A Prescriptive Machine Learning Method for Courier Scheduling on Crowdsourced Delivery Platforms

Adam Behrendt, Martin Savelsbergh, He Wang
School of Industrial & Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332, USA,
adam.behrendt@gatech.edu, martin.savelsbergh@isye.gatech.edu, he.wang@isye.gatech.edu

Crowdsourced delivery platforms face the unique challenge of meeting dynamic customer demand using couriers not employed by the platform. As a result, the delivery capacity of the platform is uncertain. To reduce the uncertainty, the platform can offer a reward to couriers that agree to be available to make deliveries for a specified period of time, i.e., to become *scheduled* couriers. We consider a scheduling problem that arises in such an environment, i.e., in which a mix of scheduled and ad-hoc couriers serves dynamically arriving pickup and delivery orders. The platform seeks a set of shifts for scheduled couriers so as to minimize total courier payments and penalty costs for expired orders. We present a prescriptive machine learning method that combines simulation optimization for offline training and a neural network for online solution prescription. In computational experiments using real-world data provided by a crowdsourced delivery platform, our prescriptive machine learning method achieves solution quality that is within 0.2%–1.9% of a bespoke sample average approximation method, while being several orders of magnitude faster in terms of online solution generation.

Key words: crowdsourced same-day delivery, machine learning, sample average approximation

1. Introduction

The rise of e-commerce has changed the landscape of logistics over the last decade. Customers expect fast and reliable delivery, often on the same day or even within a few hours after placing an order. This trend is not expected to change anytime soon, as e-commerce revenue is projected to increase approximately 48% from 2018 to 2023 (Clement 2019). With e-commerce comes an increase in the need for reliable last-mile delivery systems, with local stores often being used as fulfillment centers for online orders (Zebra Technologies 2019).

To accommodate the increase in e-commerce, retail giants like Amazon and Walmart have begun to employ *crowdsourced* delivery capacity to serve demand. Crowdsourced delivery refers to the use of independent couriers to make deliveries rather than company employees, similar to the use of independent couriers in personal transportation by companies like Uber and Lyft (Sampaio et al. 2019). Individuals acting as independent contractors to complete work without direct employment is the key characteristic of the emerging gig economy (Manyika et al. 2016). In a survey by Zebra Technologies (2019), 87% of responding companies said they would utilize crowdsourced delivery by 2028, a significant increase from the 30% who currently make use of it.

This emerging and unique commercial landscape has led to the appearance of third-party logistics (3PL) companies that rely almost exclusively on crowdsourced delivery, so-called crowdsourced delivery platforms. The key difference between traditional 3PL companies and crowdsourced delivery platforms is that traditional 3PL companies only face demand-side uncertainty, whereas crowdsourced delivery platforms face demand-side uncertainty *and* supply-side uncertainty, as couriers can opt-in and opt-out at their own discretion. Alnaggar, Gzara, and Bookbinder (2021) describe four methods by which crowdsourced delivery platforms engage with couriers: pure self-scheduling, hybrid and centralized scheduling, en-route matching, and bulletin-board type matching.

We consider a crowdsourced delivery platform that uses both centralized scheduling (for scheduled couriers) and decentralized bulletin-board type matching (for ad-hoc, unscheduled couriers). Specifically, we are concerned with the construction of shifts for scheduled couriers, in the presence of dynamically arriving orders and dynamically arriving ad-hoc couriers. This is a stochastic optimization problem in which demand and ad-hoc courier forecasts are used as input to create schedules that minimize the expected cost. Traditional solution approaches often utilize sampling methods, where potential future scenarios are generated and a deterministic version of the problem is solved in order to create candidate solutions. These solutions are then compared on their average performance over the set of scenarios, and the best is chosen. For large scale and high dimensional problems, however, generating good solutions, or even evaluating these solutions, becomes computationally prohibitive. Additionally, the platform may want/need to solve the problem several times a day for multiple markets. Therefore, generating high-quality solutions *quickly* is critical.

We propose a prescriptive machine learning method for the courier scheduling problem arising in crowdsourced delivery. The main idea of our method is to utilize the well-known sample average approximation (SAA) method *offline* to create a set of solutions to a diverse set of problem instances (i.e., demand and ad-hoc courier forecasts), and use a trained machine learning model to generate solutions to new forecasts *online*. Thus, we bypass the computationally intensive optimization step and prescribe an approximate solution with performance similar to that of the SAA method. We summarize our contributions below:

- We introduce the Crowdsourced Delivery Shift Scheduling Problem (CDSSP). In an environment in which orders and ad-hoc couriers dynamically arrive during the operating period (and where different orders have different origins and destinations and different ad-hoc couriers have different starting locations and start and end times), we seek to determine a set of shifts for scheduled couriers so as to minimize total courier payments and penalty costs for expired orders. Distinct from the existing literature, we assume the ad-hoc couriers can choose which order/order bundle they will serve (if any) based on their own preferences.

- We present a general prescriptive machine learning method to generate CDSSP solutions from demand and ad-hoc courier forecasts. The method combines sample average simulation optimization for offline training and neural networks for online solution prescription.
- In simulation experiments using real-world data provided by a crowdsourced delivery platform, the prescriptive machine learning method achieves solution quality within 0.2%–1.9% of a bespoke sample average approximation (SAA) method, while being several orders of magnitude faster than the SAA method in terms of online solution generation.
- Lastly, our prescriptive method provides a general approach to approximate solutions to optimization problems via machine learning. Variations of our method can be used in other applications settings, specifically ones where solutions to large stochastic optimization problems need to be generated quickly.

The remainder of the paper is organized as follows. In Section 2, we discuss relevant literature and highlight the contribution of our paper. In Section 3, we describe the problem of interest and introduce a formal model for it. In Section 5, we detail our prescriptive machine learning method. In Section 6, we present simulation experiments using real-world data to validate our approach. In Section 7, we provide a discussion of our results and main findings.

2. Literature Review

In the following, we discuss relevant literature in terms of the application area and methodological approach. Specifically, we review work surrounding the area of crowdsourced and same-day delivery in subsection 2.1, and literature encompassing stochastic optimization (namely, the sample average approximation method) and the emerging field of prescriptive analytics in subsection 2.2. Additionally, we highlight how our work differs from and expands upon the existing literature in each subsection, respectively.

2.1. Applications

The work in this paper falls into two main application areas: same-day and crowdsourced delivery.

Same-day Delivery. Klapp, Erera, and Toriello (2018) studied the same-day delivery problem for a single depot with a single vehicle, where multiple dispatch epochs occur over the course of a service day. The decision maker must then decide whether or not to dispatch a vehicle at each dispatch epoch, with the objective of minimizing operating costs and penalty costs of unserved service requests. Klapp, Erera, and Toriello (2018) used a dynamic programming approach to solve the deterministic variant, and utilizes this method to construct an optimal a priori policy for the stochastic case. Additionally, it is shown fully dynamic policies can outperform a priori ones, and some dynamic heuristics and solution bounds are presented. Stroh, Erera, and Toriello

(2019) studied a similar same-day delivery problem, also from a single depot, but considers the aforementioned case where a single vehicle is dispatched multiple times per day and a case where a large fleet of vehicles is managed such that each vehicle is only dispatched once per day. Stroh, Erera, and Toriello (2019) also used continuous approximations to model these same-day delivery variants, and leverages their approximation model to answer tactical system design questions. Voccia, Campbell, and Thomas (2019) considered a same-day delivery problem as a multi-vehicle dynamic pickup and delivery problem with time constraints, whose objective is to maximize the number of delivery requests served. Voccia, Campbell, and Thomas (2019) modeled the problem as a Markov Decision Process (MDP) and utilized a dynamic sample-scenario approach to solve instances of the problem. At each decision epoch, future scenarios are sampled from a distribution dependent on the current system state. These deterministic scenarios are then solved and one of the solutions is chosen by use of a consensus function (similar to Bent and Van Hentenryck (2004), albeit a fundamentally different function). Using this sample-scenario approach demonstrated the value of anticipating future demand requests. Ulmer (2020) considered a same-day delivery problem from a single depot with multiple vehicles and studied the impact of dynamically deciding the price to charge customers, in addition to the route. The problem is formulated as an MDP, and an anticipatory pricing and routing policy is implemented which uses value function approximation. Ulmer (2020) then demonstrated that such a policy results in higher revenue and more customers served for a service day when compared to benchmark policies.

In the work detailed above, dynamic pricing, routing, and dispatch policies are studied in the context of same-day delivery, in which all assume the presence of a fixed fleet of vehicles (potentially a single vehicle) available for the entirety of a service day. In this paper, we study how to choose the size of a crowdsourced fleet by deciding schedules prior to the service day. This type of courier (i.e. those that are available for set shifts and whose routes are directly controlled by the crowdsourced delivery platform) are hereby referred to as scheduled couriers. We also consider the presence of ad-hoc couriers that arrive stochastically and may serve delivery requests from a bulletin-board like posting by the crowdsourced delivery platform.

Crowdsourced Delivery. Cao, Olvera-Cravioto, and Shen (2020) considered a last-mile shared delivery problem that utilizes both ad-hoc and scheduled couriers to serve demand based on a formulation of the discrete sequential packing problem. However, they assumed that the ad-hoc couriers make deliveries for the first part of the service day, while scheduled couriers serve all unserved packages at the end of the day. Additionally, ad-hoc couriers were assumed to arrive according to a constant rate over the course of the planning period. Gurvich, Lariviere, and Moreno (2019) studied the impact of self-scheduling couriers on the profitability of a delivery platform and

the associated service level given to customers. Self-scheduling couriers are a type of crowdsourced couriers that announce their arrivals and elect to work for a fixed period of time, in which the crowdsourced delivery platform creates routes for them. Ad-hoc couriers, however, arrive at the platform and select which orders they would like to serve from a list. Yildiz and Savelsbergh (2019) modeled on-demand meal delivery platforms and investigated questions surrounding the service level and capacity of these platforms, such as the relationships among service area size, delivery offer acceptance probability, profitability, etc. Lee and Savelsbergh (2015) studied the adjacent field of dynamic ridesharing, in which traveling individuals are matched based on their origin-destination pairs to share transport, essentially a pickup and delivery environment. They specifically investigated the usefulness of having a fleet of dedicated couriers employed concurrently with private individuals with spare capacity in order to meet a minimum service guarantee to riders. Dayarian and Savelsbergh (2020) studied same-day crowdsourced delivery in which in-store customers are used to serve demand, along with a dedicated fleet of couriers. They solved this problem in a similar fashion to the previously mentioned works on same-day delivery, by implementing dynamic dispatch strategies via a rolling horizon approach. Arslan et al. (2019) considered a dynamic pickup and delivery problem with ad-hoc couriers and studied dynamic routing and assignment policies formulated as a matching problem and solved exactly with a rolling horizon approach. Santini et al. (2020) introduce the probabilistic TSP with crowdsourcing (PTSPC) and propose a solution algorithm that makes use of simulation, Monte Carlo approximation and machine learning, which has similarities to our solution approach. However, in addition to solving a different problem (routing versus schedule planning), their work uses ML to approximate the objective function while our work goes one step further and uses ML to prescribe solutions. Alnaggar, Gzara, and Bookbinder (2021) reviewed the different approaches that crowdsourced delivery platforms use to recruit and utilize couriers and provided an in-depth review of the existing literature, while Savelsbergh and Ulmer (2022) explore the literature and future research directions surrounding the main novel aspect of crowdsourced delivery: uncertain delivery capacity.

Finally, the work of Ulmer and Savelsbergh (2020) is the most relevant to our paper. Ulmer and Savelsbergh (2020) considered a dynamic crowdsourced delivery environment for a single depot with the goal of constructing shifts for scheduled couriers. Ulmer and Savelsbergh (2020) used the term "unscheduled couriers" to refer to couriers that can enter the system at any time by announcing their availability and that can leave the system at any time after that and can be offered delivery task during that period, which the courier may accept or reject. This is similar but fundamentally different from ad-hoc couriers, who can see the full list of orders and make a selection according to their own utility. The problem in Ulmer and Savelsbergh (2020) is solved for a given planning horizon instance via a value function approximation approach. We complement

the work of Ulmer and Savelsbergh (2020) by proposing a prescriptive machine learning method that can generate solutions online quickly, by generating planning horizon instance/solution pairs offline, and training a machine learning model. Thus, we bypass the more intense computation requirements of the underlying solution method while maintaining comparable solution quality. We also consider a more general model setting with multiple depots.

2.2. Methods

Sample Average Approximation (SAA). Introduced by Kleywegt, Shapiro, and Homem-de-Mello (2002), the SAA method is used for stochastic discrete optimization problems. The objective, minimizing the *expected* cost, is approximated by drawing K i.i.d. samples from a given distribution and solving the deterministic variant of the given problem over this set of realizations. Supported by promising convergence results as K grows large, SAA and its variants continue to be utilized in the field of transportation and logistics (see, e.g., Verweij et al. (2003), Schütz, Tomasgard, and Ahmed (2009), Long, Lee, and Chew (2012), Li et al. (2016), Alnaggar, Gzara, and Bookbinder (2020)). The SAA method is discussed in more detail in Section 4. In this paper, we employ the SAA method as a component in our solution method and as a means of evaluating the quality of our solutions.

Prescriptive Analytics. The problem setting of our paper is related to Bertsimas and Kallus (2020). Their paper considered a stochastic optimization problem with historical observations with unknown joint probability distributions and features. Essentially, Bertsimas and Kallus (2020) constructed a conditional probability distribution $\mathscr{Y}|X=x$ (in the form of a weight function), where $X=x$ are predictive features from a data set of N observations, by using machine learning. This conditional distribution is then used to generate samples given a new observation x, and the deterministic variant of the stochastic problem is solved with these samples. While this method has improved solution quality over a uniform weight function, or simply using the N observed y_i's directly, the computational performance remains unimproved. The scale of instances grows large quickly as the sample size increases and high dimensional problems make it difficult to efficiently generate optimal solutions. In our paper, we assume an estimate of this conditional probability distribution $\mathscr{Y}_x := \mathscr{Y}|X=x$ is known (e.g., constructed via machine learning/forecasting on a set of historical data). The main contribution is that we prescribe solutions to our conditional stochastic optimization problem *directly* from $\mathscr{Y}_x$ by using a machine learning model trained on the instance/solution pairs to a mathematical programming model, reducing the online computation time drastically.

3. Problem Description

We study a scheduling problem faced by a delivery platform that utilizes both *scheduled* and *ad-hoc* crowdsourced couriers. Prior to an operating period, the platform creates shifts to offer to crowdsourced couriers. If couriers choose to sign up for a shift at this stage, they become scheduled couriers and will be managed by the platform during the operating period. Additionally, orders can be served by ad-hoc couriers who arrive dynamically over the operating period. Fundamentally, we want to determine the *optimal* number of scheduled couriers required at each time during the operating period (to be covered by shifts) to serve orders that arrive dynamically, while accounting for the uncertain availability of ad-hoc couriers, so as to minimize the total cost of scheduled couriers, ad-hoc couriers, and expired orders (i.e., orders that cannot be served by their promised delivery time). We refer to this problem as the Crowdsourced Delivery Shift Scheduling Problem (CDSSP) and provide a more formal definition below.

3.1. Model

Let $\mathcal{T} = [0, T]$ be a given operating period (e.g., one day) during which customers in a service area place orders (see Figure 1). The operating period is divided into a set of equal-size smaller periods $\mathcal{P} = \{1, 2, ..., P\}$ (e.g., each period is 30 minutes).

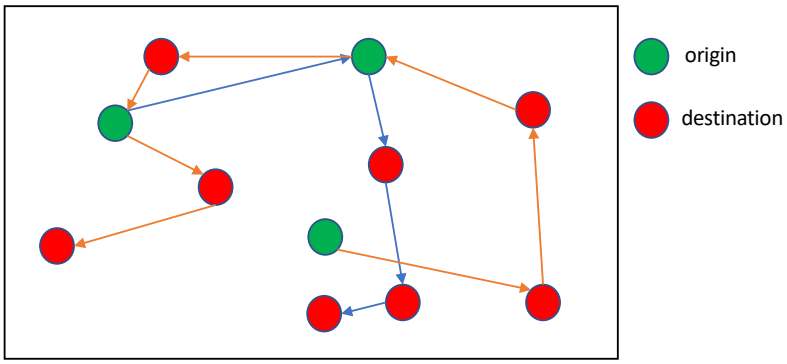

Figure 1 **Illustration of a rectangular service area with orders from multiple origins and destinations, and two potential pickup and delivery routes. Specifically, a one-to-many case is shown, where a single origin may have multiple destinations (e.g. a restaurant).**

Each order has the following characteristics:

t_p	order placement time
o	pickup location (origin)
t_o	order ready time
d	delivery location (destination)
t_d	delivery promise time

Some orders are known before the start of the operating period ($t_p = 0$), i.e., the *static orders*, and the other orders are revealed dynamically ($t_p > 0$) during the operating period, i.e., the *dynamic orders*. To predict the dynamic orders, we assume that the platform uses some predictive features X (e.g., weather forecast, seasonality, promotions, special events), which are observed before the start of the operating period. Conditional on $X = x$, the number and characteristics of dynamic orders are governed by a joint probability distribution denoted by $\mathscr{Y}_x$. Ad-hoc couriers arrive randomly over the operating period according to an arrival process $\mathscr{W}$ and are willing to serve a subset of the realized but not yet served (or assigned) orders according to a discrete choice model (see Train 2009). More specifically, we assume that the arrivals can be represented by a (potentially inhomogenous) Poisson arrival process (with known rate parameters $\lambda_p \ \forall p \in \mathcal{P}$), that the choice of order(s) to serve occurs instantaneously upon arrival (which implies that there is no interaction between the actions of ad-hoc couriers), and that after serving the chosen orders, the ad-hoc courier leaves the system. We make the aforementioned simplifying assumptions for ease of presentation and analysis, however our proposed solution approach can handle general arrival processes and custom courier choice behavior.

Let $\mathcal{N}$ and $\mathcal{A}$ represent the sets of placed orders and ad-hoc courier arrival times realized in the operating period $\mathcal{T}$, respectively. Specifically, $\mathcal{N}$ is drawn from $\mathscr{Y}_x$ and $\mathcal{A}$ is a sample path of an inhomogeneous Poisson arrival process $\mathscr{W}$ dictated by its known rate parameters. Finally, let $N := |\mathcal{N}|$ denote the number of realized orders.

Suppose the delivery platform has a service level target, which specifies that if N orders arrive in $\mathcal{T}$, then at least αN of these orders must be served for some $\alpha \in [0, 1]$. Orders are served by scheduled and ad-hoc couriers according to a dynamic assignment and routing policy π. We treat π as an input parameter and methods for optimizing π are beyond the scope of this paper; however, our proposed method allows for any general assignment and routing policy π.

Next, we describe how the platform decides how many scheduled couriers are needed. We are concerned with finding an optimal number of active scheduled couriers for each period $p \in P$ in order to meet the forecast demand while accounting for the presence of ad-hoc couriers. Let $\mathbf{z} = \{z_p \in \mathbb{Z}_{\geq 0}, \ \forall p \in \mathcal{P}\}$ be the decision vector corresponding to the number of active scheduled couriers during the operating period. Figure 2 provides an illustration of how the decision $\mathbf{z}$ is affected by the shape of a demand forecast and the presence of ad-hoc couriers.

Naturally, the purpose of $\mathbf{z}$ is to be used as a basis for the construction of shifts for scheduled couriers. We have a set of potential shifts, each defined by a start and end period, with index set $\mathcal{S} = \{1, 2, ..., S\}$. The delivery platform must decide how to convert the scheduled courier requirement $\mathbf{z}$ to a set of shifts to offer to couriers. Let $\mathbf{u} = \{u_s \in \mathbb{Z}_{\geq 0}, \ \forall s \in \mathcal{S}\}$ be the decision of how many of shift s to offer to couriers and γ_{sp} be an indicator vector representing whether shift s covers period

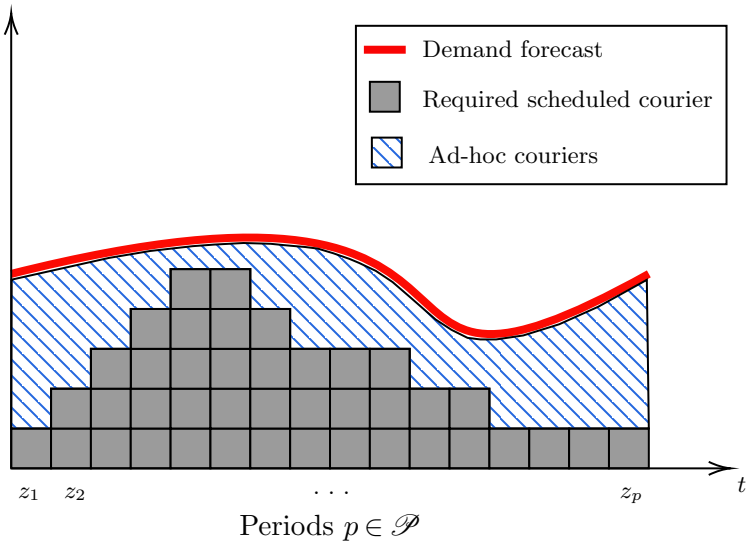

Figure 2 **Operating period $\mathcal{T}$ divided into P periods, with an example demand forecast overlaying the associated scheduled courier requirement $\mathbf{z}$. Observe that the ratio of scheduled and ad-hoc couriers may change over the course of the operating period.**

p. Then, a schedule $\mathbf{u}$ is feasible for a scheduled courier requirement $\mathbf{z}$ if $\sum_{s \in \mathcal{S}} \gamma_{sp} u_s \geq z_p \ \forall p \in \mathcal{P}$. Figure 3 illustrates a shift schedule $\mathbf{u}$ that covers the scheduled courier requirement $\mathbf{z}$ of Figure 2.

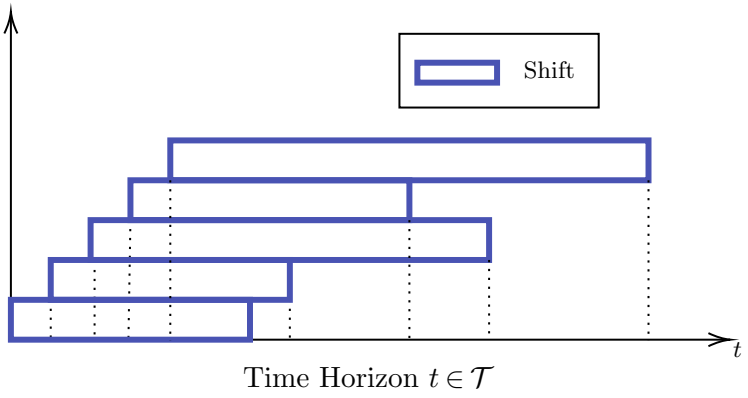

Figure 3 **Potential shift schedule for the scheduled courier requirement $\mathbf{z}$ depicted in Figure 2.**

In the setting considered in this paper, we assume that there is a large pool of couriers who prefer being a scheduled courier to being an ad-hoc courier because working as a scheduled courier offers reliable pay regardless of how many orders the courier serves. Therefore, each shift offered by the delivery platform is guaranteed to be selected by a courier. Alternatively, one can imagine a setting in which there is some probability that a shift offered by the delivery platform is left unfilled. The solution method proposed later in the paper can accommodate such a setting (by altering the simulation), but we focus on the simplified setting where couriers sign up for all shifts offered.

Specifically, shifts are offered to hedge against the uncertainty associated with order arrivals and available ad-hoc courier delivery capacity.

Let w_p be the cost of having an active scheduled courier in period $p \in \mathcal{P}$, then the cost of a shift schedule $\mathbf{u}$ is $\sum_{p\in\mathcal{P}} \sum_{s\in\mathcal{S}} w_p \gamma_{sp} u_s$. Additionally, for a shift schedule $\mathbf{u}$, an order realization $\mathcal{N}$, and an ad-hoc courier arrival realization $\mathcal{A}$, we define the cost of expired orders and ad-hoc couriers as

$$c(\mathbf{u},\mathcal{N},\mathcal{A}) := \theta \cdot \max(0, n_e - \lfloor (1-\alpha)N \rfloor) + \eta \cdot n_a, \tag{1}$$

where θ is the penalty cost for an expired order over the α threshold, n_e is the number of expired orders, η is the unit cost of an order served by an ad-hoc courier, and n_a is the number of orders delivered by ad-hoc couriers. Thus, ad hoc couriers are compensated per order whereas scheduled couriers are compensated per time unit. The counts n_e and n_a are determined by the order realization $\mathcal{N}$, the ad-hoc courier arrivals $\mathcal{A}$, and the assignment and routing policy π. Finally, observe that while the cost $c(\mathbf{u},\mathcal{N},\mathcal{A})$ depends on shift schedule $\mathbf{u}$, the cost indirectly, and more strongly, depends on scheduled courier requirement $\mathbf{z}$. We observe too that two different shift schedules $\mathbf{u}$ which both cover a scheduled courier requirement $\mathbf{z}$ may incur different costs. For example, two shift schedules with the same number of shifts, but with different shift start and end times may serve different sets of orders and thus may have different numbers of expired orders and/or orders served by ad-hoc couriers. As such, defining how to obtain a shift schedule $\mathbf{u}$ from a scheduled courier requirement $\mathbf{z}$ is important, but we will focus on how to obtain $\mathbf{z}$.

In sum, the elements of the CDSSP model include

- Parameters: service level target α and assignment and routing policy π.
- A set of orders $\mathcal{N}$ – both static and dynamic orders; the number and characteristics of dynamic orders are drawn from the joint conditional distribution $\mathscr{Y}_x$ with x a realization of the predictive features X.
- A set of ad-hoc courier arrival times $\mathcal{A}$, representing a random sample path of a Poisson arrival process $\mathscr{W}$ (dictated by rate parameters λ_p for periods $p \in \mathcal{P}$).
- Decision vectors $\mathbf{z}$ (the number of required scheduled couriers for each period) and $\mathbf{u}$ (the number of couriers assigned to each shift).

The CDSSP is then a stochastic optimization problem where we first decide the number of active scheduled couriers in each period and then construct an associated shift schedule to cover these requirements, with the goal of minimizing scheduled courier compensation and expected ad-hoc courier and penalty cost. We formulate the CDSSP as follows

$$\min_{\mathbf{z},\mathbf{u}} \sum_{p\in\mathcal{P}} \sum_{s\in\mathcal{S}} w_p \gamma_{sp} u_s + \mathbb{E}_{\mathcal{N}\sim\mathscr{Y}_x, \mathcal{A}\sim\mathscr{W}} \left[c(\mathbf{u},\mathcal{N},\mathcal{A}) \right] \tag{2a}$$

$$\text{s.t.} \sum_{s\in\mathcal{S}} \gamma_{sp} u_s \geq z_p, \quad \forall p \in \mathcal{P} \tag{2b}$$

$$u_s \in \mathbb{Z}_{\geq 0}, \quad \forall s \in \mathcal{S} \tag{2c}$$

$$z_p \in \mathbb{Z}_{\geq 0}, \quad \forall p \in \mathcal{P}. \tag{2d}$$

The objective (2a) is to minimize the cost of scheduled couriers and the expected ad-hoc courier and penalty cost. Constraint (2b) ensures that the number of scheduled couriers in each period is at least the number of scheduled couriers required for that period. Lastly, (2c) and (2d) constrain the decision variables to non-negative integers. In the next section, we detail the sample average approximation (SAA) method which has been employed to solve similar stochastic optimization problems, and highlight the key assumptions and drawbacks of such a method in the context of the CDSSP.

4. Sample Average Approximation Method

The SAA method (Kleywegt, Shapiro, and Homem-de-Mello 2002) presents an alternative formulation to approximate the stochastic optimization problem (2). Suppose we are given K i.i.d. order realizations $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_K$ and ad-hoc courier arrival realizations $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_K$. These realizations can either be obtained from historical data or sampled from $\mathscr{Y}_x$ and $\mathscr{W}$. The SAA problem is then be formulated as

$$\min_{\mathbf{z},\mathbf{u}} \sum_{p\in\mathcal{P}} \sum_{s\in\mathcal{S}} w_p \gamma_{sp} u_s + \frac{1}{K} \sum_{k=1}^{K} c(\mathbf{u}, \mathcal{N}_k, \mathcal{A}_k) \tag{3a}$$

$$\text{s.t.} \sum_{s\in\mathcal{S}} \gamma_{sp} u_s \geq z_p \quad \forall p \in \mathcal{P} \tag{3b}$$

$$u_s \in \mathbb{Z}_{\geq 0} \quad \forall s \in \mathcal{S} \tag{3c}$$

$$z_p \in \mathbb{Z}_{\geq 0} \quad \forall p \in \mathcal{P}. \tag{3d}$$

Given the evaluated cost for each order realization and ad-hoc courier arrival realization, $c(\mathbf{u}, \mathcal{N}_k, \mathcal{A}_k)$, (3) is a *deterministic* problem, which is especially useful when the expectation in the objective does not have a closed form or is difficult to evaluate. Under some regularity conditions, it can be shown that the optimal solution to (3) will converge to the solution of the stochastic optimization problem (2) when $K \to \infty$. As such, the SAA method and its variants are often used in dynamic vehicle delivery and crowdsourced delivery applications, e.g., Bent and Van Hentenryck (2004), Srour, Agatz, and Oppen (2018), Voccia, Campbell, and Thomas (2019), Dayarian and Savelsbergh (2020), Liu, He, and Shen (2020). In the following subsection, we present the SAA-SO method, which is a variant of the traditional SAA method that utilizes simulation optimization to evaluate the cost of expired orders and ad-hoc couriers ($c(\mathbf{u}, \mathcal{N}_k, \mathcal{A}_k)$) for each of the K realizations.

4.1. SAA-SO Heuristic

The usefulness of a traditionally implemented SAA approach depends on the tractability of the deterministic variant of the problem and assumes that there exists some algorithm to solve it efficiently. For large-scale or high-dimensional instances of the CDSSP, the existence of a closed-form expression for $c(\mathbf{u}, \mathcal{N}_k, \mathcal{A}_k)$ is unlikely. Evaluating the cost of expired orders and ad-hoc couriers $c(\mathbf{u}, \mathcal{N}_k, \mathcal{A}_k)$ is difficult as the set of expired orders and those selected by ad-hoc couriers depends on the assignment and routing policy π and the choice model employed by ad-hoc couriers. In order to account for this, we utilize simulation optimization (SO) to evaluate $c(\mathbf{u}, \mathcal{N}_k, \mathcal{A}_k)$. Recalling formulation (3), we note that the shift decisions $\mathbf{u}$ are dictated by the required courier capacity $\mathbf{z}$ through constraints (3b). As such, we can evaluate the total cost for a fixed $\mathbf{z}$ by solving

$$g(\mathbf{z}) := \min_{\mathbf{u}} \sum_{p \in \mathcal{P}} \sum_{s \in \mathcal{S}} w_p \gamma_{sp} u_s + \frac{1}{K} \sum_{k=1}^{K} c(\mathbf{u}, \mathcal{N}_k, \mathcal{A}_k) \tag{4a}$$

$$\text{s.t.} \sum_{s \in \mathcal{S}} \gamma_{sp} u_s \geq z_p \quad \forall p \in \mathcal{P} \tag{4b}$$

$$u_s \in \mathbb{Z}_{\geq 0} \quad \forall s \in \mathcal{S}. \tag{4c}$$

The optimization problem for the CDSSP then becomes

$$\min_{\mathbf{z}} \; g(\mathbf{z}) \tag{5a}$$

$$\text{s.t. } z_p \in \mathbb{Z}_{\geq 0} \quad \forall p \in \mathcal{P}. \tag{5b}$$

Our goal is to then solve (5) by exploring the solution space of $\mathbf{z}$ efficiently. To gain intuition, we investigate the general behavior of $g(\mathbf{z})$ through the following approximation. First, we approximate $c(\mathbf{u}, \mathcal{N}_k, \mathcal{A}_k)$ as a function of $\mathbf{z}$ by the following route assignment integer program. We assume that the scheduled couriers' start location and ad-hoc couriers' arrival location are at the center of the service region and both types of couriers are not required to return. Additionally, scheduled couriers serve demand along routes while each ad-hoc courier can serve a single order. Let $\mathcal{R}_p$ denote the index set of routes whose start (from the center of the service region) and end times fall within period $p \in \mathcal{P}$, and define $\mathcal{R} := \bigcup_{p \in \mathcal{P}} \mathcal{R}_p$. Let x_r denote whether route r is assigned to a scheduled courier, m_{ip} denote whether order i is selected by an ad-hoc courier in period p and l_i denote whether order i is unassigned and expire. We have

$$c_A(\mathbf{z}, \mathcal{N}_k, \mathcal{A}_k) := \min_{\mathbf{x},\mathbf{l},\mathbf{m}} \; \theta \cdot L + \eta \sum_{p \in \mathcal{P}} \sum_{i \in \mathcal{N}_k} m_{ip} \tag{6a}$$

$$\text{s.t. } L \geq \sum_{i \in \mathcal{N}_k} l_i - (1 - \alpha) N_k \tag{6b}$$

$$L \geq 0 \tag{6c}$$

$$\sum_{p\in\mathcal{P}}\sum_{r\in\mathcal{R}_p} a_{ir}x_r + \sum_{p\in\mathcal{P}} m_{ip} + l_i = 1 \quad \forall i \in \mathcal{N}_k \tag{6d}$$

$$m_{ip} \leq \nu_{ip} \quad \forall i \in \mathcal{N}_k,\ \forall p \in \mathcal{P} \tag{6e}$$

$$\sum_{i\in\mathcal{N}_k} m_{ip} \leq M_p \quad \forall p \in \mathcal{P} \tag{6f}$$

$$\sum_{r\in\mathcal{R}_p} x_r \leq z_p \quad \forall p \in \mathcal{P} \tag{6g}$$

$$l_i \in \{0,1\} \quad \forall i \in \mathcal{N}_k \tag{6h}$$

$$m_{ip} \in \{0,1\} \quad \forall i \in \mathcal{N}_k,\ \forall p \in \mathcal{P} \tag{6i}$$

$$x_r \in \{0,1\} \quad \forall r \in \mathcal{R}, \tag{6j}$$

where objective (6a) minimizes the penalty caused by the unassigned (and hence expired) orders L over the α service level threshold plus the wage for the ad-hoc couriers; constraints (6b) and (6c) define L and force it to be non-negative; constraint (6d) ensures that each job is present in at most one selected route, selected by an ad-hoc courier or remains unassigned (the coefficient a_{ir} is 1 if order i is present on route r and 0 otherwise); constraint (6e) ensures that ad-hoc couriers only select orders they are guaranteed to serve on-time where the parameter ν_{ip} (created from $\mathcal{N}_k$ and $\mathcal{A}_k$) is 1 if order i is has a ready time prior to period p and can be served before its deadline if started by the end of period p (that is, from the center of the service region, to the pickup, to the delivery); constraint (6f) makes certain that the number of orders selected by ad-hoc couriers in period p does not exceed the number of ad-hoc couriers that arrive in that period (denoted by M_p, constructed from $\mathcal{A}_k$); constraint (6g) guarantees the number of routes selected in each period is less than the number of scheduled couriers available during that period; and finally constraints (6i), (6h) and (6j) are binary constraints on decision variables $\mathbf{x}$ (which routes are selected by scheduled couriers), $\mathbf{m}$ (which orders are selected by ad-hoc couriers in each period) and $\mathbf{l}$ (which orders remain unassigned and expire.) Clearly, $c_A(\mathbf{z},\mathcal{N}_k,\mathcal{A}_k)$ is an upper bound on $c(\mathbf{u},\mathcal{N}_k,\mathcal{A}_k)$ as it is a restricted case of our model. Specifically, we do not allow routes of scheduled couriers to cross periods and restrict the set of orders that can be served by an ad-hoc courier.

Let $\tilde{c}_A(\mathbf{z},\mathcal{N}_k,\mathcal{A}_k)$ be the cost of the associated linear relaxation of (6). We can now approximate the cost of a solution $\mathbf{z}$ to (5) with the following nested LP

$$\tilde{g}(\mathbf{z}) := \min_{\mathbf{u}} \sum_{p\in\mathcal{P}}\sum_{s\in\mathcal{S}} w_p\gamma_{sp}u_s + \frac{1}{K}\sum_{k=1}^{K}\tilde{c}_A(\mathbf{z},\mathcal{N}_k,\mathcal{A}_k) \tag{7a}$$

$$\text{s.t.} \sum_{s\in\mathcal{S}} \gamma_{sp}u_s \geq z_p \quad \forall p \in \mathcal{P} \tag{7b}$$

$$u_s \geq 0 \quad \forall s \in \mathcal{S}. \tag{7c}$$

CLAIM 1. *The function* $\tilde{g}(\mathbf{z})$ *is convex in* $\mathbf{z}$.

Proof. First, note that the two terms of the objective are independent, as $\tilde{c}_A$ is a self-contained LP. We will show that both terms are convex in $\mathbf{z}$ by using strong duality, and hence so is their sum. Consider the LP corresponding to the first term of $\tilde{g}(\mathbf{z})$ and let y_p $(\forall p \in \mathcal{P})$ be the dual variables corresponding to constraint (7b). By strong duality, we have

$$\min_{\mathbf{u}\geq\mathbf{0}} \left\{ \sum_{p\in\mathcal{P}} \sum_{s\in\mathscr{S}} w_p \gamma_{sp} u_s, \text{ s.t. (7b)} \right\} = \max_{\mathbf{y}\geq\mathbf{0}} \left\{ \sum_{p\in\mathcal{P}} z_p y_p, \text{ s.t. } \sum_{p\in\mathcal{P}} \gamma_{sp} y_p \leq \sum_{p\in\mathcal{P}} w_p \gamma_{sp} \ \forall s \in \mathcal{S} \right\}.$$

The right-hand side of the equation is the maximum of a linear function and clearly convex in $\mathbf{z}$, as the maximum of a convex function is also convex. As for the term $\tilde{c}_A(\mathbf{z}, \mathcal{N}_k, \mathcal{A}_k)$, an almost identical argument is used. Consider the LP relaxation of formulation (6). The objective of the dual is then the maximum over an affine function of $\mathbf{z}$, which is convex in $\mathbf{z}$. □

We present the above result as our rationale for employing a variant of gradient descent in $\mathbf{z}$. We do not have a guarantee on the quality of approximation $\tilde{g}(\mathbf{z})$ and as such, we use the original formulations with integrality constraints. Algorithm 1 details an approximate gradient directed SAA-SO heuristic to solve an instance of the CDSSP.

Algorithm 1: An approximate gradient directed SAA-SO heuristic.

Data: $\boldsymbol{\mathcal{N}} \leftarrow \{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_K\}$, $\boldsymbol{\mathcal{A}} \leftarrow \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_K\}$, $\mathsf{Iter}_{\max}$

Result: Required capacity decision vector $\hat{\mathbf{z}}$

Init: $\mathbf{z}$, s.t. $z_p \leftarrow 0 \ \forall p \in \mathcal{P}$, $\hat{c} \leftarrow \infty$, $\hat{\mathbf{z}} \leftarrow \mathbf{z}$

while *non_improvement_steps* $< \mathsf{Iter}_{\max}$ **do**

- $\mathbf{u} \leftarrow \mathsf{set\_covering}(\mathbf{z})$;
- $\bar{c}, \overline{\nabla} g \leftarrow \mathsf{simulation}(\mathbf{u}, \boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{A}})$;
- **if** $\bar{c} < \hat{c}$ **then**
  - *non_improvement_steps* $\leftarrow 0$;
  - $\hat{c} \leftarrow \bar{c}$;
  - $\hat{\mathbf{z}} \leftarrow \mathbf{z}$;
  - $\mathbf{z} \leftarrow \mathsf{perturb}(\mathbf{z}, \overline{\nabla} g)$;
- **else**
  - *non_improvement_steps* $\leftarrow$ *non_improvement_steps* $+ 1$;
  - $\mathbf{z} \leftarrow \mathsf{perturb}(\mathbf{z}, \overline{\nabla} g)$;

return $\hat{\mathbf{z}}$

Let us now explain Algorithm 1. Due to the non-convexity of $g(\mathbf{z})$ arising from integrality constraints and the inherent gap between $\mathbf{z}$ and $\mathbf{u}$, we have implemented the following two modifications to gradient descent:

1. We set $\mathsf{Iter}_{\max}$ to be the maximum number of consecutive steps with non-improving cost allowed before the algorithm terminates.

2. We use an approximate gradient $\overline{\nabla} g$ found from our K simulation runs to dictate the update of $\mathbf{z}$.

Modification 1 is self-explanatory as it allows us to escape local minima with an appropriately chosen $\mathsf{Iter}_{\max}$. Modification 2 is made to avoid the excessive noise and computations of an empirically estimated gradient. Specifically, let $\mathbf{e}_p$ be a basis vector of length P with a 1 in p^{th} position and 0 in every other. Then, empirically estimating the true gradient ∇g would require solving a set covering problem and running K simulations for each of the P estimates. Additionally, the estimate contains noise, as the cost of a single order realization $c(\mathbf{u}, \mathcal{N}_k, \mathcal{A}_k)$ is a function of $\mathbf{u}$, not $\mathbf{z}$. The change in cost caused by altering $\mathbf{z}$ is masked by potential changes in shift start and end times, which may affect the difference in cost disproportionately, especially for a large penalty cost θ. Instead, we use an approximate gradient $\overline{\nabla} g$ to dictate our search direction. Let $\boldsymbol{\mathcal{N}} := \{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_K\}$ and $\mathcal{N}_e \subseteq \boldsymbol{\mathcal{N}}$ be the set of all orders which expire for a given $\mathbf{u}$ from the simulation evaluation. The p^{th} position of $\overline{\nabla} g$ is then the number of orders in $\mathcal{N}_e$ whose ready to delivery window ($[t_o, t_d]$) overlaps with period $p \in \mathcal{P}$. Intuitively, increasing $\mathbf{z}$ in periods with larger numbers of expired orders has the potential to have the most reduction in cost. As such, $\overline{\nabla} g$ provides information about the direction and step size to take at each iteration, which we utilize in the $\mathsf{perturb}(\mathbf{z}, \overline{\nabla} g)$ function. Furthermore, the construction of $\overline{\nabla} g$ occurs concurrently with our simulation evaluation step, and thus is significantly faster than constructing an empirically estimated gradient which would require an additional $P - 1$ simulation evaluations in each iteration.

Next, we will explain the functions: $\mathsf{set\_covering}$, $\mathsf{simulation}$, and $\mathsf{perturb}$. The $\mathsf{set\_covering}$ function solves the following formulation:

$$\min_{\mathbf{z},\mathbf{u}} \sum_{p \in \mathcal{P}} \sum_{s \in \mathcal{S}} w_p \gamma_{sp} u_s \tag{8a}$$

$$\text{s.t.} \sum_{s \in \mathcal{S}} \gamma_{sp} u_s \geq z_p, \quad \forall p \in \mathcal{P} \tag{8b}$$

$$u_s \in \mathbb{Z}_{\geq 0}, \quad \forall s \in \mathcal{S} \tag{8c}$$

$$z_p \in \mathbb{Z}_{\geq 0}, \quad \forall p \in \mathcal{P}. \tag{8d}$$

This is equivalent to formulation (3) without the SAA term capturing the average cost of expired orders and ad-hoc couriers over K realizations. Thus, $\mathsf{set\_covering}$ constructs a minimum cost shift schedule for a given $\mathbf{z}$ (e.g., the current value of $\mathbf{z}$ in the algorithm). The $\mathsf{simulation}$ function evaluates the SAA term for a given shift schedule by simulating the events of K realizations of demand and ad-hoc courier arrivals. The details of the simulation can be found in Section 6.2.2.

Lastly, the perturb function alters the incumbent $\mathbf{z}$ according to the approximate gradient $\overline{\nabla} g$. Specifically, it adds an additional courier to the period with the most expired orders over the K realizations. For computational efficiency, perturb may add couriers in multiple periods in early iterations when there are many expired orders (i.e., a larger step size) and only a single courier in a single period as we approach the optimal value of the objective.

4.2. Limitations

The SAA-SO method is defined for a *specific* distribution $\mathscr{Y}_x$ (from which the dynamic order realizations are drawn) and ad-hoc courier arrival process $\mathscr{W}$. In practice, the delivery platform wants/needs to solve instances of the CDSSP multiple times a day for multiple markets, each of which is associated with a distinct $\mathscr{Y}_x$ and $\mathscr{W}$. In the following section, we propose a prescriptive machine learning method that conducts a variant of the SAA method offline, and deploys a trained machine learning model online to generate solutions to the CDSSP quickly, while maintaining a similar solution quality to SAA.

While we use small to medium-sized instances (on the order of 100 orders per day) to showcase our prescriptive machine learning method, the main usefulness of this solution approach is on large instances (e.g. those that are found in the meal delivery context in large urban areas, where we may have thousands of orders per day.) Even using the SAA-SO method to solve for a single market is computationally demanding for large instances as Ruszczyński and Shapiro (2003) note that the minimum sample size to guarantee a solution is ϵ-optimal with probability at least $1-\omega$ is

$$K \geq \frac{2\sigma^2 \ln \frac{|D|}{\omega}}{(\epsilon-\delta)^2},$$

where D is the size of the solution space, σ^2 is the variance of the objective function and δ is some chosen constant. As the variance and solution space increase in the scale of the problem (i.e., size of the service region, number of orders, number of ad-hoc arrivals, etc.), the required number of samples to guarantee adequate solutions also increases, resulting in large computation times. Therefore, the sheer number of samples required may make SAA-SO impractical in this context. While these effects can be mitigated to some degree by reducing the number of samples drawn, the solution quality degrades. Our prescriptive machine learning method, on the other hand, has an online solution generation component that is not only orders of magnitude faster than SAA-SO, but the online solution time relatively constant in increasing problem scale.

5. Prescriptive Machine Learning Method

In this section, we propose a prescriptive machine learning method that addresses the shortcomings of the SAA-SO method discussed in the previous section. The proposed machine learning approach creates a trained machine learning model that approximates a solution algorithm to an optimization

problem by training on a data set of instance/solution pairs. Figure 4 illustrates the method in the context of the CDSSP.

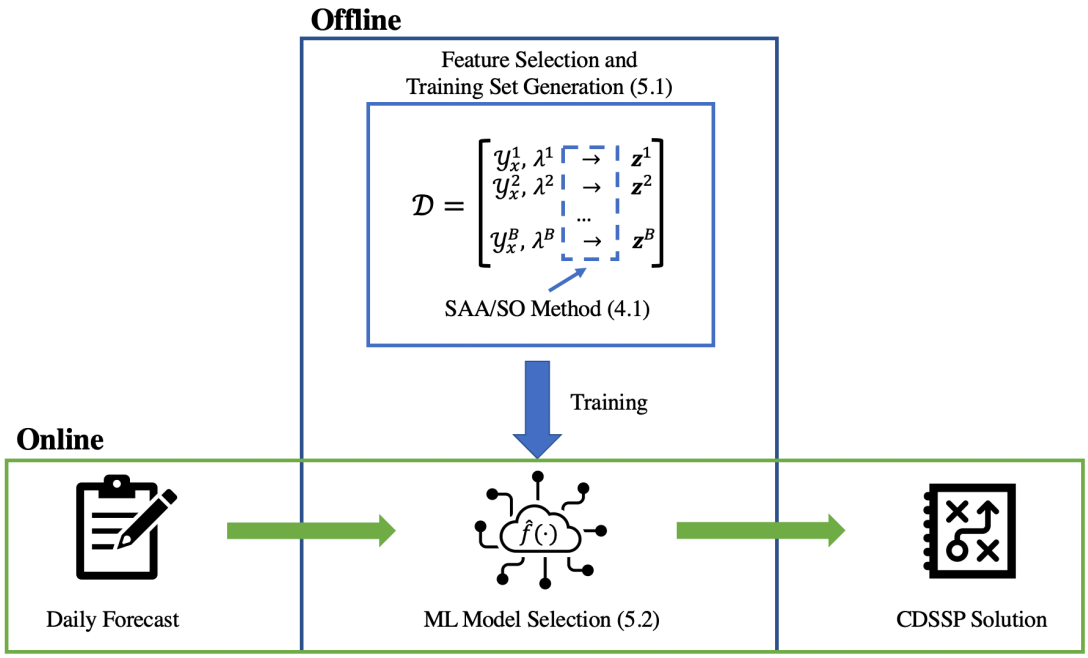

Figure 4 **Prescriptive analytics method for rapid satisfactory solution construction of large scale stochastic programming problems.**

The method is composed of an offline (training) and online (solution prescription) component. In the offline training, we first need to generate training samples, specifically, CDSSP instance/solution pairs. An instance of the CDSSP is a distribution, $\mathscr{Y}_x$, from which potential order set realizations, $\mathcal{N}$, is drawn and an ad-hoc arrival rate vector, $\boldsymbol{\lambda}$, defining a Poisson process, $\mathscr{W}$, from which ad-hoc courier arrival realizations, $\mathcal{A}$, are drawn. A solution to the CDSSP is a required scheduled courier vector $\mathbf{z}$, in which a set covering problem is solved over (after the fact) to generate a set of shifts $\mathbf{u}$. The instance of training set $\mathcal{D}$ are created from historical data (see Section 5.1) and the associated solutions to the instances are generated using the SAA-SO heuristic (described in detail in Section 4.1), which utilizes SAA to approximate the expectation in the objective of formulation (2) and simulation optimization (SO) to evaluate $c(\mathbf{u}, \mathcal{N}_k, \mathcal{A}_k)$ for the $k = 1, 2, ...K$ samples. A machine learning model is then trained on $\mathcal{D}$. In the online component, the machine learning model is used to construct solutions to instances of the CDSSP. Thus, in online deployment, the machine learning model bypasses the computationally expensive SAA-SO heuristic, while maintaining a comparable solution quality.

Recall that $\mathbf{z} = f(\mathscr{Y}_x, \boldsymbol{\lambda})$, where f is the mapping that transforms the distribution $\mathscr{Y}_x$ and $\boldsymbol{\lambda}$ into a solution $\mathbf{z}$, i.e., $f(\cdot)$ represents the SAA-SO heuristic described in the previous section. In this

section we describe how to use machine learning to estimate the function $f(\cdot)$ by an approximation $\hat{f}(\cdot)$ such that

$$\mathbf{z} = \hat{f}(\mathscr{Y}_x, \boldsymbol{\lambda}) + \epsilon, \tag{9}$$

where ϵ is the approximation error term. In the following subsections, we describe our methods of (1) feature selection, (2) training set generation, and (3) model selection.

5.1. Feature Selection and Training Set Generation

Firstly, we use the ad-hoc courier arrival rate parameters $\boldsymbol{\lambda}$ and the number of static orders directly. Next, as $\mathscr{Y}_x$ contains a forecast for a demand realization $\mathcal{N}$, it encompasses information regarding the distributions of the number and characteristics of orders. We assume the characteristics of the static orders are drawn from the same distribution as the dynamic orders. We assume these are random variables with probability density functions (pdf) fit from empirical data. As such, in our setting $\mathscr{Y}_x$ is composed of the marginal distributions of the following *independent* random variables: The number of dynamic orders that arrive in operating period $[0, \mathcal{T}]$, the placement time of an order, and the distance from the origin to the destination of an order. The fact that these random variables are independent implies that we only need to store their marginal distributions in $\mathscr{Y}_x$ to generate their realizations. Note that other scenarios with increased complexity may involve random variables that are correlated (e.g., delivery distance and promised delivery time). In such scenarios, $\mathscr{Y}_x$ is composed of the joint distributions of these variables.

We assume that we have access to the individual pdfs of the aforementioned random variables. In order to be used as input for machine learning models, this information must be converted into vector form to be used to create a training set. When creating a vector representation, we can construct discrete approximations of the pdfs. For the placement time this is accomplished by selecting a fixed number of time bins of equal length and constructing an approximate density histogram. That is, for each period $p \in \mathcal{P}$ we will have a probability that a given order's placement time falls within it. Alternatively, for random variables that approximately follow a specific parametric distribution, we can use the parameters (e.g., mean and variance) of said distribution to represent the random variable. For both the expected number of dynamic orders and the distance from the origin to the destination of orders, the data we received (detailed further in Section 6.1) indicated that the distributions are approximately normal. As such we use the mean and the variance of these random variables as input to the model.

The purpose of creating a training set is to provide a training algorithm with observations of input/output pairs to fit the model parameters of a machine learning model. These observations have a major impact on the quality of predictions for future observations. As such, the goal is

Algorithm 2: Training Set Generation

Data: $\boldsymbol{\mathcal{N}} \leftarrow \{\mathcal{N}_1, \mathcal{N}_2, \ldots\}$, $n_{\mathcal{D}}$

Result: Training set of size $n_{\mathcal{D}}$

Init: $\mathcal{D}$, an empty data set

for $1, \cdots n_{\mathcal{D}}$ **do**

- $\overline{\boldsymbol{\mathcal{N}}} \leftarrow \mathsf{random\_selection}(\boldsymbol{\mathcal{N}})$; `// $\overline{\boldsymbol{\mathcal{N}}}$ is a random subset of $\boldsymbol{\mathcal{N}}$`
- $\overline{\mathscr{Y}}_x \leftarrow \mathsf{fit}(\overline{\boldsymbol{\mathcal{N}}})$
- $\overline{\boldsymbol{\lambda}} \leftarrow \mathsf{random\_generation}()$
- $\overline{\mathbf{z}} \leftarrow \mathsf{saa\_so}(\overline{\mathscr{Y}}_x, \overline{\boldsymbol{\lambda}})$; `// Algorithm 1`
- $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\overline{\mathscr{Y}}_x, \overline{\boldsymbol{\lambda}}, \overline{\mathbf{z}})\}$

return $\mathcal{D}$

to create a large and diverse set of instance/solution pairs that are similar to expected future observations.

We present Algorithm 2 as an example of how this can be accomplished when given historical demand realizations, each of which represents a set of orders realized on a time horizon of length T. The goal is to generate $n_{\mathcal{D}}$ instance/solution pairs $((\mathscr{Y}_x, \boldsymbol{\lambda}), \mathbf{z})$ for the training set $\mathcal{D}$. To generate each instance/solution pair, we randomly select (with replacement) a subset of realizations from the historical observations and fit an approximate demand distribution to this subset. Specifically, this $\mathsf{fit}(\cdot)$ function fits approximate pdfs or parameters for a given distribution for each attribute. Next, we randomly generate a set of ad-hoc courier arrival rates $\boldsymbol{\lambda}$. As previously mentioned, the characteristic pdfs in $\mathscr{Y}_x$ are for both static and dynamic orders. As such we generate the characteristics static orders according to $\overline{\mathscr{Y}}_x$. Ad-hoc arrival rates and service levels can simply be generated uniformly at random over the desired domains of values. Afterwards, this overall distribution and set of auxiliary variables are used as input to the CDSSP optimization model and a solution is generated using Algorithm 1. The distribution $\overline{\mathscr{Y}}_x$ (in vector form), $\overline{\boldsymbol{\lambda}}$, and the associated solution ($\overline{\mathbf{z}}$) are then stored in the training set $\mathcal{D}$. This procedure is repeated until our training set is of the desired size. As we sample with replacement, as opposed to using the whole set of historical data, we are essentially performing bootstrap sampling to create our training set. In the simulation experiments (Section 6.2.1), we provide the details of our vector representation of $\mathscr{Y}_x$, discrete approximation of the pdfs, and training set creation. In the following section, we discuss the rationale behind the machine learning models considered in this paper.

5.2. Model Selection

Our goal is to approximate the function f found by the SAA-SO heuristic described in Section 4.1 by some approximate function $\hat{f}$, illustrated in Equation (9). For this purpose, we utilize artifi-

cial neural networks (ANN), specifically multilayer feedforward networks. The universal approximation theorem presented by Hornik, Stinchcombe, and White (1989) (founded on the classic Stone–Weierstrass Theorem) asserts that multilayer feedforward networks are able to approximate any measurable function to any degree of accuracy for both single and multiple output cases. Consequently, the result implies that given sufficient training and proper hyperparameters (e.g., hidden layer length and width, activation functions) artificial neural nets will produce desirable predictions.

The above concept serves as motivation to implement and test the efficacy of two main configurations of multilayer feedforward networks in the context of the CDSSP. The first configuration is illustrated in Figure 5, which depicts a multilayer feedforward network with multiple outputs. In such a configuration, all entries of $\mathbf{z}$ are prescribed *concurrently* given a single demand forecast

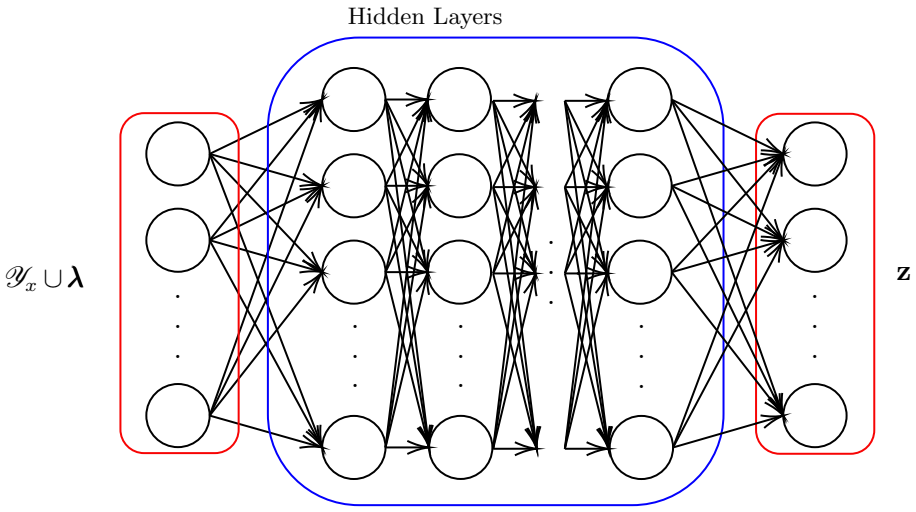

Figure 5 **Multilayer feedforward artificial neural network with multiple outputs.**

and set of auxillary variables. We hereby refer to this configuration as ANN.

Alternatively, in the second configuration illustrated in Figure 6, we construct a single output network which prescribes a single entry of $\mathbf{z}$ and is employed *recurrently*. In addition to the demand forecast and auxiliary variables, we also have a fixed size history set $\mathcal{H}_p$ to be used as input, which contains information regarding the prescriptions for the periods occurring before period p. We hereby refer to this configuration as RANN. As the individual entries of $\mathbf{z}$ are dependent on one another, explicitly including $\mathcal{H}_p$ may result in increased performance over the multiple output network. Note that given a training set of $n_{\mathcal{D}}$ observations, the recurrent configuration then has $n_{\mathcal{D}} \cdot P$ training observations. In cases where constructing and maintaining a large training set is expensive, this property is especially useful. The outputs of the predictions of the ANN and RANN are not guaranteed to be integers. That is, each entry of $\mathbf{z}$ may be a real number. As such, after

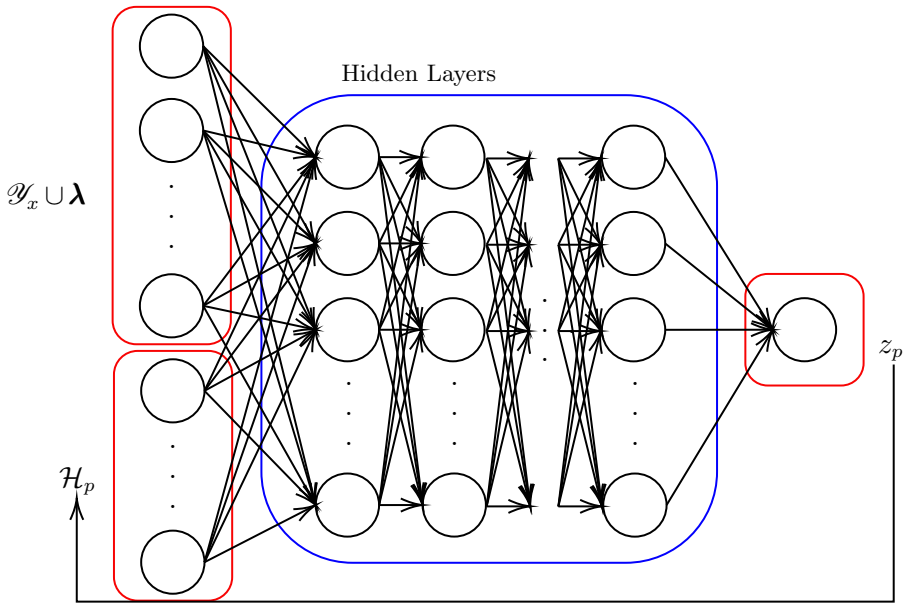

Figure 6 **Multilayer feedforward artificial neural network with a single output, including some fixed size history set $\mathcal{H}_p$ containing information surrounding predictions of previous periods.**

predicting $\mathbf{z}$ we apply a rounding function to ensure that we have integer values for the number of scheduled couriers to hire in a given period (in our simulation experiments we test the usefulness of both traditional nearest integer rounding and a ceiling function). In Section 6.2.1 we provide the details of the specific implementation of the two aforementioned multilayer feedforward network configurations.

6. Simulation Experiments

In this section, we discuss the simulation experiments conducted to provide proof of concept for our prescriptive machine learning method. We utilize real-world data from a crowdsourced delivery platform to construct instances of the CDSSP that are reflective of the crowdsourced delivery domain. In the following subsections, we describe our specific implementation of the proposed methods and present the computational performance of said methods and the quality of our solutions. We have included the data sets and source code used in this paper in the Github repository found at https://github.com/adambehrendt/Crowdsourced-Courier-Scheduling.

6.1. Data Description, Processing and Assumptions

6.1.1. Demand We received 177 days worth of real demand data from a crowdsourced delivery platform from July 2019 to January 2020. The demand data are for a single market with a single vendor and 4 unique pickup locations. For each order in the dataset, we have been provided the ID of the pickup location, the ready time, the delivery deadline, and the estimated travel time from the origin to the destination. Each order's ready time was exactly on-the hour (e.g., 8:00 a.m., 9:00

a.m., etc.) with a 1-hour delivery window. Less than 1% of these orders had a ready time outside the range of 8:00 a.m. to 7:00 p.m. and as such were removed from the data set. The average number of orders on a given day was 42.05, where the average estimated travel time from an origin to a destination was 10.17 minutes. This is a relatively low volume market, and we scale the number of orders (while preserving the relative distribution of ready times) to test the usefulness of our methods on medium-sized instances around 3 times this size, on the order of 120 orders per day (described further in subsection 6.3). In order to study the more interesting case of continuous ready times, we augment the ready times of the orders by adding a random number from 0 to 60 minutes for each order. Our processed ready times then span from 8:00 a.m. to 8:00 p.m. and the latest possible deadline is 9:00 p.m. As such, we have a planning horizon of 13 hours, where the latest ready time is guaranteed to occur no later than the 12th hour. Additionally, order placement times were not present in the data set, and we assume the placement time of each *dynamic* order occurs a fixed 45 minutes before the ready time, or is 8:00 a.m. for all orders with a ready time less than 8:45 a.m. As for *static* orders, the placement time is exactly 8:00 a.m. regardless of the ready time. We assume the service level target is 100% (i.e. we receive some penalty for every expired order), and ad-hoc couriers' choice behavior is to select an order uniformly at random from the set of orders belonging to the closest depot (i.e. minimize their detour). In Section 6.2.1 we detail our approach to training set generation, specifically how we fit distributions, dictate which orders are static or dynamic, and generate ad-hoc courier arrival times.

6.1.2. Ad-hoc Couriers Ad-hoc courier arrivals were not present in this data set. As such, we consider cases where ad-hoc couriers arrive according to either a homogeneous or inhomogeneous Poisson process with known rate parameters (which we assume can be estimated from historical data). We discuss our specific procedure for generating these arrivals in the following section. We assume that the ad-hoc couriers arrive uniformly over the service region and abide by the following simple choice model. Recall that ad-hoc drivers are compensated per order with a fixed payout. A natural utility for ad-hoc couriers is to minimize the total travel time, so as to maximize their earnings per hour. However, this would disregard any destination preferences of the ad-hoc couriers, which may be present especially for serving jobs in a one-off fashion on a pre-planned trip. As such, we model their behavior by minimizing the empty miles they drive (that is, a courier selects an order that minimizes the distance from the courier's arrival location to the order's pickup location) and choose uniformly at random among orders with the same empty miles. That is, each ad-hoc courier selects uniformly at random from the set of orders at the pickup location closest to the courier's arrival location that has active orders. If no orders are in the system upon the arrival of an ad-hoc courier, we assume that the ad-hoc courier leaves the system without serving any

orders. Upon completing an order, the ad-hoc courier also leaves the system, as we assume that an ad-hoc courier only serves a single order. We recognize that there are many variants of this simple choice model that may be reasonable in this or other contexts with different compensation schemes. However, our simple choice model takes into account the random arrival locations of ad-hoc couriers (by minimizing empty miles) and introduces variation in their choices (by randomly selecting amongst orders). As such, we believe the uncertainty introduced by the choice model is sufficient to be used as a means to test the efficacy of the solution approaches detailed in this paper.

6.2. Implementation

We consider period lengths of 30 minutes and therefore have $P = 26$ periods over the 13-hour planning horizon. Additionally, we let the minimum courier shift length be 2 hours and the maximum be 6 hours. Below we describe how we predict the required number of couriers $\mathbf{z}$ and solve a set covering problem to construct courier shifts $\mathbf{u}$.

6.2.1. Machine Learning. The features used an input to our machine learning models include a discrete probability distribution of time segments in which order ready times fall, the mean and variance of the travel times from origin to destination, the mean and variance of the number of dynamic orders that arrive on the planning horizon, and the number of static orders. We consider time segments of 15 minutes for the ready time probability distributions over the 12 hour period that order ready times can fall. As such, we have 53 features describing the demand. As for ad-hoc courier arrivals, we consider both homogeneous and inhomogeneous Poisson arrival processes and hence have a single additional feature λ in the homogenous case and a feature vector of size 26 for the inhomogenous case. The offline training portion of our proposed method involves the training set generation and the actual training of the model. The majority of the effort required for the offline training (in terms of computing time) is due to the generation of the training set because we execute the SAA-SO algorithm for each data point, which can be expensive depending on the instance size and the number of samples. The model training, on the other hand, does not depend much on the instance and sample size and generally takes a few hours.

Bootstrap Training Set Generation. As previously mentioned, we have 177 historical observations of demand (sets of orders over a time horizon). Using this set of historical data $\boldsymbol{\mathcal{N}} = \{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_{177}\}$ and a desired training set size of $n_{\mathcal{D}} = 10^4$, we generate a training set $\mathcal{D}$ by implementing Algorithm 2. As such, we now define the random_selection, fit and random_generation functions we use. The function random_selection performs bootstrap sampling, where we first uniformly select a random integer from 0 to 177 as the number of historical observations to use and then sample this number of observations with replacement. This allows us to create a training set

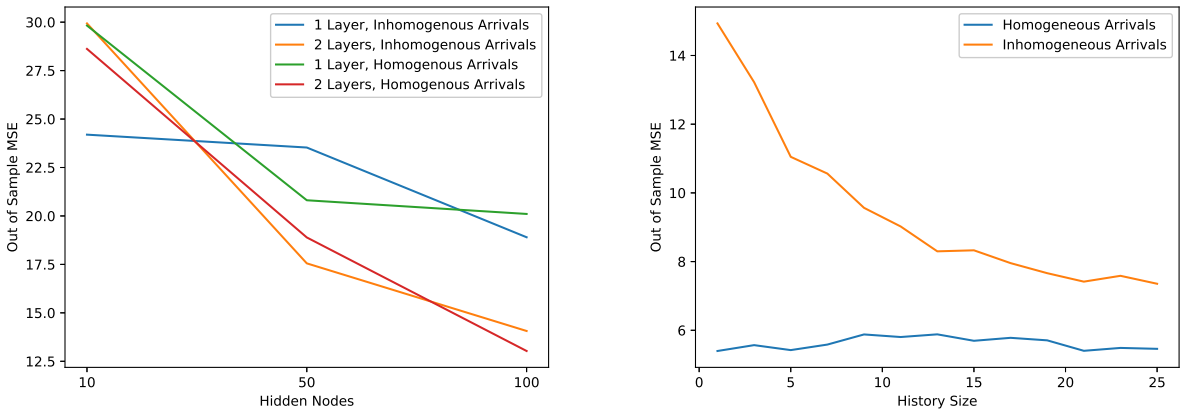

Figure 7 **Training error in relation to hidden layers, number of hidden nodes, and history size (in the recurrent case).**

much larger than that of the set of historical observations and reduces the probability of over-fitting our models to a single set of observations. The function fit then constructs the aforementioned features from this subset of observations; however, we scale the observed number of total orders by a factor of 3. We now use random_generation to construct a partition of static and dynamic orders and generate the ad-hoc courier rate parameters $\boldsymbol{\lambda}$. Firstly, we decide which orders are static or dynamic. We create a partition by the following procedure. Let μ be the mean of the total number of orders in each observation of the given subset (scaled by a factor of 3). Let β be the proportion of orders that are static, which we draw uniformly at random from $[.25, .5]$. We then record the number of static orders as $\lfloor \beta \cdot \mu \rfloor$ and the mean number of dynamic orders as $\mu - \lfloor \beta \cdot \mu \rfloor$. We let the variance of the number of dynamic orders be the variance of the total number of orders, however in practice, one would estimate this value by using a given partition directly. For homogeneous ad-hoc courier arrivals, we let $\lambda_{\max} = 2$ (per period, 26 for the entire time horizon) and draw a rate parameter λ uniformly between 0 and $\lambda_{\max}$. For inhomogeneous ad-hoc courier arrivals, we draw rate parameters λ_p uniformly between 0 and $\lambda_{\max}$ for each. Again, we set $\lambda_{max} = 2$.

Hyperparameter Selection. We completed a simple exhaustive hyperparameter search for the ANN and RANN. Figure 7 depicts the results of our search, in terms of the out of sample mean squared error. We performed K-Fold cross-validation on the data for homogenous and inhomogeneous arrivals. Each hyperparameter setup was implemented and the models were trained for a fixed number of training epochs, after which the out of sample error was averaged. The figure on the left shows the efficacy of the number of layers and the number of nodes per layer (where we used the same number of nodes for each layer). As such, we implement both our ANN and RANN with 2 hidden layers and 100 hidden nodes each. While both arrival cases have similar

average performance results, we note that the best performing models (as each training replication begins from a different random seed) for each hyperparameter configuration tend to be for the homogenous arrival case, which aligns with the slightly better performance we observe in the results presented in Section 6.4. The figure on the right shows the training error of the RANN when using different history sizes (the number of prior periods used as input). We see that the maximum history size yields the lowest out of sample error for the inhomogenous arrivals case and that the sample error remains constant for different history sizes for the homogenous case (which is what we expected). As such, we implemented a history size of 25 for the RANN for both homogeneous and inhomogeneous arrivals. Lastly, two additional inputs we used in the recurrent configuration are the current sum of $\mathbf{z}$ up until the period being predicted ($\sum_{i=0}^{p-1} z_i$, when predicting z_p) and the index of the current period being predicted.

6.2.2. Simulation. In order to evaluate the cost function $c(\mathbf{u}, \mathcal{N}_k, \mathcal{A}_k)$ for a given schedule $\mathbf{u}$ and order realization $\mathcal{N}_k$, we have implemented an agent-based and event-driven simulation. *Agent-based* refers to the fact that we model couriers and orders as individual entities. *Event-driven* signifies that the simulation clock is managed through chronological discrete events. We can then explicitly define an assignment and routing policy π by detailing the decisions of the delivery platform at each event. We are concerned with when a courier shift starts, a courier shift ends, a new order arrives, an ad-hoc courier arrives and the end of the time horizon has been reached. As previously mentioned, the optimization of π is beyond the scope of this paper, and we treat π as input to the simulation. For this reason, we implement a heuristic policy based on a cheapest_insertion function (formally defined in Algorithm 3.) This function captures the fundamental operation, i.e., deciding the least-cost insertion of an order in a route, and is used to generate and update routes.

Let r be an existing route of length n composed of a sequence of pickup and delivery locations, where locations are labelled by their position in the route. Location i has a feasible time window (in which pickup/delivery can occur) defined as $[e_i, v_i]$. (In our implementation, for pickup locations, we set e to the ready time of the associated order t_o and v to $+\infty$, and for delivery locations we set e to $-\infty$ and v to the delivery time promise of the associated order t_d.) Lastly, let $t_{i,j}$ be the travel time from location i to location j. For route r we define the following recursion

$$\hat{T}_j := \hat{T}_{j-1} + t_{j-1,j} + \max\{0, e_j - (\hat{T}_{j-1} + t_{j-1,j})\}, \quad \forall j = 1, 2, \ldots, n,$$

where $\hat{T}_j$ is the time that the courier serving route r leaves location j. We let $\hat{T}_0$ be the current time and location index 0 be the current location of the courier serving route r with feasible time window $[-\infty, +\infty]$. We can now verify the feasibility of any route r by checking the simple condition

Algorithm 3: Cheapest Insertion Operation

Data: Route r visiting $n+1$ locations, pickup location o with time window $[e_o, v_o]$, delivery location d with time window $[e_d, v_d]$

Result: Cost of insertion, update route $\hat{r}$

Init: *insertion_list*, an empty list to store feasible insertions and their cost

```
for i = 1, ··· , n do
    if feasible_insertion(o, r, i) then
        r̂, Δ_o ← insert(o, r, i) ;          // edits route, records change in cost
        for j = i, ··· , n + 1 do
            if feasible_insertion(d, r̂, j) then
                r̂, Δ_d ← insert(d, r̂, j);
                c = Δ_o + Δ_d;
                insertion_list.append(c, r̂);
                r̂ ← remove(d, r̂, j);
            else
                pass;
    else
        pass;
return minimum cost entry of insertion_list
```

$\hat{T}_j \leq v_j \ (\forall j = 1, 2, \ldots, n)$. As such, the function feasible_insertion(location, route, index) returns true if a location is feasibly inserted into a route at a given index and false otherwise. Similarly, the insert function executes the specified insertion and records the cost. As such, for a given order with pickup location o and delivery location d, Algorithm 3 evaluates the cost of all possible insertions into a route and returns the cheapest such insertion. Naturally, this algorithm can be used for route construction when used sequentially. Additionally, if there are multiple routes, we can use Algorithm 3 as a subroutine to determine which route is the cheapest to insert a given order.

Let us now explain the assignment and routing logic that takes place over the course of the simulation for a specific demand realization $\mathcal{N}_k$ and ad-hoc courier realization $\mathcal{A}_k$. As previously mentioned, a subset of orders are static ($t_p = 0$) and the other orders are dynamic ($t_p > 0$). When the simulation clock is greater than or equal to the order placement time of an order, we consider that order realized. Recall that our simulation is event-driven. We repeat the following procedure at each event time (denoted by t).

1. The set of expired orders and the set of active couriers are updated. We perform a termination check. That is, if t exceeds the the length of the time horizon T, then the simulation terminates and returns the set of expired orders.

2. All orders realized at time t are attempted to be inserted into the current courier routes (not including empty routes of new couriers, which are handled in the next step). As such, we iterate through a list of these orders, which is organized in order of non-decreasing delivery time promise. For each individual order, we call the cheapest_insertion function for each courier route and insert the order into the route that has the smallest change in cost.

3. The set of unassigned orders realized before time t (i.e., the excluding the orders from the previous step) are attempted to be inserted into the courier routes of couriers whose shift start time is exactly t in the same manner as the previous step. In doing so, we avoid attempting to insert an order into a given courier route more than once, as it is impossible for an order to be inserted into a route that it was already deemed infeasible for.

4. The number of ad-hoc couriers that arrive at time t and their locations are recorded. Each ad-hoc courier selects an order uniformly at random from the set of available orders belonging to the closest depot (that has active orders). If no orders are available at time t, then the ad-hoc courier is assumed to leave the platform and not serve any orders.

5. The time until the next event is calculated, denoted by δ. The location of each active vehicle is then updated based on this δ, the next location of their route and whether or not they must wait at the location.

The above simulation is used to evaluate $c(\mathbf{u}, \mathcal{N}_k, \mathcal{A}_k)$ and represents a routing and assignment policy π in which the use of scheduled couriers is prioritized over the use of ad-hoc couriers.

6.3. Solutions Methods

The solution methods we evaluate in the following results section are:

- SAA-SO heuristic (with varying sample size K),
- Prescriptive ANN model (with both rounding and ceiling post-processing tested),
- Prescriptive RANN model (with both rounding and ceiling post-processing tested), and
- Expected scenario (ES) heuristic.

We have described the first three solution approaches in Section 5 with implementation details in Sections 6.2.1 and 6.2.2. In order to illustrate the benefits of the SAA-SO heuristic and the prescriptive methods, we also present and evaluate a heuristic that generates solutions based on a single expected scenario which we refer to as the ES-heuristic. This heuristic does not rely on sampling or gradient based methods and as such, has a much faster offline training component. Let $\hat{\mathbf{z}}$ be a solution to the ES-heuristic. Specifically, we construct $\hat{\mathbf{z}}$ by first generating an expected scenario. That is, for a given demand distribution $\mathscr{Y}_x$ and ad-hoc arrival rate vector $\boldsymbol{\lambda}$ we estimate the expected number of ready orders in each period by multiplying the total number of expected orders (static plus expected dynamic) by the proportion of orders expected to be ready in that

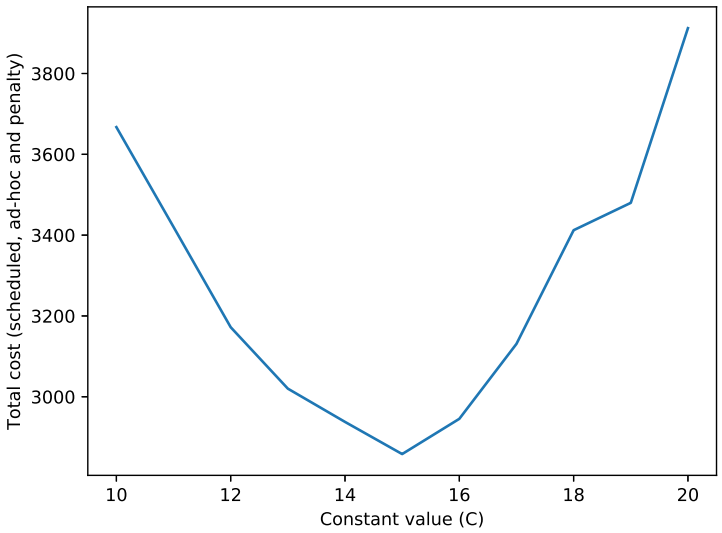

Figure 8 Search for best performing C on a homogeneous arrivals training set.

period. For ad-hoc arrivals, we simply use the expected number of arrivals directly from $\boldsymbol{\lambda}$. Additionally, we let $\bar{d}$ be the mean distance from origin to destination in $\mathscr{Y}_x$. As such, we generate a solution $\hat{\mathbf{z}}$ for an instance $(\mathscr{Y}_x, \boldsymbol{\lambda})$ by calculating

$$\hat{z}_p = \frac{1}{C} \cdot \bar{d} \cdot (\mathbb{E}[\text{ready orders in } p] - \mathbb{E}[\text{ad-hoc arrivals in } p) \quad \forall p \in \mathcal{P}, \tag{10}$$

where C is a tuning parameter that represents the approximate amount of time that a scheduled courier spends driving in a period. We then round the resulting vector $\hat{\mathbf{z}}$ to have an integer solution. Before deploying this equation online, we use our training set to search for an optimal value of C. For each observation in the training set, we generate an expected instance to test the solution generated for a given C with explicit simulation. This expected instance takes the expected number of orders and ad-hoc arrivals and assumes they are spread uniformly over their respective periods, while generating O-D pairs with distances consistent with the forecast provided in each $\mathscr{Y}_x$. As an example, we show the results of the search for a homogeneous arrivals training set in Figure 8. Based on these results, the value of C would be set to 15.

In the following section, we deploy the aforementioned solution methods online for the testing set and discuss results.

6.4. Results

We created training sets for both the homogeneous and inhomogeneous arrival cases each of size $n_{\mathcal{D}} = 10^4$. Recall that each training sample corresponds to one operating period (i.e., a day). We used 90% of the instances in each set for training and 10% of the instances for testing purposes. For

Sample size	Total cost	Gap %	Scheduled cost	Ad-hoc cost	Expired cost
K=50	3475.05	–	2444.74	224.98	805.32
K=25	3544.66	+2.0%	2415.41	229.12	900.13
K=10	3657.67	+5.3%	2371.37	235.11	1051.19
K=5	3682.46	+6.0%	2274.58	254.46	1153.43

Table 1 Change in cost of the SAA-SO method for varying number of samples (homogeneous arrivals).

Sample size	Total cost	Gap %	Scheduled cost	Ad-hoc cost	Expired cost
K=50	3384.03	–	2424.87	237.04	722.13
K=25	3357.50	-0.8%	2419.46	237.62	700.42
K=10	3498.31	+3.4%	2351.96	247.45	898.91
K=5	3684.30	+8.9%	2289.53	256.78	1137.99

Table 2 Change in cost of the SAA-SO method for varying number of samples (inhomogeneous arrivals).

the testing sets, we evaluate each respective solution on a set of 200 realizations, drawn from the conditional order distribution $\mathscr{Y}_x$ and ad-hoc courier arrival process $\mathscr{W}$. We use cost parameters $w_p = 10$ $(\forall p \in \mathcal{P})$ (scheduled courier wage per period), $n_a = 20$ (payout to ad-hoc couriers per order) and $\theta = 200$ (penalty cost of an expired order under the service level target), respectively. For the following results tables, "Gap %" is the difference in the total cost of a specific solution method and the method in the first row of the table, divided by the total cost of the first row of the table and then multiplied by 100. Tables 1 and 2 shows how the solution quality of the SAA-SO method changes with different value of K. For the homogeneous arrivals case, $K = 50$ performs the best, with the solution quality degrading as K decreases, as expected. In general, we see a similar trend in the inhomogeneous arrival case. The $K = 25$ case, however, performs slightly better ($< 1\%$) than $K = 50$, which indicates that the number of samples required to meet the maximum solution quality is met by $K = 25$, and this observation is due to randomness.

Method	Total cost	Gap %	Scheduled cost	Ad-hoc cost	Expired cost
SAA-SO (K=50)	3475.05	–	2444.74	224.98	805.32
ES-Heuristic	4055.72	+16.7%	2234.54	258.82	1562.35
ANN (rounding)	3564.63	+2.6%	2490.92	224.69	849.02
ANN (ceiling)	**3481.34**	**+0.2%**	2605.72	214.84	660.78
RANN (rounding)	3946.45	+13.6%	2458.55	238.91	1248.99
RANN (ceiling)	3810.612	+9.7%	2577.53	227.91	1005.18

Table 3 Average costs for homogeneous arrivals over for the SAA-SO heuristic and different ML model configurations of our prescriptive method.

Tables 3 and 4 detail the average costs of the ML solutions (trained on SAA-SO with $K = 50$) for the homogeneous and inhomogeneous arrival cases. In both cases, the ANN with a ceiling function applied to the output performs the best and had average total costs within 0.2% and 1.9% of the best performing SAA-SO heuristic solution, for the homogeneous and inhomogeneous cases

Method	Total cost	Gap %	Scheduled cost	Ad-hoc cost	Expired cost
SAA-SO (K=25)	3357.50	–	2419.46	237.62	700.42
ES-Heuristic	3784.67	+12.7%	2237.63	267.02	1280.02
ANN (rounding)	3489.46	+3.9%	2451.39	242.29	795.77
ANN (ceiling)	**3420.38**	**+1.9%**	2569.21	233.66	617.51
RANN (rounding)	3918.64	+16.7%	2449.16	256.50	1212.98
RANN (ceiling)	3766.96	+12.2%	2568.22	243.47	955.28

Table 4 Average costs for inhomogeneous arrivals over for the SAA-SO heuristic and different ML model configurations of our prescriptive method.

respectively. Our ML method performs better in the simpler case of homogeneous arrivals, most likely due to the uniform rate of arrivals for ad-hoc couriers. In the inhomogeneous case, the more irregular arrival pattern coupled with the random choice model introduces more variance in the solutions generated by the SAA-SO heuristic, making them more difficult to predict. Furthermore, we observe that ANN outperforms RANN and "ceiling" outperforms "rounding". The latter was expected as the ceiling function results in a larger number of required scheduled couriers and thus avoids ad-hoc courier and order expiration costs. Using a ceiling function to post-process the results tends to avoid underestimating the number of couriers required for a given period. While the additional scheduled courier cost tends to be higher than the scheduled courier cost of the SAA-SO solution, it avoids large expiration penalties which can be incurred when having too few scheduled couriers. The best performing ML models outperform the ES-heuristic in both the homogeneous and inhomogeneous cases. The ES-heuristic provides a solution based on an expected scenario. For cases with high variance (like the one we consider), this can perform even worse than SAA-SO with very few samples (see Table 1, $K = 5$). This result is not surprising because the expected scenario becomes less indicative of the overall distribution as the variance increases. Lastly, the ES-heuristic performs better in the case of inhomogeneous order arrivals (12.7% away from the best SAA-SO solution versus 16.7% for the homogeneous case). This is most likely due to the fact λ_p is known for each period, while the homogeneous case only has a single parameter. This allows equation (10) to better account for the presence of ad-hoc couriers. When compared to the homogeneous case, note the slightly increased cost of ad-hoc couriers in the inhomogeneous case and the significantly reduced expired order cost, indicating a more efficient usage of the existing ad-hoc couriers. However, the best ML model (ANN with ceiling function) still outperforms the ES-Heuristic by more than 10% in this case.

Figure 9 illustrates the difference in average online solution time between the SAA-SO heuristic and our prescriptive machine learning method (for both types of ML models). Specifically, these times do not include the training of the model, but include the creation of the required scheduled capacity $\mathbf{z}$ and the construction of courier shifts $\mathbf{u}$. Notably, the most computationally expensive

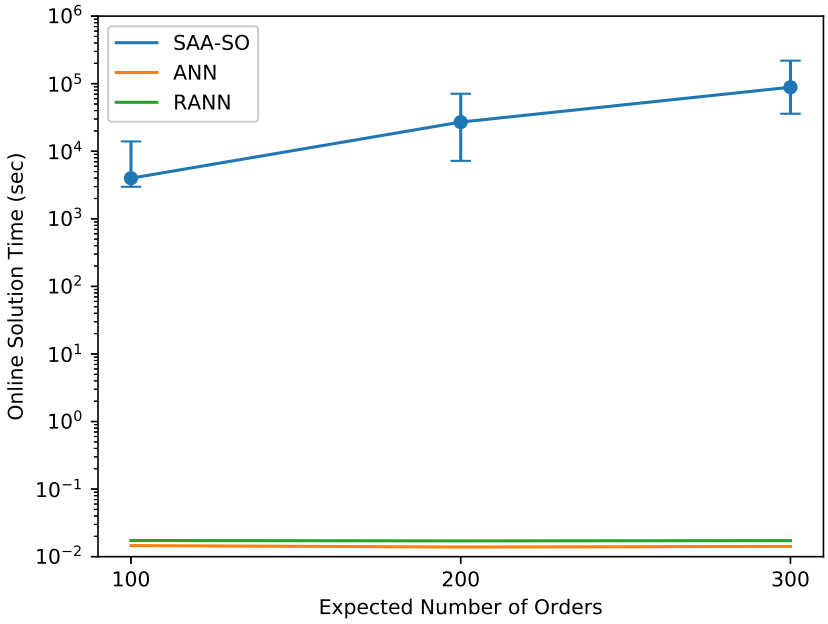

Figure 9 **Average online solution time comparison of the prescriptive ML method and the SAA-SO heuristic, for increasing instance size.**

step of the online deployment of the prescriptive method is the set covering problem. The SAA-SO times include realization generation and Algorithm 1. Expected numbers of orders of 100, 200 and 300 were tested, with standard deviations equal to one-fourth that. Additionally, we also let $K = N/4$ represent the increasing number of samples required as the variance of the objective increases. The solution time of the SAA-SO heuristic increases approximately linearly (note the log scale of the y-axis) in the instance size, while the solution time of the prescriptive machine learning methods is (almost) constant across the instance sizes considered. The use of RANN is slightly more computationally expensive than the use of ANN, as each entry of $\mathbf{z}$ must be prescribed individually, as opposed to concurrently. We note from Figure 9 that when the expected number of orders is 300, *the solution time for SAA-SO for even a single instance of CDSSP is on the order of a day, whereas the prescriptive ML methods require a fraction of a second.* As mentioned earlier, a crowdsourced delivery platform most likely needs to solve instances of CDSSP for multiple markets and, possibly, even more than once per day. Therefore, our proposed prescriptive ML method has a significant advantage over the SAA-SO heuristic, especially as the instance size increases. The intense computation is done offline and ahead of time, resulting in orders of magnitude faster online computational time, while producing comparable solutions.

7. Discussion

We have presented a prescriptive machine learning method for online construction of scheduled courier shifts on crowdsourced delivery platforms under uncertain demand and ad-hoc courier arrivals. We have shown using real crowdsourced delivery demand data that the costs of our prescribed solutions are within 0.2%–1.9% to those generated by our SAA-SO, and can be generated online in a fraction of the time (as the data and ML models can be generated and trained offline, ahead of time). For future research directions, note that although this paper focuses on crowdsourced delivery as the application setting, our prescriptive machine learning method can potentially be generalized to other applications in transportation and logistics that involve demand prediction, as the underlying simulation optimization model can easily be adapted to those settings. Additionally, in our computational experiments, we used a simple routing algorithm for scheduled couriers and a simple random choice model for ad-hoc couriers. In future work, more complex routing algorithms and choice models can be considered. In future work, complex pricing policies could be considered, namely the interaction between pricing decisions and the arrival process of ad-hoc couriers.

References

Alnaggar A, Gzara F, Bookbinder JH, 2020 *Distribution planning with random demand and recourse in a transshipment network. EURO Journal on Transportation and Logistics* 9(1):100007.

Alnaggar A, Gzara F, Bookbinder JH, 2021 *Crowdsourced delivery: A review of platforms and academic literature. Omega* 98:102139.

Arslan AM, Agatz N, Kroon L, Zuidwijk R, 2019 *Crowdsourced delivery—a dynamic pickup and delivery problem with ad hoc drivers. Transportation Science* 53(1):222–235.

Bent RW, Van Hentenryck P, 2004 *Scenario-based planning for partially dynamic vehicle routing with stochastic customers. Operations Research* 52(6):977–987.

Bertsimas D, Kallus N, 2020 *From predictive to prescriptive analytics. Management Science* 66(3):1025–1044.

Cao J, Olvera-Cravioto M, Shen ZJ, 2020 *Last-mile shared delivery: A discrete sequential packing approach. Mathematics of Operations Research* 45(4):1466–1497.

Clement J, 2019 *United States: retail e-commerce sales 2017-2023.* `https://www.statista.com/statistics/272391/us-retail-e-commerce-sales-forecast` (Accessed April 2021).

Dayarian I, Savelsbergh M, 2020 *Crowdshipping and same-day delivery: Employing in-store customers to deliver online orders. Production and Operations Management* 29(9):2153–2174.

Gurvich I, Lariviere M, Moreno A, 2019 *Operations in the on-demand economy: Staffing services with self-scheduling capacity. Sharing economy*, 249–278 (Springer).

Hornik K, Stinchcombe M, White H, 1989 *Multilayer feedforward networks are universal approximators. Neural Networks* 2(5):359–366.

Klapp MA, Erera AL, Toriello A, 2018 *The one-dimensional dynamic dispatch waves problem. Transportation Science* 52(2):402–415.

Kleywegt AJ, Shapiro A, Homem-de-Mello T, 2002 *The sample average approximation method for stochastic discrete optimization. SIAM Journal on Optimization* 12(2):479–502.

Lee A, Savelsbergh M, 2015 *Dynamic ridesharing: Is there a role for dedicated drivers? Transportation Research Part B: Methodological* 81:483–497.

Li B, Krushinsky D, Van Woensel T, Reijers HA, 2016 *The share-a-ride problem with stochastic travel times and stochastic delivery locations. Transportation Research Part C: Emerging Technologies* 67:95–108.

Liu S, He L, Shen ZJM, 2020 *On-time last-mile delivery: Order assignment with travel-time predictors. Management Science* Forthcoming.

Long Y, Lee LH, Chew EP, 2012 *The sample average approximation method for empty container repositioning with uncertainties. European Journal of Operational Research* 222(1):65–75.

Manyika J, Lund S, Bughin J, Robinson K, Mischke J, Mahajan D, 2016 *Independent work: Choice, necessity, and the gig economy. McKinsey Global Institute* 2016:1–16.

Ruszczyński A, Shapiro A, 2003 *Stochastic programming models. Handbooks in operations research and management science* 10:1–64.

Sampaio A, Savelsbergh M, Veelenturf L, van Woensel T, 2019 *Crowd-based city logistics*. Faulin J, Grasman SE, Juan AA, Hirsch P, eds., *Sustainable Transportation and Smart Logistics*, 381–400 (Elsevier).

Santini A, Viana A, Klimentova X, Pedroso JP, 2020 *The probabilistic travelling salesman problem with crowdsourcing. Optimization Online* .

Savelsbergh MW, Ulmer MW, 2022 *Challenges and opportunities in crowdsourced delivery planning and operations. 4OR* 1–21.

Schütz P, Tomasgard A, Ahmed S, 2009 *Supply chain design under uncertainty using sample average approximation and dual decomposition. European Journal of Operational Research* 199(2):409–419.

Srour FJ, Agatz N, Oppen J, 2018 *Strategies for handling temporal uncertainty in pickup and delivery problems with time windows. Transportation Science* 52(1):3–19.

Stroh AM, Erera AL, Toriello A, 2019 *Tactical design of same-day delivery systems*. Georgia Institute of Technology Working Paper.

Train KE, 2009 *Discrete choice methods with simulation* (Cambridge University Press).

Ulmer M, Savelsbergh M, 2020 *Workforce scheduling in the era of crowdsourced delivery. Transportation Science* 54(4):1113–1133.

Ulmer MW, 2020 *Dynamic pricing and routing for same-day delivery. Transportation Science* 54(4):1016–1033.

Verweij B, Ahmed S, Kleywegt AJ, Nemhauser G, Shapiro A, 2003 *The sample average approximation method applied to stochastic routing problems: a computational study. Computational Optimization and Applications* 24(2):289–333.

Voccia SA, Campbell AM, Thomas BW, 2019 *The same-day delivery problem for online purchases. Transportation Science* 53(1):167–184.

Yildiz B, Savelsbergh M, 2019 *Service and capacity planning in crowd-sourced delivery. Transportation Research Part C: Emerging Technologies* 100:177–199.

Zebra Technologies, 2019 *The future of fulfillment vision study.* `https://www.zebra.com/content/dam/zebra_new_ia/en-us/solutions-verticals/vertical-solutions/retail/vision-study/fulfillment-vision-study-report-en-us.pdf` (Accessed April 2021).