2024 European Control Conference (ECC)
June 25-28, 2024. Stockholm, Sweden

Real-time Control of Electric Autonomous Mobility-on-Demand Systems via Graph Reinforcement Learning

Aaryan Singhal[1], Daniele Gammelli[1], Justin Luke[1], Karthik Gopalakrishnan[1], Dominik Helmreich[2] and Marco Pavone[1]

Abstract— **Operators of Electric Autonomous Mobility-on-Demand (E-AMoD) fleets need to make several real-time decisions such as matching available cars to ride requests, rebalancing idle cars to areas of high demand, and charging vehicles to ensure sufficient range. While this problem can be posed as a linear program that optimizes flows over a space-charge-time graph, the size of the resulting optimization problem does not allow for real-time implementation in realistic settings. In this work, we present the E-AMoD control problem through the lens of reinforcement learning and propose a graph network-based framework to achieve drastically improved scalability and superior performance over heuristics[†]. Specifically, we adopt a bi-level formulation where we (1) leverage a graph network-based RL agent to specify a *desired next state* in the space-charge graph, and (2) solve more tractable linear programs to best achieve the desired state while ensuring feasibility. Experiments using real-world data from San Francisco and New York City show that our approach achieves up to 89% of the profits of the theoretically-optimal solution while achieving more than a 100x speedup in computational time. We further highlight promising zero-shot transfer capabilities of our learned policy on tasks such as inter-city generalization and service area expansion, thus showing the utility, scalability, and flexibility of our framework. Finally, our approach outperforms the best domain-specific heuristics with comparable runtimes, with an increase in profits by up to 3.2x.**

I. INTRODUCTION

Electric Autonomous Mobility-on-Demand (E-AMoD) systems use electric autonomous vehicles to provide on-demand ride-hailing services for customers. Operating an E-AMoD fleet involves three operations: matching available cars to customers who request rides, rebalancing idle cars to regions with high demand, and assigning cars to charging stations. In realistic settings, E-AMoD fleets can be centrally controlled and the operator can coordinate the assignment of vehicles to each of these three tasks to maximize efficiency, demand satisfaction, and profits.

It is worth emphasizing that these decisions need to be made in real time, and any offline schedule, even if computed at the start of a day using historical or predicted data, will generally be suboptimal [1] due to forecasting errors in traffic demand, vehicle energy consumption, and road congestion [2]. Thus, we seek an approach to solve the E-AMoD control problem in real-time, so that we may be able to compute updated fleet coordination decisions whenever new information is available to the operator.

[1] Stanford University, USA {aaryan04, gammelli, jthluke, gkarthik, pavone}@stanford.edu
[2] Work done while at ETH Zurich, Switzerland hedomini@student.ethz.ch
[†] The project's website can be found at: https://github.com/StanfordASL/graph-rl-for-eamod

Among other approaches, Model Predictive Control (MPC) provides a framework to make decisions in a receding horizon fashion by repeatedly solving an optimization problem based on (i) the current state of the system and (ii) a forecast of future state elements. In the context of autonomous mobility-on-demand (AMoD) systems, receding-horizon control has been used extensively to make optimal rebalancing decisions [3]. Specifically, prior work has shown that a network flow model for this problem can be successfully scaled for real-time large-scale operations [4]. However, this approach assumes that all vehicles are *indistinguishable* from each other, thereby enabling the operator to aggregate all vehicles and model their movements as a flow on a network. Electric vehicles on the other hand are *distinguishable* based on their current state of charge, which determines their maximum range. Motivated by this, Estandia et al. [5] use an augmented network flow formulation for E-AMoD systems, by including a charge dimension in the resulting graph (i.e., from a space-time graph to a space-*charge*-time graph). However, the computational complexity of the resultant optimization problem does not allow for real-time implementation, and previous work has struggled to devise effective real-time controllers even for coarse representations of space, charge, and time. For instance, Estandia et al. [5] report that solving the optimal control problem takes 42 minutes for an E-AMoD system operating across Orange County, USA over an eight-hour horizon.

In this paper, we propose a strategy to design real-time controllers for E-AMoD systems through reinforcement learning. To do so, we present the E-AMoD control problem through the lens of graph reinforcement learning (graph-RL) [6] and exploit the main strengths of graph neural networks, reinforcement learning, and classical operations research tools.

A. Related works

Existing literature on the coordination of E-AMoD systems heavily relies on solving large-scale optimization problems. Specifically, prior works approach the problem of joint optimization of charging station siting [7], joint optimization of power flows [5], and the computation of optimal rebalancing plans [8]. However, in practice, the adoption of these methods is typically limited by their computational complexity and is only considered in an offline setting, thus not immediately applicable within real-time operations. To improve the scalability of control algorithms for E-AMoD systems, previous works adopt several learning-based techniques. For example, Wan et al. [9] consider charge scheduling for personal electric vehicles. Bogyrbayeva et al. [10] optimize nightly rebalancing operations of electric vehicles to charging stations, while Shi et al. [11] propose a decentralized algorithm for the charging-constrained vehicle routing problem. Overall, although the aforementioned works cover a wide range of algorithms, there lacks a framework to deal with the joint computation of both spatial rebalancing and charging decisions within large-scale E-AMoD systems, which is a key factor in making the optimization problem prohibitively expensive and a main focus of this work.

Reinforcement learning has also been extensively used to learn fleet coordination policies that do not account for charging (i.e., AMoD systems). For example, Gueriau et al. [12] developed RL-based decentralized approaches where the action of each vehicle is determined independently through a Q-learning policy, while Holler et al. [13] developed a cooperative multi-agent approach for order dispatching and vehicle rebalancing using Deep Q-Networks and Proximal Policy Optimization. Of particular relevance to our work are methods that (i) leverage the graph structure of the underlying transportation system, and (ii) combine principled control strategies with learned components in a hierarchical way [14]–[16]. In this work, we leverage the framework of graph-RL to include charging within the range of autonomous, real-time decisions, thus substantially increasing the set of application areas.

B. Our contributions

Contribution #1. We present the first reinforcement learning agent that jointly learns the charging and rebalancing decisions for an E-AMoD fleet. This is enabled by two key design choices. First, we leverage the power of graph neural networks to capture both spatial and charge information across the system. This is critical in devising RL agents that can propagate information between different regions of the transportation network before computing a centralized decision for the entire fleet. Second, we extend the framework presented by Gammelli et al. [14] to the E-AMoD setting and develop an approach that leverages the specific strengths of direct optimization and reinforcement learning through a hierarchical formulation, which is advantageous in learning a policy that is more effective, scalable, and generalizable.

Contribution #2. We provide numerical experiments that demonstrate how our approach is highly performant, scalable, and robust to changes in operating conditions. In particular, through extensive comparisons with both classical optimization-based approaches and domain-specific heuristics, results highlight how our approach achieves close-to-optimal performance with drastic runtime improvements.

Contribution #3. This work highlights how policies learned through graph-RL exhibit a series of desirable properties of fundamental practical importance for any system operator. Specifically, results show interesting transfer performance of a trained agent in the context of (i) inter-city generalization (i.e., the agent is trained on one city and directly applied to another), and (ii) service area expansion (i.e., the agent is trained on a specific sub-graph and directly applied to additional areas of the city). We further show how the transfer capabilities achieved by agents learned through graph-RL are crucial in enabling learning within large-scale instances.

II. BACKGROUND

In this section, we introduce key terminology and notation in the context of reinforcement learning (Section II-A) and graph neural networks for network control (Section II-B).

A. Reinforcement Learning

We refer to a Markov decision process (MDP) as a tuple $\mathcal{M} = (\mathcal{S},\mathcal{A},P,r,\gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, P describes the dynamics of the system through the conditional probability distribution $P(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$, $r: \mathcal{S}\times\mathcal{A}\rightarrow\mathbb{R}$ defines a reward function, and $\gamma\in(0,1]$ is a scalar discount factor. From a reinforcement learning perspective, the final goal is to learn a policy defining a distribution over possible actions given states, $\pi(\mathbf{a}_t|\mathbf{s}_t)$ by maximizing the expected cumulative reward $J(\pi) = \mathbb{E}_{\tau\sim p_\pi(\tau)}\left[\sum_{t=0}^{H}\gamma^t r(\mathbf{s}_t,\mathbf{a}_t)\right]$, where the expectation is computed under the distribution over trajectories $p(\tau)$ induced by policy $\pi(\mathbf{a}_t|\mathbf{s}_t)$ and system dynamics $P(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$.

B. Graph Neural Networks for Network Control

Given a graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$, where $\mathcal{V} = \{v_i\}_{i=1:N_v}$ and $\mathcal{E} = \{e_k\}_{k=1:N_e}$ respectively define the sets of nodes and edges of $\mathcal{G}$, most current graph neural network models can be seen as methods attempting to learn a function taking as input (i) a D-dimensional feature description $\mathbf{x}_i$ for every node i (typically summarized in a $N_v \times D$ feature matrix $\mathbf{X}$), (ii) a representative description of the graph structure in matrix form $\mathbf{A}$ (typically in the form of an adjacency matrix), and produce an updated representation $\mathbf{x}'_i$ for all nodes in the graph. As observed in [6], graph neural networks (GNN) represent an extremely advantageous choice within network optimization problems, for three main reasons. First, GNNs are permutation invariant operators[1]. This is particularly relevant in the context of graphs, where nodes do not have a natural ordering and where non-permutation invariant computations would consider each ordering as fundamentally different, and thus have been shown to require an exponential number of input/output training examples before being able to generalize. Second, GNNs are local operators. This enables the same neural network architecture to be applied to graphs of different sizes. Third, GNNs align with the type of computations required within network optimization problems, which has been shown to lead to better performance and increased data efficiency.

III. THE E-AMOD CONTROL PROBLEM

In this section, we introduce the charge-expanded network flow model characterizing the E-AMoD control problem. Towards this aim, we partition the region of operation for an E-AMoD fleet (e.g., a city) into a set of discrete non-overlapping regions denoted $\mathcal{A}$. The time horizon is discretized into a set of discrete intervals $\mathcal{T} = \{1,2,\cdots,T\}$ of a given length ΔT. We consider a set of equally spaced discrete charge levels for each vehicle denoted by $\mathcal{C} = \{1,...,C\}$, where C is the highest charge level. When a car travels from region i to region j, it takes l_{ij} time steps and loses η_{ij} units of charge. While charging, a car moves up t_c discrete charge levels per time step. We assume that our E-AMoD fleet has a fixed set of N autonomous electric cars and that every region $a\in\mathcal{A}$ has a charging station with a finite number of charging plugs, denoted as N_c^a, with $N_c^a > 0$, which is reasonable for most E-AMoD settings. We denote customer demand from region $i\in\mathcal{A}$ to $j\in\mathcal{A}$ at time $t\in\mathcal{T}$ as d_{ij}^t and define the arrival process of passengers for each

[1]We will refer to a computation as permutation invariant if its output is independent of the ordering of its inputs.

origin-destination pair as a time-dependent Poisson process that is independent of the arrival processes of other origin-destination pairs and the coordination of E-AMoD fleets [17]. We denote the total demand from region $i \in \mathcal{A}$ at time t as $d_i^t = \sum_{j\in\mathcal{A}} d_{ij}^t$. Note that customers might request a ride within the same spatial region, i.e., d_{ii}^t can be non-zero. We assume that a customer that is not assigned a ride within one time step will leave the system.

We denote the cost for increasing the charge level of a vehicle by one discrete level at time t to be p_e^t, assumed to be known. We denote the cost of operating a vehicle from region $i \in \mathcal{A}$ to region $j \in \mathcal{A}$ as o_{ij}^t. This cost is a function of the distance traveled, captures the amortized cost of maintenance, and is assumed to be given. Finally, the revenue for the operator generated by serving a passenger traveling from region i to j at time t is denoted by ρ_{ij}^t.

A. The Space-Charge Graph

Having formally defined the relevant notation and assumptions used in this work, this section describes the graph structure characterizing the E-AMoD network flow problem (Figure 1). In this graph, nodes represent a *(spatial region, charge level)* tuple and are used to capture the state of a vehicle. Multiple vehicles may have the same spatial region and charge level. Over time, vehicles transition from one state to another as they satisfy passenger demands, perform spatial rebalancing, or get recharged. At any time, vehicles can transition from one node (i.e., spatial region and charge state) to another through edges that capture valid transitions. Formally, we denote the graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \mathcal{A} \times \mathcal{C}$ is the set of nodes and $\mathcal{E} = \{(i,j) \subset \mathcal{V} \times \mathcal{V}\}$ is the set of edges. We denote an edge from node i to node j as e_{ij}. The graph $\mathcal{G}$ is characterized by two types of edges – those representing the physical movement of vehicles from one region to another, denoted by $\mathcal{E}_{road}$, and those representing the charging of vehicles, denoted by $\mathcal{E}_{charge}$. Thus $\mathcal{E} = \mathcal{E}_{charge} \cup \mathcal{E}_{road}$. For any node $i \in \mathcal{V}$, we denote the corresponding spatial region and charge level as i_r and i_c, respectively. Using this notation, we define $\mathcal{E}_{charge} = \{e_{ij} | i_r = j_r$ and $(j_c - i_c) = kt_c$ for some $k \in \mathbb{N}\}$. This set represents all possible transitions that can happen as a result of charging. Similarly, we define the edges corresponding to roads as $\mathcal{E}_{road} = \{e_{ij} | j_c = i_c - \eta_{ij}\}$. For every edge $e = (i,j) \in \mathcal{E}$ on the graph, we associate a time-variant cost c_{ij}^t to traverse it. For road edges, $c_{ij}^t = o_{i_r, j_r}$, while for a charging edge $e = (i,j) \in \mathcal{E}_{charge}$, the cost is $c_{ij}^t = (j_c - i_c) p_e^t$. Lastly, we define the travel time for all road and charging edges as τ_{ij} as $\tau_{ij} = l_{i_r j_r}$ and $\tau_{ij} = \lceil \frac{(j_c - i_c)}{t_c} \rceil$, respectively.

B. The Optimal Control Problem

The E-AMoD control problem is naturally posed as a network flow problem. Formally, let x_{ij}^t be the number of passengers who started traveling from $i \in \mathcal{V}$ to $j \in \mathcal{V}$ at time $t \in \mathcal{T}$, and let y_{ij}^t be the number of vehicles that started rebalancing or charging at time t from node i to node j. The objective of the E-AMoD control problem is to maximize the profits over a pre-specified time horizon T, and is defined as follows:

$$\max \sum_{t=1}^{T} \sum_{(i,j): e_{ij} \in \mathcal{E}} [(\rho_{ij}^t - c_{ij}^t) x_{ij}^t - c_{ij}^t y_{ij}^t] \tag{1a}$$

s.t.

$$\sum_{(i,j)\in\mathcal{E}_{road}: i_r = u, j_r = v} x_{ij}^t \le d_{uv}^t \quad \forall i,j \in \mathcal{V}, t \in \mathcal{T} \tag{1b}$$

$$\sum_{i\in\mathcal{V}} (x_{ij}^{t-\tau_{ij}} + y_{ij}^{t-\tau_{ij}}) = \sum_{k\in\mathcal{V}} (x_{jk}^t + y_{jk}^t) \forall t \in \mathcal{T}, j \in \mathcal{V} \tag{1c}$$

$$\sum_{(i,j): i_r = j_r = a} \sum_{k=0}^{\tau_{ij}} y_{ij}^{t-k} \le N_a^t \quad \forall a \in \mathcal{A}, t \in \mathcal{T} \tag{1d}$$

$$x_{ii}^0 = x_i^{\text{init}}, y_{ii}^0 = 0 \quad \forall i \in \mathcal{V} \tag{1e}$$

$$x_{ij}^t \ge 0, y_{ij}^t \ge 0, \tag{1f}$$

where, the objective term (1a) represents the total profit, constraint (1b) ensures that passenger flow does not exceed demand, (1c) enforces flow conservation, (1d) ensures that the number of vehicles charging at any point in time does not exceed the capacity of that station, and (1e) and (1f) set the initial conditions and ensure non-negativity of the decision variables, respectively.

Note that the objective in (1a) involves $|\mathcal{E}| \times T$ decision variables. Since, the number of edges $|\mathcal{E}| = O(|\mathcal{V}|^2)$, and $\mathcal{V} = \mathcal{A} \times \mathcal{C}$, the number of decision variables in the optimization problem described by Eqn. 1a is $O(|\mathcal{A}|^2 |\mathcal{C}|^2 T)$. Crucially, the rapid growth of the underlying optimization problem with respect to the spatial resolution, charge levels, and planning horizon T results in poor computation performance for real-time applications, as observed in [5].

Our goal is to reduce the complexity of problem (1a)-(1f) to enable real-time control. To achieve this, we formulate the problem in Eqn. (1a) as a sequential decision-making problem [14], [18]. Our hypothesis is that we can express effective E-AMoD policies as a composition of policies: a higher-level policy trained through RL to maximize long-term reward, and a lower-level local approximation of the problem (1a)-(1f) to compute feasible, fleet-wide decisions. This local correspondence is central to our formulation: we exploit exact optimization when it is useful, and otherwise push the complexities of optimizing for long-term performance to the learned policy. In the next section, we will present our approach via a tri-level framework, and discuss specific details about the design of our RL agent.

IV. GRAPH-RL FOR E-AMoD CONTROL

In this section, we introduce the proposed graph-RL framework for E-AMoD systems. Specifically, we first describe a tri-level formulation used to approximate the problem (1a)-(1f) (Section IV-A) together with the definition of a Markov Decision Process (MDP) within this formulation (Section IV-B). Lastly, we describe the proposed graph-RL agent (Section IV-C).

A. The Tri-level Framework

Similarly to [18] and [14], we approximate the problem in Eqn. (1a) through the following three-step decomposition: (i) derive passenger flows x_{ij}^t by solving a matching problem, (ii) compute a desired distribution of idle vehicles across spatial and charge dimensions through reinforcement learning, and (iii) solve a minimal cost rebalancing problem to compute the desired flows y_{ij}^t that would better achieve the desired distribution from the previous step (Figure 1). Notice that solving this approximation drastically reduces the number of decision variables in the optimization problem. Specifically, rather than solving a single planning problem with $|\mathcal{E}| \times T$ decision variables, we now solve

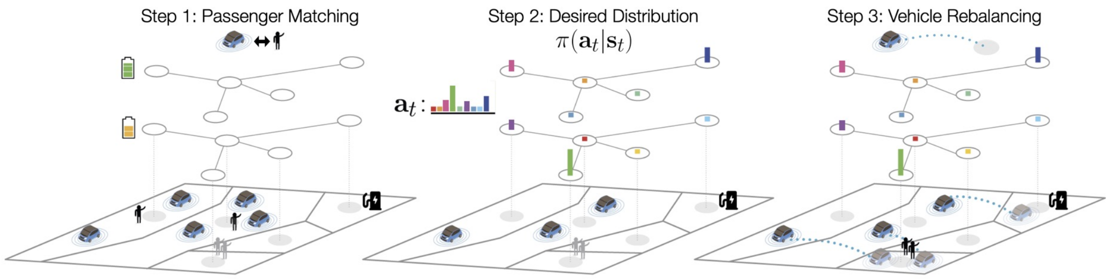
Step 1: Passenger Matching
Step 2: Desired Distribution
$\pi(\mathbf{a}_t|\mathbf{s}_t)$
$\mathbf{a}_t$:
Step 3: Vehicle Rebalancing

Fig. 1. A visual representation of the tri-level framework for a given time step t. Step 1 (left) involves matching ride requests to cars. Step 2 (center) uses the policy learned through RL to compute an ideal redistribution of cars over the space-charge graph $\mathcal{G}$, i.e., $\mathbf{a}_t$. Step 3 (right) computes the spatial rebalancing and charging flows to achieve (as best as possible) the target distribution given by $\mathbf{a}_t$.

three simpler problems that do not scale with the length of the planning horizon T, i.e., characterized by $|\mathcal{E}|$, $|\mathcal{V}|$, and $|\mathcal{E}|$ variables, respectively. Crucially, we propose RL as an appealing learning paradigm to compensate for the lack of explicit planning caused by the three-step approximation and learn a control policy that optimizes long-term reward; in particular, we extend the framework proposed in [14] to handle *space-charge* graphs (as opposed to *space-only* graphs), developing RL-based approaches for hierarchical, large-scale network topologies. In what follows, we introduce the three-step framework in more detail.

Step 1: Passenger Matching. The first step is passenger matching, wherein the following matching problem is solved to derive passenger flows:

$$\max_{x_{ij}^t \geq 0} \sum_{(i,j):e_{ij}\in\mathcal{E}_{road}} (\rho_{ij}^t - c_{ij}^t) x_{ij}^t \tag{2a}$$

$$\text{s.t.} \sum_{j:e_{ij}\in\mathcal{E}_{road}} x_{ij}^t \leq n_{init}^t[i] \quad \forall i \in \mathcal{V} \tag{2b}$$

$$\sum_{\substack{(i,j):e_{ij}\in\mathcal{E}_{road}\\ i_r=a_1, j_r=a_2}} x_{ij}^t \leq d_{a_1 a_2}^t \quad \forall a_1, a_2 \in \mathcal{A}, \tag{2c}$$

where $n_{init}^t[i]$ defines the number of idle vehicles in node i at time t before matching, constraint (2a) denotes the difference between the revenue and cost of traversing all the edges, (2b) limits the maximum flow from each node to the number of available vehicles, and (2c) ensures that the passenger flow between any two nodes does not exceed demand. Notice that since the constraint matrix is totally unimodular, the resulting passenger flows are positive integers, i.e., $x_{ij}^t \in \mathbb{Z}_+$ if $d_{ij}^t \in \mathbb{Z}_+, \forall i,j \in \mathcal{V}$. .

Step 2: Desired Distribution. The second step entails determining the desired number of idle vehicles $n_{target}^t[i]$ at each node. Let us denote the number of idle vehicles at node $i \in \mathcal{V}$, after the matching step as $n_{idle}^t[i]$, i.e., $n_{idle}^t[i] = n_{init}^t[i] - \sum_{j\in\mathcal{V}} x_{ij}^t$. In this work, we compute $n_{target}^t[i]$ in two steps. First, we determine a desired *distribution* of vehicles (i.e., the action for the RL agent) $\mathbf{a}_t = \{a_t[i]\}_{i\in\mathcal{V}}$, where $a_t[i] \in [0,1]$ defines the percentage of currently available vehicles to be moved to node i at time t, and $\sum_{i\in\mathcal{V}} a_t[i] = 1$. Second, we use the desired distribution to compute $n_{target}^t[i] = \lfloor a_t[i] \sum_{i\in\mathcal{V}} n_{idle}^t[i] \rfloor$ as the actual number of vehicles at each node i. Here, the floor function $\lfloor \cdot \rfloor$ ensures that the desired number of vehicles is integer and always available. It is important to highlight how this action representation is scale-invariant by construction, as it acts on *ratios* as opposed to raw vehicle counts: a strategy that has been shown to lead to increased learning stability and better generalization [14]. Crucially, the goal of our formulation is to use reinforcement learning to learn a policy over desired distributions $\mathbf{a}_t$ that is capable of steering the myopic behavior of steps 1 and 3 towards long-term optimality.

Step 3: Vehicle Rebalancing. Henceforth, we refer to "rebalancing" as both spatial rebalancing and charge rebalancing (i.e., charging) for brevity. In this third step, we use a linear program to compute the rebalancing flows y_{ij}^t that (i) achieve the desired distribution from step 2 in the minimum cost, and (ii) satisfy domain-specific constraints. Specifically, this is achieved by solving the following problem:

$$\min_{\substack{y_{ij}^t \geq 0\\ s_v \geq 0}} \sum_{(i,j):e_{ij}\in\mathcal{E}} c_{ij}^t y_{ij}^t + M \sum_{v\in\mathcal{V}} |s_v| \tag{3a}$$

$$s.t. \sum_{j\in\mathcal{V}} y_{ij}^t \leq n_{idle}^t[i] \quad , \forall i \in \mathcal{V} \tag{3b}$$

$$\sum_{i\in\mathcal{V}} (y_{ij}^t - y_{ji}^t) + s_j = n_{target}^t[j] - n_{idle}^t[j] \quad , \forall j \in \mathcal{V}, \tag{3c}$$

where Eqn. (3a) represents the rebalancing cost plus a penalty for deviations from the desired distribution, with s_v defined as a slack variable for vehicle deviation and M as a large penalty factor, constraint (3b) limits the rebalancing flows from a region to the vehicles available in that region, and (3c) ensures that the resulting number of vehicles (the left-hand side) is close to the desired number of vehicles (the right-hand side).

B. The E-AMoD Markov Decision Process

In this section, we formulate the E-AMoD control problem as an MDP. In what follows, we define the elements characterizing the MDP for the E-AMoD control problem.

State space. We define the state of the system $\mathbf{s}_t$ to capture relevant information required to express effective fleet control strategies. To do so, we define the state representation to encode information about (i) the topology of the space-charge network through an adjacency matrix $\mathbf{A}$, and (ii) local information about each node in the network through a feature matrix $\mathbf{X}$. On one hand, the topology of the space-charge network is fully characterized by the adjacency matrix $\mathbf{A}$ of graph $\mathcal{G}$, as introduced in Section III-A. On the other, we choose the feature matrix $\mathbf{X}$ to be defined by three main sources of information. Firstly, we characterize the state of the E-AMoD system by the current and *projected* number of idle vehicles across all nodes, $n_{idle}^t[i]$ for $t = t,...,t = t+K, \forall i \in \mathcal{V}$. Note that the projected number of idle vehicles is readily estimated given past matching and rebalancing actions (i.e., x_{ij}^k and y_{ij}^k for all $k < t$), as well as travel times τ_{ij}. Secondly, profit-maximizing control policies will necessarily depend on information regarding the potential revenue that can be obtained across different regions. To do so, we express the potential revenue across all regions i over the next K time steps as the sum of all revenues from estimated trips originating from that region: $\left(\sum_{j\in\mathcal{A}} \hat{d}_{ij}^{t+1} \hat{\rho}_{ij}^{t+1},...,\sum_{j\in\mathcal{A}} \hat{d}_{ij}^{t+K} \hat{\rho}_{ij}^{t+K}\right)$, where $\hat{d}_{ij}^{t+1}$ and $\hat{\rho}_{ij}^{t+1}$ are the estimated demand and trip revenue between regions i and j. Finally, to enable proactive charging policies, we distinguish between nodes at different charge levels through the fractional charge level $\frac{i_c}{|\mathcal{C}|}$ of the node.

Action space. In this work, we consider the problem of learning a *desired distribution of idle vehicles* across all nodes in the graph $\mathbf{a}_t \in \mathbb{R}_{\geq 0}^{|\mathcal{V}|}$. Specifically, we define the action $\mathbf{a}_t$ to describe a probability distribution over vehicle charge level and location.

Reward. We define the reward function for the MDP such that the RL agent learns actions that maximize the global objective described in Equation (1a). To do so, the instantaneous reward should reflect the revenue from passenger trips as well as the rebalancing costs associated with fleet management. Specifically, given trip revenues and costs (i.e., ρ_{ij}^t and c_{ij}^t) together with the number of passenger and rebalancing trips (i.e., x_{ij}^t and y_{ij}^t) we define the reward as:

$$r(\mathbf{s}_t,\mathbf{a}_t) = \sum_{i,j} \left[(\rho_{ij}^t - c_{ij}^t) x_{ij}^t - c_{ij}^t y_{ij}^t \right]. \tag{4}$$

Dynamics. The dynamics characterizing the E-AMoD MDP describe both the stochastic evolution of the system, as well as how fleet management decisions influence future state elements, such as the availability and distribution of idle vehicles. Specifically, we assume the evolution of travel demand between regions d_{ij}^t to be independent of the operator decisions and follows a time-dependent Poisson process (in our experiments, estimated from real trip travel data). On the other hand, some of the state variable's transitions deterministically depend on the chosen action. For example, the projected availability $n_{idle}^{t'}[i], \forall i \in \mathcal{V}, t' > t$ is uniquely defined as the sum of the current availability $n_{idle}^t[i]$ together with the projected number of incoming vehicles at time t' (from both passenger and rebalancing trips), minus the vehicles currently chosen to be rebalanced. Finally, state variables related to provider information, such as trip price ρ_{ij}^t and cost c_{ij}^t are assumed to be exogenous and known (hence, independent from the actions selected by the operator).

C. Graph-RL Agent

After having introduced the E-AMoD control problem and the related MDP formulation, we now formally describe the graph network-based architecture characterizing the proposed RL agent. In this work, we learn E-AMoD control policies through the Soft-Actor-Critic (SAC) [19] algorithm and define the following architectures for policy (i.e., the actor) and value function estimator (i.e., the critic).

Actor. As described in Section IV-B, the goal of the policy network is to learn a mapping from the current state of the system $\mathbf{s}_t$ to a desired distribution of idle vehicles $\mathbf{a}_t$. We define $\pi(\mathbf{a}_t|\mathbf{s}_t)$ as a Dirichlet distribution over nodes in the graph (i.e., $\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t) = \text{Dir}(\mathbf{a}_t|\alpha_t)$), with the policy network parametrizing the concentration parameters $\alpha_t \in \mathbb{R}_+^{|\mathcal{V}|}$. The neural network used in our implementation consists of one layer of graph convolution network [20] with skip-connections and ReLU activations, whose output is then aggregated across neighboring nodes using a sum-pooling function, and finally passed to three MLP layers of 32 hidden units to produce the Dirichlet concentration parameters.

Critic. The architecture of the critic mostly overlaps with the one used to define the policy network. The main difference between the two architectures lies in an additional *global* sum-pooling performed on the output of the graph convolution to compute a single value function estimate for the entire network, i.e., opposed to the actor that computes an output for every node.

V. Experiments

In this section, we present simulation results that demonstrate the performance of our proposed approach. Specifically, the goal of our experiments is to answer the following questions: (1) Can the proposed graph-RL framework learn effective fleet management strategies in real-world urban mobility scenarios? (2) Computationally, what are the advantages of graph-RL approaches compared to traditional optimization-based strategies and domain-specific heuristics? (3) What are the generalization capabilities of behavior policies learned through our approach?

A. Simulation Environment and Baselines

We model an E-AMoD system serving passenger travel in San Francisco and New York City for 12 hours, from 8am-8pm. In San Francisco, travel demand d_{ij}^t and travel times l_{ij} is based on origin-destination travel data for all passenger travel for an average weekday in 2019, provided by StreetLight Data[2]. We use this data to calibrate a Poisson process of travel demand, which we use to generate unseen (but realistic) demand patterns for both training and test scenarios. In New York City, travel demand and travel times are based on High Volume For-Hire Vehicles origin-destination data provided by the New York City Taxi and Limousine Commission[3]. Each spatial region is approximately 6 traffic analysis zones (TAZ). In each experiment, we assume fleet size is 20% of the peak total travel demand. Additionally, we assume there are a total number of 50 kW charging stations equal to 20% of the fleet size, which are distributed uniformly across all spatial regions to determine N_c^a. We model the fleet vehicle based on the Chevrolet Bolt EV, with 65 kWh and an energy consumption of 0.4037 kWh/mi that includes a 30% de-rating due to charging losses and autonomous vehicle auxiliary loads, which we use to determine η_{ij}. Reserving 40% of the vehicle battery capacity for operational uncertainty in energy consumption, and setting the charge level step size to be 2 kWh, we result in $C = 19$ charge levels. Setting $\Delta T = 15$ minutes, we have $t_c = 6$ charge levels. The electricity price p_e^t is based from Pacific Gas & Electric's Business Electric Vehicle[4] time-of-use energy rates in 2022, which promotes charging when

[2]https://www.streetlightdata.com/

[3]https://www.nyc.gov/site/tlc/index.page

[4]https://www.pge.com/tariffs/assets/pdf/tariffbook/ELEC_SCHEDS_BEV.pdf

solar generation is most plentiful. Its rates are 0.16872 \$/kWh from 8am-9am and 2pm-4pm, 0.14545 \$/kWh from 9am-2pm, and peak price of 0.38195 \$/kWh from 4pm-8pm. The amortized cost of maintenance o_{i_r,j_r} is calculated using 0.077 \$/mi from the American Automobile Association[5]. The revenue from serving passengers ρ_{ij}^t is based on Lyft pricing for the San Francisco Bay Area[6] in 2022, with a base fare and service fee of \$4.90, price per mile of 0.90 \$/mi, and price per minute of 0.39 \$/min.

In our experiments, we compare the proposed graph-RL framework with heuristic and MPC-based methods. All three approaches are targeting to solve the E-AMoD control problem within a real-time constraint of 10 seconds [21], which may be demanded by a lower-level vehicle dispatch algorithm. The heuristic and MPC-based methods are implemented as follows:

Heuristics. We focus on measuring the performance of realistic, domain-specific fleet management heuristics. This class of methods also adopts the tri-level framework outlined in IV-A, but determines the desired distribution in Step 2 heuristically.

First, vehicles are recharged using one of the following methods:

1) *Empty-to-full*: all vehicles that reach a charge level below the average trip's energy consumption are recharged to full.
2) *Off-peak Absolute*: when electricity price is not at its peak, all vehicles below 30% charge level are recharged for one time step (i.e., recharge t_c charge levels). During peak price, vehicles with charge level below the average trip's energy consumption are recharged for one time step.
3) *Off-peak Relative*: equivalent to Off-peak Absolute except that during off-peaks the lowest 30% of vehicles by charge level in each region are recharged.

Second, idle vehicles are spatially rebalanced to uniformly distribute them across all spatial regions. These benchmarks provide a measure of performance for methods that are computationally feasible and simplest to implement by real-world operators.

MPC-based. Within this class of methods, we focus on measuring performance of MPC approaches that serve as state-of-the-art benchmarks for the E-AMoD control problem.

4) *MPC-Oracle*: the MPC is based on Eqn (1a) that assumes perfect foresight information of future user requests for all time steps. This approach serves as an *oracle* that provides a performance upper bound for any fleet management policy. Notice that MPC-Oracle does not scale well as the number of regions increases, since solving the optimization model is extremely computationally expensive.
5) *MPC*: we relax the assumption of perfect foresight information in MPC-Oracle. Additionally, in an attempt to approach the real-time constraint of 10 seconds, the planning horizon is reduced to three time steps of look-ahead for scenarios with 5 spatial regions, and a single time step look-ahead for scenarios with 10, 15, and 20 spatial regions. This approach is a more realistic optimization-based benchmark, but might still violate real-time constraints.

In our experiments, we monitor a set of metrics not directly included in the reward function. Specifically, we report performance with respect to (i) *Served demand*: defined as the number of completed passenger trips, (ii) *Operating cost*: defined as the overall cost induced on the system by non-passenger trips (i.e., accounting for both charging and spatial rebalancing), and (iii) *Percentage of oracle performance*.

B. Learning to Control E-AMoD Fleets

In our first simulation experiment, we study system performance on both San Francisco and New York scenarios, across increasing spatial coverage (i.e., from 5 spatial regions to 20). Results in Table I, Part A show that policies learned through graph-RL achieve, on average, $\approx 75\%$ of MPC-Oracle, which assumes perfect foresight of future demand and unlimited computation time. Table I and Figure 2 also highlight how the proposed approach is comparable in performance with the more realistic implementation of MPC, outperforming it in the SF15, SF20 scenarios and with a slight loss in performance on NY5, NY10 and SF5, SF10. Crucially, despite it being outperformed by MPC in some of the above instances, the graph-RL policy is the only method (together with the three heuristics) that can successfully satisfy the computation time constraints: this is of critical importance for the operator, as it would simply not be able to execute MPC in real-time beyond 5 spatial regions. Thus, results in Table I, Part A and Figure 2 show how graph-RL is able to maintain the performance of optimization-based methods, while substantially outperforming heuristics with comparable computation time.

Computational analysis.

After having discussed system performance in the previous experiment, we further study the computational cost of the proposed graph-RL approach compared to both heuristics and optimization-based approaches. As shown in Fig 3, we compare the time necessary to compute a single decision across varying dimensions of the underlying space-charge graph, ranging from 5 spatial regions and 19 charge levels up to 20 spatial regions and 19 charge levels. The results show how policies learned through graph-RL are approximately on par with heuristics, as opposed to optimization-based methods that scale super-linearly with the dimensionality of the problem. In practice, we compare the proposed graph-RL approach to (i) Off-peak Relative as a representative heuristic, as all heuristics considered in this work have comparable runtime, and (ii) MPC-Oracle to highlight the theoretical complexity of the underlying control problem. Crucially, Fig 3 highlights the appealing scalability of RL-based methods and shows that learning-based approaches allow for real-time control by forward-propagation of the current system state through the learned policy $\pi(\mathbf{a}|\mathbf{s})$, essentially amortizing the cost of optimization.

C. Transfer and Generalization

Inter-city transfer. To assess the transfer capabilities of graph-RL within E-AMoD systems, we also study the degree to which a policy learned on one city can be applied *zero-shot* (i.e., without further training) to a different city. Concretely, we do so by selecting a policy trained in New York and then deploying it in San Francisco (and vice-versa). As for the rest of our experiments, we repeat this procedure across varying levels of spatial coverage. Despite the lower overall performance, results in Table I, Part B show how policies learned through graph-RL exhibit an interesting

[5]https://newsroom.aaa.com/wp-content/uploads/2021/08/2021-YDC-Brochure-Live.pdf

[6]https://www.lyft.com/pricing/SFO

			Heuristics			RL	Optimization	
			Empty-to-full	Off-peak Abs.	Off-peak Relative	Graph-RL (ours)	MPC	MPC-Oracle
A	San Francisco	5	32.1% (0.5s)	32.1% (0.5s)	32.2% (0.5s)	87.4% (0.4s)	**94.2% (9.7s)**	971.7 (1:41min)
		10	27.0% (1.6s)	30.5% (1.6s)	30.6% (1.6s)	81.6% (1.9s)	**88.4% (15.2s)**	2271.2 (6:51min)
		15	27.9% (4.4s)	31.9% (4.4s)	32.0% (4.4s)	**76.4% (3.5s)**	67.7% (33.3s)	3544.8 (16:01min)
		20	26.7% (6.5s)	30.7% (6.6s)	30.7% (6.7s)	**78.1% (7.8s)**	73.6% (58.6s)	5261.9 (28:47min)
	New York	5	26.6% (0.4s)	30.5% (0.4s)	30.5% (0.4s)	89.0% (0.5s)	**95.1% (9.6s)**	84.5 (1:37min)
		10	18.8% (1.8s)	22.5% (1.8s)	22.9% (1.8s)	74.3% (1.8s)	**81.5% (14.4s)**	292.1 (6:19min)
		15	18.7% (4.2s)	22.3% (4.2s)	22.3% (4.2s)	63.4% (3.3s)	**79.2% (31.6s)**	528.6 (14:34min)
		20	18.1% (7.5s)	21.7% (7.6s)	21.7% (7.7s)	55.5% (7.39s)	**75.0% (54.4s)**	930.8 (24:58min)
B	SF → NY	5	-	-	-	77.0% (0.5s)	-	-
		10	-	-	-	35.6% (1.8s)	-	-
		15	-	-	-	54.7% (4.0s)	-	-
		20	-	-	-	30.3% (7.5s)	-	-
	NY → SF	5	-	-	-	55.1% (0.5s)	-	-
		10	-	-	-	48.8% (1.9s)	-	-
		15	-	-	-	46.1% (4.1s)	-	-
		20	-	-	-	44.5% (7.6s)	-	-
C	San Francisco	5 → 20	-	-	-	47.7% (7.7s)	-	-
		10 → 20	-	-	-	66.7% (7.6s)	-	-
		15 → 20	-	-	-	74.6% (7.7s)	-	-

TABLE I

PERCENTAGE OF ORACLE REWARD (PROFIT, THOUSANDS OF DOLLARS) AND COMPUTATION TIME PER DECISION ON TEST SCENARIOS. **BLACK-BOLD** AND RED HIGHLIGHT THE BEST-PERFORMING (NON-ORACLE) MODEL AND THE BEST-PERFORMING MODEL THAT SATISFIES REAL-TIME CONSTRAINTS (I.E., 10 SEC.), RESPECTIVELY. **RED-BOLD** IS USED IN CASE THE TWO COINCIDE.

degree of portability, substantially outperforming all domain-specific heuristics without having been explicitly trained for transfer performance, resulting in an average improvement of 1.75x.

Service area expansion. To further study how well policies learned through graph-RL can generalize to conditions unseen during training, we now consider the case of a hypothetical service area expansion. Specifically, we do so by selecting a policy trained within a specific spatial coverage in San Francisco and then deploying it within a larger spatial region (e.g., deploying the policy trained on SF 5 within the SF 20 scenario). As in the case of inter-city transfer, results in Table I, Part C highlight how the proposed graph-RL framework exhibits strong intrinsic generalization capabilities and outperforms all domain-specific heuristics, with an improvement ranging between 1.5x and 2.5x of heuristics performance. Moreover, by comparing results on SF5→SF20 and NY20→SF20, our experiments indicate that transfer across cities is more challenging than transfer within the same city: an observation that aligns with intuition, as different cities are typically characterized by more substantial differences (e.g., topology, travel times, etc.).

Together, these experiments highlight the benefits of the inductive biases introduced by graph neural networks and show huge promise in extending these analyses by explicitly considering transfer and generalization in the design of neural architectures and training strategies, e.g., by considering meta-RL [16].

Transfer to enable learning of large-scale instances. Lastly, we focus on quantifying the potential benefits of the transfer capabilities of graph-RL agents operating within a single city. Specifically, in Table I, Part C, and Figure 4, we measure the zero-shot performance on SF20 of policies trained on smaller scenarios (i.e., SF5, SF10, SF15) as opposed to the performance of training

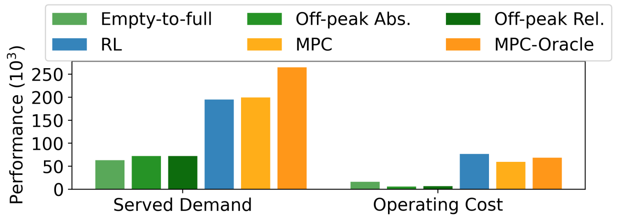

Fig. 2. Average served demand and operational cost comparison on San Francisco and New York (5, 10, 15, 20) scenarios.

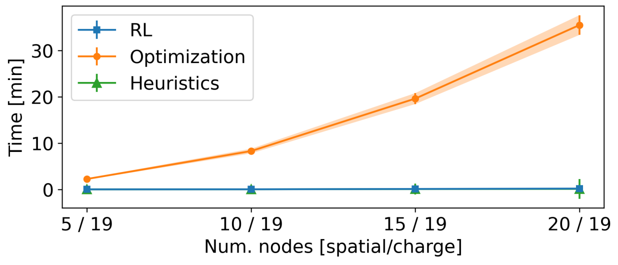

Fig. 3. Comparison of computation times between optimization (MPC-oracle, orange), graph-RL (blue), and heuristics (green).

from scratch a new control policy (i.e., SF20). Results show how the agents trained on SF5, SF10, and SF15 achieve 61.1%, 85.4%, and 95.5% of the agent fully trained on SF20, respectively. Not only does this quantify the benefits of curriculum learning within fleet control problems, whereby more similar environments allow for better transfer, but also opens several promising directions for future work toward the use of agents trained on small, computationally efficient environments as a starting point for successive fine-tuning on larger (and computationally intensive) instances.

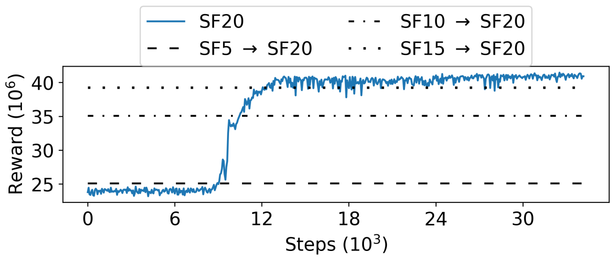

Fig. 4. Reward curve when training SF20 from scratch compared with zero-shot performance of SF5, SF10, and SF15 agents when deployed on SF20.

VI. CONCLUSIONS

Existing literature on the coordination of E-AMoD systems heavily relies on either optimization-based approaches or domain-specific heuristics. Among these two classes of techniques, methods belonging to the first have been shown to be extremely performant, although typically not scalable; on the other hand, methods belonging to the second make real-time implementation feasible by sacrificing on performance. In this paper, we present a graph-RL framework to achieve the best of both worlds: the scalability of heuristics, while maintaining high performance. We do so by introducing an RL agent that leverages the benefits of graph neural networks, reinforcement learning, and optimization for the real-time scheduling of E-AMoD fleets. Our experiments operating an E-AMoD fleet in NYC and SF using realistic data show that graph-RL policies can achieve performance comparable to the one of (real-time infeasible) optimization-based approaches while maintaining the scalability of domain-specific heuristics. Crucially, we show how graph-RL enables reinforcement learning agents to recover highly flexible, generalizable, and scalable behavior policies. There are several avenues for future research. First, to further validate the applicability of our method to real-world large-scale E-AMoD systems, a lower-level vehicle dispatch algorithm that is guided by our framework can be integrated with the simulation environment. Furthermore, the environment can be reconfigured to reward control policies that ensure periodicity in the fleet state for daily repeatability of the operations. Second, investigating ways to explicitly consider transfer in the design of neural architectures and training strategies, (e.g., meta-learning) is extremely promising. More broadly, the idea that large-scale network control problems can be approximated using a sequence of learning-guided linear approximations that are easier to solve is very promising and merits further exploration.

ACKNOWLEDGMENT

The authors thank Edward Schmerling, Ruolin Li and James Harrison for their insightful feedback, and Matteo Zallio for help in making the figures. The authors thank StreetLight Data, Inc. for providing travel demand data and the Stanford Research Computing Center computational resources on the Sherlock cluster. This research was supported by the National Science Foundation under the CPS program, Stanford Bits & Watts EV50 Project, the Center for Automotive Research at Stanford, and the NASA University Leadership Initiative (grant #80NSSC20M0163). This article solely reflects the opinions and conclusions of its authors and not NSF, NASA, or Stanford University.

REFERENCES

[1] B. Turan, R. Pedarsani, and M. Alizadeh, "Dynamic pricing and fleet management for electric autonomous mobility on demand systems," *Transportation Research Part C: Emerging Technologies*, vol. 121, p. 102829, 2020.

[2] J. Wen, N. Nassir, and J. Zhao, "Value of demand information in autonomous mobility-on-demand systems," *Transportation Research Part A: Policy and Practice*, vol. 121, pp. 346–359, 2019.

[3] G. Zardini, N. Lanzetti, M. Pavone, and E. Frazzoli, "Analysis and control of autonomous mobility-on-demand systems," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, pp. 633–658, 2022.

[4] M. Tsao, R. Iglesias, and M. Pavone, "Stochastic model predictive control for autonomous mobility on demand," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 3941–3948.

[5] A. Estandia, M. Schiffer, F. Rossi, J. Luke, E. C. Kara, R. Rajagopal, and M. Pavone, "On the interaction between autonomous mobility on demand systems and power distribution networks – an optimal power flow approach," *IEEE Transactions on Control of Network Systems*, vol. 8, no. 3, pp. 1163–1176, 2021. [Online]. Available: https://arxiv.org/abs/1905.00200

[6] D. Gammelli, J. Harrison, K. Yang, M. Pavone, F. Rodrigues, and P. C. Francisco, "Graph reinforcement learning for network control via bi-level optimization," in *Int. Conf. on Machine Learning*, 2023.

[7] J. Luke, M. Salazar, R. Rajagopal, and M. Pavone, "Joint optimization of autonomous electric vehicle fleet operations and charging station siting," in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2021, pp. 3340–3347.

[8] H. Bang and A. A. Malikopoulos, "Congestion-aware routing, rebalancing, and charging scheduling for electric autonomous mobility-on-demand system," in *2022 American Control Conference (ACC)*. IEEE, 2022, pp. 3152–3157.

[9] Z. Wan, H. Li, H. He, and D. Prokhorov, "Model-free real-time ev charging scheduling based on deep reinforcement learning," *IEEE Transactions on Smart Grid*, vol. 10, no. 5, pp. 5246–5257, 2018.

[10] A. Bogyrbayeva, S. Jang, A. Shah, Y. J. Jang, and C. Kwon, "A reinforcement learning approach for rebalancing electric vehicle sharing systems," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[11] J. Shi, Y. Gao, W. Wang, N. Yu, and P. A. Ioannou, "Operating electric vehicle fleet for ride-hailing services with reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 11, pp. 4822–4834, 2019.

[12] M. Guériau and I. Dusparic, "Samod: Shared autonomous mobility-on-demand using decentralized reinforcement learning," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 1558–1563.

[13] J. Holler, R. Vuorio, Z. Qin, X. Tang, Y. Jiao, T. Jin, S. Singh, C. Wang, and J. Ye, "Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem," in *IEEE Int. Conf. on Data Mining*, 2019.

[14] D. Gammelli, K. Yang, J. Harrison, F. Rodrigues, F. C. Pereira, and M. Pavone, "Graph neural network reinforcement learning for autonomous mobility-on-demand systems," in *Proc. IEEE Conf. on Decision and Control*, 2021. [Online]. Available: https://arxiv.org/abs/2104.11434

[15] Z. Yu and M. Hu, "Deep reinforcement learning with graph representation for vehicle repositioning," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[16] D. Gammelli, K. Yang, J. Harrison, F. Rodrigues, F. Pereira, and M. Pavone, "Graph meta-reinforcement learning for transferable autonomous mobility-on-demand," in *ACM Int. Conf. on Knowledge Discovery and Data Mining*, 2022. [Online]. Available: https://arxiv.org/abs/2202.07147

[17] C.-F. Daganzo, "An approximate analytic model of many-to-many demand responsive transportation systems," *Transportation Research*, vol. 12, no. 5, pp. 325–333, 1978.

[18] C. Fluri, C. Ruch, J. Zilly, J. Hakenberg, and E. Frazzoli, "Learning to operate a fleet of cars," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 2292–2298.

[19] H. T., A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Int. Conf. on Machine Learning*, 2018.

[20] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[21] H. Luo, Z. Bao, F. M. Choudhury, and J. S. Culpepper, "Dynamic ridesharing in peak travel periods," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 7, pp. 2888–2902, 2019.