Dynamic Ridehailing with Electric Vehicles

Nicholas D Kullman, Martin Cousineau, Justin Goodson, Jorge E. Mendoza

Dynamic Ridehailing with Electric Vehicles

Nicholas D. Kullman, Martin Cousineau, Justin C. Goodson, and Jorge E. Mendoza

Abstract

We consider the problem of an operator controlling a fleet of electric vehicles for use in a ridehailing service. The operator, seeking to maximize revenue, must assign vehicles to requests as they arise and recharge and reposition vehicles in anticipation of future requests. To solve this problem, we employ deep reinforcement learning, developing policies whose decision making uses Q-value approximations learned by deep neural networks. We compare these policies against a common taxi dispatching heuristic and against dual bounds on the value of an optimal policy, including the value of an optimal policy with perfect information which we establish using a Benders-based decomposition. We assess performance on instances derived from real data for the island of Manhattan in New York City. We find that, across instances of varying size, our best policy trained with deep reinforcement learning outperforms the taxi dispatching heuristic. We also provide evidence that this policy may be effectively scaled and deployed on larger instances without retraining.

1 Introduction

Governmental regulations as well as a growing population of environmentally conscious consumers have led to increased pressure for firms to act sustainably. This pressure is particularly high in the logistics domain, which accounts for about one third of emissions in the United States (Office of Transportation and Air Quality 2019). Ridehailing services offer a means to more sustainable transportation, promising to reduce the need for vehicle ownership, offering higher vehicle utilization (Lyft 2018), allowing transit authorities to streamline services (Bahrami et al. 2017), and stimulating the adoption of new vehicle technologies (Jones and Leibowicz 2019) In recent years, ridehailing services have seen rapid and widespread adoption, with the number of daily ridehailing trips more than quadrupling in New York City from November 2015 to November 2019 (New York City Taxi & Limousine Commission 2018).

Simultaneously, electric vehicles (EVs) are beginning to replace internal-combustion engine (ICE) vehicles, commanding increasingly more market share (Edison Electric Institute 2019). Coupling the pressures to act sustainably with EVs' promise of lower operating costs, ridehail companies are likely to be among the largest and earliest adopters of EV technology. Indeed most major ridehail companies have made public commitments to significant EV adoption (Slowik et al. 2019). However, EVs pose technological challenges to which their ICE counterparts are immune, such as long recharging times and limited recharging infrastructure (Pelletier et al. 2016).

In this work, we consider these challenges as posed to an operator of a ridehail company whose fleet consists of EVs. We further assume that the EVs in the fleet are centrally controlled and coordinated. While fleet control is somewhat centralized today, it is likely to become increasingly centralized as ridehail companies adopt autonomous vehicles (AVs). As with EVs, ridehail companies are likely to be among the earliest and largest adopters of AVs, as they stand to offer many benefits including reduced operating costs and greater efficiency and predictability (Fagnant and Kockelman 2015). For brevity, we refer to

this problem as the Electric Ridehail Problem with Centralized control, or E-RPC.

Our contributions in this work are as follows: 1) We offer the first application of deep reinforcement learning (RL) to the E-RPC, developing policies that respond in real time to serve customer requests and anticipate uncertain future demand under the additional constraints of fleet electrification. The policies are *model-free*, meaning they learn to anticipate this demand without any prior knowledge of its shape. We compare these deep RL-based policies to a common heuristic in the taxi dispatching literature. 2) We evaluate these policies on instances constructed with real data reflecting ridehailing operations from New York City in 2018. 3) We establish a dual bound for the dynamic policies using a perfect information relaxation that we solve using a Benders-like decomposition. We compare this complex dual bound against a simpler dual bound and provide an analysis of when the additional complexity of the perfect information bound is valuable. 4) We show that our best deep RL-based solution significantly outperforms the benchmark dispatching heuristic and comes within 19% of an optimal policy with perfect information. 5) We further demonstrate that the best-performing deep RL-based policy can be scaled to larger problem instances without additional training. This is encouraging for operators of ridehail companies, as it suggests robustness to changes in the scale of operations: in the event of atypical demand or a change in the number of vehicles (e.g., due to fleet maintenance), the policy should still provide reliable service.

We begin by reviewing related literature in §2, then provide a formal problem and model definition in §3. We describe our solution methods in §4, the bounds established to gauge the effectiveness of these methods in §5, and demonstrate their application in computational experiments in §6. We offer brief concluding remarks in §7.

2 Related Literature

Ridehail problems (RPs), those addressing the operation of a ridehailing company, fall under the broader category of dynamic vehicle routing problems (VRPs). Within dynamic VRPs, they may be classified as a special case of the dynamic VRP with pickups and deliveries or, more precisely, a special case of the dynamic dial-a-ride problem. Recent technological advances in mobile communications have driven new opportunities in ridehailing and other *mobility-on-demand* (MoD) services, reinvigorating research in this area. As a result, there now exists a substantial body of literature specifically pertaining to MoD applications within the dial-a-ride domain. This literature is the focus of our review. For a broader survey of the dynamic dial-a-ride literature, we refer the reader to Ho et al. (2018), and, similarly, for the dynamic VRP literature, to Psaraftis et al. (2016).

Often studies of RPs focus their investigation on the assignment problem, wherein the operator must choose how to assign fleet vehicles to new requests. For example, Lee et al. (2004) propose the use of real-time traffic data to assign the vehicle which can reach the request fastest. A study by Bischoff and Maciejewski (2016) uses a variant of this assignment heuristic that accounts for whether demand exceeds supply (or vice versa), as well as vehicles' locations within the service region. Seow et al. (2009) propose a decentralized heuristic that gathers multiple requests and allows vehicles to negotiate with one another to determine which vehicles serve the new requests. They show that it outperforms heuristics like those in Lee et al. (2004) and Bischoff and Maciejewski (2016). Hyland and Mahmassani (2018) introduce a suite of optimization-based assignment strategies and show that they outperform heuristics that do not employ optimization. Following suit, Bertsimas et al. (2019) describe ways to reduce the complexity of optimization-based approaches for problems with large fleets and many customers. They demonstrate

their proposed methods on realistically-sized problem instances that reflect ridehailing operations in New York City.

The aforementioned studies largely ignore the task of repositioning idle vehicles in anticipation of future demand. This is in contrast to studies such as Miao et al. (2016), Zhang and Pavone (2016), and Braverman et al. (2019) who address their RP from the opposing perspective of the fleet repositioning problem. Miao et al. (2016) do so using learned demand data with a receding horizon control approach. Braverman et al. (2019) use fluid-based optimization methods for the repositioning of idle vehicles to maximize the expected value of requests served. They establish the optimal static policy and prove that it serves as a dual bound for all (static or dynamic) policies under certain conditions. Other studies such as Fagnant and Kockelman (2014) and Alonso-Mora et al. (2017) consider strategies for both the assignment and repositioning tasks. The former does so using rule-based heuristic strategies, evaluating them using an agent-based model; the latter using optimization-based methods.

A recent competing method to address ridehail problems is reinforcement learning (RL), predominantly deep reinforcement learning. As it is well-suited to address problems with larger fleet sizes, many studies employing RL take a multi-agent RL (MARL) approach (Oda and Tachibana (2018), Oda and Joe-Wong (2018), Holler et al. (2019), Li et al. (2019), Singh et al. (2019)). In work sharing many similarities to ours, Holler et al. (2019) use deep MARL with an attention mechanism to address both the assignment and repositioning tasks. They compare performance under two different MARL approaches: one in which a system-level agent coordinates vehicle actions to maximize total reward; and one in which vehicle-level agents act individually, each maximizing its own reward. Oda and Joe-Wong (2018) use a deep MARL approach to tackle fleet repositioning, but rely on a myopic heuristic to perform assignment. Oda and Tachibana (2018) do so as well, but employ special network architectures with soft-Q learning which they argue better accommodates the inherent complexities in road networks and traffic conditions. Li et al. (2019) employ a multi-agent RL framework, comparing two approaches that vary in what individual vehicles know regarding the remainder of the fleet; however, in contrast to most MARL applications to RPs, they address only the assignment problem, assuming that the vehicles are de-centralized and therefore not repositioned by the operator. Similarly assuming de-centralized vehicles and addressing only the assignment problem, Xu et al. (2018) combine deep RL with the optimization-based methods used in, e.g., Alonso-Mora et al. (2017), Zhang et al. (2017), and Hyland and Mahmassani (2018). In their work, Xu et al. use deep RL to learn the objective coefficients in a math program that is used in an optimization framework to assign vehicles to requests.

Missing from all aforementioned studies is fleet electrification — none take into account the technical challenges associated with electric vehicles, such as long recharging times or limited recharging infrastructure. The number of studies that consider these constraints is limited but growing. Most (e.g., Jung et al. (2012), Chen et al. (2016), Kang et al. (2017), Iacobucci et al. (2019), La Rocca and Cordeau (2019)) use the heuristics and optimization methods previously mentioned. We focus here on three works closely related to ours that employ learning-based approaches. First, Al-Kanj et al. (2018) use approximate dynamic programming to learn a hierarchical interpretation of a value function that is used to determine whether or not to recharge vehicles at their current location, in addition to repositioning decisions to neighboring grid cells and the assignment of vehicles to nearby requests. The study considers the possibility for riders or the operator to reject a trip assignment at various costs, and it also considers the problem of fleet sizing. Second, Pettit et al. (2019) use deep RL to learn a policy determining when an EV should recharge its battery and when it should serve a new request. The study considers time-dependent energy costs. However, it only considers a single vehicle, and it ignores the task of repositioning the vehicle in anticipation of future requests. Finally, Shi et al. (2019) apply deep

reinforcement learning to the RP with a community-owned fleet, in which they seek to minimize customer waiting times and electricity and operating costs. Similar to many of the RL studies cited previously, the study employs a multi-agent RL framework that produces actions for each vehicle individually which are then used in a centralized decision-making mechanism to produce a joint action for the fleet. The joint actions assign vehicles to requests and specify when vehicles should recharge; however the study also ignores repositioning actions.

Thus, ours marks the first application of deep reinforcement learning to the E-RPC in consideration of EV constraints alongside both fleet repositioning and assignment tasks. We also argue that we consider a more realistic problem environment than in most RL-based approaches to ridehail problems. Indeed, the vast majority of studies (e.g., Al-Kanj et al. (2018), Oda and Tachibana (2018), Holler et al. (2019), Shi et al. (2019), Singh et al. (2019)) rely on a coarse grid-like discretization to describe the underlying service region. This serves as the basis for their repositioning actions, which are often restricted to only the adjacent grid cells. This is in contrast to the current work, in which we allow repositioning to non-adjacent sites that correspond to real charging station locations. Time discretization can also be overly coarse. Many studies choose one minute between decision epochs, as in, for example, Oda and Joe-Wong (2018); but some choose far longer between decision epochs, such as six minutes in Shi et al. (2019) and 15 minutes in Al-Kanj et al. (2018). Given that current ridehailing applications provide fast, sub-minute responses, such an arrangement would likely lead to customer dissatisfaction. We are free of such a discretization here, allowing agents to respond to customer requests immediately.

3 Problem Definition and Model

We consider a central operator controlling a homogeneous fleet of electric vehicles $\mathcal{V} = \{1, 2, \ldots, V\}$ that serve trip requests arising randomly across a given geographical region and over an operating horizon T. Across the region are charging stations (CSs, or simply *stations*) $\mathcal{C} = \{1, 2, \ldots, C\}$ at which vehicles may recharge or idle when not in service. The operator assigns vehicles to requests as they arise. Additionally, the operator manages recharging and repositioning operations in anticipation of future requests. The objective is to maximize expected revenue across the operating horizon.

Several assumptions and conditions connect the problem to real-world operations. When a new request arises, the operator responds immediately, either rejecting the request or assigning it to a vehicle. A vehicle is eligible to serve a new request if it can pickup within w time units, the amount of time customers are willing to wait. Each vehicle maintains at most one pending trip request. Thus, a vehicle either serves a newly assigned request immediately or does so after completing work-in-process. Assignments of requests to vehicles cannot be canceled, rescheduled, or reassigned. Each vehicle has a maximum battery capacity Q and known charging rate, energy consumption rate, and speed. The operator ensures assignments and repositioning movements are energy feasible.

We formulate the problem as a Markov decision process (MDP) whose components are defined as follows.

States.

A *decision epoch* $k \in \{0, 1, \ldots, K\}$ begins with a new trip request or when a vehicle finishes its work-in-process, whichever occurs first. The system state is captured by the tuple

$$s = (s_t, s_r, s_{\mathcal{V}}). \tag{1}$$

The current time $s_t = (t, d)$ is the time of day in seconds t and the day of week d. If the epoch was triggered by a new request, then $s_r = ((o_x, o_y), (d_x, d_y))$ consists of the Cartesian coordinates for the request's origin and destination, otherwise $s_r = \emptyset$. The vector $s_{\mathcal{V}} = (s_v)_{v \in \mathcal{V}}$ describes the state of each vehicle in the fleet. For a vehicle v, $s_v = (x, y, q, j_m^{(1)}, j_o^{(1)}, j_d^{(1)}, j_m^{(2)}, j_o^{(2)}, j_d^{(2)}, j_m^{(3)}, j_o^{(3)}, j_d^{(3)})$, consisting of the Cartesian coordinates of the vehicle's current position x, y; its charge $q \in [0, Q]$; and a description of its current activity (or *job*) $j^{(1)}$, as well as potential subsequent jobs $j^{(2)}$ and $j^{(3)}$, which may or may not exist, as determined by the operator. The description of a job $j^{(i)}$ consists of its type $j_m^{(i)} \in \{\text{idle, charge, reposition, preprocess, serve}, \emptyset\}$ (equivalently, $\{0, 1, 2, 3, 4, \emptyset\}$; "preprocess" refers to when a vehicle is en route to pickup a customer), as well as the coordinates of the job's origin $j_o^{(i)} = (j_{o,x}^{(i)}, j_{o,y}^{(i)})$ and destination $j_d^{(i)} = (j_{d,x}^{(i)}, j_{d,y}^{(i)})$. We limit the number of tracked jobs in the state to three, because this is the maximum number of scheduled jobs a vehicle can have given the constraint that a vehicle may have at most one pending trip request: if a vehicle is currently serving a request and has a pending request, then it will have jobs of type $j_m^{(1)} = 4$, $j_m^{(2)} = 3$, and $j_m^{(3)} = 4$. Note that we do not allow vehicles currently preprocessing for one request to be assigned a second (this would indeed require tracking a fourth job). With a sufficiently strict customer service requirement (i.e., the time between receipt of a customer's request and a vehicle's arrival to the customer), the number of opportunities to meet that requirement after undergoing the necessary (preprocessing, serving, preprocessing) sequence is small enough as to warrant the situation's exclusion. If a vehicle has $n < 3$ scheduled jobs, then we set $j_m^{(i)} = j_o^{(i)} = j_d^{(i)} = \emptyset$ for $n < i \leq 3$.

An *episode* begins with the system initialized in state s_0 in epoch 0 at time 0 on some day d with no new request ($s_r = \emptyset$) and all vehicles idling with some charge at a station ($v \in \mathcal{C}$, $q \in [0, Q]$, $j_m^{(1)} = \text{idle}$, $j_m^{(2)} = j_m^{(3)} = \emptyset$ for all $v \in \mathcal{V}$). The episode terminates at some epoch K in state s_K when the time horizon T has been reached.

Actions.

When the process occupies state s, the set of actions $\mathcal{A}(s)$ defines feasible assignments of vehicles to a new request as well as potential repositioning and recharging movements. An action $\mathbf{a} = (a_r, a_1, \ldots, a_v, \ldots, a_V)$ is a vector comprised of components $a_r \in \mathcal{V} \cup \{\emptyset\}$ denoting which vehicle serves new request s_r as well as a repositioning/recharging assignment a_v for each vehicle v indicating the station to which it should be repositioned and begin recharging $a_v \in \mathcal{C} \cup \{\emptyset\}$. $a_r = \emptyset$ denotes rejection of the new request (if it exists) and $a_v = \emptyset$ denotes the "no operation" (NOOP) action for vehicle v, meaning it proceeds to carry out its currently assigned jobs. We refer to the vehicle-specific repositioning/recharging/NOOP actions a_v as RNR actions. Jobs of type *idle* (0), *charge* (1), *reposition* (2), and $\emptyset$ are preemptable, meaning they can be interrupted by a new assignment from the operator, whereas jobs of type *preprocess* (3) and *serve* (4) are non-preemptable.

The following conditions characterize $\mathcal{A}(s)$. If there is no new request ($s_r = \emptyset$), then $a_r = \emptyset$. If there is a new request, then a vehicle is eligible for assignment if it can be dispatched immediately ($j_m^{(1)} \in \{0, 1, 2, \emptyset\}$) or if it is currently servicing a customer ($j_m^{(1)} = 4$) and can then be dispatched ($j_m^{(2)} \notin \{3, 4\}$). Further, a vehicle-request assignment must be energy feasible and must arrive to the customer within w time units, the maximum amount of time a customer is willing to wait. Let $f_q(s_v, s_r)$ be the maximum charge with which vehicle v can reach a station after serving request s_r. $f_q(s_v, s_r)$ is equal to the current charge of vehicle v, minus the charge required for the vehicle to travel to the origin of the new request (minus the charge needed to first drop off its current customer, if $j_m^{(1)} = 4$), to the destination of the new request, and then to the nearest $c \in \mathcal{C}$ from the destination. Let $f_t(s_v, s_r)$ be

Table 1: Eligible actions for an EV v.

Action	Eligibility conditions
Serve request	Request exists, no pending service ($j_m^{(2)} < 3$), energy feasible, time feasible
Reposition to a_v	No active or pending service ($j_m^{(1)} < 3$), energy feasible
NOOP	Always, unless just finished serving and no pending service ($j_m^{(1)} = \emptyset$)

the duration of time required for vehicle v to arrive at the customer. $f_t(s_v, s_r)$ is equal to the time for vehicle v to travel to the origin of the new request from its current location (plus the time to first drop off its current customer, if $j_m^{(1)} = 4$). An assignment of vehicle v to request s_r is feasible if $f_q(s_v, s_r) \geq 0$ and $f_t(s_v, s_r) \leq w$. Repositioning and recharging movements must also be energy feasible and may not preempt existing service assignments. If a vehicle is currently serving or preparing to serve a request ($j_m^{(1)} \in \{3, 4\}$), or if the vehicle has been selected to serve the new request ($a_r = v$), then we only allow the NOOP action $a_v = \emptyset$. Otherwise, we allow repositioning to any station $c \in \mathcal{C}$ that can be reached given the vehicle's current charge level. Lastly, if a vehicle completes a trip request, triggering a new epoch, and that vehicle has not been assigned a subsequent trip request, then it must receive a repositioning action ($a_v \neq \emptyset$). This condition forces vehicles to idle only at stations. Eligible actions for an EV are summarized in Table 1.

Reward Function.

When the process occupies state s and action $\mathbf{a}$ is selected, we earn a reward $C(s, \mathbf{a})$ if we assign a vehicle to serve the new trip request s_r. The reward consists of a base fare plus a charge proportional to the distance of the request. Letting $d(s_r)$ be the distance from the request's origin to its destination, then the reward is

$$C(s, \mathbf{a}) = \begin{cases} 0 & a_r = \emptyset \\ C_b + C_d d(s_r) & \text{otherwise,} \end{cases} \tag{2}$$

where C_b is the base fare, and C_d is the charge per unit distance.

Transition Function.

The transition from one state to the next is a function of the selected action and the event triggering the next epoch. The subsequent state s' is constructed by updating the current state s to reflect changes to vehicles' job descriptions based on selected action $\mathbf{a}$. Then, at time s'_t, we observe either a new request s'_r or a vehicle completing its work-in-process. At that time, we update the vehicle states $s'_{\mathcal{V}}$ to reflect new positions and charges. We also update vehicles' job descriptions: if a vehicle has completed n jobs between time s_t and s'_t, we left-shift vehicles' job descriptions by n ($j^{(i)} \leftarrow j^{(i+n)}$ for $1 \leq i \leq 3-n$) and backfill the vacated entries with null jobs ($j_m^{(i)} = j_o^{(i)} = j_d^{(i)} = \emptyset$ for $3 - n < i \leq 3$). See Appendix A for a more detailed description.

Objective Function.

Define a policy π to be a sequence of decision rules $(X_0^\pi, X_1^\pi, \ldots, X_K^\pi)$, where X_k^π is a function mapping state s in epoch k to an action $\mathbf{a}$ in $\mathcal{A}(s)$. We seek to provide the fleet operator with a policy $\pi^\star$ that

maximizes the expected total rewards earned during the time horizon, conditional on the initial state:

$$\tau(\pi^\star) = \max_{\pi \in \Pi} \mathbb{E}\left[\sum_{k=0}^{K} C(s_k, X_k^\pi(s_k)) \middle| s_0 \right], \tag{3}$$

where Π is the set of all policies. As is common in reinforcement learning, we will often refer to the user or implementer of a policy as an *agent*.

4 Solution Methods

We develop four policies to solve E-RPC, including a random policy that serves as a lower bound. We describe these policies in §4.2 after first providing a brief description of deep reinforcement learning (in §4.1) as it is the foundation of two of our policies.

4.1 Deep Reinforcement Learning

As defined in Sutton and Barto (2018), reinforcement learning (RL) refers to the process through which an *agent*, sequentially interacting with some environment, learns what to do so as to maximize a numerical reward signal from the environment; that is, learns a policy satisfying equation (3). In practice, the agent often achieves this by learning the value of choosing an action a from some state s, known as the state-action pair's *Q-value* ($Q(s,a)$), equal to the immediate reward plus the expected sum of future rewards earned by taking action a from state s. We can express this relationship recursively following from the Bellman equation:

$$Q(s,a) = C(s,a) + \gamma \mathbb{E}\left[\max_{a' \in \mathcal{A}(s')} Q(s',a') \middle| s, a \right], \tag{4}$$

where s' is the subsequent state reached by taking action a from state s and $\gamma \in [0,1)$ is a discount factor applied to the value of future rewards. With knowledge of Q-values for all state-action pairs it may encounter, the agent's policy is then to choose the action with the largest Q-value. However, as the number of unique state-action pairs is too large to learn and store a value for each, a functional approximation of these Q-values is learned. When deep artificial neural networks are used for this approximation, the method is called *deep reinforcement learning* or, more specifically, *deep Q-learning* (we use the terms interchangeably here). The neural network used in this process is referred to as the *deep Q-network* (DQN).

Beginning with arbitrary Q-value approximations, the agent's DQN improves through its interactions with the environment, remembering observed rewards and state transitions, and drawing on these memories to update weights θ defining its DQN. Specifically, with each *step* (completion of an epoch) in the environment, the agent stores a memory which is a tuple of the form (s, a, r, s'), consisting of a state s, the action a taken from s, the reward earned $r = C(s,a)$, and the subsequent state reached s'. Once a sufficient number of memories (defined by the hyperparameter M_{start}) have been accumulated, the agent begins undergoing *experience replay* every M_{freq} steps (Lin 1992). In experience replay, the agent draws a random sample (of size M_{batch}) from its accumulated memories and uses them to update its DQN via stochastic gradient descent (SGD) on its weights θ using a loss function based on the difference

$$r + \gamma \max_{a' \in \mathcal{A}(s')} Q(s',a') - Q(s,a) \tag{5}$$

(or simply $r - Q(s, a)$ if s' is terminal), where the Q-values in equation (5) are estimated using the agent's DQN. The specific loss function and SGD optimizer used to update the weights may vary — we provide the specific implementations chosen for this work in §B.

We utilize the learned Q-values via an *ϵ-greedy* policy, wherein the agent chooses actions randomly with some probability ϵ and chooses the action with the highest predicted Q-value with probability $1 - \epsilon$. The value of ϵ is decayed from some initially large value ϵ_i at the beginning of training to some small final value ϵ_f over the course of some number ϵ_N of training steps. This encourages exploration when Q-values are unknown and exploitation as its predictions improve.

4.1.1 Implemented extensions to deep RL.

We adopt several well established extensions that improve the standard deep RL process just described. First, we utilize a more sophisticated sampling method during experience replay known as *prioritized experience replay* (PER) (Schaul et al. 2016). In PER, memories are more likely to be sampled if they are deemed more important, i.e., if the loss associated with a memory is not yet known (because the memory has not yet been sampled during replay) or if the loss is large (if the agent was more "surprised" by the memory). Second, we use the double DQN (DDQN) architecture (Hasselt et al. 2016), which employs a "target" DQN for action evaluation and a "primary" DQN for action selection. The target DQN is a clone of the primary DQN that gets updated less often (every M_{update} steps), which helps break the tendency for DQNs to overestimate Q-values. Third, we employ dueling DDQN (D3QN) architecture (Wang et al. 2016), in which the DQN produces an estimate both of the value of the state and of the relative *advantage* of each action. By producing these values independently (which together combine to yield Q-values), dueling architecture allows the agent to forgo the learning of action-specific values in states in which its decision-making is largely irrelevant. Finally, we use n-step learning (Sutton 1988), which helps to reduce error in learned Q-values. With n-step learning, the rewards r and subsequent states s' stored in an agent's memories are modified: rather than using the state encountered one step into the future from s (s'), the subsequent state stored in the memory is that encountered n-steps into the future ($s^{(n)}$). Similarly, the reward r is replaced by the (properly discounted) sum of the next n rewards, i.e., those accumulated between s and $s^{(n)}$.

The process of choosing which deep RL extensions to implement and values for associated hyperparameters is not unlike traditional heuristic approaches in transportation optimization. While simple heuristics may suffice in simpler problems (see, e.g., Clarke and Wright (1964)), more complex problems often demand more complex (meta)heuristics specifically tuned for a particular application (see, e.g., Gendreau et al. (2002)). Analogously, off-the-shelf deep RL may be successful in simpler problems (e.g., Karpathy (2016)), but success in more complex problems requires enhancements (e.g., Hessel et al. (2018)) and is dependent on hyperparameter values (Henderson et al. 2018). The suite of deep RL extensions adopted here and our selection of hyperparameter values (§B) are guided by standard values where available and ultimately chosen empirically over the course of the work.

4.2 Policies

We begin by describing *Nearest*, a heuristic policy whose action selection is derived from distance-based rules commonly employed in dynamic taxi dispatching (Maciejewski et al. 2016). We then describe an augmentation of this policy that maintains the Nearest heuristic for RNR decisions but uses deep RL to determine which vehicle to assign to new trip requests. We refer to this policy as *deep ART* (for "Assigns Requests To vehicles") or just *Dart*. Finally, we describe the *deep RAFTR* ("Request Assignments and

FleeT Repositioning") or *Drafter* agent, which uses deep RL both to assign vehicles to requests and to reposition vehicles in the fleet.

4.2.1 Nearest.

Given a state s with new request s_r and vehicle states $(s_v)_{v\in\mathcal{V}}$, the Nearest policy chooses an action $\mathbf{a}$ as follows. To serve the new request, it selects the vehicle which can reach the customer fastest: $a_r = \arg\min_{v\in\bar{\mathcal{V}}} f_t(s_v, s_r)$, where $\bar{\mathcal{V}}$ is the set of vehicles that are eligible to serve the request given time, energy, and job constraints. For vehicle-specific RNR decisions a_v, all vehicles that do not have work-in-process receive instructions to reposition to the nearest station, which we may denote $c_v^\star$ for vehicle v. This yields RNR actions

$$a_v = \begin{cases} c_v^\star & \text{if } j_m^{(1)} \notin \{3,4\} \\ \emptyset & \text{otherwise.} \end{cases} \tag{6}$$

4.2.2 Dart.

The Dart agent uses a combination of heuristics and deep Q-learning to choose an action $\mathbf{a} = (a_r, (a_v)_{v\in\mathcal{V}})$ from state $s = (s_t, s_r, s_\mathcal{V})$. Specifically, Dart selects RNR actions a_v using the same logic as Nearest (equation (6)); however, the action a_r pairing a vehicle with the new request now employs a deep Q-network. We denote by $\mathcal{A}_{\text{Dart}} = \mathcal{V} \cup \{\emptyset\}$ the action space for Dart's DQN. The DQN takes as input the concatenation of a subset of features from the state s which we represent by $x_{\text{Dart}} = x_t \oplus x_r \oplus x_\mathcal{V}$ ("$\oplus$" is the concatenation operator). These components are defined as follows:

x_t **System time** : the concatenation of the relative time elapsed t/T and a *one-hot vector* indicating the day of the week d (e.g., for zero-indexed d with $d = 0$ corresponding to Monday, $d = 6$ corresponding to Sunday, and d currently equal to Thursday ($d = 3$), the vector $(0, 0, 0, 1, 0, 0, 0)$).

x_r **Request information** : the concatenation of a binary indicator for whether there is a new request $\mathbf{1}_{s_r\neq\emptyset}$; a one-hot vector indicating the location of the request's origin; a one-hot vector indicating the request's destination; and for each vehicle v, the distance v would have to travel to reach the request's origin (scaled by the maximum such distance so values are in $[0, 1]$ (ineligible vehicles are given value 1)).

$x_\mathcal{V}$ **Vehicle information** : the concatenation of x_v for all vehicles $v \in \mathcal{V}$ ($x_\mathcal{V} = x_1 \oplus \cdots \oplus x_V$), where the vehicle-specific x_v is itself the concatenation of the time at which its last non-preemptable job ends (scaled by $1/T$), the charge with which it will finish its last non-preemp
job (scaled by $1/Q$), and a one-hot vector indicating its current location.

One-hot vectors in x_{Dart} that indicate location rely on a discretization of the fleet's operating region, as is common in other dynamic ridehail problems in the literature (e.g., Al-Kanj et al. (2018), Holler et al. (2019)). We denote the set of discrete locations (*taxi zones* (TZs)) constituting the region by $\mathcal{L}$.

x_{Dart} is passed to the agent's DQN to produce Q-value predictions. It then uses an ϵ-greedy policy as described in §4.1 to choose an $a_r \in \mathcal{A}_{\text{Dart}}$ (infeasible vehicles are ignored). A schematic of Dart's DQN is shown at left in Figure 1.

4.2.3 Drafter.

From a state s, the Drafter agent uses a DQN to choose a vehicle to serve new requests a_r and to provide RNR instructions a_v for each vehicle $v \in \mathcal{V}$. Whereas the number of unique actions under the control of

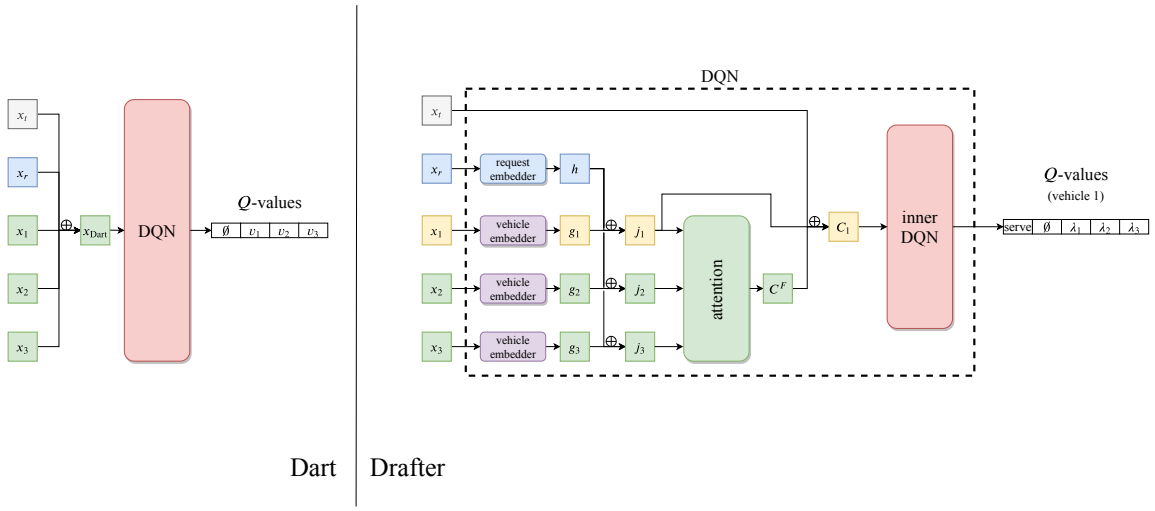


Figure 1: Schematics of the Dart and Drafter agents in the case of three vehicles and three TZs ($V = 3$, $\mathcal{L} = 3$). Elements are concatenated at intersections marked with $\oplus$. The Drafter schematic shows the case where Q-values are being predicted for vehicle 1 (in yellow). Note: in practice, a single forward pass with x_{Drafter} can be used to generate Q-values for all vehicles. We depict the process for a single vehicle here for the sake of clarity.

Dart's DQN is $V+1$, for Drafter it is $(V+1)\times(C+1)^V$. For non-trivial fleet sizes V, this is intractably large. In response, we employ a multi-agent reinforcement learning (MARL) framework which avoids this intractability by making separate, de-centralized Q-value predictions for each vehicle. This reduces the number of actions under the control of Drafter's DQN to $C+2$, corresponding to the vehicle-specific actions of relocating to each station $c \in \mathcal{C}$, serving the request, and doing nothing. We further reduce the number of actions by using the discretization into TZs $\mathcal{L}$ described in §4.2.2, yielding the action space $\mathcal{A}_{\text{Drafter}} = \{\text{serve}, \emptyset\} \cup \mathcal{L}$ for Drafter's DQN. These actions correspond respectively to serving the request, doing nothing, and relocating to each TZ $\lambda \in \mathcal{L}$. For vehicle v, we define its DQN input by $x^v_{\text{Drafter}} = (x_t, x_r, x_v, x_{\mathcal{V}})$ where components are as defined for Dart in §4.2.2 and the redundant x_v is included just to indicate the vehicle for which we are generating Q-values. With vehicle input x^v_{Drafter}, the DQN makes Q-value predictions $Q(x^v_{\text{Drafter}}, a)$ for $a \in \mathcal{A}_{\text{Drafter}}$. These predictions are collected for all vehicles and used to make a centralized decision. A schematic of Drafter's DQN is shown at right in Figure 1.

DQN and attention mechanism. Let us consider the prediction of Q-values for a vehicle $v^\star$. In the example in Figure 1, $v^\star = v_1$. We now also incorporate into the DQN an *attention mechanism* (Mnih et al. 2014) to improve the agent's ability to distinguish the most relevant vehicles in the current state. We expand on the attention mechanism used in Holler et al. (2019) by incorporating information about the current request, which is likely to influence vehicles' relevancy. Details of our attention mechanism can be found in the appendix, §B.1. The attention is used to provide an alternative representation (an *embedding*) of each vehicle $g_v \in \mathbb{R}^l$ and the request $h \in \mathbb{R}^m$, as well as a description of the fleet $C^F \in \mathbb{R}^n$ known as the *fleet context* (here $l = n = 128$, $m = 64$; see §B.1). These components are used to form the vector $C_{v^\star} = C^F \oplus g_{v^\star} \oplus h \oplus x_t$ which is fed to an inner DQN. This inner DQN, which has a structure similar to Dart's DQN, outputs Q-values $Q(x^{v^\star}_{\text{Drafter}}, a)$ for $a \in \mathcal{A}_{\text{Drafter}}$. The process is repeated for

each $v \in \mathcal{V}$, with each vehicle using the same attention mechanism and inner DQN (i.e., all weights are shared).

Drafter is made scalable both through its use of an MARL approach as described above, and also by the attention mechanism. While the MARL approach ensures that the output of the DQN does not depend on the size of the fleet, the attention mechanism ensures the same for the input. More specifically, the attention mechanism makes it possible to predict Q-values (via the inner DQN) using a representation of the fleet C^F whose size is not determined by the number of vehicles in the fleet. This structure allows Drafter to be applied to instances of a different size than those on which it was trained. For example, given a Drafter agent trained to operate a fleet of size V, if a new vehicle is purchased and added to the fleet, or if a vehicle malfunctions and must be temporarily removed, Drafter is capable of continuing to provide service for this modified fleet of size $V' \neq V$. We assess the scalability of the Drafter agent in our computational experiments, §6.

Centralized decision-making. Given Q-value predictions for all vehicles, Drafter's centralized decision-making proceeds as follows. Let $\bar{\mathcal{V}}_{\text{Drafter}}$ be the set of vehicles for whom the service action has the largest Q-value. To serve the request, we choose the vehicle in $\bar{\mathcal{V}}_{\text{Drafter}}$ with the largest such Q-value; this vehicle then receives NOOP for its RNR action. The remaining vehicles perform the RNR action with the largest Q-value, with NOOP prioritized in the event of a tie (ties between repositioning actions are broken arbitrarily). Some consideration must be given to repositioning assignments a_v, as the DQN produces outputs in $\mathcal{L}$ (TZs), while actions a_v take values in $\mathcal{C}$ (stations). Given a repositioning instruction to some TZ $\lambda^\star \in \mathcal{L}$ for vehicle v, we set a_v to the station in $\lambda^\star$ that is nearest to the vehicle.

Given an action $\mathbf{a} = (a_r, (a_v)_{v\in\mathcal{V}})$, define a^v_{Drafter} to be the action in Drafter's action space $\mathcal{A}_{\text{Drafter}}$ assigned to vehicle v:

$$a^v_{\text{Drafter}} = \begin{cases} \text{serve} & a_r = v \\ L(a_v) & \text{otherwise,} \end{cases} \tag{7}$$

where $L(a_v)$ represents the TZ that contains station a_v.

Experience Replay. As described in §4.1.1, Drafter saves state-transition memories $(s, \mathbf{a}, r, s^{(n)})$ at each step which are later used to train its DQN via prioritized experience replay. Let $x^{v(n)}_{\text{Drafter}}$ be the DQN input for vehicle v from state $s^{(n)}$. Because Drafter makes V predictions at each step, our sample of M_{batch} memories leads to an effective sample size of $V * M_{\text{batch}}$, where each memory $(s, \mathbf{a}, r, s^{(n)}) = (x^v_{\text{Drafter}}, a^v_{\text{Drafter}}, r, x^{v(n)}_{\text{Drafter}})_{v\in\mathcal{V}}$. Note that the reward r is shared equally among all vehicles during training, with each vehicle receiving the full amount. This was chosen to encourage cooperation among the fleet. Thus, for Drafter, the difference that forms the basis of its loss function (c.f. equation (5)) is

$$r + \gamma \max_{a' \in \mathcal{A}_{\text{Drafter}}} Q(x^{v(n)}_{\text{Drafter}}, a') - Q(x^v_{\text{Drafter}}, a^v_{\text{Drafter}}). \tag{8}$$

4.2.4 Random.

Finally, we consider the *Random* policy which chooses a vehicle to serve the request (a_r) uniformly from $\mathcal{V} \cup \{\emptyset\}$ and chooses repositioning locations a_v uniformly from $\mathcal{C} \cup \{\emptyset\}$ for each $v \in \mathcal{V}$. We offer Random simply to serve as a lower bound.

5 Dual Bounds

Assessing policy quality is hampered by the lack of a strong bound on the value of an optimal policy, a dual bound. Without an absolute performance benchmark it is difficult to know if a policy's performance is "good enough" for practice or if additional research is required. Here we offer two dual bounds. First is the value of an optimal policy that can serve all requests (§5.1), and the second is the value of an optimal policy with perfect information, i.e., the performance achieved via a clairvoyant decision maker (§5.2).

5.1 Serve-All Bound.

We first consider the *Serve-all* (SA) dual bound in which we assume that new requests can always be assigned to some vehicle $v \in \mathcal{V}$. That is, we assume that the size of the fleet is sufficient to be able to meet demand. We know that, given a fleet of sufficient size, an optimal policy will serve all requests, because policies seek to maximize the sum of rewards, and rewards are non-negative and earned only by serving requests ($a_r = \emptyset \Rightarrow C(s, \mathbf{a}) = 0$)). Therefore, to compute the value of this policy, we can simply sum the rewards of all observed requests. The gap between the SA dual bound and the best policy is a reflection of the adequacy of fleet size relative to demand and can serve as justification for a ridehailing company to invest (or not) in additional vehicles.

5.2 Perfect Information Bound.

In practice, it is likely that not all requests can be feasibly served, making the Serve-all bound loose. In an attempt to establish a tighter dual bound, we consider the value of an optimal policy under a *perfect information* (PI) relaxation. Under the PI relaxation, the agent is clairvoyant, aware of all uncertainty a priori. In the E-RPC, to have access to PI is to know in advance all details regarding requests: their origins, destinations, and when they will arise. Denote such a set of known requests by $\mathcal{R} \in \mathrm{P}$, where P is the set of all request sets.

In the absence of uncertainty, we can rewrite the objective function as

$$\max_{\pi \in \Pi} \mathbb{E}\left[\sum_{k=0}^{K} C(s_k, X_k^\pi(s_k)) \middle| s_0\right] = \mathbb{E}\left[\max_{\pi \in \Pi} \sum_{k=0}^{K} C(s_k, X_k^\pi(s_k)) \middle| s_0\right]. \tag{9}$$

Notice that the *perfect information problem* (right-hand side of equation (9)) can be solved with the aid of simulation. We may rely on the law of large numbers — drawing random realizations of uncertainty (request sets $\mathcal{R} \in \mathrm{P}$), solving the inner maximization for each, and computing a sample average — to achieve an unbiased and consistent estimate of the true objective value. This value is the *perfect information bound*. It remains to solve the inner maximization.

In the absence of uncertainty that results from having access to PI, the inner maximization can be solved deterministically: with all information known upfront, no information is revealed to an agent during the execution of a policy. As a result, there is no advantage in making decisions dynamically (step by step) rather than statically (making all decisions at time 0). This permits the use of an exact solution via math programming, which we pursue using a *Benders-based branch-and-cut* algorithm in which at each integer node of the branch-and-bound tree of the *master problem*, the solution is sent to the *subproblem* for the generation of Benders cuts. Here, the master problem is responsible for assigning vehicles to requests in a time-feasible manner, and the subproblem is responsible for ensuring the energy feasibility of these assignments. The use of the Benders-based decomposition enables the solution of

large PI problems which may not be otherwise feasible. We discuss the master problem in more detail in §5.2.1, the subproblem and the generation of cuts in §5.2.2, and comment on the bound's tractability in §5.2.3.

5.2.1 Master problem.

The master problem, responsible for time-feasible assignments of customer requests to vehicles, is the mixed integer-linear program (MIP) defined by equations (10)-(16). In it, requests are represented as nodes in a directed graph $\mathcal{G}$. Request nodes i and j are connected by a directed arc (i, j) when a vehicle can feasibly serve request j after request i (ignoring energy requirements). The graph $\mathcal{G}$ also contains a dummy node for each vehicle. These dummy nodes are connected to request nodes for which the assignment of vehicles to requests is time-feasible. The problem involves choosing arcs in $\mathcal{G}$, starting from vehicles' dummy nodes, that form (non-overlapping) paths for the vehicles which indicate the sequence of requests that vehicles will serve. If a request i is a member of some vehicle's path, it contributes c_i to the objective function, its value as given by the reward function (equation (2)).

In the master problem, $\mathcal{R}$ is the set of all requests (known a priori given PI), $\mathcal{V}$ is the set of vehicles, c_i is the reward associated with serving request i, h_i is a binary variable taking value 1 if request i is assigned to any vehicle, y_{vi} is a binary variable taking value 1 if job i is the first request assigned to vehicle v, x_{ij} is a binary variable taking value 1 if a vehicle is assigned serve request j immediately after request i, z_i is a continuous non-negative variable equal to the time at which a vehicle arrives to pick up request i, t_i^r is the time at which request i begins requesting service, w is the maximum amount of time a customer may wait between when they submit their request and when a vehicle arrives, t_i^p is the travel time between the origin and destination of request i, and t_{ij}^s is the travel time between the destination of request i and the origin of request j. For vehicles, t_v^p is equal to the earliest time at which they can depart from their initial locations, and t_{vi}^s is equal to the travel time between their initial locations and the origin of request i. We formally define the master problem as

$$\text{maximize} \quad \sum_{i \in \mathcal{R}} c_i h_i \tag{10}$$

$$\text{subject to} \quad \sum_{v \in \mathcal{V}} y_{vi} + \sum_{j \in \mathcal{R} \setminus \{i\}} x_{ji} \geq h_i, \quad \forall i \in \mathcal{R} \tag{11}$$

$$\sum_{j \in \mathcal{R} \setminus \{i\}} x_{ij} \leq h_i, \quad \forall i \in \mathcal{R} \tag{12}$$

$$\sum_{j \in \mathcal{R}} y_{vj} \leq 1, \quad \forall v \in \mathcal{V} \tag{13}$$

$$z_i \geq t_i^r + (t_v^p + t_{vi}^s - t_i^r) y_{vi}, \quad \forall i \in \mathcal{R}, \forall v \in \mathcal{V} \tag{14}$$

$$z_i - z_j \geq t_i^r - t_j^r - w + (t_j^p + t_{ji}^s - t_i^r + t_j^r + w) x_{ji}, \quad \forall i, j \in \mathcal{R}\,(i \neq j) \tag{15}$$

$$h_i, x_{ij}, y_{vi} \in \{0, 1\};\ t_i^r \leq z_i \leq t_i^r + w \tag{16}$$

The objective (10) maximizes the sum of rewards earned by assigning requests to vehicles. Equation (11) manages the binary assignment variable h_i, requiring that it be included in some vehicle's path to take value 1. Similarly, equation (12) manages the variables for the outgoing arcs from request i, forcing them to take value 0 if the request has not been assigned to a vehicle. Equation (13) ensures that each vehicle has at most one request assigned to be its first. Equation (14) sets a lower bound for the time at which vehicles' can arrive to their initial requests, and equation (15) sets a lower bound for the time at which vehicles' can arrive to subsequent requests. Finally, equation (16) defines variables

scopes' and bounds the earliest and latest possible start times for requests z_i.

As mentioned, we only connect request nodes i and j via directed arc (i, j) if it is time-feasible to serve request j after request i; that is, if $t_i^r + t_i^p + t_{ij}^s \leq t_j^r + w$. While energy-feasibility is ultimately ensured by the subproblem, we can facilitate the master problem by eliminating additional arcs in $\mathcal{G}$ using known energy consumptions to perform stronger feasibility checks. Let ρ be the maximum rate at which vehicles acquire energy when recharging, q_i^p be the energy required to travel from the origin to the destination of request i, q_{ij}^s be the energy required to travel from the destination of request i to the origin of request j, q_i^d be the energy required to travel from the destination of request i to the nearest charging station, and q_i^o be the energy required to travel to the origin of request i from its nearest charging station. Then for arc (i, j) to exist, it must be that $t_j^r + w - (t_i^r + t_i^p + t_{ij}^s) \geq \frac{1}{\rho} \max\{0, (q_j^p + q_j^d - (Q - q_i^o - q_i^p))\}$, which states that the maximum down time between i and j (left-hand side) must be sufficient to accommodate any recharging that must occur between these requests (right-hand side). We provide similar feasibility checks for the arcs (v, i) connecting vehicle dummy nodes to requests. Specifically, the simpler time-feasible check requires that vehicle v can arrive to request i in time: $t_v^p + t_{vi}^s \leq t_i^r + w$. In consideration of energy requirements, we ensure $t_i^r + w - (t_v^p + t_{vi}^s) \geq \frac{1}{\rho} \max\{0, q_{vi}^s + q_i^p + q_i^d - q_v^p\}$, which states that the time for the vehicle to perform any required charging (right-hand side) must be no greater than the available time before it must arrive to request i. Finally, we force the underlying graph to be acyclic, meaning we prohibit pairs of requests i, j such that $(t_i^r + t_i^p + t_{ij}^s < t_j^r + w) \wedge (t_j^r + t_j^p + t_{ji}^s < t_i^r + w)$.

5.2.2 Subproblem.

A solution to the master problem is a sequence of request assignments $r_v = (r_v^1, r_v^2, \ldots)$ for each vehicle $v \in \mathcal{V}$. r_v^1 is taken to be the element in the singleton $\{j \,|\, y_{vj} = 1\} \cup \emptyset$; subsequent entries r_v^i are similarly elements in singletons $\{j | x_{r_v^{i-1} j} = 1\} \cup \emptyset$ (r_v terminates with the first null element). Given sequences for each vehicle, the subproblem must ensure they are energy feasible. To do so, we use a modified version of the labeling algorithm developed by Froger et al. (2019) to solve the Fixed Route Vehicle Charging Problem (FRVCP) (Montoya et al. 2017). The FRVCP entails providing charging instructions (where to charge and to what amount) for an electric vehicle traversing a fixed sequence of customers. The algorithm takes as input a sequence like r_v and, if successful, terminates with the minimum duration path through the sequence that includes instructions specifying at which charging stations to recharge and to what amount. When unsuccessful, the labeling algorithm returns the first unreachable node in the sequence. In the original implementation customers did not have time constraints as they do here, where vehicles are required to arrive to requests in the window $[t_i^r, t_i^r + w]$. We further discuss the algorithm and our modifications to it to accommodate this difference in the appendix, §C.

Unsuccessful termination of the algorithm for a sequence r_v indicates that the sequence cannot be traversed energy-feasibly. To remove it from the search space, we add the following *feasibility cut* to the master problem:

$$y_{v,r_v^1} + \sum_{i=2}^{j^\star} x_{r_v^{i-1}, r_v^i} < |r_v|, \tag{17}$$

where $j^\star \leq |r_v|$ is the index of the first unreachable request node in r_v to which the algorithm was unable to feasibly extend a label. This cut (17) enforces that not all variables defining the sequence be selected.

5.2.3 Tractability of the PI Bound.

The PI bound is often computationally intractable to obtain. This is because the estimation of the expected value with perfect information entails repeated solutions to the inner maximization of equation (9), a challenging problem despite the absence of uncertainty. Even if the Benders-based method of §5.2.1 and 5.2.2 does not return an optimal solution for a given realization of uncertainty, we may still get a valid bound by using the best (upper) bound produced by the solver (Gurobi v8.1.1). The solver's bound, typically attained via linear relaxations to the master problem, serves as a bound on the value of an optimal policy with PI, and therefore also as a bound on an optimal policy. This mixed bound, effectively combining both an information and a linear relaxation, is weaker than a bound based on the information relaxation alone. However, we show in computational experiments that it is still useful. In the experiments, when the bound results only from the information relaxation (i.e., we were able to solve all realizations of uncertainty to optimality), we denote the bound by an asterisk (*); otherwise, the bound is mixed.

6 Computational Experiments

To test the solution methods proposed in §4, we perform computational experiments modeled after business-day ridehailing operations on the island of Manhattan in New York City. We describe the experimental setup in §6.1, present our results in §6.2, and offer a brief discussion of the results in §6.3.

6.1 Experimental Setup

We generate problem instances (equivalently, *episodes* over which the agents act) using data publicly available for the island of Manhattan in New York City. New York City Taxi & Limousine Commission (2018) provide a dataset consisting of all ridehail, Yellow Taxi, and Green Taxi trips taken in 2018. We filter the data to only those trips that originate and terminate in Manhattan and those that occur on business days ($d \in \{0, 1, 2, 3, 4\} = \{\text{Mon, Tues, Weds, Thurs, Fri}\}$, excluding public holidays). The average daily profile for these data is shown in Figure 2. Based on this profile, we set the episode length T to 24 hours, with episodes beginning and ending at 3:00 am, as this time corresponds to a natural lull in demand from the previous day and gives agents time to perform proactive recharging before the morning demand begins. Data for each trip includes the request's pickup time as well as the location of its origin and destination. Requests' origins and destinations are given by their corresponding *taxi zones*, an official division of New York City provided by the city government (NYC OpenData 2019) that roughly divides the city into neighborhoods. We use this as the basis for our geographical discretization $\mathcal{L}$ as described in §4.2, resulting in $|\mathcal{L}| = 61$ TZs for Manhattan (see Figure 3). Coordinates $((o_x, o_y), (d_x, d_y))$ for requests' exact origins and destinations are drawn randomly from inside their corresponding TZs.

We fix the set of stations $\mathcal{C}$ to all CSs currently available or under construction in Manhattan, as listed by National Renewable Energy Laboratory (2019), yielding a set of $|\mathcal{C}| = 302$ CSs (blue marks in left map of Figure 3). We assume that all CSs dispense energy at a constant rate of 72kW, equal to that of a Tesla urban supercharger (Tesla 2017). Vehicles' batteries have similar traits to that of a mid-range Tesla Model 3, with a capacity of 62 kWh and an energy consumption of 0.15 kWh/km. Further, we assume vehicles travel at a speed of 16.1 km/hr (10 mi/hr), and, when serving a request, receive a fixed reward of $C_b = \$7.75$ and distance-dependent reward of $C_d = \$1.9$/km, under the assumption of Euclidean distances. We set the maximum time that customers are willing to wait for a vehicle to be $w = 5$ minutes.

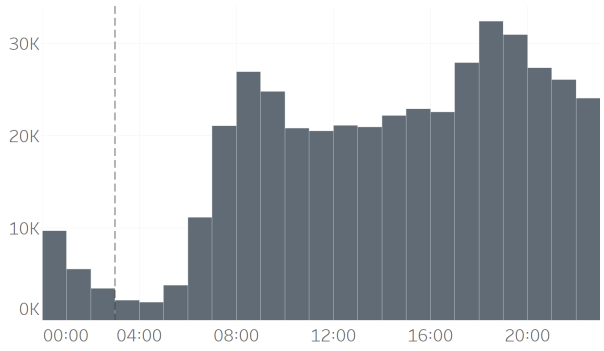


Figure 2: Mean number of requests by hour in Manhattan on a business day in 2018. The dashed vertical line indicates episodes' 03:00 start time.

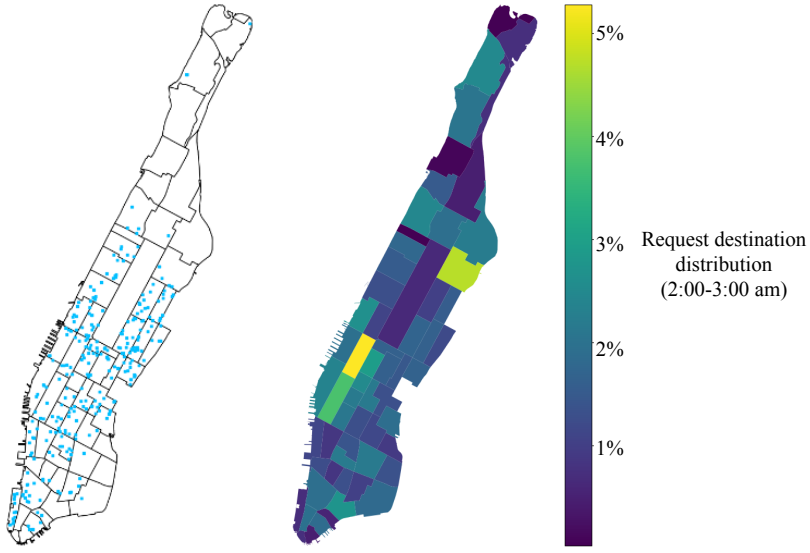


Figure 3: (Left) The island of Manhattan divided into taxi zones with charging station locations shown in blue. (Right) Distribution of requests' desintations by taxi zone during episodes' last hour (02:00-03:00).

Vehicles' initial locations are determined by drawing a TZ randomly according to the distribution of drop-off locations during the final hour of the previous day's operation (2:00-3:00am). See right map in Figure 3. We then choose a charging station uniformly from within the TZ (TZs without CSs are excluded). Vehicles' initial charges are drawn uniformly from $[0, Q]$. To randomly generate a set of R requests for an instance, we first sample a business day from 2018 then draw R requests from that day.

We consider problem instances of three size classes as determined by the number of daily requests R and vehicles V; we will refer to these size classes by their ratio R/V. The first instance size 1400/43 was chosen such that the mean number of requests per hour during the busiest time of day (between 6:00-7:00pm, see Figure 2) is approximately 100. The number of vehicles in this set was then chosen to align with the (hourly) R/V ratio used in experiments in Bertsimas et al. (2019). Experiments on 1400/43 instances include results for all agents as well as the SA and PI bounds. The agents and SA bound are evaluated over a set of 200 episodes, while the PI bound — given its higher computational demand — is established using an average over 50 episodes. Prior to evaluation, the Dart and Drafter agents are first trained on a separate set of 750 episodes. We note that the PI bound can be solved to optimality for these instances, so we denote it by PI*.

Given a daily request set of size $R = 1400$, a fleet size of $V = 14$ is arguably more realistic than $V = 43$. This number is derived from a scaling of $R^\star/V^\star$, where $R^\star = 449,121$ is the total number of trip requests (taxi and ridehail) served on an average business day in Manhattan in 2018, and $V^\star = 4,400$ is an approximation of the number of simultaneously active Yellow Taxis in Manhattan at any given time in 2018 (see data aggregations in, e.g., Schneider (2019)). Consequently, the second size class of instances we consider is 1400/14. With the same demand but a reduced supply, these instances represent a more challenging problem environment for the agents than 1400/43. Experiments for this set include results for all agents and the SA and PI bounds; however, we can no longer solve the PI bound to optimality, so we use the mixed version of this bound as described in §5.2.3. Again, the SA bound and all agents are evaluated on a set of 200 episodes, with Dart and Drafter first trained on a set of 750 episodes, and the PI bound is an average over 50 episodes.

Finally, we consider an instance set ten times larger (14000/140) to assess Drafter's ability to scale and generalize. That is, we evaluate the Drafter agent directly on 200 instances of size 14000/140 without any additional training — it simply uses its DQN as trained on the 1400/14 instances. We compare the Drafter agent to the Random and Nearest agents, as well as the SA bound (we forgo the PI bound completely due to the size of the problem, $\sim 10^8$ variables and constraints). Dart is absent from this analysis as its scalability is inherently handicapped, with its DQN's output (that is $\mathcal{A}_{\text{Dart}}$) dependent on the number of vehicles in the instance. This is in contrast to Drafter, for which the size of its DQN output ($|\mathcal{A}_{\text{Drafter}}|$) is indifferent to the size of the fleet. Details on the hyperparameters used in the training of the Dart and Drafter agents for all instance size classes are provided in §B.

6.2 Results

We divide our analysis of the results by instance size, beginning with 1400/43 in §6.2.2, 1400/14 in §6.2.3, and 14000/140 in §6.2.4. Prior to describing agents' performance on the instance sets, we first offer a comparison of the dual bounds in §6.2.1.

6.2.1 Comparison of Dual Bounds.

The first proposed dual bound, SA, is simple to compute but may be loose when not all requests can be feasibly served. In contrast, the PI bound promises to be tighter, but it is significantly more challenging

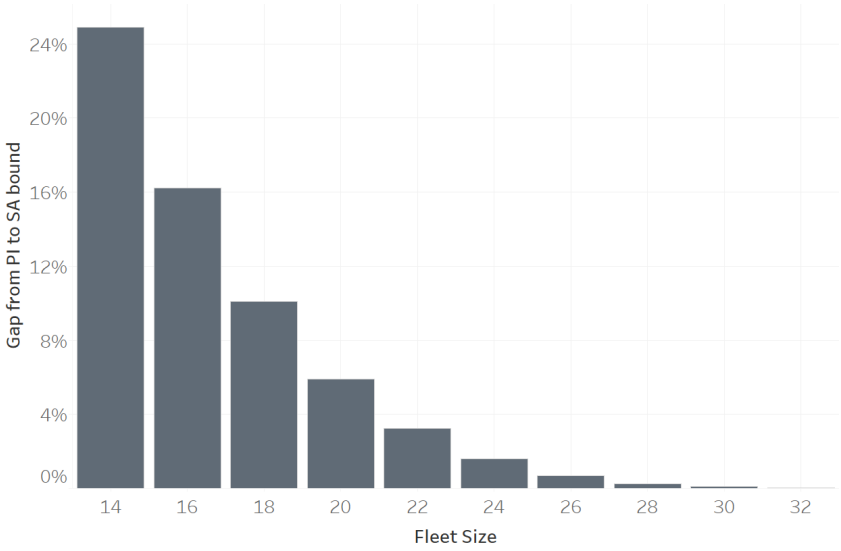


Figure 4: The mean percent difference of the SA bound relative to the PI bound over 20 episodes of an instance with 1400 requests for varying fleet sizes. Note: computed as $\frac{SA-PI}{PI}$, where SA is the revenue earned if all requests are served and PI is (the upper bound on) the revenue earned by an optimal poicy with perfect information.

to compute. Here we aim to quantify the value of the additional computation required for the PI bound.

Intuitively, with a larger fleet size V, more requests can be served. At some V, a policy with PI should be able to serve all requests, so the PI and SA bounds will be equal. Conversely, with decreasing fleet size, even with PI, it should become impossible to serve all requests, so the bounds will diverge. We test this intuition in an experiment comparing the value of these two bounds as a function of fleet size. The results are summarized in Figure 4. We begin with instances of size 1400/14, for which (as shown in §6.2.3) the gap between the PI and SA bounds is large. We then increment the fleet size by 2 ($V' = V + 2$), and resolve 20 episodes of the $1400/V'$ instances. We find that the results confirm intuition – the gap between the bounds decreases exponentially with increasing fleet size, reaching equivalence (100% of requests served by an optimal policy with PI) at a fleet size of 32. We note that the baseline PI values here reflect the mixed PI bound discussed in §5.2.3, implying that the gaps may actually be larger than these results suggest.

6.2.2 Instance size class 1400/43.

The Dart and Drafter *training curves*, showing their increasing objective performance over the 750 training episodes, are provided in Figure 5. We see that learning happens quickly for both agents, stabilizing after approximately 250 training episodes at which point they outperform the Nearest policy, whose performance over the training episodes is also shown for reference in Figure 5. In Figure 6 we compare agents' performance on the 50 evaluation episodes for which the PI bound is available. For each of the 50 evaluation episodes, we are able to solve the PI bound to optimality. We find that the Drafter agent is the best performing with a mean daily revenue of \$14,642, although there remains an 18.8% gap between this policy and both the PI and SA dual bounds, whose values are identical. As anticipated by the results in §6.2.1, an optimal policy with perfect information is able to serve all requests. After Drafter, Dart is the second-best performing agent, earning a mean daily revenue of \$14,435 and beating

Figure 5: Training curves for the Dart (orange) and Drafter (blue) agents over 750 training instances of size 1400/43. Nearest (red) is provided as a reference. Darker curves are smoothed representations of the lighter curves.

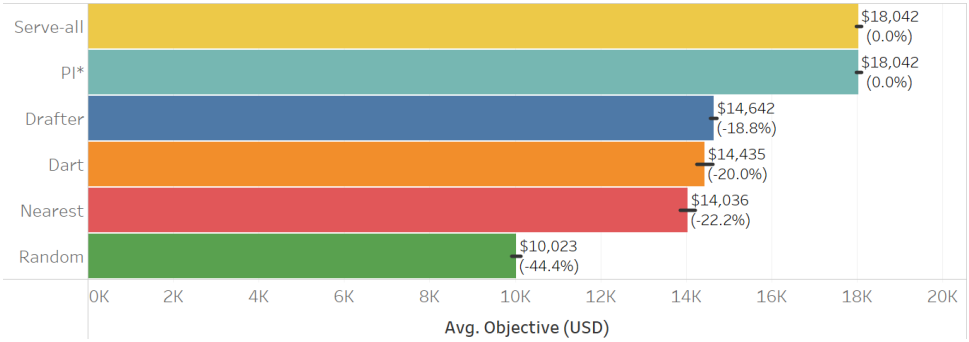


Figure 6: Agents' objective performance relative to the PI bound over the 50 evaluation instances of size 1400/43 for which the PI bound is available. Parenthesized values indicate the gap to the PI bound, which is equivalent to the SA bound. Black marks indicate 95% confidence intervals.

Nearest (at $14,036) by 2.8%.

We provide further comparison between the agents in Figure 7 which shows three valuable metrics for a fleet operator: as measures of customer service, we consider average customer waiting time and number of requests served; and as a measure of sustainability, we consider *fleet occupancy rates* (the percent of time, averaged across all vehicles, that they are either preprocessing or serving a request). We see that by always assigning the nearest eligible vehicle to serve a request, Nearest achieves the shortest average waiting time of 145 s. This is in contrast to the Dart and Drafter agents which have average customer waiting times of 198 and 200 s respectively. The similarity of Random's average wait time (202 s) to Drafter and Dart is due to the fact that it is in effect drawing a waiting time randomly from a distribution that is bounded below by Nearest (which always chooses the minimum) and above by the maximum waiting time $w = 300$ s. By being able to assign vehicles further away, we see that Drafter and Dart are both able to serve more requests on average, resulting in higher fleet occupancy rates and greater objective values.

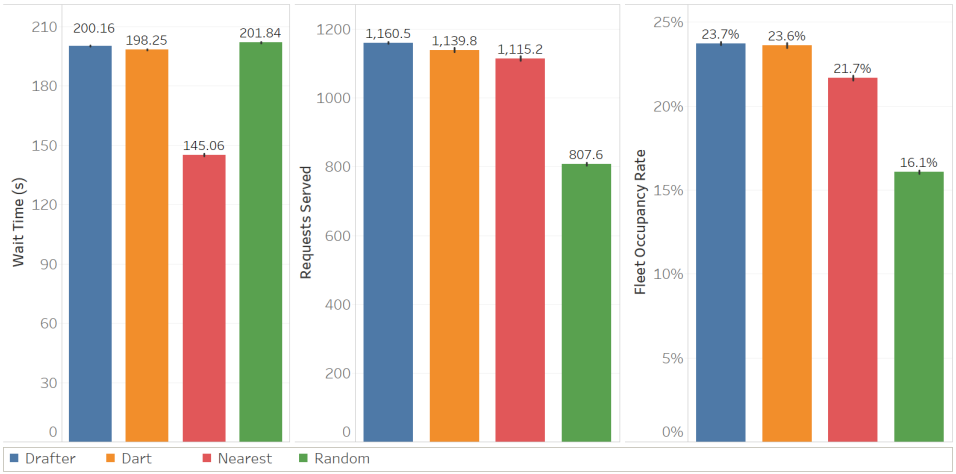


Figure 7: Over 200 evaluation instances of size 1400/43, agents' performance as measured by (Left) average customer waiting time (Center) number of requests served, and (Right) fleet occupancy rate. Black marks indicate 95% confidence intervals.

6.2.3 Instance size class 1400/14.

Training curves for Dart and Drafter on the 1400/14 instances are provided in Figure 8. We see again rapid learning for both agents, plateauing after approximately 200 training episodes. Comparing agents' objective achievement over 200 evaluation episodes (Figure 9), we find that Drafter now achieves a significantly higher objective than both the Dart and Nearest agents, outperforming the former by \$1,597 (16.8%) and the latter by \$1,557 (16.4%). We find the agents' mean performance relative to the PI bound to be looser (34.3%) than for the 1400/43 instances, suggesting that access to information about the future becomes increasingly more valuable as supply shrinks relative to demand. We also find, as anticipated by the results in §6.2.1, that the gap between the PI and SA bounds has increased from 0% to 25%. Figure 6.2.3 offers further comparison of the agents. We again find that Nearest achieves the shortest waiting time of approximately 3 minutes, compared to approximately 3.3 minutes for Dart and Drafter. In contrast to before, this does not equate to a greater number of requests served by Dart, nor to an appreciable increase in its fleet occupancy rate. Drafter, however, serves on average 137 more requests per day than Dart and 134 more than Nearest. This results in the highest fleet occupancy rate of 46.7%, relative to 39.1% for Dart and 38.5% for Nearest.

6.2.4 Instance size class 14000/140.

We apply Drafter directly to the 14000/140 instances after undergoing no additional training after the 1400/14 instances, and we find that it is again the best performing agent, achieving an objective 4.5% (\$5,649) better than Nearest and coming within 26.6% of the SA dual bound (see Figure 10). Figure 11 offers a more detailed analysis, showing that on average Drafter serves 5.7% (570) more requests per day than Nearest and has a 5% higher fleet occupancy rate. It does so while offering a customer waiting time of 212 s, only 38 s longer than that of Nearest (174 s).

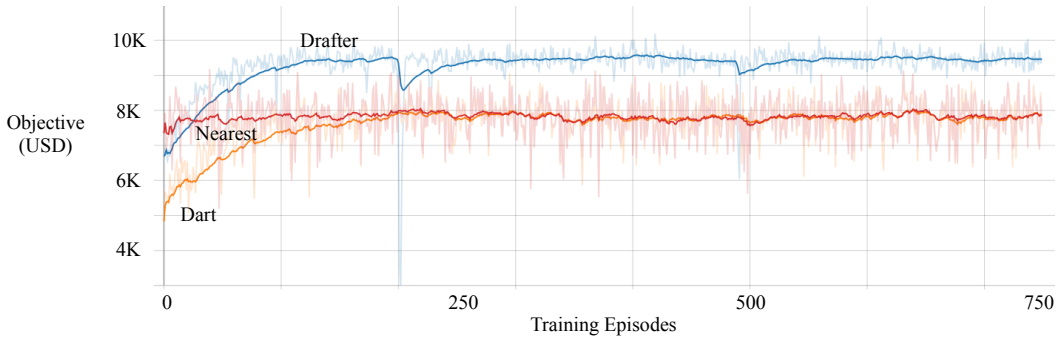


Figure 8: Training curves for the Dart (orange) and Drafter (blue) agents over 750 instances of size 1400/14. Nearest (red) is provided as a reference. Darker curves are smoothed representations of the lighter curves.

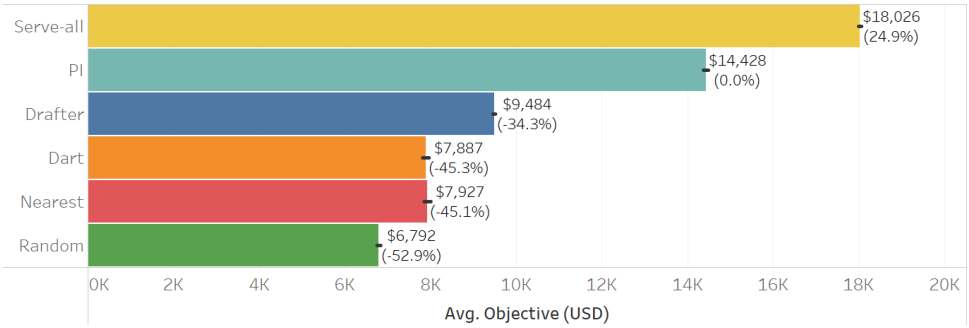
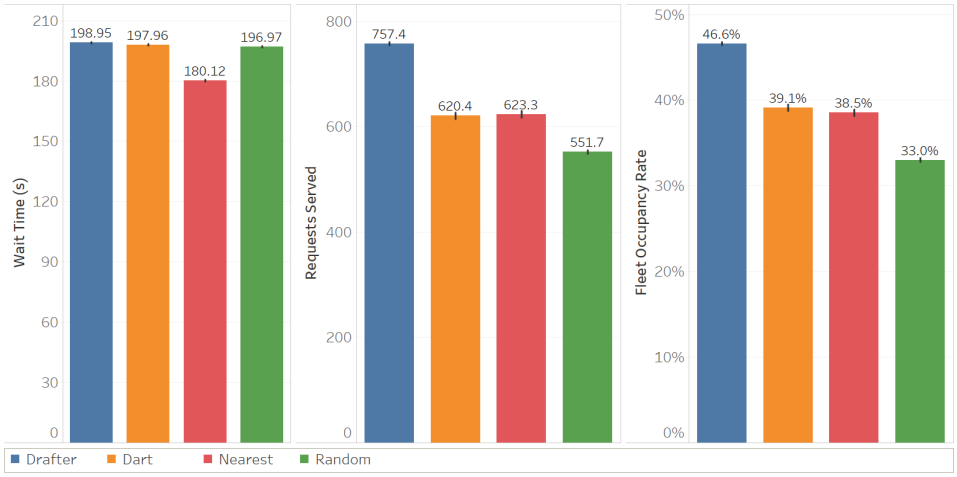


Figure 9: Agents' objective performance over 200 evaluation instances of size 1400/14 with the gap to the SA bound in parentheses. Black marks indicate 95% confidence intervals.



Over 200 evaluation instances of size 1400/14, agents' performance as measured by (Left) average customer waiting time (Center) number of requests served, and (Right) fleet occupancy rate. Black marks indicate 95% confidence intervals.

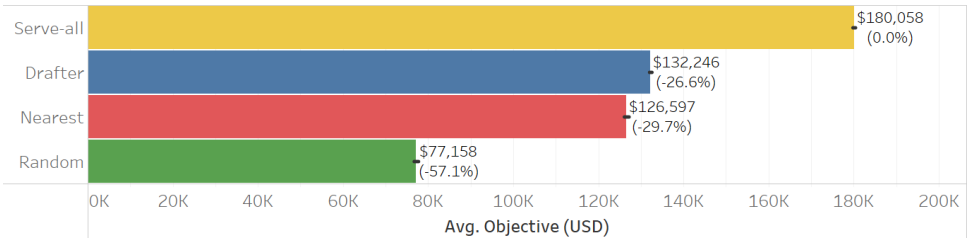


Figure 10: Agents' objective performance over 200 evaluation instances of size 14000/140 with the gap to the SA bound in parentheses. Black marks indicate 95% confidence intervals.

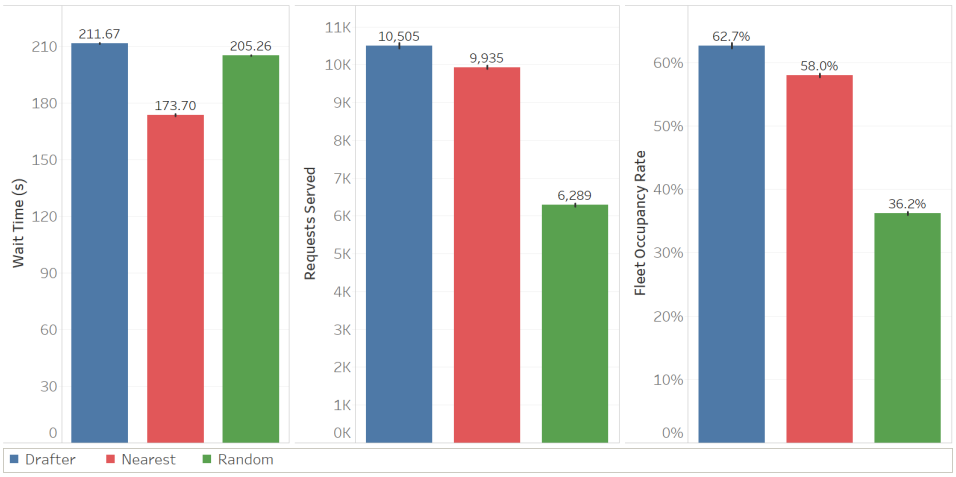


Figure 11: Over 200 evaluation instances of size 14000/140, agents' performance as measured by (Left) average customer waiting time (Center) number of requests served, and (Right) fleet occupancy rate. Black marks indicate 95% confidence intervals.

6.3 Discussion

While often in dynamic optimization no dual bound is available against which to compare policy performance, we offer two here. Although the gaps from these bounds to our agents are sometimes large, the results serve to illustrate that even a loose bound has utility. For example, agents' widening gaps to the SA bound from the 1400/43 to the 1400/14 instances demonstrate the potential for the bound to gauge the size of the fleet relative to demand. Having such a measure allows the decision-maker to justify (or not) investments in additional vehicles. Comparative analysis of the bounds also suggests that access to perfect information is of significant value in the E-RPC: for instances with a V/R ratio of 1400/32 and greater, the bounds are equivalent, indicating that an optimal policy with PI is consistently able to serve all requests.

Most notable from the computational experiments is the performance of the Drafter agent. Drafter achieves the best objective value in all size classes, outperforming the second-best agent by 1.4% in the 1400/43 instances, 16.4% in the 1400/14 instances, and 4.5% in the 14000/140 instances. The results demonstrate Drafter's ability to conduct fleet repositioning actions that anticipate future demand and allow it to serve more customers. The increase in its performance relative to other agents between the 1400/43 and 1400/14 instances suggests that this anticipation is especially valuable in supply-limited regimes. Additionally, Drafter requires no prior knowledge or assumptions about the shape of the demand — in the language of reinforcement learning, it is *model-free*. Rather than relying on, e.g., historical data or domain knowledge regarding parameters describing the demand's spatial and temporal distributions, Drafter garners sufficient working knowledge of the demand during its training to determine which actions best allow it to serve future requests and ultimately achieve greater objective values. Being model-free also implies that Drafter is flexible to various relationships between actions and rewards, because this relationship is learned by the agent. That is, while we trained and evaluated Drafter on the reward function defined by equation (2), we could have equally used, e.g., a stochastic reward function or one which incorporated additional revenue or cost metrics. Although displaying inferior performance than Drafter, Dart is also model-free and therefore enjoys these same benefits. Finally, we also highlight Drafter's scalability. After undergoing training on instances of size 1400/14, the agent was directly applied to the 14000/140 instances, ten times larger than those on which it was trained. Despite not receiving any training on instances of this size, Drafter still managed to outperform the other agents in this size class. This is encouraging for operators of ridehail companies, as it suggests that Drafter may be robust to changes in problem instances. That is, in the event that demand is slow on a particular day or one or more vehicles are unavailable (e.g., due to repairs), then Drafter should still provide reliable service.

7 Concluding Remarks

We considered the E-RPC, the problem faced by an operator of a ridehail service with a fleet composed of centrally-controlled electric vehicles. To solve the E-RPC, we developed two policies, Dart and Drafter, derived with deep reinforcement learning. To serve as a comparison, we consider a heuristic policy common in dynamic taxi dispatching, as well as a randomly-acting policy that serves as a lower bound. We develop two dual bounds on the value of an optimal policy, including the value of an optimal policy with perfect information. To establish this bound, we decompose a relaxed version of the E-RPC into time- and energy-feasibility subproblems, and solve them via a Benders-like decomposition. We offer an analysis of when this more complex dual bound offers value beyond that of a simpler proposed dual

bound. The two dual bounds provide utility in assessing fleet size relative to demand and the value of perfect information.

We perform computational experiments on instances derived from real world data for ridehail operations in New York City in 2018. Across instances of varying sizes, we consistently find the deep RL-based Drafter agent to be the best performing. On the most realistic instances for which it was specifically trained, it achieved an objective 16.4% better than any other policy. We further show that Drafter is scalable, finding that it outperforms other policies by 4.5% on instances ten times larger than any on which it was trained. These results ultimately suggest there are opportunities for deep reinforcement learning in solving problems like the E-RPC. By more efficiently utilizing finite resources, these methods promote greater sustainability in the domain of transportation and logistics.

Acknowledgements

Nicholas Kullman is grateful for support from the Société des Professeurs Français et Francophones d'Amérique. Justin Goodson wishes to express appreciation for the support from the Center for Supply Chain Excellence at the Richard A. Chaifetz School of Business. Martin Cousineau and Jorge Mendoza would like to thank HEC Montréal and the Institute for Data Valorization (IVADO) for funding that contributed to this project. This research was also partly funded by the French Agence Nationale de la Recherche through project e-VRO (ANR-15-CE22-0005-01).

A Updating Vehicles' Job Descriptions

Upon choosing action $\mathbf{a}$ from state s, we immediately update vehicles' job descriptions. We describe this process here. For the following discussion, let the current location of vehicle v be (v_x, v_y).

NOOP.

Vehicles receiving the NOOP instruction ($a_v = \emptyset$) see no change to their existing job descriptions.

Repositioning/Recharging.

Let the location to which the vehicle is instructed to reposition be $a_v = c$ with location (c_x, c_y). We set the first job's type to repositioning ($j_m^{(1)} = 2$), its origin to the vehicle's current location ($j_o^{(1)} = (v_x, v_y)$), and its destination to c ($j_d^{(1)} = (c_x, c_y)$). We then set the second job type to charging ($j_m^{(2)} = 0$) which begins and ends at c ($j_o^{(2)} = j_d^{(2)} = (c_x, c_y)$). After recharging, the vehicle then idles at the station: $j_m^{(3)} = 0$ and $j_o^{(3)} = j_d^{(3)} = (c_x, c_y)$.

Serving Request.

Let the new request have origin (o_x, o_y) and destination (d_x, d_y).

No work-in-process. If the vehicle is not already serving a request ($j_m^{(1)} \neq 4$), then its first job is updated to type *preprocess* ($j_m^{(1)} = 3$) with origin $j_o^{(1)} = (v_x, v_y)$ and destination $j_d^{(1)} = (o_x, o_y)$. Its second job is then serving the customer, so we set the type to serve ($j_m^{(2)} = 4$), the origin $j_o^{(2)} = (o_x, o_y)$, and the destination $j_d^{(2)} = (d_x, d_y)$. The vehicle's third job is then empty, so we set $j_m^{(3)} = j_o^{(3)} = j_d^{(3)} = \emptyset$.

Work-in-process. If the vehicle is currently serving an existing customer ($j_m^{(1)} = 4$), then the first job is unchanged. The second job is *preprocess* ($j_m^{(2)} = 3$), as the vehicle moves from the drop-off location of its current customer ($j_o^{(2)} = j_d^{(1)}$) to the pick-up location of the new customer ($j_d^{(2)} = (o_x, o_y)$). The vehicle's third job is then to serve the customer request, so we set the type to serve ($j_m^{(3)} = 4$), the origin $j_o^{(3)} = (o_x, o_y)$, and the destination $j_d^{(2)} = (d_x, d_y)$.

B Agent Details

We provide here additional details about the training and implementation of the Dart and Drafter agents. Hyperparameters, chosen based on empirical results, are given in Table 2. We describe the attention mechanism used in Drafter's DQN in §B.1.

B.1 Attention Mechanism in Drafter

The attention mechanism used by the Drafter agent is depicted in Figure 1. We begin by creating an *embedding* g_v of each vehicle's representation x_v using a single dense layer called the "vehicle embedder." An embedding of some input is simply an alternative representation of that input. For example, for a vehicle representation $x_v \in [0,1]^2 \times \{0,1\}^{|\mathcal{L}|}$ (see §4.2.2) the embedder maps it to $g_v \in \mathbb{R}^{128}$ via $f(W_a x_v + \mathbf{b})$, where W_a is a $128 \times (2 + |\mathcal{L}|)$ matrix of trainable weights, $\mathbf{b}$ is a 128-length vector of trainable weights, and f is the activation function (here a rectified linear unit, *ReLU*). Similar to vehicle embeddings g_v, we create an embedding $h \in \mathbb{R}^{64}$ of the request x_r using another single dense layer (the

Table 2: Agents' hyperparameters.

Parameter	Dart	Drafter
Optimizer (learning rate)	Adam (0.001)	Adam (0.001)
Loss function	Huber	Huber
Discount factor γ	0.999	0.999
Memory capacity	1,000,000	1,000,000
Steps prior to learning M_{start}	2,000	2,000
Training frequency M_{freq}	100	100
Batch size M_{batch}	32	32
Initial epsilon ϵ_i	1	1
Final epsilon ϵ_f	0.01	0.1
Epsilon decay steps ϵ_N	75,000	75,000
PER type	None (uniform)	proportional
PER α	-	0.6
PER β_0	-	0.4
PER beta decay steps	-	600,000
Target network update frequency M_{update}	5,000	5,000
n-step learning	3	3
DQN activation functions	ReLU	ReLU*
Number of hidden layers (pre-dueling, advantage stack, value stack)	(1,2,2)	(2,2,2)**

* With the exception of the layers in the attention mechanism as described in §B.1.

** For the inner DQN (see Figure 1).

"request embedder"). We then concatenate each vehicle's embedding g_v with the request embedding h to produce joint embeddings $j_v = g_v \oplus h$. The j_vs and g_vs are then used to perform attention over the vehicles, creating a *fleet context* vector $C^F \in \mathbb{R}^{128}$ via

$$C^F = \sum_{v \in \mathcal{V}} \alpha_v g_v, \tag{18}$$

where $\alpha_v = \sigma(\mathbf{w} \cdot \tanh(W \cdot j_v))$ is a scalar weighting vehicle v, σ is the sigmoid activation function, $\mathbf{w}$ is a trainable vector of weights, and W is a trainable matrix of weights (the summation in equation (18) is performed element-wise). This attention mechanism is similar to the one used to produce the fleet context in Holler et al. (2019), with the notable difference that here we also incorporate the request. As the request s_r is likely to influence which vehicles are relevant in a given state, its inclusion in the attention mechanism should yield a more descriptive fleet context C^F. The fleet context vector is then used in the production of Q-values for each vehicle as described in §4.2.3.

C Froger et al. (2019) FRVCP Labeling Algorithm

We provide here additional details on the labeling algorithm from Froger et al. (2019) and our modifications to accommodate time constraints on the customers. We will refer to the sequence of requests to be served by vehicle v, as determined by the master problem (§5.2), by $r_v = (r_v^1, r_v^2, \ldots)$.

C.1 Algorithm Overview

To find the optimal recharging instructions given a request sequence r_v, the FRVCP is reformulated as a resource-constrained shortest path problem. The algorithm then works by setting labels at nodes on a graph $\mathcal{G}'_v$ (see example in Figure 12) that reflects the vehicle's assigned request sequence r_v and possible charging station visits. Labels are defined by *state-of-charge* (SoC) *functions*. SoC functions are piecewise-linear functions comprised of *supporting points* $z = (z^t, z^q)$ that describe a state of departure from a node in $\mathcal{G}'_v$ in terms of time z^t and battery level z^q. See Figure 13 for an example.

During the algorithm's execution, labels are extended along nodes in $\mathcal{G}'_v$. When extending labels, SoC functions are shifted by the travel time and energy between nodes, and supporting points resulting in infeasible (negative) charges are pruned. When extending a label to a charging station node, we create new supporting points that correspond to the *breakpoints* in the charging curve at that station – energy levels at which the charging rate changes. (Note that here, we assume linear charging until the vehicle reaches full battery, at which point it begins idling. This breakpoint corresponds to z_2 in the left graph of Figure 13.) We continue to extend labels along nodes in $\mathcal{G}'_v$ until either the destination node $r_v^{|r_v|}$ is reached or there are no feasible label extensions. If the destination node is reached, the algorithm returns the earliest arrival time to that node based on the label's SoC function and the sequence is deemed to be energy feasible; if the destination node cannot be reached, the sequence is deemed energy-infeasible and the algorithm returns the first request node to which it could not extend a label. Bounds on energy and time are established in pre-processing and are used alongside dominance rules during the algorithm's execution in order to improve its efficiency. For complete details on the algorithm, we refer the reader to Froger et al. (2019).

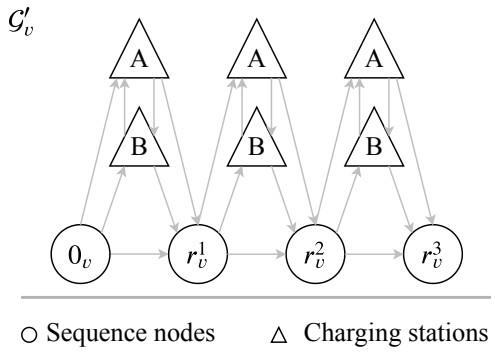


Figure 12: A depiction of the graph G'_v for a vehicle assigned to serve three customer requests. In the example, we assume there are two stations, A and B, at which the vehicle can recharge.

C.2 Algorithm Modifications

We modify the algorithm to accommodate time constraints for the requests in r_v, which are not considered in the original implementation. We assume that the vehicle is eligible to depart its initial location, denoted 0_v, at time t_v^p with charge q_v^p. Moving between locations a and b requires energy $q_{a,b}^s$ and time $t_{a,b}^s$ (if b is a request, the values reflect the time and energy to reach its destination via its origin). For this discussion, additional information about the algorithm beyond the overview in §C.1 may be necessary, for which we refer the reader to the description of Algorithm 3 in §5.3 and Appendix E of Froger et al. (2019).

The first modification serves to include the option of idling at charging stations. We add a point in the charging function at (∞, Q) as shown in Figure 13. This allows the vehicle to idle at a CS after it has fully charged its battery. Second, when extending a label to a request, we prune supporting points at the request node based on the request's time constraints. Specifically, all supporting points that are too early are shifted later in time. This may result in multiple supporting points with the same time but different energy levels, in which case we keep only the supporting point with maximum energy. Supporting points whose times are too late are eliminated, and a new supporting point is established where the SoC function intersects the end of the customer's time window. These processes are shown in Figure 13. Finally, when extending a label from one request node directly to another, if a supporting point has been shifted later in time by some amount Δt (like z_1'' in Figure 13), then the point incurs a charge penalty Δq. This reflects the constraint that vehicles may not idle at request nodes, so we assume that the vehicle has continued driving (e.g., "circling the block") during the time Δt and has thus drained its battery by the amount Δq. Note that such penalties are not incurred when extending to or from charging station nodes in $\mathcal{G}'_v$ as these nodes have no time constraints.

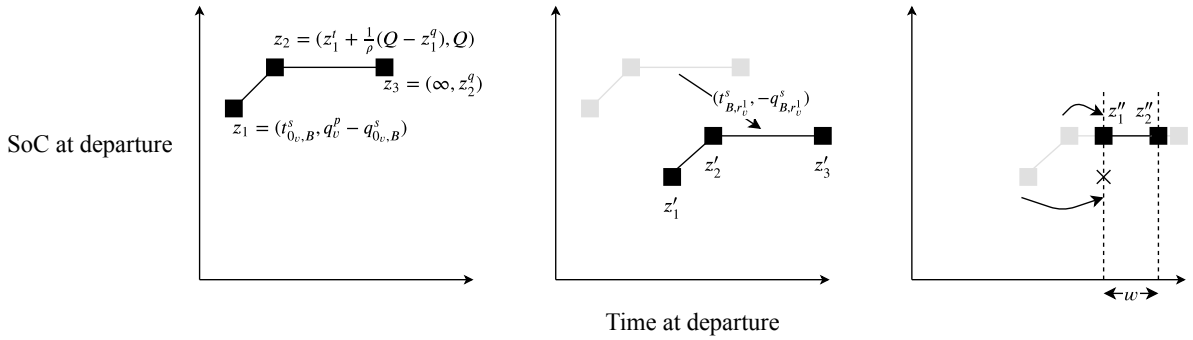


Figure 13: SoC functions for labels extended from 0_v to B to r_v^1. (Left) The SoC function at B after arriving directly from 0_v with supporting points corresponding to immediate departure without recharging (z_1), recharging completely (z_2), and subsequent indefinite idling (z_3). (Center) These points are then shifted by $(t^s_{B,r_v^1}, -q^s_{B,r_v^1})$ as we extend a label to r_v^1, resulting in points z'_1, z'_2, z'_3. (Right) Supporting points must lie within customer r_v^1's time window w. Point z'_1 is eliminated since it is dominated after this shift (indicated by the "x"), having less charge than the new point z''_1. Point z'_3 is eliminated and the new point z''_2 is created where the SoC function intersects the end of the time window. Note that there is no charge penalty incurred here, as the label is being extended from a charging station.

References

Lina Al-Kanj, Juliana Nascimento, and Warren B Powell. Approximate dynamic programming for planning a ride-sharing system using autonomous fleets of electric vehicles. *arXiv preprint arXiv:1810.08124*, 2018.

Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, 2017.

Omid Bahrami, Jim Gawron, Jack Kramer, and Bob Kraynak. Flexbus: Improving public transit with ride-hailing technology, December 2017. `http://sustainability.umich.edu/media/files/dow/Dow-Masters-Report-FlexBus.pdf`, Accessed December 2019.

Dimitris Bertsimas, Patrick Jaillet, and Sébastien Martin. Online vehicle routing: The edge of optimization in large-scale applications. *Operations Research*, 67(1):143–162, 2019.

Joschka Bischoff and Michal Maciejewski. Simulation of city-wide replacement of private cars with autonomous taxis in berlin. *Procedia computer science*, 83:237–244, 2016.

Anton Braverman, Jim G Dai, Xin Liu, and Lei Ying. Empty-car routing in ridesharing systems. *Operations Research*, 67(5):1437–1452, 2019.

T Donna Chen, Kara M Kockelman, and Josiah P Hanna. Operations of a shared, autonomous, electric vehicle fleet: Implications of vehicle & charging infrastructure decisions. *Transportation Research Part A: Policy and Practice*, 94:243–254, 2016.

Geoff Clarke and John W Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

Edison Electric Institute. Electric vehicle sales: Facts and figures, October 2019. `https://www.eei.org/issuesandpolicy/electrictransportation/Documents/FINAL_EV_Sales_Update_Oct2019.pdf`, Accessed January 2020.

Daniel J Fagnant and Kara Kockelman. Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. *Transportation Research Part A: Policy and Practice*, 77:167–181, 2015.

Daniel J Fagnant and Kara M Kockelman. The travel and environmental implications of shared autonomous vehicles, using agent-based model scenarios. *Transportation Research Part C: Emerging Technologies*, 40: 1–13, 2014.

Aurélien Froger, Jorge E Mendoza, Ola Jabali, and Gilbert Laporte. Improved formulations and algorithmic components for the electric vehicle routing problem with nonlinear charging functions. *Computers & Operations Research*, 104:256–294, 2019.

Michel Gendreau, Gilbert Laporte, and Jean-Yves Potvin. Metaheuristics for the capacitated vrp. In Paolo Toth and Daniele Vigo, editors, *The vehicle routing problem*, chapter 6, pages 129–154. SIAM, 2002.

H.V. Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pages 2094–2100. AAAI Press, 2016.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *32nd AAAI Conference on Artificial Intelligence*, pages 3207–3214. AAAI Press, 2018.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *32nd AAAI Conference on Artificial Intelligence*, pages 3215–3222. AAAI Press, 2018.

Sin C Ho, WY Szeto, Yong-Hong Kuo, Janny MY Leung, Matthew Petering, and Terence WH Tou. A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, 111:395–421, 2018.

John Holler, Risto Vuorio, Zhiwei Qin, Xiaocheng Tang, Yan Jiao, Tiancheng Jin, Satinder Singh, Chenxi Wang, and Jieping Ye. Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem. *arXiv preprint arXiv:1911.11260*, 2019.

Michael Hyland and Hani S Mahmassani. Dynamic autonomous vehicle fleet operations: Optimization-based strategies to assign avs to immediate traveler demand requests. *Transportation Research Part C: Emerging Technologies*, 92:278–297, 2018.

Riccardo Iacobucci, Benjamin McLellan, and Tetsuo Tezuka. Optimization of shared autonomous electric vehicles operations with charge scheduling and vehicle-to-grid. *Transportation Research Part C: Emerging Technologies*, 100:34–52, 2019.

Erick C Jones and Benjamin D Leibowicz. Contributions of shared autonomous vehicles to climate change mitigation. *Transportation Research Part D: Transport and Environment*, 72:279–298, 2019.

Jaeyoung Jung, R Jayakrishnan, and Keechoo Choi. Shared-taxi operations with electric vehicles. *Institute of Transportation Studies Working Paper Series, Irvine, Calif*, 2012.

Namwoo Kang, Fred M Feinberg, and Panos Y Papalambros. Autonomous electric vehicle sharing system design. *Journal of Mechanical Design*, 139(1):011402, 2017.

Andrej Karpathy. Deep reinforcement learning: Pong from pixels, May 2016. `http://karpathy.github.io/2016/05/31/rl/`, Accessed January 2020.

Charly Robinson La Rocca and Jean-François Cordeau. Heuristics for electric taxi fleet management at teo taxi. *INFOR: Information Systems and Operational Research*, 0(0):1–25, 2019.

Der-Horng Lee, Hao Wang, Ruey Long Cheu, and Siew Hoon Teo. Taxi dispatch system based on current demands and real-time traffic conditions. *Transportation Research Record*, 1882(1):193–200, 2004.

Minne Li, Zhiwei Qin, Yan Jiao, Yaodong Yang, Jun Wang, Chenxi Wang, Guobin Wu, and Jieping Ye. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. In *The World Wide Web Conference*, pages 983–994. ACM, 2019.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

Lyft. Why 130,000 lyft passengers were ready to ditch their personal cars in less than 24 hours, December 2018. `https://blog.lyft.com/posts/ditchyourcardata`, Accessed December 2019.

Michal Maciejewski, Joschka Bischoff, and Kai Nagel. An assignment-based approach to efficient real-time city-scale taxi dispatching. *IEEE Intelligent Systems*, 31(1):68–77, 2016.

Fei Miao, Shuo Han, Shan Lin, John A Stankovic, Desheng Zhang, Sirajum Munir, Hua Huang, Tian He, and George J Pappas. Taxi dispatch with real-time sensing data in metropolitan areas: A receding horizon control approach. *IEEE Transactions on Automation Science and Engineering*, 13(2):463–478, 2016.

Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent models of visual attention. In *Proceedings of the 27th International Conference on Neural Information Processing Systems-Volume 2*, pages 2204–2212. MIT Press, 2014.

Alejandro Montoya, Christelle Guéret, Jorge E Mendoza, and Juan G Villegas. The electric vehicle routing problem with nonlinear charging function. *Transportation Research Part B: Methodological*, 103:87–110, 2017.

National Renewable Energy Laboratory. Alternative fuels data center, 2019. Data retrieved 09/2019 from `https://www.nyserda.ny.gov/All-Programs/Programs/Drive-Clean-Rebate/Charging-Options/Electric-Vehicle-Station-Locator`.

New York City Taxi & Limousine Commission. Tlc trip record data, 2018. Data retrieved 09/2019 from `https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page`.

NYC OpenData. Nyc taxi zones, 2019. Data retrieved 09/2019 from `https://data.cityofnewyork.us/Transportation/NYC-Taxi-Zones/d3c5-ddgc`.

Takuma Oda and Carlee Joe-Wong. Movi: A model-free approach to dynamic fleet management. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 2708–2716. IEEE, 2018.

Takuma Oda and Yulia Tachibana. Distributed fleet control with maximum entropy deep reinforcement learning. In *NeurIPS 2018 Machine Learning for Intelligent Transportation Systems Workshop*, 2018.

Office of Transportation and Air Quality. U.S. Transportation Sector Greenhouse Gas Emissions 1990-2017, June 2019. URL `https://nepis.epa.gov/Exe/ZyPDF.cgi?Dockey=P100WUHR.pdf`. EPA-420-F-19-047.

Samuel Pelletier, Ola Jabali, and Gilbert Laporte. 50th anniversary invited article - goods distribution with electric vehicles: review and research perspectives. *Transportation Science*, 50(1):3–22, 2016.

Jacob F Pettit, Ruben Glatt, Jonathan R Donadee, and Brenden K Petersen. Increasing performance of electric vehicles in ride-hailing services using deep reinforcement learning. *arXiv preprint arXiv:1912.03408*, 2019.

Harilaos N Psaraftis, Min Wen, and Christos A Kontovas. Dynamic vehicle routing problems: Three decades and counting. *Networks*, 67(1):3–31, 2016.

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *4th International Conference on Learning Representations*, 2016.

Todd W. Schneider. Taxi and ridehailing usage in new york city, 2019. `https://toddwschneider.com/dashboards/nyc-taxi-ridehailing-uber-lyft-data/`, Accessed September 2019.

Kiam Tian Seow, Nam Hai Dang, and Der-Horng Lee. A collaborative multiagent taxi-dispatch system. *IEEE Transactions on Automation Science and Engineering*, 7(3):607–616, 2009.

Jie Shi, Yuanqi Gao, Wei Wang, Nanpeng Yu, and Petros A Ioannou. Operating electric vehicle fleet for ride-hailing services with reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

Ashutosh Singh, Abubakr Alabbasi, and Vaneet Aggarwal. A distributed model-free algorithm for multi-hop ride-sharing using deep reinforcement learning. In *NeurIPS 2019 Machine Learning for Autonomous Driving Workshop*, 2019.

Peter Slowik, Lina Fedirko, and Nic Lutsey. Assessing ride-hailing company commitments to electrification, February 2019. `https://theicct.org/sites/default/files/publications/EV_Ridehailing_Commitment_20190220.pdf`, Accessed January 2020.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press, 2018. ISBN 9780262352703.

Tesla. Supercharging cities, 2017. `https://www.tesla.com/blog/supercharging-cities`, Accessed December 2019.

Z. Wang, T. Schaul, M. Hessel, H.V. Hasselt, M. Lanctot, and N.D. Freitas. Dueling network architectures for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 1995–2003, New York, New York, USA, 20–22 Jun 2016. PMLR.

Zhe Xu, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, and Jieping Ye. Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 905–913. ACM, 2018.

Lingyu Zhang, Tao Hu, Yue Min, Guobin Wu, Junying Zhang, Pengcheng Feng, Pinghua Gong, and Jieping Ye. A taxi order dispatch model based on combinatorial optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2151–2159. ACM, 2017.

Rick Zhang and Marco Pavone. Control of robotic mobility-on-demand systems: a queueing-theoretical perspective. *The International Journal of Robotics Research*, 35(1-3):186–203, 2016.