Neural Approximate Dynamic Programming for the Ultra-fast Order Dispatching Problem

Arash Dehghan[a], Mucahit Cevik[a,1,*], Merve Bodur[b]

[a]*Toronto Metropolitan University, Toronto, ON, Canada*
[b]*University of Edinburgh, Edinburgh, UK*

Abstract

Same-Day Delivery (SDD) services aim to maximize the fulfillment of online orders while minimizing delivery delays but are beset by operational uncertainties such as those in order volumes and courier planning. Our work aims to enhance the operational efficiency of SDD by focusing on the ultra-fast Order Dispatching Problem (ODP), which involves matching and dispatching orders to couriers within a centralized warehouse setting, and completing the delivery within a strict timeline (e.g., within minutes). We introduce important extensions to ultra-fast ODP such as order batching and explicit courier assignments to provide a more realistic representation of dispatching operations and improve delivery efficiency. As a solution method, we primarily focus on NeurADP, a methodology that combines Approximate Dynamic Programming (ADP) and Deep Reinforcement Learning (DRL), and our work constitutes the first application of NeurADP outside of the ride-pool matching problem. NeurADP is particularly suitable for ultra-fast ODP as it addresses complex one-to-many matching and routing intricacies through a neural network-based VFA that captures high-dimensional problem dynamics without requiring manual feature engineering as in generic ADP methods. We test our proposed approach using four distinct realistic datasets tailored for ODP and compare the performance of NeurADP against myopic and DRL baselines by also making use of non-trivial bounds to assess the quality of the policies. Our numerical results indicate that the inclusion of order batching and courier queues enhances the efficiency of deliv-

---

*Corresponding author

Email addresses: `arash.dehghan@torontomu.ca` (Arash Dehghan), `mcevik@torontomu.ca` (Mucahit Cevik), `merve.bodur@ed.ac.uk` (Merve Bodur)

[1]Toronto Metropolitan University, Toronto, ON, Canada

ery operations and that NeurADP significantly outperforms other methods. Detailed sensitivity analysis with important parameters confirms the robustness of NeurADP under different scenarios, including variations in courier numbers, spatial setup, vehicle capacity, and permitted delay time.
Keywords: Order dispatching, Ultra-fast delivery, NeurADP, Value function approximation

1. Introduction

The widespread adoption of online shopping, particularly accelerated by the COVID-19 pandemic, has transformed traditional markets in recent years and compelled many businesses to embrace streamlined direct delivery of products to customers [7]. One notable consequence of this shift is the emergence of Same-Day Delivery (SDD) services, which have fundamentally changed shopping behaviors by offering the convenience of online ordering and near-instant access to products. The SDD has seen a remarkable growth in recent years, with a valuation of \$5.77 billion in the United States in 2019 and a projected value of \$15.6 billion by 2024 [20]. Recognizing the evolving dynamics of the retail landscape, major players such as Target, Walmart, and Amazon have all acknowledged the significance of providing competitive same-day shipping options and have looked to expand their same-day shipping services [21]. As a result of this rapid expansion, centralized warehouses have become the central hub for managing incoming online orders and dispatching fleets of couriers, all with the goal of providing efficient and prompt service.

With the growing popularity of SDD operations, it is crucial to prioritize operational efficiency. The primary goal of SDD operations is to maximize the fulfillment of online orders while minimizing delivery delays. Nevertheless, SDD services naturally encompass several considerations that need to be taken into account in managing delivery operations. The courier shift schedules, vehicle capacities and dynamic routing of delivery couriers are some of the important considerations in this regard. Additionally, SDD operations involve a substantial level of uncertainty that stems from factors such as the timing, volume, deadlines, and destination locations of orders. These multitude of factors pose significant challenges for SDD operators, who must navigate them to provide efficient services. To this effect, Voccia et al. [28] introduce the Same-Day Delivery Problem (SDDP) as a framework to define the complex decision-making and routing logistics involved in ensuring the timely delivery of online orders within strict time constraints.

The SDDP may be decomposed into two distinct sub-problems: the Vehicle Routing Problem (VRP) and the Order Dispatching Problem (ODP). The VRP addresses the routing aspect, while the ODP concentrates on the matching and dispatching components of the problem. Both sub-problems are usually relevant in the context of a centralized warehouse handling stochastic order arrivals for cost-effective and timely delivery to customers. The VRP involves minimizing total vehicle travel distance, time, or cost while accounting for congestion, capacity, and time windows. On the other hand, the ODP involves assigning orders to couriers, minimizing fulfillment time and avoiding capacity breaches, factoring in location, time windows, and order size.

In this paper, we focus on the ultra-fast ODP, which involves the matching and dispatching aspects of the SDDP. Specifically, we explore a centralized decision-making problem in which a warehouse dispatches fleets of couriers, each with their own shift schedules, to maximize the number of orders served throughout the day. These orders arrive stochastically, and the warehouse's primary objective is to ensure *ultra-fast* deliveries, e.g., completing them within minutes. This requirement of urgent delivery introduces a critical time constraint, distinguishing it from other SDDP/ODP works which allow for more lenient delivery timelines. Furthermore, despite the widespread adoption of this rapid delivery approach by global delivery giants such as *Getir*, a renowned Turkish delivery service, and *Gorillas*, a Germany-based platform specializing in swift grocery and essential item deliveries, this particular delivery setting is not well-studied in the literature. In this regard, our paper contributes to the existing literature on the SDDP and ODP by building upon the work of Kavuk et al. [9], which focuses on an order dispatching problem based on Getir's operations. Specifically, they develop a deep reinforcement learning (DRL) approach for ODP to make informed decisions regarding only the acceptance or rejection of incoming orders to a single depot while the order assignments to the couriers are based on predefined rules. The main contributions of our paper are summarized in what follows.

We propose a novel Markov decision process (MDP) model that introduces several innovative features and capabilities to the single-depot ODP. In particular, in terms of operational enhancements (1) we employ *batching* as a means of enhancing the efficiency of the order dispatching operations and we utilize *courier queues* which enable the concurrent handling and emptying of all orders within the queue, rather than being limited to serving one order at a time, (2) we make *explicit courier assignments* to optimize the allocation of orders to couriers, and (3) we enforce *hard deadlines* to ensure timely delivery of orders. By incorporating these considerations, our proposed MDP model for ODP provides a more comprehensive and realistic representation of the dispatching process, thereby enhancing its practical relevance.

Inspired by its effective application to the ride-pool matching problem, and observing the suitability of our problem's structure to leverage its strengths, we adopt the Neural Approximate Dynamic Programming (NeurADP) as the solution approach. NeurADP is an innovative methodology introduced by Shah et al. [18] which combines Approximate Dynamic Programming (ADP) and DRL techniques and has exclusively been applied within the ride-sharing framework. Our study constitutes the first application of NeurADP beyond its original context, expanding its potential applications and further demonstrating its effectiveness in addressing real-world dynamic decision-making problems. We note that NeurADP is well-suited for ultra-fast ODP as it can address complex one-to-many matching and routing intricacies through a neural network-based value function approximation (VFA) that captures high-dimensional problem dynamics without requiring manual feature construction as in many other ADP frameworks. In order to demonstrate the effectiveness of NeurADP, we compare it with a large set of myopic and DRL baselines. We also conduct a sensitivity analysis to investigate the influence of various factors on the performance of NeurADP.

In our numerical study, to support our research and facilitate comprehensive evaluations, we introduce three original datasets specifically tailored for order dispatching operations in addition to considering a commonly used dataset from the literature. These datasets capture diverse real-world scenarios and provide a rich environment for training and testing our proposed methods. The availability of these datasets benefits future researchers in the field, fostering further advancements in the study of ODP and SDDP. Furthermore, we explore the effects of the number of agents considered, the spatial setup, the allowed vehicle capacity, and the permitted delay time. This analysis enhances our understanding of the robustness and adaptability of NeurADP in diverse scenarios, offering valuable insights to practitioners seeking to implement this approach in real-world applications. Importantly, our analysis also provides several managerial insights related to the ultra-fast ODP:

- NeurADP improves order fulfillment by 6.7%- 16.9% in the baseline configuration compared to the benchmark policies thanks to its ability to intelligently batch orders and optimize courier utilization. Furthermore, the greatest benefits of NeurADP-based policies are observed when there are fewer couriers who are working at or near full capacity or when the operation faces tighter delivery schedules.

- In terms of fulfillment strategy, NeurADP policy shows that the companies can enhance their efficiency by strategically batching the orders and ensuring the swift return of couriers to the warehouse, rather than attempting to maximize the number of orders loaded onto each courier's vehicle. That is, the companies can potentially serve more orders with quicker turnaround which could be more beneficial than simply loading couriers with as many orders as possible, especially in a high-demand and fast-paced delivery environment.

- The significance of policy intelligence in optimizing order fulfillment varies depending on the operation environment, particularly based on the sparsity of the delivery locations and their distance to the central warehouse. In cases where deliveries are more dispersed, the effectiveness of a policy becomes markedly more crucial. On the other hand, when delivery locations are densely clustered and closer to the warehouse, the relative importance of having an intelligent policy diminishes. This suggests that companies operating in diverse geographic settings should tailor their dispatching policies to the specific distribution characteristics of each area to optimize courier efficiency and order fulfillment rates.

The remainder of the paper is organized as follows. Section 2 offers a comprehensive review of the relevant SDD literature in the context of both the VRP and ODP, better positioning our research within the existing body of work. Section 3 provides a formal description of the problem setting for our ODP. In Section 4, we describe the NeurADP solution methodology. Details regarding the datasets and benchmark policies used in the experiments are provided in Section 5. The results of the computational experiments are presented in Section 6, followed by a conclusion in

Section 7 that summarizes the research findings and suggests avenues for future research.

2. Literature Review

We review the relevant literature by exploring the challenges and solutions related to the SDDP and its sub-problems, particularly ODP, highlighting the latest research and findings in this field, as well as placing our work and contributions within the broader context of the existing literature. In order to tackle the challenges in the SDDP, various strategies have been proposed, encompassing heuristic algorithms, machine learning models, combinatorial optimization models, reinforcement learning (RL) algorithms, and market-based mechanisms. Table 1 presents the most relevant studies to ours from the SDDP literature. This table comprises six indicators regarding the problem context and solution methodology. These are "Solution Technique", which describes the approach employed to solve the problem, "Large Capacity", which denotes whether couriers are allowed to carry multiple sets of orders simultaneously, "Multi-Courier", which indicates consideration of more than one courier, "Hard Deadlines", which signifies the presence of strict delivery deadlines, "Bundling", which pertains to the possibility of bundling/batching orders at the same decision-making step, and "Shifts", which indicates the incorporation of courier shifts into the respective problem formulation.

VRP has been explored within the context of the SSDP in various works. Ulmer et al. [26] presented an ADP-based order assignment/dispatching and routing policy which allows same-day delivery vehicles to better integrate dynamic requests into delivery routes through preemptive depot returns. Joe and Lau [8] combined DRL with a simulated annealing-based routing heuristic for a dynamic VRP. Their method uses a state representation based on the total cost of the remaining routes of the vehicles. Additionally, Côté et al. [6] proposed a re-optimization heuristic and a branch-and-regret heuristic that uses sampled scenarios to anticipate future events to address a variation of the VRP that involves urgent deliveries of time-sensitive orders. Ngu et al. [12] presented a decentralized multi-agent RL approach in formulating and solving the VRP using a parameter-sharing deep Q-network. Ulmer [22] proposed a method called anticipatory pricing and routing policy to improve the cost-efficiency of same-day delivery for e-commerce retailers. By dynamically adjusting prices based on delivery deadlines and using a guided offline VFA, this

Table 1: Summary of relevant studies. (ADP: Approximate Dynamic Programming, DRL: Deep Reinforcement Learning, CH: Combinatorial Heuristic, MIP: Mixed Integer Programming, PFA: Policy Function Approximation, NeurADP: Neural Approximate Dynamic Programming)

Study	**Problem**	**Solution Technique**	**Large Capacity**	**Multi-Courier**	**Hard Deadlines**	**Bundling**	**Shifts**
Ulmer et al. [26]	VRP	ADP	✓			✓	
Joe and Lau [8]	VRP	DRL	✓	✓			
Côté et al. [6]	VRP	CH	✓	✓	✓		
Ngu et al. [12]	VRP	DRL	✓	✓			
Klapp et al. [10]	ODP	MIP	✓			✓	
Kavuk et al. [9]	ODP	DRL		✓			✓
Ulmer and Streng [23]	ODP	PFA	✓	✓	✓	✓	
Ulmer and Thomas [24]	ODP	PFA	✓	✓	✓		
Chen et al. [5]	ODP	DRL	✓	✓	✓		
Cardona Peláez et al. [4]	ODP	PFA	✓	✓	✓		
Our Work	ODP	NeurADP	✓	✓	✓	✓	✓

policy incentivizes customers to select efficient delivery options, allowing the fleet to serve more orders and increase revenue. Finally, Dayarian et al. [7] explored the concept of drone replenishment in the context of same-day home delivery. They introduced the VRP with drone resupply and proposed various algorithms to optimize the delivery process, quantifying the potential benefits of using drones for delivery vehicles.

Several studies have specifically focused on the dispatching aspect of the SDDP. Cardona Peláez et al. [4] explored a two-echelon fleet approach that utilizes intra-route replenishment and policy function approximation (PFA) based on real-life geographical distributions to optimize fleet configuration and maintain service levels. Whereas Ulmer and Thomas [24] used fleets of heterogeneous drones and vehicles to perform deliveries, utilizing PFA based on geographical districting to decide which delivery method of transportation to use. Similarly, Chen et al. [5] proposed a same-day delivery system using both vehicles and drones and presented a deep Q-learning approach to learn the value of assigning customer orders to either drones, vehicles, or not offering service at all. To enable real-time dispatch decisions that balance speedy delivery with consolidation, Ulmer and Streng [23] introduced a novel same-day delivery approach that combines autonomous vehicles with pickup stations and utilized a PFA approach. In addition, Klapp et al. [10] formulated an arc-based Mixed Integer Programming (MIP) model and designed local search heuristics to solve the deterministic version of the ODP and derived an apriori solution for the stochastic case. Finally, Kavuk et al. [9] presented a DRL approach to solving the ODP for ultra-fast delivery, using deep Q-networks to learn the actions of warehouses and considering two reward functions: one related to the number of orders served and the other to minimize delivery delays.

In this paper, we consider the matching and dispatching components of the SDDP, particularly ODP, differentiating our focus from prior research that primarily concentrated on the routing elements [6, 8, 12, 26]. In particular, we consider a centralized decision-making scenario where a warehouse is tasked with coordinating fleets of couriers, each operating on its own shift schedule in order to optimize the total number of orders fulfilled throughout the day. These orders are subject to stochastic arrivals, and the primary goal of the warehouse is to achieve ultra-fast deliveries, aiming to complete them within a matter of minutes. This imperative for rapid delivery imposes a crucial time constraint, setting it apart from previous SDDP and ODP works which permit more relaxed delivery time-frames [8, 10, 12, 26]. The imperative to handle deliveries within minutes necessitates the capability of real-time decision-making, thereby challenging the feasibility of past traditional offline solutions [10]. The dynamic nature of the problem is further accentuated by the stochastic arrival of orders, requiring dispatching and matching strategies to flexibly adapt to varying patterns, distinguishing it from the approaches seen in previous works [17, 26]. Moreover, the added complexity arises from the individual courier shift schedules, which demand meticulous coordination and optimization to meet the stringent ultra-fast delivery criteria. This inclusion of courier shifts sets our work apart from prior studies such as [4, 5, 23, 24] which did not encompass this facet. In the pursuit of achieving such rapid deliveries, the allocation of couriers to orders becomes a task of precise resource allocation optimization, a contrast to the focus of traditional ODP studies, which primarily revolve around dispatching timing and transportation mode decisions (e.g., see [4, 23, 24]). Despite the widespread adoption of this rapid delivery approach by delivery corporations such as *Getir* and *Gorillas*, this particular delivery setting has not been well-studied in the existing literature.

We introduce a comprehensive set of improvements for order dispatching operations in the context of ultra-fast delivery, building upon the work of Kavuk et al. [9]. Their research closely aligns with ours, particularly as they focus on addressing the ODP encountered by *Getir*. In their study, Kavuk et al. [9] employed DRL to determine only the acceptance or rejection decisions for incoming orders. However, their work does not take into account important ODP considerations including courier assignment and order batching, and importantly, their framework does not impose strict delivery deadlines, rather penalizing the delays. Furthermore, their empirical analysis is limited to a single dataset and the rule-based baselines for comparative analysis. To further enhance the existing problem framework, our work introduces innovative features and capabilities, namely, order batching, explicit courier assignment and hard deadlines. Hence, it helps streamline the coordination and efficiency of order dispatching, contributing to the practical relevance of this problem. Moreover, we extend the scope of the problem to encompass different urban settings and larger-scale dispatching operations, involving more agents, orders, and a broader geographical area. This expansion enables us to capture the intricacies and challenges of managing substantial dispatching tasks, providing valuable insights for real-world applications. To tackle these challenges, we adapt the innovative NeurADP approach for order dispatching, which was originally designed for ride-sharing, marking its first application outside its original context and showcasing its effectiveness in dynamic decision-making problems. Moreover, to support our research and facilitate comprehensive evaluations, we introduce three novel tailored datasets for order dispatching.

3. Problem Description and Formulation

We present a dynamic order dispatching model that aims to efficiently match couriers with incoming batches of online orders. Our model considers the spatial and temporal demand patterns of the orders, which arrive dynamically over a 24-hour decision horizon and are served by a single centralized depot. This choice of a single central depot is particularly important in the context of ultra-fast delivery, where efficiency and speed are paramount. For instance, the logistics of coordinating multiple depots may introduce unnecessary delays and complexities, ultimately hindering the goal of rapid order fulfillment. Orders are generated stochastically and have specific delivery deadlines based on their arrival time. Once an order is assigned to a courier, it is accepted into the system and its delivery prior to its designated drop-off deadline is guaranteed. Moreover, the model takes into account a predetermined group of heterogeneous couriers available during the planning horizon, considering their capacity constraints and shift schedules. All couriers have individual shift start times, with each shift lasting six hours (excluding breaks) to reflect real-world shift lengths.

Our model incorporates several key problem specifications in the ultra-fast delivery setting to efficiently manage the dispatch and delivery process. First, multiple orders may arrive at any decision epoch, and couriers located at the warehouse are promptly dispatched upon being matched to these orders. Secondly, once a courier is dispatched with a set of orders, they must complete all assigned deliveries before returning to the depot, precluding any preemptive returns. To aid in this process, a queue is maintained for each courier with a capacity equivalent to their vehicle's limit. This queue accommodates both pending orders awaiting pickup and delivery, as well as new orders which may be matched to an on-shift courier as they continue their deliveries. Orders are incorporated into the queue only if their inclusion maintains adherence to constraints regarding timely delivery of all orders within the queue, steering clear of overloading courier vehicles, and staying within courier shift duration during order deliveries. Furthermore, the queue of orders is rearranged prior to the courier dispatching from the depot so as to optimize the route from the warehouse to each order destination location and back. However, once an order is assigned to a specific courier's queue, it cannot be transferred to another courier's queue. Lastly, in line with prior research on ODP, unmatched orders beyond their arrival period are assumed to exit the system [9]. This assumption reflects customers' general expectation of timely confirmation regarding the acceptance of their requests.

The primary objective of our model is to maximize the total number of online orders fulfilled within the decision horizon. To achieve this, our assignment decisions consider future order arrival uncertainties and the potential downstream impact of current decisions. To handle the complexity of these decisions, we formulate an MDP model and adapt a NeurADP solution framework, enabling effective real-time decision-making under uncertainty. To this end, we partition the finite planning horizon into discrete time intervals, with each interval having a duration of δ (e.g., five minutes). We assume that decisions are made at the onset of each interval, while exogenous information is observed continuously throughout. Following each interval, the state of the system is updated by incorporating the decisions and the observed external information. The collection of epochs for decision-making is denoted as $\mathcal{T} := \{0, \ldots, T\}$. At each decision epoch, the aim is to match "available" couriers with incoming orders. The availability of a courier is determined by several factors, including whether they are on their shift, their available capacity, and whether adding a new order to their assigned order set would comply with the maximum allowed delay for any order and would not extend the courier beyond their shift end time. Both the couriers who are stationed at the warehouse, as well as those who are away from the warehouse making deliveries, are eligible to be paired with incoming orders, provided they satisfy the availability constraints. Once paired with a batch of incoming orders, couriers located at the warehouse are promptly dispatched so as to adhere to the ultra-fast delivery requirements, while those making deliveries maintain a queue for orders to pick up and deliver after completing their ongoing assignments. The courier's queue accepts orders up until the moment the courier returns to the warehouse, at which point it is emptied, and the courier is promptly dispatched with any orders that may have accumulated in their queue. Couriers who are off-shift cannot be matched with orders, keeping their queues empty.

Each courier has a predetermined start time for their shift, with each shift spanning a duration of six hours, without any scheduled breaks. Incoming orders are typically associated with a specific delivery deadline which can be set in different ways depending on the company policies. For instance, in their paper, Kavuk et al. [9] consider a 45-minute delivery time for any given order and employ a reward function that promotes fast deliveries below this 45-minute target. Lastly, it should be noted that, at each time step, orders have the potential to be consolidated (i.e., batched) either with other concurrent orders or with previously assigned orders for each respective courier. The network housing the depot is characterized as $\mathcal{N} = (\mathcal{L}, \mathcal{E})$, where $\mathcal{L} = \{0, 1, \ldots, L\}$ corresponds to the depot and customer locations, and $\mathcal{E} = \mathcal{L} \times \mathcal{L}$ represents the distance between each respective pair of locations, determined by the Haversine distances. Given two locations ℓ, $\ell' \in \mathcal{L}$, we denote the travel time between ℓ and ℓ', leaving ℓ at time t, by $\mathtt{time}_t(\ell, \ell')$, such that $\ell = 0$ always denotes the depot location. Next, we present the components that make up the MDP model.

3.1. State Variables

The state of the system at time $t \in \mathcal{T}$ is defined by $S_t = (C_t, O_t)$, such that C_t represents the state of all couriers, and O_t the state of all incoming orders awaiting delivery. The state of an individual courier may be represented as a three-dimensional attribute vector defined by $c = (c_{\mathtt{shift}}, c_{\mathtt{ret}}, c_{\mathtt{ords}}) \in \mathcal{C}$ with $\mathcal{C}$ denoting the set of possible courier states. In this representation, $c_{\mathtt{shift}}$ indicates the time at which the shift of the courier starts. Moreover, if the courier is away from the warehouse fulfilling deliveries, $c_{\mathtt{ret}}$ signifies the time required for them to complete their deliveries and return to the warehouse. This value is set to zero if the courier is not on shift or is already at the warehouse. Lastly, $c_{\mathtt{ords}}$ represents the courier's queue and encompasses significant details about the courier's current tasks, including the orders they are currently assigned for delivery upon their return to the warehouse. The sequence of online orders assigned to the queue is optimized to minimize travel time for delivering all orders, and it is rearranged prior to a courier departing from the depot to make deliveries. Subsequently, the state of an online order is represented by a two-dimensional attribute vector denoted as $o = (o_{\mathtt{dest}}, o_{\mathtt{dead}}) \in \mathcal{O}$ with $\mathcal{O}$ denoting the set of possible incoming orders. Here, $o_{\mathtt{dest}}$ denotes the destination of the order, while $o_{\mathtt{dead}}$ corresponds to the specific delivery deadline time. When an order is received between decision epochs $t-1$ and t, the deadline attribute $o_{\mathtt{dead}}$ is determined at the start of epoch t using the equation

$$o_{\mathtt{dead}} = t + \mathtt{time}_t(0, o_{\mathtt{dest}}) + \mathtt{delay}_{\max}. \tag{1}$$

where $\mathtt{delay}_{\max}$indicates the maximum allowed time beyond the original travel duration from the depot to the order's drop-off location. While this method of calculating the delivery deadlines is slightly different than Kavuk et al. [9]'s approach, we note that our proposed framework can accommodate alternative ways of setting the delivery deadlines. Whereas we note that explicitly setting a delay parameter can help setting more realistic customer expectations.

3.2. Decision Variables

At each decision epoch $t \in \mathcal{T}$, we determine the matching between available couriers (i.e., those idly waiting in the warehouse or the busy couriers with available space in their queues) and incoming online orders considering the current system state. More specifically, we begin by examining the feasibility of grouping the set of incoming orders into batches, taking into account the order drop-off deadlines, and then evaluate the potential for assigning a specific batch to a courier, taking into consideration both capacity and timing limitations. To evaluate the feasibility of batching a set of orders together, or taking a single order by itself, we make sure whether a batch is able to be delivered before each order's respective drop-off deadline. In other words, a batching is feasible if there exists a viable route for a courier to deliver each order in the batch, as well as the orders it is currently assigned to, prior to each order's respective deadline beginning from the warehouse. Furthermore, when deciding if a courier can be paired with a batch of orders, we consider the following factors: (i) the courier's active status and current shift, (ii) whether the newly assigned batch pushes the courier's queue beyond the allowed limit, (iii) the courier's ability to deliver all orders before their specific deadlines, and (iv) whether the courier can complete all deliveries and return to the depot before their shift ends. We subsequently define the collection of actions taken at time t as $\mathbf{a}_t \in \mathbf{A}_t(S_t)$, such that $\mathbf{A}_t(S_t)$ denotes the set of all feasible actions for state S_t. These actions encompass both the matching of couriers to order batches, as well as the determination of delivery sequencing within each courier's assigned orders.

We define the reward collected at time step $t \in \mathcal{T}$ as follows:

$$R_t(\mathbf{a}_t) = \sum_{c \in C_t} \left(\beta \cdot q_t(\mathbf{a}_{tc}) \; - \; \omega_t(\mathbf{a}_{tc})\right). \tag{2}$$

Here, $q_t(\cdot)$ provides the number of orders fulfilled by a courier by taking the input action at time t, whereas $\omega_t(\cdot)$ represents the time required for a courier to deliver all the orders in its queue, including the ones associated with its current task and return to the warehouse. To ensure that the first term has more weight than the second one in the objective function, the multiplier of β is introduced. This parameter serves as a constant which takes into consideration elements such as maximum allowed queue size and geographical area. Its computation involves determining the longest conceivable queue duration, encompassing travel duration between different points and from those points to the warehouse on the map. For instance, if we consider a maximum queue size of three, β is determined by computing the three longest travel durations between different locations on the map. The incorporation of β is thus aimed at giving precedence to the maximization of the total number of orders fulfilled in each time interval. Nevertheless, when two feasible actions serve an equal number of orders, the decision rests on the option that enables the courier to finish their tasks in the least amount of time. This emphasis ensures that couriers become available more swiftly to handle new groups of orders.

3.3. Exogenous Information and Transition Function

During each time step within the decision horizon, the system receives a collection of online orders which constitute the exogenous information. The orders arriving between time t and $t+1$ are denoted as W_{t+1}. Moreover, W_0 represents the orders which have accumulated during the time between the final time step in the previous day and the initial time step $t = 0$, given a 24-hour planning horizon. Subsequently, the evolution of the system state from time t to $t+1$ is determined by the transition function that depends on the arrival of online orders and the decision tuple $\mathbf{a}_t \in \mathbf{A}_t(S_t)$. By introducing the post-decision state [15], the state transition can be divided into two distinct parts. The post-decision state captures the system state immediately after a decision has been made but prior to the arrival of exogenous information in the subsequent time step. The initial transition (3a) leads to the post-decision state via the action $\mathbf{a}_t$, and is denoted by $S_t^{\texttt{Courier-Post}}$. As unfulfilled orders exit the system at each time step, the post-decision state consists solely of information related to the couriers. The subsequent transition (3b) occurs from the post-decision state to the next state, influenced by the arrival of exogenous information W_{t+1}:

$$S_t^{\texttt{Courier-Post}} = \texttt{statepost}(S_t, \mathbf{a}_t) \tag{3a}$$

$$S_{t+1} = \texttt{statenext}(S_t^{\texttt{Courier-Post}}, W_{t+1}) \tag{3b}$$

Due to the assumption that unassigned orders exit the system at the end of each decision epoch, the state of orders at time $t+1$ is defined as follows:

$$O_{t+1} = W_{t+1} \tag{4}$$

Furthermore, the state of couriers at time $t+1$ is described as follows:

$$C_{t+1} = S_t^{\texttt{Courier-Post}} \tag{5}$$

such that $S_t^{\texttt{Courier-Post}}$ denotes the state of all couriers after taking the actions $\mathbf{a}_t$ and simulating their movements forward in time by one period, prior to the arrival of new exogenous information. For a courier, their state remains unchanged in the next time step if they are not on their shift or if they are at the warehouse without any assigned orders. Yet, if a courier is at the depot and receives new orders, they are sent out with their state updated to show the estimated return time. Similarly, if a courier is already out delivering and receives new online orders, their queue adapts while they continue their ongoing delivery route.

3.4. Optimal Policy

The objective in our order dispatching problem is to maximize the expected number of online orders served throughout the operation horizon:

$$\max_{\pi \in \Pi} \mathbb{E}_{W=(W_0,\ldots,W_T)} \left[\sum_{t \in \mathcal{T}} R_t \left(\mathbf{A}_t^{\pi}(S_t^{\pi}(W)) \right) \middle| S_0 \right]. \tag{6}$$

Through the solution of Equation (6), we can identify a policy π from a set of feasible policies Π which maximizes the reward when its recommended actions $\mathbf{A}_t^{\pi}(S_t)$ are sequentially implemented at realized states. The realized states are defined as follows:

$$S_0^{\pi}(W) = S_0 \tag{7a}$$

$$S_{t+1}^{\pi}(W) = \texttt{statenext}(\texttt{statepost}(S_t^{\pi}(W), \mathbf{A}_t^{\pi}(S_t^{\pi}(W))), W_{t+1}),\ t = 0, \ldots, T-1 \tag{7b}$$

such that S_0 corresponds to the initialized couriers, as well as the orders which have accumulated during the time interval from the final time step in the previous day up to the starting point at $t = 0$ within the initial state of the decision horizon, provided a 24-hour decision horizon. The future reward is determined by taking the expectation with respect to the stochastic process described by W. The actions and states encountered during each decision epoch depend solely on the revealed random variables up to that point, and accordingly the overall reward relies on the realization of the complete vector W. By solving the Bellman optimality equations, the optimal values $V_t(S_t)$ at each state S_t can be calculated as

$$V_t(S_t) = \max_{\mathbf{a}_t \in \mathbf{A}_t(S_t)} \left\{ R_t(\mathbf{a}_t) + \mathbb{E}_{W_{t+1}}[V_{t+1}(S_{t+1})] \right\} \tag{8}$$

where $S_{t+1} = \texttt{statenext}(\texttt{statepost}(S_t, \mathbf{a}_t), W_{t+1})$. To compute the value function $V_t(\cdot)$, a backward induction procedure can be employed, which involves working backward in time from the final epoch T [15]. This procedure considers the rewards associated with taking the optimal actions and the probabilities of transitioning between states. The recursive process continues until reaching the first stage, $t = 0$. However, this approach becomes impractical even for small instances due to the requirement of enumerating all possible outcomes and actions. Accordingly, ADP-based methods can be used to solve such problems.

4. Solution Methodology

In this section, we first describe an ADP approach for the ODP, highlighting its handling of the curses of dimensionality, VFA and updating methods. Subsequently, we introduce NeurADP as the more suitable method for our order dispatching problem and discuss its main distinctions from the ADP approach. Lastly, we provide our adaptation of the NeurADP algorithm for our problem setting.

4.1. ADP and VFA for Ultra-fast ODP

As outlined in Section 3, feasible decisions in the ODP involve not only the acceptance and rejection of incoming orders, but also the assignment of accepted order batches to courier queues and the determination of their respective routes. Recall the feasible set of decisions denoted by $\mathbf{A}_t(S_t)$. The definition of this set ensures that couriers are on their shifts, have adequate capacity for assigned orders, have viable routes for timely delivery, including any previously assigned orders, and can return to the depot before their shift concludes. Given this, the optimal policy for the ODP may be obtained using the Bellman optimality equations defined in Equation (8). However, computing $V_t(S_t)$ exactly proves intractable for complex large-scale problems such as the ODP due to what Powell [15] classifies as the "three curses of dimensionality", referring to the challenges of managing the state, action, and outcome space. More specifically, solving the Bellman optimality equation for a state S_t requires computing the anticipated downstream reward. This involves multiplying the value of each possible outcome S_{t+1} by the probability determined by the exogenous information W_{t+1}. However, for large-scale problems such as the ODP, the outcome space becomes excessively large, resulting in the first curse of dimensionality. To overcome this, ADP utilizes the concept of post-decision states, dividing the dynamic programming equation into two parts:

$$V_t(S_t) = \max_{\mathbf{a}_t \in \mathbf{A}_t(S_t)} \{R_t(\mathbf{a}_t) + V_t^{\texttt{Post}}(S_t^{\texttt{Courier-Post}})\} \tag{9a}$$

$$V_t^{\texttt{Post}}(S_t^{\texttt{Courier-Post}}) = \mathbb{E}_{W_{t+1}}\left[V_{t+1}(S_{t+1}) \middle| S_t^{\texttt{Courier-Post}}\right] \tag{9b}$$

To avoid enumerating the entire outcome space and evaluating future values, Equation (9a) establishes a deterministic optimality equation based on the post-decision state. Hence, it eliminates the need for such exhaustive computations. Equation (9b) expresses the post-decision state value function as the expected value of downstream rewards, where $S_{t+1} = \texttt{statenext}(S_t^{\texttt{Courier-Post}}, W_{t+1})$. However, the computational challenge arises from the high-dimensional state space, making it difficult to compute value functions for all feasible post-decision states. In the ODP, the post-decision state is influenced by various factors related to couriers and their potential states, including their current locations and shift-times, as well as their assigned orders and their respective deadlines. This complexity increase, known as the "second curse of dimensionality", is associated with the exponential increase in the state space. To address this, an approximation of the post-decision state value function, $V_t^{\texttt{Post}}(S_t^{\texttt{Courier-Post}})$, can be used.

There exist several types of value function approximations [16]. One approach to VFA involves the utilization of lookup tables in conjunction with state-wise aggregation. More specifically, a unique entry is assigned to each state in the lookup table, applying varying levels of aggregation to state values and using weighted summations to improve the accuracy of VFAs obtained from the aggregation levels. The utilization of basis functions offers an alternative approach for performing VFAs. These functions serve the purpose of transforming the original state space into a typically lower-dimensional form, with the goal of capturing influential state features which impact their values.

Another method employed for VFA is the dual heuristic approach, which relies upon the concept of marginal values. More specifically, rather than exclusively focusing on the inherent value associated with occupying a particular state, this method places greater importance on assessing how the value function changes concerning that state's derivative, hence enabling the prioritization of the rate of change in value rather than the absolute value itself. This often leads to more efficient problem-solving across a wide array of practical applications. Furthermore, this approach is particularly valuable for resource allocation problems, such as the ODP, where vector-valued decision problems, namely (9a) in this framework, are typically addressed, e.g., using linear, nonlinear, or integer programming. The dual heuristic approach has found widespread application in the transportation domain, for problems such as the ride-pool matching problem [29], taxi-on-demand [1, 19], and crowd-shipping [11]. However, we observe that this approach would have some important drawbacks when applied to the ODP. Since this observation motivates our proposal of instead employing NeurADP for the ODP, we next briefly explain the dual heuristic approach.

In its common practice, we would define the linear approximation of the courier-based post-decision value function using the linear decomposition of the function $\bar{V}_t^{\texttt{Post}}(S_t^{\texttt{Courier-Post}})$. This function incorporates the courier vector attributes in the post-decision state, specifically $\{S_{tc}^{\texttt{Courier-Post}}\}_{c\in\mathcal{C}}$, and is formally defined as follows:

$$\bar{V}_t^{\texttt{Post}}(S_t^{\texttt{Courier-Post}}) := \sum_{c\in\mathcal{C}} \bar{v}_{tc}^{\texttt{Post}} S_{tc}^{\texttt{Courier-Post}} \tag{10}$$

wherein $\bar{v}_{tc}^{\texttt{Post}}$ represents the expected down-stream reward associated with a courier being in the post-decision state of c at time t. This representation provides a considerable computational benefit, since rather than computing value functions for each possible post-decision state at the current time t, we need only $\left|\mathcal{C}\right|$ variables, which are denoted as $\bar{v}_{tc}^{\texttt{Post}}$. In the subsequent ADP algorithm, where n represents the iteration number and is used to index all the relevant components, the update for $\bar{v}_{tc}^{\texttt{Post}}$ is defined as follows:

$$\bar{v}_{tc}^{\texttt{Post},n} = (1-\alpha^n)\, \bar{v}_{tc}^{\texttt{Post},n-1} + \alpha^n\, \hat{v}_{tc}^{\texttt{Post},n} \tag{11}$$

such that α^n represents the step-size at iteration n of the algorithm, $\bar{v}_{tc}^{\texttt{Post},n-1}$ signifies the current approximate value of $\bar{v}_{tc}^{\texttt{Post}}$, and $\hat{v}_{tc}^{\texttt{Post},n}$ represents the observed marginal values associated with having an additional courier of type c at time t. The partial derivative values $\hat{v}_{tc}^{\texttt{Post},n}$ may be obtained as the numerical derivative of the following MIP model:

$$\max \quad R_{t+1}(S_{t+1}^n, \mathbf{a}_{t+1}) + \sum_{c\in\mathcal{C}} \bar{v}_{t+1,c}^{\texttt{Post},n-1} S_{t+1,c}^{\texttt{Courier-Post},n} \tag{12a}$$

$$\text{s.t.} \quad \mathbf{a}_{t+1} \in \mathbf{A}_{t+1}^{\texttt{MIP}}(S_{t+1}^n) \tag{12b}$$

where $\mathbf{A}_t^{\texttt{MIP}}(\cdot)$ represents an MIP formulation of the ODP feasible decisions. Such an MIP model would necessitate introducing decision variables for order acceptance/rejection, assignment of order batches to couriers, as well as those to decide courier routes along with various sets of constraints to ensure their feasibility. As such, this would not be a computationally viable approach, in particular due to the need to solve this MIP to optimality a large number of times. Therefore, in the literature, the common approach has been to transform the MIP into its linear programming (LP) relaxation, e.g., $\mathbf{a}_{t+1} \in \mathbf{A}_{t+1}^{\texttt{LP}}(S_{t+1}^n)$, and using LP duals; in our case this would be dual values associated with the constraints pertaining to courier flow conservation.

4.2. Motivation for NeurADP

The ADP methodology described above is not suitable for our ODP due to several reasons. Firstly, as noted for $\mathbf{A}_{t+1}^{\texttt{MIP}}(\cdot)$, our problem setting involves a complex decision-making process which necessitates a more intricate one-to-many matching between couriers and batches of online orders, as opposed to a straightforward one-to-one courier-order matching. Additionally, complex routing decisions have to be made for each courier adhering to respective order deadlines, further complicating the decision space. On the other hand, due to poor LP relaxations, updating the VFA parameters with the dual values of the matching LP is not a preferable option either. Furthermore, to mitigate the curses of dimensionality, ADP commonly employs an aggregated attribute space. For the ODP, this approach involves consolidating courier-related attributes, such as their locations, rather than considering each courier's individual state. However, crafting these state attributes manually requires domain expertise, which is challenging in the context of the ODP. This primarily stems from the complexity of the ODP as it involves numerous couriers with varying shift times, different numbers of orders, each having unique deadlines, and traveling along distinct trajectories. While such a manual approach may be reasonable in transportation problems such as the taxi-on-demand problem with a single request per driver, it becomes impractical when dealing with couriers who have personalized shifts and multiple orders to manage. Finally, although linear and piece-wise linear VFAs offer simplicity in their integration into MIP models, this simplicity may diminish modeling accuracy and representational power. This becomes particularly evident in intricate, high-dimensional problem settings characterized by non-linear dynamics and complex attribute dependencies, as observed in the ODP, rendering them inferior options for our specific problem setting.

NeurADP, introduced by Shah et al. [18] to address the one-to-many case of the ride-pool matching problem, is an innovative ADP-based algorithm explicitly designed to overcome the limitations of traditional ADP methods when dealing with large-scale problems. While both NeurADP and ADP aim to solve sequential decision-making problems by approximating the value functions of post-decision states, they differ in their approach. As mentioned, ADP typically relies on linear or piece-wise linear VFAs. In contrast, NeurADP utilizes a non-linear neural network-based VFA. This allows for an automatic compact low-dimensional state-space representation without the need for domain expertise for state-space aggregation as a means of dealing with high state space dimensionality. The neural network-based non-linear value function is then innovatively integrated into the MIP-based framework through a *two-step decomposition*: (1) the set of feasible actions for each courier is enumerated, (2) a matching integer program (much simpler than the aforementioned MIP consisting of all the decisions of the ODP) is solved over all couriers, with the values associated with each action integrated into the integer programming (IP) model as constants. Furthermore, rather than using LP-based duals to update these approximations as in ADP, NeurADP leverages DRL techniques for updating its value function approximations. More specifically, the gradients associated with the network parameters are computed and adjusted by minimizing the L2-norm between the current value function estimate and a one-step projection of the return derived from the Bellman equation. To enhance stability, NeurADP incorporates off-policy updates along with DRL techniques such as the implementation of a target network and Double Q-learning [27]. These additions further refine the algorithm and contribute to its improved performance. We explain these concepts in our ODP adaptation in more detail next.

4.3. NeurADP Solution Methodology

We next detail the NeurADP algorithm for our problem setting. We first describe the two-step decomposition enabled by NeurADP and explain identifying feasible courier-order matchings and the IP model for obtaining optimal matching. Then, the NeurADP-based VFA is explained, which is followed up by the description of the overall algorithm. Lastly, a brief discussion on the neural network architecture is provided.

4.3.1. Two-step decomposition

At every decision epoch t, given the state of the system $S_t = (C_t, O_t)$, the NeurADP solution methodology begins with enumerating feasible matchings between couriers and incoming batches of online orders. To evaluate the feasibility of batching a set of orders together or delivering a single order by itself, we take into consideration whether a batch can be delivered before each order's respective drop-off deadline. In other words, a batching is feasible if there exists a viable route for a courier to deliver each order in the batch, as well as the orders it is currently assigned to, prior to each order's respective deadline, beginning from the warehouse. With respect to the orders present in O_t, we first define B_t to represent the set of all order batchings with the minimum batch size of one and the maximum batch size of the available capacity of an empty courier's queue, denoted by $\texttt{queue}_{\texttt{max}}$. It is important to note that while we denote the maximum queue size to be equal for all couriers in our model, this simplification is made for the sake of notational clarity, and it may be varied for each courier in practice. To assess whether it is possible to match a courier c to a new order batch b, we consider the following constraints:

$$c_{\texttt{shift}} \leq t \tag{13a}$$

$$c_{\texttt{shift}} + \texttt{shift}_{\texttt{length}} > t \tag{13b}$$

$$\left|c_{\texttt{ords}}\right| + \left|b\right| \leq \texttt{queue}_{\texttt{max}} \tag{13c}$$

Here, the constraints presented in equations (13a) and (13b) ensure that the courier is actively working during the time when the batch of orders is assigned to them, with $\texttt{shift}_{\texttt{length}}$ denoting the shift length of couriers, while constraint (13c) guarantees that adding the orders from the newly assigned order batch to the courier's existing queue of previously matched orders does not exceed the maximum allowed capacity. Assuming the courier is currently on duty and has adequate storage capacity for both their new and previously assigned orders, we next confirm the existence of a feasible sequence for these combined orders. This sequence must ensure that each order delivery is completed before the respective drop-off deadline, and that the courier is able to successfully complete all deliveries and return to the warehouse prior to the end of their shift. We establish the new queue of online orders for courier c, encompassing both the previously assigned orders as well as those within batch b, by $c_{\texttt{ords}'}$, such that $c_{\texttt{ords}'} = c_{\texttt{ords}} \cup b$. We then define $D = |c_{\texttt{ords}'}|$, and introduce $\mathscr{Z}$ as the set of all permutations of $c_{\texttt{ords}'}$ where each $\sigma \in \mathscr{Z}$ represents a unique sequence for delivering the online orders in $c_{\texttt{ords}'}$. For instance, if $c_{\texttt{ords}'} = \{4, 6, 12\}$ and $\sigma = (12, 4, 6)$, then order 12 from $c_{\texttt{ords}'}$ is delivered first, followed by the order 4, and finally the order 6. Furthermore, given permutation σ, we let $o^{\sigma(d)}_{drop}$ indicate the earliest drop-off time of the order at the drop-off index d in σ, and $o^{\sigma(d)}_{\texttt{dest}}$ to represent its drop-off destination. Note that, given σ, these earliest drop-off times can be calculated in a forward manner starting from $d = 1$ to $d = D$, where $o^{\sigma(1)}_{drop} = t + \texttt{time}_t(o, o^{\sigma(1)}_{\texttt{dest}})$, $o^{\sigma(2)}_{drop} = o^{\sigma(1)}_{drop} + \texttt{time}_t(o^{\sigma(1)}_{\texttt{dest}}, o^{\sigma(2)}_{\texttt{dest}})$ and so on. Our goal is to ensure the existence of a permutation $\sigma \in \mathcal{Z}$ of online orders which meets the following constraints:

$$o^{\sigma(d)}_{dead} \leq o^{\sigma(d)}_{drop} \qquad \forall d \in \{1, \ldots, D\} \tag{14a}$$

$$c_{\texttt{shift}} + \texttt{shift}_{\texttt{length}} > o^{\sigma(D)}_{drop} + \texttt{time}_t(o^{\sigma(D)}_{\texttt{dest}}, 0) \tag{14b}$$

Constraint (14a) ensures that the courier is able to deliver all of their assigned orders prior to their individual deadlines. Furthermore, constraint (14b) guarantees that the courier is able to fulfill all deliveries and return the depot prior to the end of their shift. Then, the feasible set of matches at time t between couriers and order batchings may be described as follows:

$$F_t = \{(c, b) \in C_t \times B_t : (13a) - (13c),\ (14a) - (14b)\} \tag{15}$$

We note that the feasible set of matches, F_t, can be enumerated efficiently for the ultra-fast ODP with the capacitated couriers. Whereas for higher dimensional or less restricted problem settings (e.g., see [18]), full enumeration might not be achievable, in which case a subset of feasible matches can be heuristically generated.

In the second step, the NeurADP algorithm builds the following matching IP model to determine the decisions for each courier:

$$\texttt{MatchingIP:} \quad \max \sum_{c \in C_t} \sum_{f \in F_t \cup \{\varnothing\}} r_{tcf} \cdot a_{tcf} + \texttt{score}_t(c \leftrightarrow f) \cdot a_{tcf} \tag{16a}$$

$$\text{s.t.} \sum_{b \in B_t : (c,b) \in F_t} a_{tcb} + a_{tc\varnothing} = 1 \qquad \forall c \in C_t \tag{16b}$$

$$\sum_{c \in C_t} \sum_{b \in B_t : (c,b) \in F_t ; o \in b} a_{tcb} \leq 1 \qquad \forall o \in O_t \tag{16c}$$

$$a_{tcf} \in \{0, 1\} \qquad \forall c \in C_t, f \in F_t \cup \{\varnothing\} \tag{16d}$$

At each time step, there exists a default action for each courier denoted by $\varnothing$, indicating that they will not be assigned a new batch of orders. The constraints represented by (16b) guarantee that each individual courier c at time t is assigned to exactly one feasible action. Similarly, (16c) ensures that each individual order o is assigned to at most one courier. Furthermore, (16d) ensures that all decision variables are binary. (Importantly, we note that the set of actions for feasible matchings, F_t, may be further reduced by considering only the permutation for each updated queue $c_{\texttt{ords}'}$ which minimizes the time taken to make all deliveries in the queue and return to the depot.) Finally, the objective in (16a) calculates the total reward over the immediate reward of assigning courier c to the feasible matching f, which is calculated as in Equation (2) and denoted by r_{tcf}, as well as the downstream reward, denoted by $\texttt{score}_t(c \leftrightarrow f)$, gained from matching a courier c to a feasible decision f at time t.

4.3.2. NeurADP-based VFA

Next, given the `MatchingIP`, we detail how NeurADP approximates the value functions, linking it to the ADP content reviewed in Section 4.1. To derive the dispatching policy, NeurADP looks to solve the Bellman optimality equations introduced in Equation (8). Similar to ADP, NeurADP utilizes the concept of post-decision states, and divides the dynamic programming equation into two parts, namely Equation (9a) and Equation (9b). Furthermore, it aims to approximate the post-decision state value function, denoted by $V_t^{\texttt{Post}}(S_t^{\texttt{Courier-Post}})$, also by first performing a courier-based decomposition. More specifically, the value function of the courier-based post-decision state is decomposed into the individual couriers' value functions as follows:

$$\bar{V}_t^{\texttt{Post}}(S_t^{\texttt{Courier-Post}}) \approx \sum_{c \in \mathcal{C}} \widetilde{V}_{tc}^{\texttt{Post}}(S_{tc}^{\texttt{Courier-Post}}). \tag{17}$$

NeurADP then approximates the value functions of individual couriers. In doing so, the assumption is made that a courier's long-term reward is minimally affected by the actions of other couriers in the current decision epoch. This assumption, rooted in the idea that long-term rewards primarily stem from the interaction between delivery routes, enables modeling that focuses on the pre-decision state of other couriers. Thus, it simplifies and expedites the optimization of delivery routes and schedules by reducing computational complexity and not heavily weighing the numerous possible actions of other couriers. The approximation of individual courier value functions can be succinctly expressed as:

$$\widetilde{V}_{tc}^{\texttt{Post}}(S_{tc}^{\texttt{Courier-Post}}) \approx \hat{V}_{tc}^{\texttt{Post}}(S_{tc}^{\texttt{Courier-Post}}, \{S_{tc'}\}_{c' \neq c}). \tag{18}$$

Here, the approximated post-decision state value function for courier c, denoted by $\bar{V}_{tc}^{\texttt{Post}}$, accepts input data about its own post-decision state, $S_{tc}^{\texttt{Courier-Post}}$, as well as auxiliary pre-decision state information from the other couriers, $\{S_{tc'}\}_{c' \neq c}$. This auxiliary data provides context about the environment in which courier c operates before taking an action, including the number of couriers on break, those at the warehouse, the average occupied capacity of other couriers, and the volume of incoming orders at that time step. Incorporating this additional information allows for a more accurate evaluation of the post-decision state value, considering that order acceptance is influenced not just by a courier's state, but also by the competitive nature of their operational environment. The overarching value function may thus be written as follows:

$$\bar{V}_{t}^{\texttt{Post}}(S_{t}^{\texttt{Courier-Post}}) \approx \sum_{c \in \mathcal{C}} \hat{V}_{tc}^{\texttt{Post}}(S_{tc}^{\texttt{Courier-Post}}, \{S_{tc'}\}_{c' \neq c}). \tag{19}$$

The individual value functions, $\hat{V}_{tc}^{\texttt{Post}}(\cdot)$, are linearly integrated into the overall value function within the `MatchingIP`. That is, $\texttt{score}_t(c \leftrightarrow f)$ terms in the `MatchingIP` objective, which reflect the estimated long-term value of assigning a specific courier (c) to a particular feasible matching (f), are derived from $\hat{V}_{tc}^{\texttt{Post}}(\cdot)$. Specifically, for each (c, f) pair, first the post-decision state, $S_{tc}^{\texttt{Courier-Post}}$, is determined and then the corresponding value from the approximated value function, $\hat{V}_{tc}^{\texttt{Post}}(\cdot)$, is obtained from the trained neural network to calculate the $\texttt{score}_t(c \leftrightarrow f)$ values. This approach reduces the evaluations of the non-linear value function from an exponential to a linear scale with respect to the number of couriers.

To update the individual courier-based VFAs, NeurADP explicitly calculates the gradients associated with each parameter using standard symbolic differentiation libraries. It then adjusts these parameters to minimize the L2 distance between the one-step return estimate of the Bellman equation and the current value function estimate. In order to tackle stability and scalability issues, which are particularly vital in neural network value function learning, NeurADP employs a combination of methodological and practical strategies. These include the use of off-policy updates to stabilize Bellman updates and addressing data scarcity by directly storing sets of feasible actions. NeurADP additionally utilizes a singular neural network for individual courier value functions and employs prioritized experience replay to reuse experience efficiently. Moreover, practical simplifications such as utilizing low-dimensional embeddings for discrete locations and introducing Gaussian noise for exploration during training are strategically implemented. This ensures that learning remains manageable and is precisely tailored to the complexities and subtleties of the underlying problem space.

4.3.3. Overall algorithm

The overall NeurADP algorithm for the ODP is presented in Figure 1 in the form of a flowchart. Initially, the system, along with prediction and target neural networks (used as value functions for courier post-decision states), and couriers with their shifts are initialized. Orders stochastically arrive and, depending on courier availability, feasible actions between couriers and orders are enumerated. During training, these actions are stored as future training experiences. The prediction neural network scores each action based on immediate rewards and the value of the resultant post-decision state, with Gaussian noise added for exploratory purposes during training [13].

The `MatchingIP` defined in (16a)-(16d) is utilized to determine optimal actions, maximizing immediate and anticipated downstream rewards. If training, the replay buffer is checked for sufficient experiences to begin sampling and value function training. When utilizing a prioritized replay buffer, the associated weights with each experience are retrieved. The target neural network scores feasible actions from the experience, and the `MatchingIP` chooses the best actions. The prediction network updates the value of each post-decision state at time t through gradient descent, using the `MatchingIP`-selected best action value at time $t+1$. If a prioritized replay is in use, experience weights are updated. After sampling experiences, the algorithm collates rewards from the current iteration, simulates courier movements, and advances in time. If the subsequent time step marks the day's end, the iteration concludes; otherwise, it progresses to the new time step.

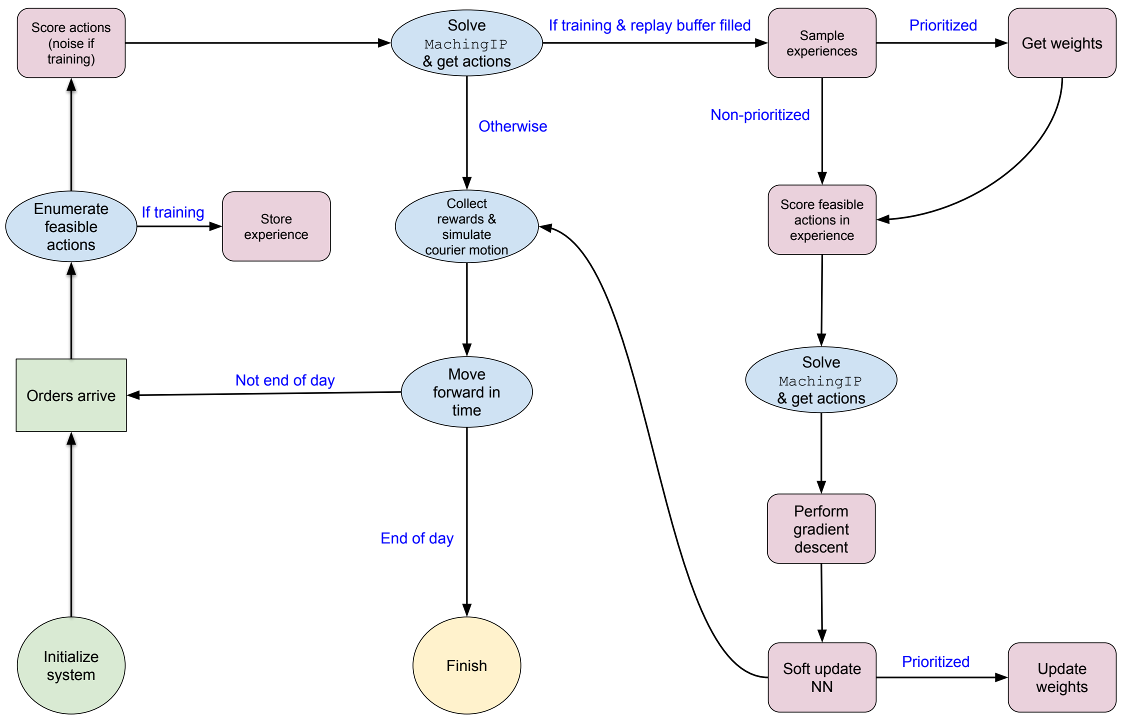


Figure 1: NeurADP algorithm flowchart.

4.3.4. *Neural network architecture*

The architecture of the underlying neural network value function begins with an embedding layer which takes as input the current location of the courier as well as the destination locations of its matched orders. From here, these embedded location representations, complemented by their associated delays, are inputted into an LSTM layer. The output is then combined with additional pertinent auxiliary information and proceeds through several dense layers, ultimately yielding a single value. Furthermore, parameter tuning is undertaken throughout the network's architecture, encompassing modifications to embedding sizes and variations in the number of dense layers.

5. Experimental Setup

In this section, we describe the datasets and the benchmark policies employed in our numerical study. All the experiments are conducted using Python 3.6.13 on Google Cloud servers and we use IBM ILOG CPLEX Optimization Studio version 12.10.0 to solve the IP models.

5.1. Datasets

In our numerical study, we examine four distinct geographical-based datasets containing delivery information of online orders: the Brooklyn, Chicago, Bangalore, and Iowa datasets. The Brooklyn and Chicago datasets [2] encompass delivery data for online DoorDash requests within their respective urban cities, while the Bangalore dataset [14] incorporates order requests from restaurants in Bangalore, India. Finally, the Iowa dataset, which was introduced by Ulmer et al. [25], is comprised of destination locations for meal deliveries within Iowa City. Each dataset includes latitude-longitude points for drop-off locations associated with real-world order requests from which we extract a frequency distribution of the popularity of each delivery location within the city from which we sample. The datasets for Brooklyn, Iowa, Chicago, and Bangalore consist of 988, 500, 117, and 77 unique destination locations, respectively. A warehouse is located at the centre of each distribution of order destinations and the distance between each pair of locations is calculated via the haversine formula. To account for real-world travel time variations influenced by travel direction and accurately depict the unevenness in travel between two points, randomness is incorporated into the travel time values. This involves introducing an additional noise of up to 10% to each value.

We consider a 24-hour problem horizon which is broken into 5-minute decision epochs (i.e., $\delta = 5$ minutes). For all the datasets, the number of requests which arrive between each decision epoch is sampled based upon a distribution of real-world order requests, as presented in Kavuk et al. [9], with a 1-order request standard deviation band. Figure 2 shows a series of box plots that illustrate the distribution of distances between drop-off locations and the warehouse in each dataset. The calculation of the multiplier term β in the reward function (2) depends on the courier capacity parameter and the dataset as described in Section 3.2. For instance, for the Brooklyn dataset, for the capacity of 1, 2, 3 and 4, the β values are 50, 73, 96, and 119, respectively. Similarly, for the courier capacity value of 3, the β values are 467, 43, and 260 for Bangalore, Chicago and Iowa datasets, respectively.

Figure 3 displays the geographic distribution of delivery locations in the Brooklyn dataset (as a representative dataset) along with the distribution of order arrivals throughout the day. The

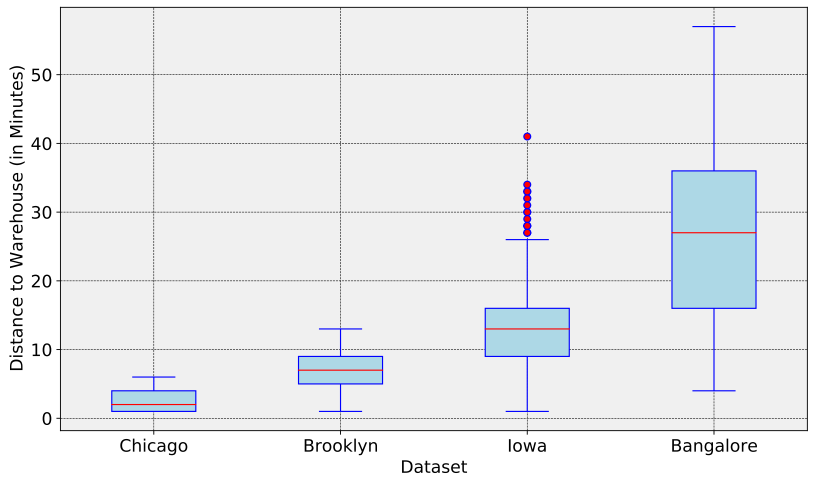
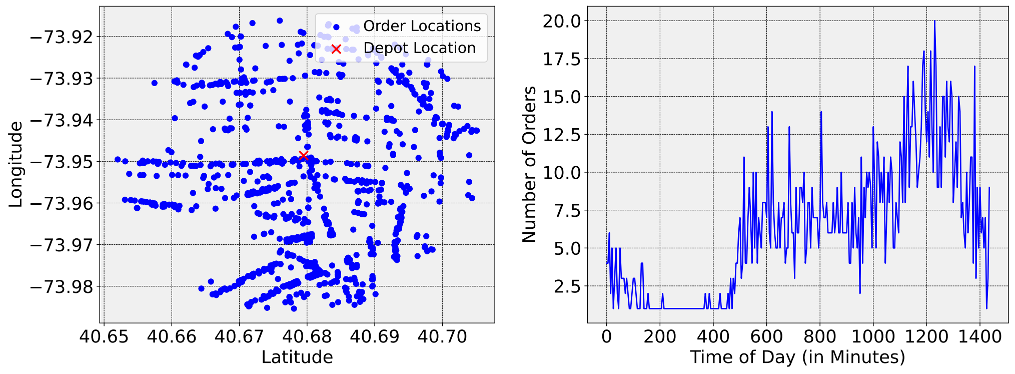


Figure 2: Distribution of drop-off points for each dataset.

schedules of couriers are manually planned in advance to accommodate the anticipated fluctuations in order volume throughout the day. This entails scheduling fewer shifts during expected periods of low demand and scheduling more shifts during peak hours. More specifically, the quantity of couriers on duty closely mirrors the pattern of order arrivals depicted in Figure 3b. Between the hours of 3 AM and 6 AM, when the order volume is at its lowest, the courier staffing reaches its minimum. Conversely, the staffing level reaches its peak during the rush-hour period of 7 PM to 9 PM.



(a) Location points

(b) Order request distribution

Figure 3: Brooklyn dataset specifications.

5.2. Benchmark Policies

We consider a family of benchmark policies in our comparative analysis with `NeurADP`, namely, a set of myopic policies, which we denote by `Myopic`, as well as a group of DRL policies, labeled as `DRL`. Generally, myopic policies encompass greedy strategies which prioritize immediate rewards obtained from actions taken in the present time step, while disregarding any future consequences of these decisions. These policies facilitate a streamlined decision-making process, offering benefits when addressing complex and dynamic problems where it may not be feasible to calculate a globally optimal policy, as seen in the case of our order dispatching problem. However, as a result of their greedy nature, myopic policies may not always produce optimal long-term policies, and are thus often used as baselines. The employed `Myopic` policies follow a similar pattern and are implemented as follows. Available couriers are sorted so as to prioritize varying system dynamics, such as distance to the warehouse and available capacity. Each courier is then examined individually, with the aim of identifying the action between the courier and the set of incoming orders which maximizes the courier's order fulfillment while minimizing delivery time. For instance, in the case where couriers are sorted based upon proximity to the warehouse, we begin by evaluating feasible actions for the nearest courier. Among the available actions, we select the one which maximizes the number of orders which are matched, while minimizing the delivery time for both newly assigned and previously assigned orders. In cases where multiple actions serve an equal number of orders, the preference is given to the action which allows the courier to complete their assigned deliveries more quickly. Once an action is determined for a courier, we finalize the matching and move on to the next available courier in the sorted queue.

We examine a collection of `DRL` policies the ultra-fast ODP, which are derived from those introduced in [9]. At each time step, we employ a trained Double Deep-Q Network (DDQN) to make accept-reject determinations for incoming online orders. The problem and decision dynamics are outlined as follows. The orders are sorted based upon their estimated delivery durations (direct delivery time). We evaluate each order individually and utilize the DDQN network to determine whether to accept or reject the order. If the decision is to accept, we employ a straightforward heuristic to match the order with an available courier, taking into account varying system dynamics such as distance to the warehouse and available capacity. A reward of 1 is accrued if an order is accepted and successfully matched with an available courier, and 0 reward is accrued otherwise. In the event that an order is accepted but cannot be fulfilled due to courier unavailability caused by capacity or time constraints, the order is disregarded, resulting in no reward. Once a decision is made and the order is either matched or ignored, we proceed to the next order in the sorted queue. The DDQN network, responsible for learning the accept-reject actions, is a feed-forward neural network and it is trained with the same dataset used for `NeurADP`. It receives inputs regarding the state of the couriers, including their distances to the warehouse, their currently assigned orders, and the number of couriers on not on shift or at the warehouse. Additionally, information about the specific order under consideration is incorporated, encompassing details such as the destination location and delivery deadline. The network comprises four hidden layers with 32, 64, 64, and 32 neurons, respectively, and an output layer with two neurons representing the accept and reject decisions.

6. Results

In this section, we present results from our detailed numerical study for the ultra-fast ODP. We primarily consider the average number of orders fulfilled within the 24-hour decision horizon while comparing the performance of the `NeurADP` policy against the two classes of benchmark policies. More specifically, we assess each policy over 20 days of testing data and subsequently compute the average total number of orders that each policy has encountered and fulfilled across these test days. Orders are generated via sampling at every decision epoch, and, according to a given policy, a feasible decision is made to batch them together and match them with couriers. Upon matching, the couriers' assigned orders are updated, and a simulation of the couriers' movement toward their next destination takes place. Below, we first introduce two novel artificial bounds on the achievable performance level that enhance our understanding of the quality of our solutions. Then, we delve into the analysis of our overall findings.

6.1. Artificial Bounds on Achievable Performance Level

Considering that problems such as the ODP often involve large, complex and highly dynamic systems, deriving optimal solutions for such problems is often infeasible and methodologies for deriving theoretical bounds are usually unsatisfactory [3]. As such, practitioners typically evaluate policy performance based on the total potential reward available throughout the problem horizon. In the case of the ODP, this corresponds to the total number of online orders received in the system. However, depending upon the problem context and parameters, this overarching upper limit may not be realistic or reasonable to achieve, as it may be infeasible to come close to the maximum potential reward. For instance, in the ODP, though the system may receive a large number of orders throughout the day, provided the capacity constraints of couriers, order deadlines, number of couriers, and courier shift times, it may not be feasible to serve even half of those orders. As such, we introduce two novel benchmark ceilings.

For our first benchmark ceiling, we consider a scenario where orders are served immediately upon being matched with couriers, which we define as `Direct`. Specifically, we maintain the same constraints based on courier capacities and order deadlines. However, once a batch of orders is matched with the available couriers, we assume that all orders are delivered by the start of the subsequent decision epoch. This implies that all available couriers, as long as they are on their shift, will be ready in the following time step with an empty queue. This allows us to establish a lower bound, relative to the total number of orders observed, on how well our `NeurADP` and benchmark policies could have performed, even in this unrealistic setting. The second benchmark ceiling involves applying the NeurADP framework to each individual day's worth of orders in our testing dataset. Instead of training our `NeurADP` policy on a separate training dataset, deriving a policy that maximizes the expected reward, and subsequently testing it on a separate testing dataset, we train the `NeurADP` policy on the same testing dataset that we ultimately evaluate it on. This results in an alternative `NeurADP` policy, which we refer to as `NeurADP-Fixed`. This policy aims to derive the best policy based on a deterministic set of orders for a specific day. This approach provides us with additional insights into the performance of alternative policies for ultra-fast ODP.

6.2. Order Dispatching Performance

We evaluate the results from our experiments with respect to four primary inputs: the number of couriers, delay time, courier capacity, and geographical location. The number of couriers is obtained by accounting for all the couriers working within a 24-hour period, while the delay time represents the maximum duration of time a courier has from an order's entry into the system to their drop-off. This duration is used to determine the order deadline, calculated using Equation (1). The courier capacity specifies the maximum number of orders a courier can carry simultaneously, while the geographic location pertains to the spatial dataset and the distribution of requests based on geography. In our *baseline configuration*, we utilize the Brooklyn dataset and set the parameters to include 15 couriers, a maximum allowable delay time of 10 minutes, and a maximum courier capacity of 3 orders. Due to the extensive computational time required to derive the `NeurADP-Fixed` ceiling values for each experiment, we consider it exclusively for the experiments related to our baseline configuration and the varying number of couriers and utilize the `Direct` ceiling for the remaining experiments. We begin by examining the baseline configuration to identify the most suitable benchmark policies from the `Myopic` and `DRL` policy classes, which are later used in the comparative analysis with the `NeurADP` policy.

6.2.1. Baseline Configuration

Our primary benchmark policies exhibit several variations in the matching process between orders and couriers. In the `DRL` policy, we have the flexibility to match accepted orders with couriers based on either their distance or current queue capacity. In terms of distance, the order can be assigned to the closest available courier or the farthest one. Regarding capacity, we can allocate the order to the courier with the least occupied queue or the most occupied queue. Similarly, for the `Myopic` policy, the matching of a batch of orders can be determined by considering both the proximity of distance and the capacity of the courier queue. We consider four variations of each benchmark policy. "DC" represents the utilization of distance in the matching process, where the closest courier is chosen. Conversely, "DF" signifies that the farthest courier is selected based on distance. Moreover, "CE" denotes the utilization of capacity for matching, with the emptiest courier being selected, while "CF" indicates the selection of the fullest courier based on capacity.

Table 2 provides the outcomes of the considered policy variants for the baseline configuration of our experimental setup. The table includes the `NeurADP-Fixed` value, which indicates the average number of orders served using the fixed ceiling, as well as the percentage of orders fulfilled by each policy, denoted as "% Filled". More specifically, we calculate the average number of orders served by each policy and divide it by the fixed ceiling value. This result is then multiplied by 100, and a standard deviation is provided for each policy. Furthermore, the percentage increase of the `NeurADP` policy compared to the other benchmark policies is included in the final right-most column labeled "% Incr. NeurADP". This metric is calculated by subtracting the average number of orders fulfilled by the benchmark policies from the average number of orders fulfilled by the `NeurADP` policy, then dividing the result by the "Fixed Ceiling" value, multiplied by 100.

Table 2: Performance of different policies for ultra-fast ODP for *baseline configuration* (avg. number of orders fulfilled over 20-day test window is reported for the Fixed Ceiling; performance of other policies are w.r.t. Fixed Ceiling, provided as mean±stdev)

Policy	***Fixed Ceiling***	***% Filled***	***% Incr. NeurADP***
NeurADP	955.00	97.96 ± 1.97	-
DRL-DC	-	86.04 ± 1.53	+11.92
DRL-DF	-	83.65 ± 1.74	+14.31
DRL-CE	-	81.04 ± 1.75	+16.92
DRL-CF	-	85.81 ± 1.64	+12.15
Myopic-DC	-	91.27 ± 1.64	+6.69
Myopic-DF	-	87.79 ± 2.15	+10.17
Myopic-CE	-	88.58 ± 1.32	+9.38
Myopic-CF	-	90.77 ± 2.18	+7.19

In general, we observe that the `NeurADP` policy consistently outperforms all variations of benchmark policies for both the `DRL` and `Myopic` cases. This can be attributed to its enhanced ability to efficiently match batches of orders with available couriers, which we explore in more detail in further experiments below. Additionally, we find that the `Myopic` policies exhibit consistently superior performance compared to the `DRL`-based policies. One possible explanation for this trend is that the `Myopic` policy settings prioritize maximizing the number of incoming orders batched together when matching them with a given courier. On the other hand, the `DRL` policy tends to make simpler accept/reject decisions for each individual order and matches them individually as well (i.e., `DRL`-based approach does not try to identify the best batch of orders). Furthermore, we note that among the various benchmark policy variations, '`DRL-DC`' and '`Myopic-DC`' consistently yield the best performance. As a result, we utilize '`DRL-DC`' as our `DRL` policy and '`Myopic-DC`' as our `Myopic` policy for the remainder of the experiments.

6.2.2. Impact of Number of Couriers

We examine the impact of the number of available couriers on the number of fulfilled orders throughout the 24-hour problem horizon. The results of these experiments are presented in Table 3 for 10, 15, and 20 couriers. We once again utilize the `NeurADP-Fixed` ceiling, whose values of average orders served are shown under "Fixed Ceiling", and present the percentage of orders each policy fulfills based upon this ceiling in columns labeled "% NeurADP Filled", "% Myopic Filled", and "% DRL Filled". As before, we calculate the average number of orders served by each policy and divide it by the ceiling value. This result is then multiplied by 100, and a standard deviation is provided for each policy. The final two columns illustrate the percentage increase in the average number of orders fulfilled by the `NeurADP` policy compared to that of the `Myopic` and `DRL` policies, respectively. This metric is calculated by subtracting the average number of orders fulfilled by the benchmark policies from the average number of orders fulfilled by the `NeurADP` policy, then dividing the result by the "Fixed Ceiling" value, multiplied by 100. These column definitions remain consistent throughout the subsequent tables presented.

Table 3: Impact of number of couriers on order fulfillment for the Brooklyn dataset (avg. number of orders fulfilled over 20-day test window is reported for the Fixed Ceiling; performance of other settings are w.r.t. Fixed Ceiling, provided as mean$\pm$stdev).

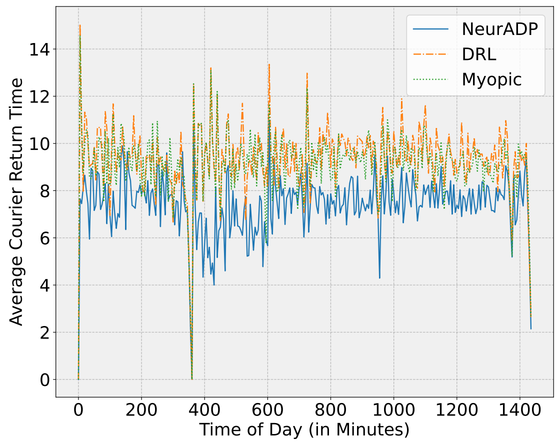
Number of Couriers	*Fixed Ceiling*	*% NeurADP Filled*	*% Myopic Filled*	*% DRL Filled*	*% Incr. Over Myopic*	*% Incr. Over DRL*
10 couriers	691.85	97.89 ± 1.43	86.37 ± 1.06	82.14 ± 1.52	+11.52	+15.75
15 couriers	955.00	97.96 ± 1.97	91.27 ± 1.64	86.04 ± 1.53	+6.69	+11.92
20 couriers	1134.70	98.30 ± 1.98	95.21 ± 2.23	90.22 ± 1.79	+3.09	+8.08

We observe that, once again, `NeurADP` is able to consistently outperform the benchmark policies. This superiority in performance is due to several factors. First, the `NeurADP` policy enables, on average, a greater number of agents to be available at the warehouse, compared to the benchmark policies. More specifically, in the scenario with 10 couriers, there are on average 0.37

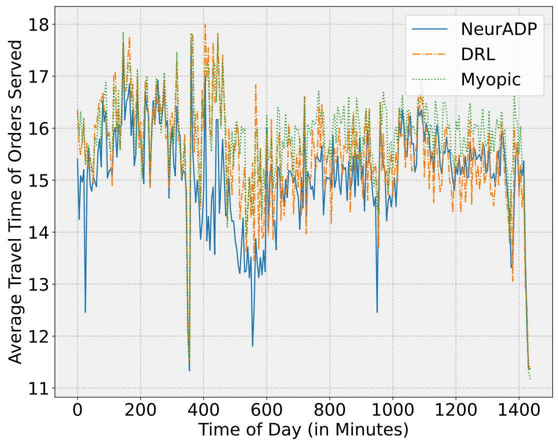
couriers available at the warehouse at each decision epoch throughout the day, while there are only 0.06 for each of the `DRL` and `Myopic` policies. This is due to `NeurADP`'s ability to match batches of orders to couriers which minimize their travel time away from the warehouse, yet maximize their orders fulfilled. More specifically, unlike the other policies which look to fill the queue of each courier at each time step, the `NeurADP` policy looks to rather match couriers with orders which best allow them to return to the warehouse as soon as possible. Doing so allows them to be able to be available to more incoming orders in subsequent time steps. This can be better seen in Figure 4, in which we see that the average return time of a courier making a delivery is consistently lower than both benchmark policies for the `NeurADP`. Additionally, we see that `NeurADP` accepts orders which on average require less travel time to be delivered. More specifically, the average direct travel time from the depot to the delivery location for an accepted order is 14.41 minutes for `NeurADP`, while it is 15.24 and 15.61 minutes for the `DRL` and `Myopic` policies, respectively. Furthermore, the average queue size of couriers making deliveries is 1.42 orders for `NeurADP`, while it is respectively 1.84 and 2.02 for the `DRL` and `Myopic` cases. Thus, by ignoring orders at each time step, which may be out of the way for couriers, and by prioritizing return times to the depot, `NeurADP` is able to more efficiently serve orders throughout the day. This superiority in performance deteriorates, however, as more couriers are incorporated into the environment. This can be attributed to the diminishing significance of the quality of the employed policy as more couriers become available. In other words, when there are a large number of couriers available to handle order deliveries at each time step, the specific policy being used becomes less critical since most policies are capable of performing relatively well in such scenarios.

6.2.3. Impact of Delay Time

We next assess the impact of the permitted delay time on the number of orders served. The summary results for this experiment are provided in Table 4. We employ the `Direct` ceiling to establish the upper bound on the number of orders fulfilled and illustrate the percentage of fulfilled orders by each policy based on this ceiling. Similar to previous experiments, we observe that the `NeurADP` policy outperforms the two benchmark policies for all variations of delay time. Moreover, we once again notice a decline in relative performance improvements attributed to the



(a) Average return time (in minutes) of couriers making deliveries.



(b) Average travel time (in minutes) of orders accepted.

Figure 4: Brooklyn dataset base-case auxiliary statistics.

NeurADP policy compared to the two benchmark policies as the problem setting becomes less restrictive (i.e., when the maximum allowed delay time is increased).

Table 4: Impact of delay time on order fulfillment for the Brooklyn dataset (avg. number of orders fulfilled over 20-day test window is reported for the Direct Ceiling; performance of other settings are w.r.t. Direct Ceiling, provided as mean±stdev).

Delay Time	***Direct Ceiling***	***% NeurADP Filled***	***% Myopic Filled***	***% DRL Filled***	***% Incr. Over Myopic***	***% Incr. Over DRL***
5 minutes	1560.75	55.80 ± 0.95	49.95 ± 0.84	46.33 ± 0.76	+5.85	+9.47
10 minutes	1578.65	59.26 ± 0.62	55.21 ± 0.92	52.04 ± 0.50	+4.04	+7.22
15 minutes	1584.75	59.65 ± 0.50	56.84 ± 0.47	53.51 ± 0.47	+2.81	+6.14

Figure 5 illustrates the fulfillment of orders by each policy over the problem horizon in the 10-minute delay scenario. Initially, when there are relatively few incoming orders, all policies exhibit similar performance in fulfilling the incoming orders. However, as the day progresses and a higher volume of orders arrives at each decision epoch, the NeurADP policy surpasses the benchmark policies, especially during peak hours. This observation highlights that when the number of orders is low and there are sufficient couriers available, all policies perform relatively well. However, as the number of orders increases and couriers become busier, making informed and intelligent decisions becomes crucial. In such situations involving ultra-fast delivery during peak periods, the

`NeurADP` policy demonstrates superior performance, effectively outperforming the other policies.

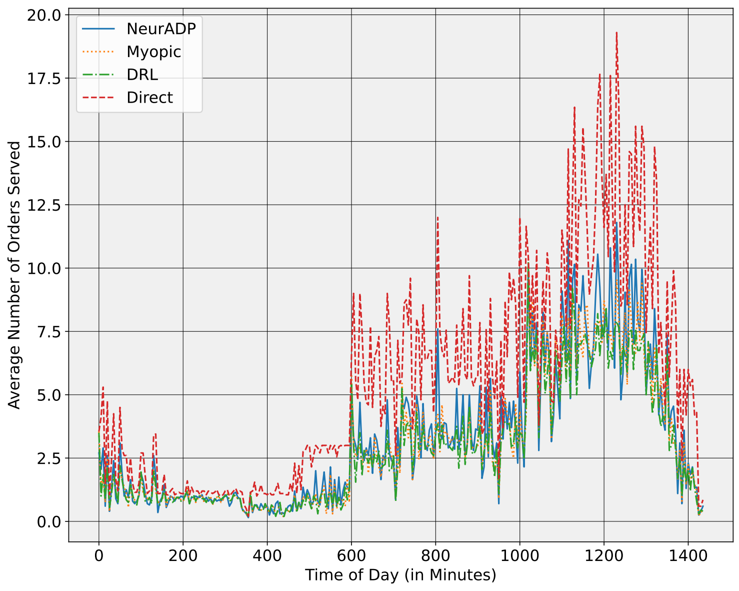


Figure 5: Orders seen and fulfilled throughout the day for the Brooklyn dataset.

6.2.4. Impact of Courier Capacity

Table 5 presents the results on the impact of the maximum capacity size of a courier and the number of orders fulfilled. These results reaffirm the superiority of the `NeurADP` policy compared to the benchmark policies across different capacity variations. Interestingly, the findings suggest that the addition of extra capacity yields diminishing returns. Specifically, increasing the capacity from 1 to 2 orders results in an average increase of 172.51 orders served among the three policies. However, the subsequent increases in capacity from 2 to 3 orders and from 3 to 4 orders correspond to smaller increases of 107.67 and 64.68 orders served, respectively. This trend is also observed in the average filled queue size of couriers making deliveries. The increase in filled queue size from a maximum capacity of 1 to 2 orders is 0.44 orders on average, while the increases from 2 to 3 orders and 3 to 4 orders are only 0.27 and 0.11 orders, respectively. It is important to note that capacity depends on various factors, such as delay time, and increasing capacity may not necessarily translate into serving more orders due to time constraints or couriers available. Thus, our findings suggest that while the `NeurADP` policy exhibits improvement over the `DRL` policy up to a capacity of 3 orders, the improvement diminishes for a capacity of 4 orders. This indicates that as the capacity increases, the problem setting becomes less restrictive, allowing for a wider range of effective policies, which benefits the `NeurADP` policy and enables it to benefit from smarter decision-making. However, as the capacity continues to increase, the improvement of `NeurADP` over the `DRL` policy declines, suggesting that well-performing policies can be more easily obtained and that the quality of the policy becomes less crucial when the problem setting becomes too non-restrictive. The improvement of `NeurADP` over the `Myopic` policy follows a similar general trend as well.

Table 5: Impact of courier capacity on order fulfillment for the Brooklyn dataset (avg. number of orders fulfilled over 20-day test window is reported for the Direct Ceiling; performance of other settings are w.r.t. Direct Ceiling, provided as mean±stdev).

Courier Capacity	*Direct Ceiling*	*% NeurADP Filled*	*% Myopic Filled*	*% DRL Filled*	*% Incr. Over Myopic*	*% Incr. Over DRL*
1 order	993.10	62.82 ± 0.96	58.75 ± 1.01	58.48 ± 1.05	+4.07	+4.34
2 orders	1481.95	55.03 ± 0.69	51.47 ± 0.65	49.08 ± 0.43	+3.56	+5.95
3 orders	1578.65	59.26 ± 0.62	55.21 ± 0.92	52.04 ± 0.50	+4.04	+7.22
4 orders	1614.00	61.10 ± 0.55	58.54 ± 0.60	55.25 ± 0.45	+2.56	+5.85

6.2.5. Impact of Geographic Location

The experimental results related to different geographic locations are presented in Table 6, which include Chicago, Brooklyn, Iowa, and Bangalore datasets. Each dataset has its unique distribution of order and delivery locations, as depicted in Figure 2 and Figure 3. We observe that in datasets where the delivery area is more concentrated, like the Chicago dataset, all policies perform well in fulfilling order requests, and the performance advantage of the `NeurADP` policy over the benchmark policies is relatively small. However, as the delivery area expands and the delivery locations become more scattered, the improvement of the `NeurADP` policy over the benchmark policies becomes more significant. This can be explained by the following reasoning: in dense areas where delivery locations are close to the depot, making smarter decisions in matching couriers to orders or rejecting certain orders to wait for the next time step becomes less critical. This is because even with sub-optimal matching, the courier will still be able to fulfill all assigned orders and return to the warehouse on time for the next time step. However, in datasets with sparse delivery locations, such as the Bangalore dataset, where travel time from the depot to each location is longer, making sub-optimal matching decisions between couriers and incoming orders becomes more costly. A courier being occupied with sub-optimal assignments for a longer duration means they are unavailable to serve new orders, leading to delays. This becomes more apparent when comparing the average return times of couriers for each policy across the different datasets. For the Chicago dataset, the average return time for couriers making deliveries under the `NeurADP` policy is 3.63 minutes, while it is 4.63 and 4.22 minutes for the `DRL` and `Myopic` policies, respectively, showing a relatively small difference. In contrast, for the Bangalore dataset, the average return time for a courier in the `NeurADP` policy is 18.27 minutes, while it is 30.66 and 30.03 minutes for the `DRL` and `Myopic` policies, respectively, demonstrating a much larger disparity. As such, making poor matching decisions in sparse scenarios incurs higher costs, and having a smarter policy becomes significantly more important.

Table 6: Impact of geographic location on order fulfillment (avg. number of orders fulfilled over 20-day test window is reported for the Direct Ceiling for each location; performance of the policies are w.r.t. Direct Ceiling, provided as mean±stdev).

Geographic Location	*Direct Ceiling*	*% NeurADP Filled*	*% Myopic Filled*	*% DRL Filled*	*% Incr. Over Myopic*	*% Incr. Over DRL*
Chicago	1601.60	96.93 ± 0.37	96.61 ± 0.32	94.51 ± 0.35	+0.33	+2.43
Brooklyn	1578.65	59.26 ± 0.62	55.21 ± 0.92	52.04 ± 0.50	+4.04	+7.22
Iowa	1549.90	38.19 ± 0.46	30.63 ± 0.50	30.02 ± 0.61	+7.56	+8.17
Bangalore	1468.80	24.14 ± 0.24	16.88 ± 0.33	15.54 ± 0.52	+7.25	+8.60

6.3. Computational Performance of NeurADP

Lastly, we provide a comprehensive overview of the auxiliary statistics related to the NeurADP algorithm drawn from our experiments. Notably, the algorithm takes approximately 8 hours to execute in our baseline experiment. Within a NeurADP iteration, the computational time is predominantly consumed by three tasks: data preparation for the neural network and evaluation of feasible actions (42.48%), the generation of feasible actions while satisfying the constraints (28.83%), and the `MatchingIP`-driven action selection process (26.39%). Other tasks collectively account for less than 1% of the total computational time. The considerable time spent on generating feasible actions highlights the value of experience sampling, suggesting that storing such actions for future reference could lead to significant computational savings. Our results also show that including auxiliary information in our post-decision state gives a modest performance enhancement of 0.61%. Considering the less pronounced geographic competition among couriers in the ODP compared to the ride-pool matching problem, it can be expected that such additional information about other system agents might not yield significant benefits. Nevertheless, given the slight performance boost and minimal computational overhead, we have opted to incorporate this auxiliary information in our experiments.

7. Conclusion

This paper addresses the challenges and complexities of the same-day delivery problem by focusing on the order dispatching and matching aspects. Our work builds upon existing research and contributes to the literature by introducing innovative features and capabilities. It proposes the incorporation of batching and courier queues to enhance dispatching operations, providing a more realistic representation of the order dispatching process. Additionally, the scope of the problem is expanded to consider larger problem sizes, capturing the complexities of managing larger-scale dispatching operations. Furthermore, our paper introduces the application of the NeurADP approach to solving ultra-fast ODP, extending the potential applications of NeurADP beyond its original context. The effectiveness of NeurADP is demonstrated through implementation and comparison with myopic and DRL baselines, highlighting its advantages. Original datasets tailored for order dispatching operations are introduced to support the research and facilitate comprehensive evaluations. The paper also presents artificial bounds for evaluating solution quality and conducts a sensitivity analysis to investigate the performance of NeurADP under various factors. Overall, this work contributes to advancing the understanding and applicability of solution methodologies in the field of order dispatching and same-day delivery, providing valuable insights for practitioners and future research endeavors.

Future work may aim to enhance the representation of uncertainty in this problem setting by introducing loading and pickup delays, thereby capturing real-world dynamics more accurately. Additionally, exploring the incorporation of time-dependent uncertainty in order arrivals would provide insights into how service efficiency and quality impact the frequency of incoming order requests. Furthermore, multi-modal delivery within this scenario, encompassing various transportation modes like drones, autonomous vehicles, and traditional couriers can be investigated in future works as well. This approach would address the emerging trend of utilizing diverse delivery methods to optimize efficiency and address a range of delivery scenarios. Finally, fairness aspects of delivery services in this expedited delivery setting, aiming to ensure equitable access to timely deliveries across all geographical areas constitute an interesting research avenue.

Disclosure statement

No potential conflict of interest was reported by the authors.

Data Availability Statement

Full research data can be accessed through following link:
https://tinyurl.com/yc32pwrp

References

[1] Al-Kanj, L., Nascimento, J., Powell, W.B., 2020. Approximate dynamic programming for planning a ride-hailing system using autonomous fleets of electric vehicles. European Journal of Operational Research 284, 1088–1106.

[2] Barkingdata, 2022. Doordash restaurant data. https://www.kaggle.com/datasets/polartech/doordash-restaurant-data.

[3] Bertsekas, D.P., 2005. Dynamic programming and suboptimal control: A survey from ADP to MPC. European Journal of Control 11, 310–334.

[4] Cardona Peláez, S., et al., 2022. Same-day delivery routing problem with intraroute resource replenishment in a heterogeneous fleet.

[5] Chen, X., Ulmer, M.W., Thomas, B.W., 2022. Deep Q-learning for same-day delivery with vehicles and drones. European Journal of Operational Research 298, 939–952.

[6] Côté, J.F., de Queiroz, T.A., Gallesi, F., Iori, M., 2021. Dynamic optimization algorithms for same-day delivery problems. Bureau de Montreal, Université de Montreal.

[7] Dayarian, I., Savelsbergh, M., Clarke, J.P., 2020. Same-day delivery with drone resupply. Transportation Science 54, 229–249.

[8] Joe, W., Lau, H.C., 2020. Deep reinforcement learning approach to solve dynamic vehicle routing problem with stochastic customers, in: Proceedings of the International Conference on Automated Planning and Scheduling, pp. 394–402.

[9] Kavuk, E.M., Tosun, A., Cevik, M., Bozanta, A., Sonuć, S.B., Tutuncu, M., Kosucu, B., Basar, A., 2022. Order dispatching for an ultra-fast delivery service via deep reinforcement learning. Applied Intelligence 52, 1–26.

[10] Klapp, M.A., Erera, A.L., Toriello, A., 2018. The dynamic dispatch waves problem for same-day delivery. European Journal of Operational Research 271, 519–534.

[11] Mousavi, K., Bodur, M., Cevik, M., Roorda, M.J., 2021. Approximate dynamic programming for crowd-shipping with in-store customers.

[12] Ngu, E., Parada, L., Macias, J.J.E., Angeloudis, P., 2022. A decentralised multi-agent reinforcement learning approach for the same-day delivery problem. arXiv preprint arXiv:2203.11658 .

[13] Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R.Y., Chen, X., Asfour, T., Abbeel, P., Andrychowicz, M., 2017. Parameter space noise for exploration. arXiv preprint arXiv:1706.01905 .

[14] Poddar, H., 2019. Zomato Bangalore restaurants. https://www.kaggle.com/datasets/himanshupoddar/zomato-bangalore-restaurants.

[15] Powell, W.B., 2007. Approximate Dynamic Programming: Solving the curses of dimensionality. volume 703. John Wiley & Sons.

[16] Powell, W.B., 2010. Approximate dynamic programming-ii: algorithms. Wiley Encyclopedia of Operations Research and Management Science .

[17] Restrepo, M.I., Semet, F., Pocreau, T., 2019. Integrated shift scheduling and load assignment optimization for attended home delivery. Transportation Science 53, 1150–1174.

[18] Shah, S., Lowalekar, M., Varakantham, P., 2020. Neural approximate dynamic programming for on-demand ride-pooling, in: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 507–515.

[19] Simao, H.P., Day, J., George, A.P., Gifford, T., Nienow, J., Powell, W.B., 2009. An approximate dynamic programming algorithm for large-scale fleet management: A case application. Transportation Science 43, 178–197.

[20] Statista, 2021. Same-day delivery market size in the U.S. from 2019 to 2024. https://tinyurl.com/4zdcxx6a.

[21] Thomas, L., 2019. Target expands same-day shipping option in the latest move in the delivery wars with Walmart and Amazon. CNBC. https://tinyurl.com/bdf2h9kv.

[22] Ulmer, M.W., 2020. Dynamic pricing and routing for same-day delivery. Transportation Science 54, 1016–1033.

[23] Ulmer, M.W., Streng, S., 2019. Same-day delivery with pickup stations and autonomous vehicles. Computers & Operations Research 108, 1–19.

[24] Ulmer, M.W., Thomas, B.W., 2018. Same-day delivery with heterogeneous fleets of drones and vehicles. Networks 72, 475–505.

[25] Ulmer, M.W., Thomas, B.W., Campbell, A.M., Woyak, N., 2021. The restaurant meal delivery problem: Dynamic pickup and delivery with deadlines and random ready times. Transportation Science 55, 75–100.

[26] Ulmer, M.W., Thomas, B.W., Mattfeld, D.C., 2019. Preemptive depot returns for dynamic same-day delivery. EURO Journal on Transportation and Logistics 8, 327–361.

[27] Van Hasselt, H., Guez, A., Silver, D., 2016. Deep reinforcement learning with double Q-learning, in: Proceedings of the AAAI conference on artificial intelligence.

[28] Voccia, S.A., Campbell, A.M., Thomas, B.W., 2019. The same-day delivery problem for online purchases. Transportation Science 53, 167–184.

[29] Yu, X., Shen, S., 2019. An integrated decomposition and approximate dynamic programming approach for on-demand ride pooling. IEEE Transactions on Intelligent Transportation Systems 21, 3811–3820.