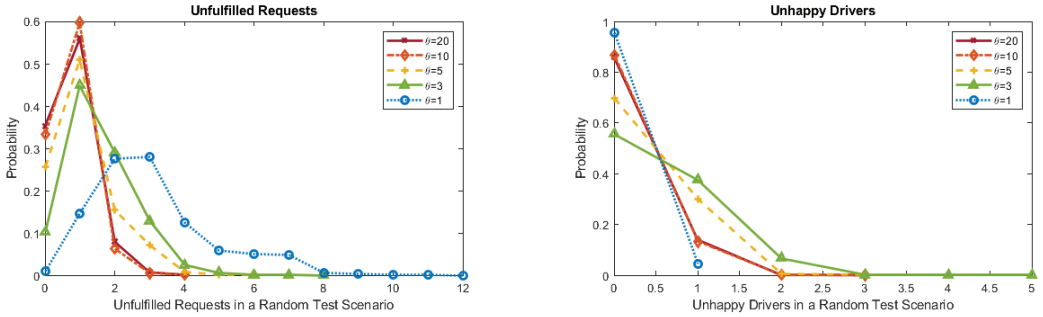
As for the contents of these menus, there is a strong but not perfect overlap between requests recommended with varying $\theta$, as exhibited in Table 5. The smallest overlap between two $\theta$ values is 86.6%, and the portion of overlap between two menus strictly increases as the larger maximum menu size increases. This is likely because as $\theta$ increases, the menu is able to have more requests so it has more chances to include the requests from the smaller $\theta$'s menu.

Figure 8 provides some insights as to the unfulfilled requests distributions and the unhappy driver

Table 5: Recommendation overlap in menus in the same instance made with different $\theta$ values, requiring that $\theta_1 < \theta_2$. For example, the 2nd row ($\theta_1 = 3$) 3rd column ($\theta_2 = 10$) entry signifies that of the set of driver-request pairs offered in a $\theta = 3$ menu, on average 92.2% of them are also offered in the corresponding $\theta = 10$ menu.

| | Portion of $\theta_1$ menu pairs in $\theta_2$ menus | | | |
|---|---|---|---|---|
| | $\theta_2 = 3$ | $\theta_2 = 5$ | $\theta_2 = 10$ | $\theta_2 = 20$ |
| $\theta_1 = 1$ | 0.876 | 0.923 | 0.969 | 0.974 |
| $\theta_1 = 3$ | – | 0.866 | 0.922 | 0.943 |
| $\theta_1 = 5$ | – | – | 0.905 | 0.913 |
| $\theta_1 = 10$ | – | – | – | 0.915 |

distributions. There are marked improvements in the number of unfulfilled requests as $\theta$ increases from 1 to 3 to 5 to 10, then there are slightly more unfulfilled requests as $\theta$ increases to 20. The probability that a driver becomes an unhappy driver in this experiment is 0.23%, 2.6%, 1.6%, 0.7%, and 0.7% for $\theta = 1$, 3, 5, 10, and 20 respectively. For the unhappy drivers distributions, the $\theta = 1$ menus have by far the fewest unhappy drivers, as the menus so rarely recommend a request to more than one driver. The $\theta = 10$ and $\theta = 20$ menus have the next fewest unhappy drivers, with essentially the same distribution, followed by $\theta = 5$ menus. Larger menus are expected to have more drivers that accept at least a few requests, so even if there is overlap in accepted requests, there will be more assignment options, which can reduce or eliminate the number of unhappy drivers. The $\theta = 3$ menus have the highest number of unhappy drivers, because the menus are large enough to have overlap in accepted requests but small enough that there are not many assignment options.




(a) (b)

Figure 8: Graph (a) displays the distributions of the number of unfulfilled requests for the menus with each of the five different $\theta$ values. Graph (b) displays the distributions of the number of unhappy drivers for the menus with each of the five different $\theta$ values.

The runtime information for this experiment is shown in Table 6. The variability is high for all $\theta$ values besides $\theta = 1$, with the median time being much lower than the average time for these $\theta$ values. The $\theta = 3$ menus require the most time by far, and $\theta = 10$ menus have the lowest average time. Overall, our default value $\theta = 5$ is a well-rounded choice, as there are some disadvantages to having a low $\theta$ value, such as a higher number of unfulfilled requests, while having a higher $\theta$ value only slightly improves

performance in the metrics we examined, but requires the drivers to spend more time going through their menu to make decisions.

Table 6: Mean and Median runtimes of SLSF for each $\theta$ value across the ten instances.

| | | $\theta = 1$ | $\theta = 3$ | $\theta = 5$ | $\theta = 10$ | $\theta = 20$ |
|---|---|---|---|---|---|---|
| Time (seconds) | Mean | 10.65 | 97.44 | 14.67 | 5.64 | 9.48 |
| | Median | 9.65 | 34.64 | 7.24 | 2.43 | 2.35 |

### 5.5.3 Driver Compensation

Our final experiment tests the efficacy of the platform under different driver compensation policies. In previous experiments, the wage for drivers is 80% of the fare. In this experiment we create and analyze menus using six different compensation levels: a driver's wage for completing request $i$ can be 40%, 50%, 60%, 70%, 80%, or 90% of $fare_i$, where $fare_i$ follows the formula in Table 2. We create ten parameter instances, which include driver and request OD pairs, the $d_{ij}$, the random match bonus component of each $c_{ij}$, i.e., all parameters except for the $c_{ij}$ and $p_{ij}$ as these depend on the wage level of the drivers. Once the parameters have been generated, for each wage level, $c_{ij}$ and $p_{ij}$ are calculated, a corresponding set of 100 training scenarios are generated, SLSF produces a menu with $\theta = 5$, and 5,000 test scenarios are generated and used to complete performance analysis. The test scenarios depend on the $p_{ij}$ values, so the set of test scenarios is different for each menu.

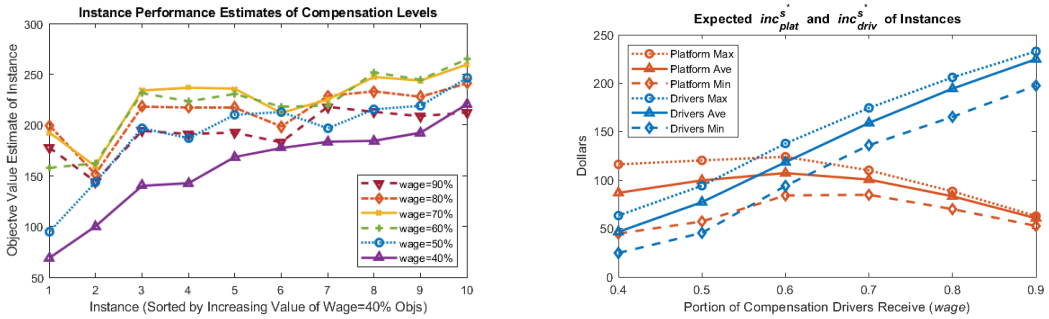
The objective values of the six compensation levels across the ten instances are displayed in Figure 9a. The 40% wage menus have the lowest objective value in every instance, and the 50% and 90% wage menus are always out-performed by at least one other wage level menu. The 60% wage menus have the highest objective value out of any menu in five of the ten instances, followed by the 70% wage menus that are the highest in three of the ten instances. The expected income of the drivers and the platform is displayed in Figure 9b. The drivers' total income $inc^{s^*}_{driv}$ in dollars for test scenario $s^*$ is the portion of the fare that the driver receives summed across all assigned driver-request pairs. The platform's total income $inc^{s^*}_{plat}$ in dollars for test scenario $s^*$ is calculated as the \$1.85 booking fee plus the portion of the fare that is not given to the driver. The formulas for drivers and platform income are given by (33) and (34) respectively.

$$inc^{s^*}_{driv} = \sum_{i \in M} \sum_{j \in N} (wage \cdot fare_i) v^{s^*}_{ij} \tag{33}$$

$$inc^{s^*}_{plat} = \sum_{i \in M} \sum_{j \in N} \left(1.85 + (1 - wage) \cdot fare_i\right) v^{s^*}_{ij} \tag{34}$$

Figure 9b shows that the drivers' total income grows with the compensation level, as they are paid a higher percent of the fare and the willingness to participate increases so more requests are assigned, as confirmed by Figures 10 and 11a. Platform income is the highest when the drivers are compensated 60%

of the fare, and the variability of expected income across the ten instances is higher for the lower wage level menus.



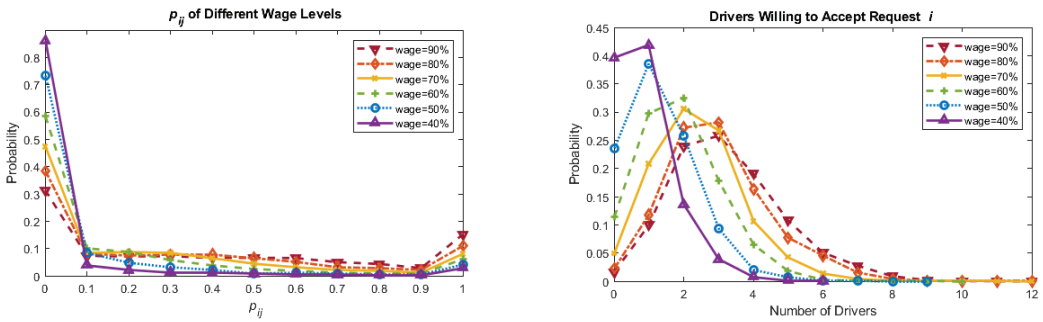

(a) (b)

Figure 9: Graphs comparing the objective values and expected dollars received of the menus produced with different wage levels. In (a), the instance performance estimate of the six menus in the same instance are plotted on the same $x$-axis value, so direct comparison between the objective values of menus in the same instance is possible. They are plotted by increasing objective estimate value of the wage= 40% menu of each instance. Graph (b) displays the expected $inc^{s^*}_{plat}$ and $inc^{s^*}_{driv}$ for each wage level. The expected dollar amounts are calculated across the 5,000 test scenarios for each of the ten instances. The maximum, average, and minimum values in the graph correspond to the highest, average, and lowest expected platform and driver income values among the ten instances respectively.

Figure 10 provides insight to how the drivers' compensation affects their behavior. Figure 10a displays the $p_{ij}$ distributions for each wage level, and 10b displays the distributions of the number of drivers willing to accept each request. These graphs illustrate the magnitude by which the potential recommendation and assignment options dwindle as reducing wages results in fewer drivers being compensated enough to accept requests. If drivers are paid 60% or less of the fare, a majority of driver-request pairs have a $p_{ij}$ value of 0, and of the pairs that have a positive $p_{ij}$ value, most have $p_{ij} \leq 0.5$. This results in few drivers being willing to accept a given request, for example with 40% compensation, on average 40% of requests in a scenario have zero drivers willing to accept those requests. Yet, if drivers are paid 80% or 90% of the fare, on average only 2.3% and 1.6% of requests have zero drivers willing to accept them, respectively.

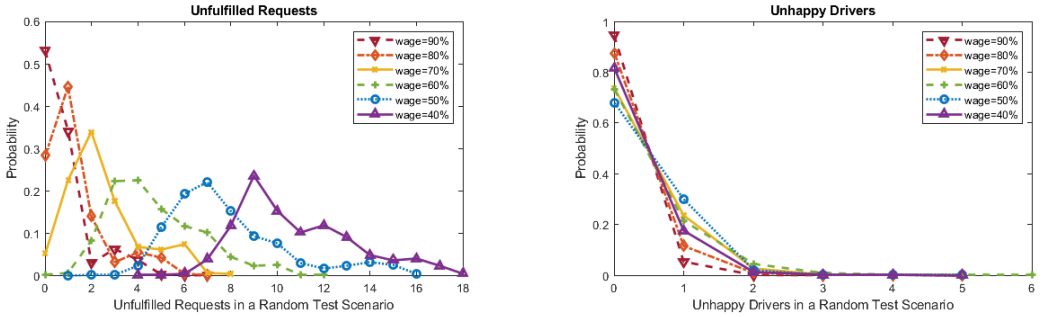
The unfilled requests and unhappy driver distributions are displayed in Figure 11. The number of unfilled requests increases as drivers are paid less, with an expected 3.7% of requests unfulfilled in the 90% wage menus and an expected 54.0% of requests unfulfilled in the 40% wage menus. The number of unfulfilled requests for low wage menus is quite high. However, the probability that the drivers have a negative experience by becoming an unhappy driver is fairly low, with the highest expected probability of becoming an unhappy driver between the six wage levels being 1.7% for the 50% wage menus. The probability of becoming an unhappy driver decreases as the wage level increases, because more requests are accepted so there are more assignment options that reduce the number of unhappy drivers. An exception is the lower number of unhappy drivers for 40% wage menus, likely because the number of accepted requests are low enough that there is less overlap in accepted requests between drivers.

Considering the results of our analysis, neither the platform nor the drivers benefit from paying




(a) (b)

Figure 10: Graph (a) displays the distributions of the $p_{ij}$ values of all driver-request pairs in the ten instances for each wage level. Graph (b) displays the distributions of the number of drivers in the random test scenarios that are willing to accept a given request across the ten instances. The number of willing drivers is calculated regardless of which drivers' menus contain the given request. Test scenarios are used instead of SAA scenarios to avoid the bias of mutated scenarios.




(a) (b)

Figure 11: Graphs (a) and (b) display the distributions of the number of unfulfilled requests and the number of unhappy drivers for each of the six different wage levels, respectively.

drivers only 40% or 50% of the fare. The loss in driver willingness to participate means the platform has few recommendation and assignment options, so even though they keep a higher percentage of the fare, they lose out due to a large percent of requests being unfulfilled. On the other hand, paying the drivers 90% of the fare provides quality service with high fulfillment rates and happy drivers, but the platform's revenue and objective value are lower than they could be under a different wage level. The platform's objective and revenue are highest at 60% compensation, but the quality of service suffers and could lead to the platform not having a satisfied customer and driver base in the long term. While minimally sacrificing objective value and income, increasing the compensation level to 70% notably improves the quality of service for drivers and customers. To make a stronger commitment to the customer and driver experience in the long term, the platform could choose to raise the compensation to 80% or higher.

# 6 Conclusions and Future Directions

In this paper we formulate a mathematical model to create personalized menus of requests recommended to drivers participating with a ridesharing or crowdsourced delivery platform. This is the first-known method able to create supply-side hierarchically-motivated menus under stochastic driver selections. For computational viability, we reformulate our bilevel optimization model as a single level MILP that is equivalent to our original formulation when drivers are allowed to signal their willingness to accept an unlimited number of requests. We directly capture the stochastic nature of the driver selection behavior in the optimization model using the Sample Average Approximation (SAA) method. The Chicago Regional Transportation Network data is used as a ridesharing case study.

Because of the platform's limited knowledge of drivers' request preferences, stochastically optimizing the menus sent to drivers yields efficient allocation of driver participation that captures uncertainty in supplier selections and interdependencies among drivers not achievable by generating menus via deterministic optimization nor benchmark policies. This is confirmed by our comparison of our stochastic approach with deterministic and current approaches, as our formulation has a better objective value, more matches, fewer unhappy drivers, and a higher platform profit than the compared alternative methods. Out of a maximum menu size of 5, the menu size average is 4.6 requests, indicating the platform benefits from recommending multiple requests to each driver when the platform's knowledge about driver selections is stochastic. On average, 91.9% of requests are successfully matched, meaning that drivers being allowed the autonomy to decide what request(s) to accept from a menu does not prevent the platform from fulfilling a high percentage of incoming requests. Increasing the maximum menu size to be 10 or 20 only slightly increases solution performance but requires the drivers to spend more time making selections on larger menus. Compensating drivers 40% or 50% of the fare can result in over half of requests remaining unmatched, as drivers are unwilling to accept most requests. Compensating 60% or 70% yields the highest utility to the platform, though compensating 80% or 90% provides the highest quality of service to drivers and customers.

Computational tests recommend a training sample size for inputted SAA scenarios and a test sample size for performance analysis. Removing the unhappy driver variables and constraints from our SLSF formulation yields an approach that performs nearly as well as SLSF and significantly reduces the runtime. While the recommended training sample size used for our analysis may result in a runtime longer than is suited for practical implementation of ride sharing applications, using a smaller training sample size or removing the unhappy driver variables are both viable options for producing quality menus quickly. With this in mind, a platform with around 20 drivers and 20 requests in each geographical decision epoch can attain a dispatch window of 5 seconds to decide driver menus drivers (see Sections 5.2 and 5.5.1) and then an assignment window of around 0.2 seconds (see Appendix B). The remaining time in the epoch would go to drivers to make their selections, making a decision window of almost 25 seconds for a 30 second epoch or almost 55 seconds for a 60 second epoch.

Future extensions of this work have a number of possible directions. Future research could develop tractable methodologies when drivers respond back to the platform with their preferred requests to fulfill, rather than their general willingness. Alternate assumptions for driver behavior models could be

explored and implemented, or selections could be based on driver preference data previously gathered by the platform. Alternate data sets could be used to learn drivers' request preferences over time. Our approach currently models a single decision epoch, so a next step could include multiple epochs. Finally, the characteristics of optimal menus found via our proposed approach could be exploited in the design of heuristics that produces quality menus more quickly.

## Acknowledgements

This work was partially funded by the National Science Foundation CAREER award 1751801 and by the Johnson & Johnson WiSTEM2D program.

## References

Agatz, Niels, Erera, Alan, Savelsbergh, Martin, & Wang, Xing. 2012. Optimization for dynamic ride-sharing: A review. *European Journal of Operational Research*, **223**(2), 295–303. DOI: 10.1016/j.ejor.2012.05.028.

Alnaggar, Aliaa, Gzara, Fatma, & Bookbinder, James H. 2019. Crowdsourced Delivery: A Review of Platforms and Academic Literature. *Omega*, 102139. DOI: 10.1016/j.omega.2019.102139.

Arslan, Alp M, Agatz, Niels, Kroon, Leo, & Zuidwijk, Rob. 2018. Crowdsourced Delivery– A dynamic pickup and delivery problem with ad hoc drivers. *Transportation Science*, **53**(1), 222–235. DOI: 10.1287/trsc.2017.0803.

Banerjee, Siddhartha, Riquelme, Carlos, & Johari, Ramesh. 2015. Pricing in ride-share platforms: A queueing-theoretic approach. *Available at SSRN 2568258*. DOI: 10.2139/ssrn.2568258.

Barbosa, Miguel Moreira da Silva Lima. 2019. *A data-driven compensation scheme for last-mile delivery with crowdsourcing*. M.Phil. thesis, University of Porto. https://repositorio-aberto.up.pt/bitstream/10216/124212/2/367287.pdf (Retrieved on May 14 2020).

Bimpikis, Kostas, Candogan, Ozan, & Saban, Daniela. 2019. Spatial pricing in ride-sharing networks. *Operations Research*, **67**(3), 744–769. DOI: 10.1287/opre.2018.1800.

Captain, Sean. 2019. *Drive for Instacart and You Could Make $29.05 for an Hour's Work-—or $2.74*. Fast Company. https://www.fastcompany.com/90309043/instacart-drivers-say-this-data-proves-theyre-still-being-underpaid?partner=rss&utm_source=rss&utm_medium=feed&utm_campaign=rss+fastcompany&utm_content=rss (Retrieved on April 19, 2021).

Chen, Lu, Gao, Yunjun, Liu, Zixian, Xiao, Xiaokui, Jensen, Christian S, & Zhu, Yifan. 2018. PTRider: a price-and-time-aware ridesharing system. *Proceedings of the VLDB Endowment*, **11**(12), 1938–1941. DOI: 10.14778/3229863.3236229.

Cheng, Xi, Gou, Qinglong, Yue, Jinfeng, & Zhang, Yan. 2019. Equilibrium decisions for an innovation crowdsourcing platform. *Transportation Research Part E: Logistics and Transportation Review*, **125**, 241–260. DOI: 10.1016/j.tre.2019.03.006.

Chicago Area Transportation Study (CATS). 2016. *Chicago Regional Network*. https://github.com/bstabler/TransportationNetworks/tree/master/ch regional (Retrieved on August 1, 2019).

Cook, Heather. 2019. *Amazon Flex App: Everything You Need to Know*. Ridester. https://www.ridester.com/amazon-flex-app/ (Retrieved on April 30, 2020).

Delfino, Devon. 2017. *Make Money As an Instacart Shopper: What to Expect.* Nerdwallet. https://www.nerdwallet.com/blog/finance/make-money-as-an-instacart-shopper-what-to-expect/ (Retrieved on April 30, 2020).

Di Febbraro, A, Gattorna, E, & Sacco, N. 2013. Optimization of dynamic ridesharing systems. *Transportation research record*, **2359**(1), 44–50. DOI: 10.3141/2359-06.

DoorDash. 2019. *How Do I Claim Drive Orders in Advance.* https://help.doordash.com/dashers/s/article/How-do-I-claim-Drive-orders-in-advance?language=en_US (Retrieved April 30, 2020).

Einav, Liran, Farronato, Chiara, & Levin, Jonathan. 2016. Peer-to-Peer Markets. *Annual Review of Economics*, **8**, 615–635. DOI: 10.1146/annurev-economics-080315-015334.

EstimateFares.com. 2019. *Chicago Lyft Rates.* EstimateFares.com. https://estimatefares.com/rates/chicago (Retrieved on August 1, 2019).

Furuhata, Masabumi, Dessouky, Maged, Ordóñez, Fernando, Brunet, Marc-Etienne, Wang, Xiaoqing, & Koenig, Sven. 2013. Ridesharing: The State-of-the-Art and Future Directions. *Transportation Research Part B: Methodological*, **57**, 28–46. DOI: 10.1016/j.trb.2013.08.012.

Gdowska, Katarzyna, Viana, Ana, & Pedroso, João Pedro. 2018. Stochastic last-mile delivery with crowdshipping. *Transportation Research Procedia*, **30**, 90–100. DOI: 10.1016/j.trpro.2018.09.011.

Gesing, Ben. 2017. *Sharing Economy Logistics.* DHL Trend Research. https://www.dhl.com/content/dam/downloads/g0/about_us/logistics_insights/dhl_ trendreport_sharing_economy_final.pdf (Retrieved on April 30,2020).

Godil, Ayesha. 2020. *I Became a Postmate for a Week to Fulfill Your Food Delivery Needs.* Spoon University. https://spoonuniversity.com/lifestyle/postmates-driver-experience-i-became-a-postmate-for-a-week-to-fulfill-your-food-delivery-needs (Retrieved on April 30, 2020).

Helling, Brett. 2019. *Uber Fees: How Much Does Uber Pay, Actually? (With Case Studies).* Ridester. https://www.ridester.com/uber-fees/ (Retrieved on April 30, 2020).

Hong, Huiting, Li, Xin, He, Daqing, Zhang, Yiwei, & Wang, Mingzhong. 2019. Crowdsourcing Incentives for Multi-Hop Urban Parcel Delivery Network. *IEEE Access*, **7**, 26268–26277. DOI: 10.1109/ACCESS.2019.2896912.

Hou, Liwen, Li, Dong, & Zhang, Dali. 2018. Ride-matching and routing optimisation: Models and a large neighbourhood search heuristic. *Transportation Research Part E: Logistics and Transportation Review*, **118**, 143–162. DOI: 10.1016/j.tre.2018.07.003.

JC. 2019. *Have You Been Sidelined by Uber.* Ridester. https://www.ridester.com/have-you-been-sidelined-by-uber/ (Retrieved on April 30, 2020).

Kleywegt, Anton J, Shapiro, Alexander, & Homem-de Mello, Tito. 2002. The Sample Average Approximation Method for Stochastic Discrete Optimization. *SIAM Journal on Optimization*, **12**(2), 479–502. DOI: 10.1137/S1052623499363220.

Le, Tho V, Stathopoulos, Amanda, Van Woensel, Tom, & Ukkusuri, Satish V. 2019. Supply, demand, operations, and management of crowd-shipping services: A review and empirical evidence. *Transportation Research Part C: Emerging Technologies*, **103**, 83–103. DOI: 10.1016/j.trc.2019.03.023.

Lei, Chao, Jiang, Zhoutong, & Ouyang, Yanfeng. 2019. Path-based dynamic pricing for vehicle allocation in ridesharing systems with fully compliant drivers. *Transportation Research Part B: Methodological.* DOI: 10.1016/j.trb.2019.01.017.

Li, Minne, Qin, Zhiwei, Jiao, Yan, Yang, Yaodong, Wang, Jun, Wang, Chenxi, Wu, Guobin, & Ye, Jieping. 2019a. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. *Pages 983–994 of: The World Wide Web Conference.* DOI: 10.1145/3308558.3313433.

Li, Yafei, Wan, Ji, Chen, Rui, Xu, Jianliang, Fu, Xiaoyi, Gu, Hongyan, Lv, Pei, & Xu, Mingliang. 2019b. Top-$k$ Vehicle Matching in Social Ridesharing: A Price-aware Approach. *IEEE Transactions on Knowledge and Data Engineering*. DOI: 10.1109/TKDE.2019.2937031.

Lin, Qiulin, Xu, Wenjie, Chen, Minghua, & Lin, Xiaojun. 2019. A Probabilistic Approach for Demand-Aware Ride-Sharing Optimization. *Pages 141–150 of: Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*. DOI: 10.1145/3323679.3326512.

Long, Jiancheng, Tan, Weimin, Szeto, WY, & Li, Yao. 2018. Ride-sharing with travel time uncertainty. *Transportation Research Part B: Methodological*, **118**, 143–171. DOI: 10.1016/j.trb.2018.10.004.

Lowalekar, Meghna, Varakantham, Pradeep, & Jaillet, Patrick. 2018. Online spatio-temporal matching in stochastic and dynamic domains. *Artificial Intelligence*, **261**, 71–112. DOI: 10.1016/j.artint.2018.04.005.

Luo, Qi, & Saigal, Romesh. 2017. Dynamic pricing for on-demand ride-sharing: A continuous approach. *Available at SSRN 3056498*. DOI: 10.2139/ssrn.3056498.

Lyft. 2018. *How to Give a Lyft Ride*. Lyft. https://help.lyft.com/hc/en-us/articles/115013080028-How-to-give-a-Lyft-ride#app (Retrieved on April 30, 2020).

Marshall, Aarian. 2020. *Uber Changes Its Rules, and Drivers Adjust Their Strategies*. Wired. https://www.wired.com/story/uber-changes-rules-drivers-adjust-strategies/. (Retrieved on April 27, 2021).

Masoud, Neda, & Jayakrishnan, R. 2017. A decomposition algorithm to solve the multi-hop Peer-to-Peer ride-matching problem. *Transportation Research Part B: Methodological*, **99**, 1–29. DOI: 10.1016/j.trb.2017.01.004.

McInerney, James, Rogers, Alex, & Jennings, Nicholas R. 2013. Learning periodic human behaviour models from sparse data for crowdsourcing aid delivery in developing countries. *arXiv preprint arXiv:1309.6846*. https://arxiv.org/ftp/arxiv/papers/1309/1309.6846.pdf (Retrieved on May 12, 2020).

Mofidi, Seyed Shahab, & Pazour, Jennifer A. 2019. When is it beneficial to provide freelance suppliers with choice? A hierarchical approach for peer-to-peer logistics platforms. *Transportation Research Part B: Methodological*, **126**, 1–23. DOI: 10.1016/j.trb.2019.05.008.

Mourad, Abood, Puchinger, Jakob, & Chu, Chengbin. 2019. A Survey of Models and Algorithms for Optimizing Shared Mobility. *Transportation Research Part B: Methodological*. DOI: 10.1016/j.trb.2019.02.003.

Najmi, Ali, Rey, David, & Rashidi, Taha H. 2017. Novel dynamic formulations for real-time ride-sharing systems. *Transportation Research Rart E: logistics and transportation review*, **108**, 122–140. DOI: 10.1016/j.tre.2017.10.009.

Newton, Casey. 2014. *TaskRabbit is Blowing Up its Business Model and Becoming the Uber for Everything*. The Verge. https://www.theverge.com/2014/6/17/5816254/taskrabbit-blows-up-its-auction-house-to-offer-services-on-demand (Retrieved on April 30, 2020).

Nourinejad, Mehdi, & Ramezani, Mohsen. 2019. Ride-Sourcing modeling and pricing in non-equilibrium two-sided markets. *Transportation Research Part B: Methodological*. DOI: 10.1016/j.trb.2019.05.019.

Özkan, Erhun, & Ward, Amy R. 2020. Dynamic Matching for Real-Time Ride Sharing. *Stochastic Systems*, **10**(1), 29–70. DOI: 10.1287/stsy.2019.0037.

Postmates. 2019. *Accepting and Completing Deliveries*. https://support.postmates.com/fleet/articles/220058207-article-Accepting-and-completing-deliveries (Retrieved on April 30, 2020).

Punel, Aymeric, & Stathopoulos, Amanda. 2017. Modeling the acceptability of crowdsourced goods deliveries: Role of context and experience effects. *Transportation Research Part E: Logistics and Transportation Review*, **105**, 18–38. DOI: 10.1016/j.tre.2017.06.007.

Qin, Zhiwei, Tang, Xiaocheng, Jiao, Yan, Zhang, Fan, Xu, Zhe, Zhu, Hongtu, & Ye, Jieping. 2020. Ride-Hailing Order Dispatching at DiDi via Reinforcement Learning. *INFORMS Journal on Applied Analytics*, **50**(5), 272–286. DOI:10.1287/inte.2020.1047.

Ride Share Guy. 2019. *Uber Finally Lets Drivers See Where Passengers Are Going!* Ride Share Guy. https://therideshareguy.com/uber-rolling-out-new-driver-features/ (Retrieved on December 19, 2019).

Rideguru. 2018. *How Can You Determine the Rider's Destination Before Accepting the Ride?* Rideguru. https://ride.guru/lounge/p/how-can-you-determine-the-riders-destination-before-accepting-the-ride (Retrieved on April 30, 2020).

Roadie. 2018. *Drivers.* Roadie. https://www.roadie.com/drivewithroadie (Retrieved on April 30, 2020).

Schreieck, Maximilian, Safetli, Hazem, Siddiqui, Sajjad Ali, Pflügler, Christoph, Wiesche, Manuel, & Krcmar, Helmut. 2016. A matching algorithm for dynamic ridesharing. *Transportation Research Procedia*, **19**, 272–285. DOI: 10.1016/j.trpro.2016.12.087.

Sheffi, Yosef. 1985. *Urban transportation networks.* Prentice-Hall, Englewood Cliffs, NJ. http://web.mit.edu/sheffi/www/selectedMedia/sheffi_urban_trans_networks.pdf (Retrieved on March 3, 2019).

Soto Setzke, David, Pflügler, Christoph, Schreieck, Maximilian, Fröhlich, Sven, Wiesche, Manuel, & Krcmar, Helmut. 2017. Matching drivers and transportation requests in crowdsourced delivery systems. *In: Twenty-third Americas Conference on Information Systems.* https://pdfs.semanticscholar.org/8192/1e5ff7c5e4a63cfab432299a02b71d9b3fcc.pdf?_ga=2.220587288.1808169234.1588275359-636657769.1569443277 (Retrieved on April 10, 2019).

Stiglic, Mitja, Agatz, Niels, Savelsbergh, Martin, & Gradisar, Mirko. 2015. The benefits of meeting points in ride-sharing systems. *Transportation Research Part B: Methodological*, **82**, 36–53. DOI: 10.2139/ssrn.2567274.

Wang, Hai, & Yang, Hai. 2019. Ridesourcing systems: A framework and review. *Transportation Research Part B: Methodological*, **129**, 122–155. DOI: 10.1016/j.trb.2019.07.009.

Wang, Yuan, Zhang, Dongxiang, Liu, Qing, Shen, Fumin, & Lee, Loo Hay. 2016. Towards enhancing the last-mile delivery: An effective crowd-tasking model with scalable solutions. *Transportation Research Part E: Logistics and Transportation Review*, **93**, 279–293. DOI: 10.1016/j.tre.2016.06.002.

Zha, Liteng, Yin, Yafeng, & Du, Yuchuan. 2018. Surge pricing and labor supply in the ride-sourcing market. *Transportation Research Part B: Methodological*, **117**, 708–722. DOI: 10.1016/j.trb.2017.09.010.

# A Proof of Theorem 4.1

**Theorem 4.1.** *Our SLSF formulation is equivalent to BLSF when* (11) *is relaxed and scenarios are categorized using sets of* $\hat{y}^s_{ij}$ *values.*

*Proof.* First, we prove any feasible solution to BLSF is feasible in SLSF with the same objective value, and then prove that any feasible solution to SLSF is feasible in BLSF with the same objective value. We assume that $\mathbb{P}(s)$, $c_{ij}$, $d_{ij}$, $\ell$, $\theta$, $q_j$, and $\hat{y}^s_{ij}$ for all $i \in M$, $j \in N$, and $s \in \Omega$ are fixed and the same for BLSF and SLSF. BLSF does not contain $\hat{y}^s_{ij}$, but we use the information from $\hat{y}^s_{ij}$ to determine if $u^s_{ij} - \phi^s_j > 0$

which allows the program to solve the lower level problem. Let $[\boldsymbol{x}^*, \boldsymbol{y}^*, \boldsymbol{v}^*, \boldsymbol{z}^*]$ represent a full set of $x_{ij}$, $y_{ij}^s$, $v_{ij}^s$, and $z_{ij}^s$ values for all $i \in M$, $j \in N$, and $s \in \Omega$.

Suppose $[\boldsymbol{x}^*, \boldsymbol{y}^*, \boldsymbol{v}^*, \boldsymbol{z}^*]$ is feasible in BLSF. It's clear that constraints (15), (18), (19), (21), and (22) are met. The remaining constraints are (16), (17), and (20). We have that $y_{ij}^s = 1$ if $x_{ij} = \hat{y}_{ij}^s = 1$ and equals 0 otherwise, and $x_{ij}$ and $\hat{y}_{ij}^s$ are both restricted to binary values. We therefore know that the logic constraints $y_{ij}^s \leq x_{ij}$, $y_{ij}^s \leq \hat{y}_{ij}^s$, and $y_{ij}^s \geq -1 + \hat{y}_{ij}^s + x_{ij}$ hold for any feasible values of $x_{ij}$ and $\hat{y}_{ij}^s$. Using these along with constraints (3) and (6), it follows that $v_{ij}^s \leq y_{ij}^s \leq x_{ij}$, that $v_{ij}^s \leq y_{ij}^s \leq \hat{y}_{ij}^s$, and that

$$z_{ij}^s \geq -1 + y_{ij}^s + (1 - \sum_{k \in M} v_{kj}^s) \geq -2 + \hat{y}_{ij}^s + x_{ij} + (1 - \sum_{k \in M} v_{kj}^s), \tag{A.1}$$

so (16), (17) and (20) are upheld, respectively. The objective value corresponding to $[\boldsymbol{x}^*, \boldsymbol{y}^*, \boldsymbol{v}^*, \boldsymbol{z}^*]$ in SLSF is the same as it is in BLSF, as the objective function and the value of the variables in the objective function are the same.

Next, we let $[\boldsymbol{x}^*, \boldsymbol{y}^*, \boldsymbol{v}^*, \boldsymbol{z}^*]$ be feasible in SLSF. This formulation doesn't contain $y_{ij}^s$, so instead $y_{ij}^s$ values are calculated based on the previously-defined logical formula using $x_{ij}$ and $\hat{y}_{ij}^s$. First, we claim this calculation of $y_{ij}^s$ is feasible in BLSF, that is, they form the optimal solution to the lower level problem when (11) is dropped. The calculated $y_{ij}^s$ are binary and are shown in the first half of the proof to follow the logic constraint $y_{ij}^s \leq x_{ij}$, so constraint (10) is met. Further, this choice of $y_{ij}^s$ maximizes the objective value, as for each $i \in M$, $j \in N$, and $s \in \Omega$, the quantity $(u_{ij}^s - \phi_j^s) y_{ij}^s$ is maximized: if $u_{ij}^s - \phi_j^s$ is positive, then $\hat{y}_{ij}^s = 1$ so $y_{ij}^s$ is it's maximum feasible value, i.e. is 1 if $x_{ij} = 1$ and otherwise is 0. If $u_{ij}^s - \phi_j^s$ is nonpositive, then by definition $\hat{y}_{ij}^s = 0$ so $y_{ij}^s$ is its minimum feasible value, 0. Because every term in the objective formula sum is maximized, we conclude our set of $y_{ij}^s$ values are optimal.

We observe that constraints (2), (4), (5), (7), and (8) are already met. Two constraints remain to be verified, constraints (3) and (6). We begin with constraint (3) for some $i \in M$, $j \in N$, and $s \in \Omega$. In the case where $y_{ij}^s = 0$, at least one of $x_{ij}$ and $\hat{y}_{ij}^s$ equal 0. This means at least one of constraints (16) and (17) can be rewritten as $v_{ij}^s \leq 0$, so the inequality $v_{ij}^s \leq y_{ij}^s$ is valid. In the case where $y_{ij}^s = 1$, we note that $v_{ij}^s$ is binary so it cannot be greater than one, so (3) holds for this value of $y_{ij}^s$ as well. These are the only possible values for $y_{ij}^s$ so we conclude that (3) is valid. We repeat this process to verify (6), starting with the case where $y_{ij}^s = 0$. For this value of $y_{ij}^s$, we can simplify (6) to be $z_{ij}^s \geq -\sum_{k \in M} v_{kj}^s$. This constraint is easily verified by combining the $z_{ij}^s$ nonnegativity constraint (21) with the observation that $\sum_{i \in M} v_{ij}^s \geq 0$ as all $v_{ij}^s$ values are binary. Lastly we examine the case when $y_{ij}^s = 1$. We have that $x_{ij}$ and $\hat{y}_{ij}^s$ both equal 1, so it follows that the right-hand side of both (20) and (6) have value $1 - \sum_{k \in M} v_{kj}^s$. The left-hand sides are also identical so (6) is valid.

With this we conclude for all possible cases, $[\boldsymbol{x}^*, \boldsymbol{y}^*, \boldsymbol{v}^*, \boldsymbol{z}^*]$ is feasible in BLSF. The objective value is also unchanged between the two formulations, as the objective function and the value of the variables in the objective function are the same. □

## B Phase 0: Performance Analysis Sample Size

To evaluate the performance of any menus produced, we must first determine a test scenario sample size that is cost effective but also sufficiently accurate. Thus, we run a series of experiments to compare objective estimates made for the same menus, but with different sizes of test scenario sets. Each objective estimate is the weighted average of (23) when (23)-(29) is run for each test scenario in the set. The test sample sizes are 500, 1,000, 5,000, and 10,000 scenarios. The quality of each estimate is determined in one of two ways. One set of experiments uses a small number of drivers and requests and limits the menu size $\theta$. These problems are small enough to compute the true performance of a menu by analyzing it across all possible test scenarios as defined in Section 4.3. This exhaustive analysis allows us to determine the true error of our performance estimates. For the larger problems where it is intractable to do exhaustive performance, we instead compare to performance with 100,000 test samples, which is ten times larger than the largest sample size we include in our experiments.

Our exhaustive performance experiments include three problem sizes: $6\times6$, $8\times8$, and $10\times10$, where the first number in each pair corresponds to the number of requests, and the second number corresponds to the number of drivers. For all $6\times6$ and $8\times8$ problems, the maximum menu size is $\theta=3$, and for all $10\times10$ problems $\theta=2$. After generating each menu, the size of the complete test scenario set $S^*$ is determined. For our exhaustive performance experiments we require between $2^{14}$ and $2^{18}$ distinct test scenarios so that there are enough scenarios for the problem to be somewhat complex, but few enough that the true performance can be calculated in a reasonable amount of time. The larger problem sizes we test are $20\times20$, $20\times40$, $40\times20$, and $40\times40$, and all set $\theta=5$. For every small and large problem size, three sets of parameter instances are generated.

For every parameter instance, a menu is produced using SLSF with 100 training scenarios. To ensure that no drivers nor requests are trivial, we require $\sum_{i=1}^{m} p_{ij}>0$ for $j=1,...,n$ and $\sum_{j=1}^{n} p_{ij}>0$ for $i=1,...,m$, i.e., for every request there exists at least one driver with a nonzero willingness to accept that request, and for each driver there exists a request that driver has a nonzero willingness to accept. Additionally, we require at least half of all $p_{ij}$ values to be in $(0,1)$ to ensure a certain level of stochasticity in the problem.

For each menu produced and for each test sample size, the corresponding number of test scenarios are drawn randomly and a performance estimate is calculated and compared with the true performance calculation obtained from the exhaustive set of test scenarios for small cases, or with the estimate from 100,000 test samples for large cases. This process is repeated three times, creating three trials for each test sample size for each menu. The accuracy of our performance estimates for small and large problems are summarized in Figure 12 and the runtime summary is displayed in Table 7. In Figure 12 the performance estimate errors vary between problems, but in general we observe a low error rate for the tested sample sizes. With the exception of one of the three $10\times10$ instances, all performance estimates for small and large problem instances have less than 5% error. For our small problems, the true percent error of the performance estimates is quite low, with the average of the maximum error being 2.16%. For the larger problems, while we do not know the true performance estimate errors, our estimates are similar to estimates made with 100,000 samples. For these problems the average maximum error across

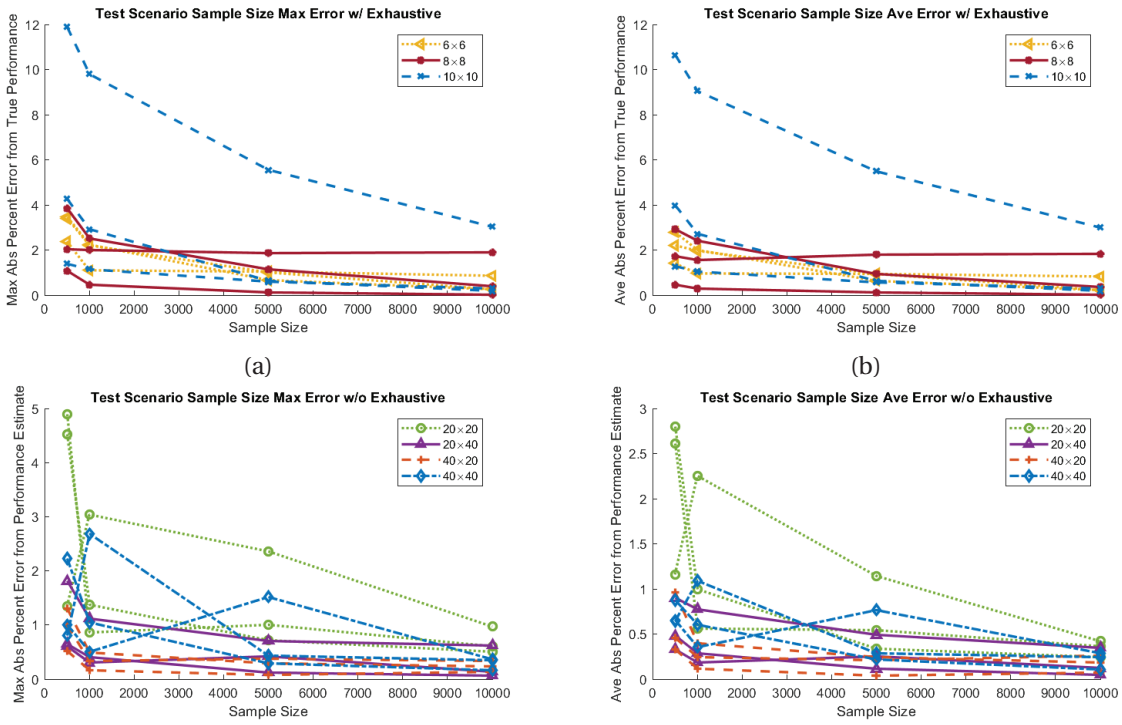



Figure 12: Absolute values of error rates of Phase 0 performance estimates. Graphs (a) and (b) are the small problems where error is calculated by comparing the performance estimate of the menu to the true performance; graphs (c) and (d) are the large problems where error is calculated by comparing the performance estimate to an estimate made with 100,000 scenarios. Each line on a graph represents data from a parameter instance and corresponding menu, and for each test sample size, the error of the three trials is either averaged ((b) and (d)) or the maximum value is displayed ((a) and (c)).

the sample sizes is 0.95%, but require a fraction of the computational time.

On average the maximum and average errors decrease as sample size increases, though we observe reduced improvement for the larger sample sizes for most problems. While there is variability in error, there is not a noticeable correlation between percent error and problem size (the number of drivers and requests). Instead the parameter settings and corresponding menu seem to cause this variability rather than just problem size. In contrast, a clear correlation between problem size and runtime exists, as larger problems take longer to solve on average. Though there is a high variability between problem sizes and problem instances, estimates made for the same menu with the same number of samples have almost identical runtimes. Additionally, estimates made for the same menu with different sample sizes scale linearly with the number of samples, i.e. an estimate made for a menu using 5,000 samples will take ten times the amount of time required to make an estimate for the same menu using 500 samples. The remainder of this report contains a large number of experiments so we use 5,000 samples, as it will cut the performance analysis runtime in half compared to 10,000 samples but still have a high level of accuracy, with an observed average maximum error of 1.00%.

Table 7: Runtimes in Phase 0 of the small and large problems. Column 1 separates the problems by size, with the three small problems listed first. Column 2 displays the average log base 2 value of the number of distinct test scenarios for each menu across the three problem instances. Column 3 displays the average runtime of either the exhaustive performance calculation for small problems or the performance estimate using 100,000 scenarios for large problems. Columns 4-7 display, for each of the four test scenario sample sizes, the average runtime of the three trials averaged across the three problem instances for the given problem size.

| | | Average Time (seconds) | | | | |
|---|---|---|---|---|---|---|
| Size | Ave $\log_2(\|S^*\|)$ | Exh./Est. | 500 scens | 1,000 scens | 5,000 scens | 10,000 scens |
| 6×6 | 14.7 | 864.2 | 18.7 | 36.6 | 175.9 | 349.7 |
| 8×8 | 16.0 | 3879.1 | 17.2 | 34.6 | 175.2 | 351.7 |
| 10×10 | 15.7 | 1827.2 | 15.3 | 30.4 | 153.8 | 306.7 |
| 20×20 | 61.3 | 3890.7 | 19.6 | 39.3 | 200.8 | 402.3 |
| 20×40 | 34.3 | 5592.7 | 27.9 | 55.7 | 278.8 | 559.1 |
| 40×20 | 62.0 | 5655.3 | 28.5 | 56.9 | 284.0 | 571.8 |
| 40×40 | 117.0 | 11869.8 | 56.3 | 111.7 | 562.8 | 1130.4 |

# C Proof of Theorem 5.1

**Theorem 5.1.** *There always exists a menu with menu size $\theta$ for all drivers that is in the optimal solution set of SLSFnoZ.*

*Proof.* This is a simple proof by construction. We start with any optimal solution $[\boldsymbol{x}^*, \boldsymbol{v}^*]$ of SLSFnoZ. If $\sum_{i\in M} x^*_{ij} = \theta$ for all $j \in N$, we are done. If not, there is some subset $\bar{N} \subset N$ such that $\sum_{i\in M} x^*_{ij} < \theta$ for all $j \in \bar{N}$ and $\sum_{i\in M} x^*_{ij} = \theta$ for all $j \in N \setminus \bar{N}$. Because $\theta \leq m$, there exists at least one set of driver-request pairs we can add to the current menu set that would create a new menu with menu size $\theta$ for all drivers. That is, we can claim that for each $j \in \bar{N}$, there exists at least one set $\bar{M}_j$ such that $|\bar{M}_j| = \theta - \sum_{i\in M} x^*_{ij}$ and $x^*_{ij} = 0$ for all $i \in \bar{M}_j$. We define the new menu set $\bar{\boldsymbol{x}}$ such that $\bar{x}_{ij} = 1$ if $x^*_{ij} = 1$ or if $j \in \bar{N}$ and $i \in \bar{M}_j$, and $\bar{x}_{ij} = 0$ otherwise. The solution $[\bar{\boldsymbol{x}}, \boldsymbol{v}^*]$ is still feasible in SLSFnoZ, as $\boldsymbol{v}^*$ is unchanged, therefore (17)-(19) hold. Additionally, $\sum_{i\in M} \bar{x}_{ij} = \theta$ for all $j \in N$ and $\bar{\boldsymbol{x}}$ is binary, so (15) and (22) hold. Next, $x^*_{ij} \leq \bar{x}_{ij}$ for all $i \in M$ and $j \in N$ so (16) holds. The objective function value for both $[\boldsymbol{x}^*, \boldsymbol{v}^*]$ and $[\bar{\boldsymbol{x}}, \boldsymbol{v}^*]$ is the same, as the objective value only contains $\boldsymbol{v}$ terms and no $\boldsymbol{x}$ terms. The solution is feasible and objective value is unchanged, so $[\bar{\boldsymbol{x}}, \boldsymbol{v}^*]$ is also an optimal solution and has a menu size $\theta$ for all drivers.

□