Deployment Optimization for Shared e-Mobility Systems with Multi-agent Deep Neural Search

Man Luo[1,2], Bowen Du[2], Konstantin Klemmer[2], Hongming Zhu[3] and Hongkai Wen[1,2*]
[1] The Alan Turing Institute, UK [2] University of Warwick, UK [3] Tongji University, China

Shared e-mobility services have been widely tested and piloted in cities across the globe, and already woven into the fabric of modern urban planning. This paper studies a practical yet important problem in those systems: how to deploy and manage their infrastructure across space and time, so that the services are *ubiquitous* to the users while *sustainable* in profitability. However, in real-world systems evaluating the performance of different deployment strategies and then finding the optimal plan is prohibitively expensive, as it is often infeasible to conduct many iterations of trial-and-error. We tackle this by designing a high-fidelity simulation environment, which abstracts the key operation details of the shared e-mobility systems at fine-granularity, and is calibrated using data collected from the real-world. This allows us to try out arbitrary deployment plans to learn the optimal given specific context, before actually implementing any in the real-world systems. In particular, we propose a novel multi-agent neural search approach, in which we design a hierarchical controller to produce tentative deployment plans. The generated deployment plans are then tested using a multi-simulation paradigm, i.e., evaluated in parallel, where the results are used to train the controller with deep reinforcement learning. With this closed loop, the controller can be steered to have higher probability of generating better deployment plans in future iterations. The proposed approach has been evaluated extensively in our simulation environment, and experimental results show that it outperforms baselines e.g., human knowledge, and state-of-the-art heuristic-based optimization approaches in both service coverage and net revenue.

Index Terms—**Shared Mobility Systems, Electric Vehicles, Deployment Optimization, Deep Reinforcement Learning**

I. Introduction

Shared electric mobility (e-mobility) systems are becoming ubiquitous and forming a considerable part of our transportation paradigm in urban environments. By integrating them with other modalities, cities could optimize commuting experience, mitigate congestion and elevate air quality, thereby improving living standards. Therefore, deploying more such services is bound to offer an improved convenience and raise acceptance levels among users, thus boosting utilization and profitability over the long run. In this paper, we study the problem of *deployment optimization* for shared e-mobility systems, which aims to find the optimal way of deploying the shared e-mobility infrastructure (stations in our case) during operation, achieving the desired balance between gains and cost.

This has become increasingly important as the services mature: they have now passed the initial fast growing stage in which high deployment cost can be tolerated, but are more keen to fine-tune themselves to increase profit margin, while improving coverage and service level if possible. To achieve that, the shared e-mobility systems need to carefully select which stations to open or close and when, so that the overall performance is optimize throughout lifespan. Specifically, this particular problem falls into a broader class of optimization tasks in the context of e-mobility systems, which have attracted attentions from various communities. For instance, the work in [1] addresses a similar problem of charger planning for electric vehicle (EV) sharing services. It proposes a demand-aware planning approach and uses heuristics to approximate the optimal plan which ensures both pervasive coverage (in terms of reaching as many POIs as possible) and satisfies sufficient user charging demand. However, this approach only performs one-off optimization relying on aggregated historical demand estimates, which won't be able to adapt as situation changes. On the other hand, the recent work in [2] considers the incremental cases, but essentially it uses greedy-based approaches to re-compute for charger planning, which may not be efficient in many cases. In addition, most of the existing work aims to maximizing certain objectives given fixed budgets, such as service coverage and satisfied demand.

We consider a more realistic case, where the deployment of the shared e-mobility system could be optimized dynamically as they operate, with budget that could change over time. Concretely, we would like to find the dynamic deployment plan that guides system deployment through time, which maximizes given objectives (e.g., service coverage or revenue) while satisfing the constraints on budget. In particular, we consider the budget for infrastructure deployment as a function of time, which could depend on the expected income generated by the potential deployment plan, i.e., the revenue of the system under that plan should at least not exceed the cost of adjusting its stations (the system is self-sustaining). In reality, comparing to fixed budgets, such strategies could better cater for varying user demand and provide substantial improvement in performance, e.g., we could temporally close stations that are often quiet on specific days (weekdays vs. weekends), while deploying "overflow" stations in the presence of demand surges. However, it is not trivial to find the optimal deployment plans, in that i) for such mobility systems operating at city scales, the search space of station deployment can be prohibitively large; and ii) for a given deployment plan, accurately evaluating its performance could be challenging. This may be easy for static metrics such as service coverage, but for those that involve interactions between users and the systems, e.g., revenue of orders, it is often difficult to evaluate without actually running the systems with the deployment plan. Using historical data to extrapolate is one option (as in [1] and [2]),

[*] Corresponding author.

but as we show later the performance can be very limited.

To address the challenges, in this paper we build a high-fidelity simulation environment for shared e-mobility systems, which is able to capture the operation details of such systems at fine-granularity. The simulation environment is calibrated with data collected from a real-world shared e-mobility system for a year, to ensure that it can faithfully simulate the real system behaviors and user interactions. With the simulation environment, we design a novel multi-agent neural search algorithm to address the deployment optimization problem. Specifically, we consider a hierarchical controller, which iteratively proposes deployment plans, and evaluates their performance. The results are propagated back to the controller as rewards, whose parameters are updated accordingly so it can generate better plans in the future, as shown in Fig. 1. Concretely, the technical contributions of this paper are as follows:

- We design a high-fidelity simulation environment for shared e-mobility systems at city scales. Our simulation environment abstracts the key functionalities that are crucial in practice, while is able to capture the fine-grained details of operations in the shared e-mobility system. We calibrate the simulation with multi-modal real world data, which provides support for training and testing the proposed algorithms.
- We propose a novel multi-agent neural search algorithm to address the deployment optimization problem. We formulate the problem with multi-agent reinforcement learning (MARL) framework, and develop a novel hierarchical controller architecture, which learns to search for optimal deployment plans efficiently.
- We evaluate the proposed approach extensively, and show that development plans proposed by our approach significantly outperform the actual plan used in the real-world, and the state-of-the-art optimization approaches, achieving improvements in both revenue and coverage.

II. Related Work

Shared Mobility Services. Recently there is a surging research interests in shared mobility systems in cities, e.g., from understanding taxi demand [3] to reposition shared bikes [4] and optimising shared mobility systems to achieving the desired balance between profit and fairness [5]. EV-sharing in particular, as one of the environmental friendly services, has attracted extensive attention from various angles and incubated many interesting problems, such as energy consumption estimation [6], charging scheduling [7], [8] and infrastructure planning [9]. These studies mainly consider public transportation, e.g., electric taxis or buses, while our work focuses on privately-owned EV sharing services, which is fundamentally different. For instance, for electric taxis their networks of stations [10] are mainly used for charging, whose service coverage depends on individual taxi drivers. However in our case, the users can only access (renting, returning and charging) vehicles at the available stations, and thus the deployment of stations will have much more direct impact on the entire system. Moreover, comparing to traditional vehicle sharing systems, optimizing the operation of EV sharing services is more complex due to the unique properties of EVs.

Urban Mobility Systems Simulation. Building simulation environments to study urban mobility systems has become a popular approach in existing work [11], [12], [13]. However, most of those solutions rely on oversimplified assumptions, while only considering the historical data of the mobility system for training and testing. For instance, the simulator in [11] first simulates potential users randomly across space, and then those generated users need to keep "walking" to different stations until an available bike is found, which is not realistic nor efficient. The simulation environment for taxi in [13] assumes the taxis need to move randomly on the road network, while in [12] the potential users have to wait until they are picked up by an arriving taxi, both of which are often not true in practice. In the context of simulating facility/infrastructure deployment, most of the existing work only considers one-time [1], or multiple-stages [2] of infrastructure deployment or expansion. Our simulation environment, on the other hand, simulates the system operation and the actual interactions between users and the system with much finer details, and supports dynamic/continuous infrastructure deployment, i.e., stations can be opened or shut down at arbitrary times.

Deep Learning in Mobility Applications. Deep learning techniques have been used in various mobility applications due to their superior performance, e.g., improving service levels of taxi [14], predicting user demand [15], and reposition bikes or vehicles [16]. In particular, deep reinforcement learning has been introduced to solve various challenging mobility problems, such as road management [17], order dispatching [18], and rebalancing [19]. Due to their distributed nature, many of those mobility applications can be modeled as multi-agent games, which can be well solved by deep reinforcement learning. For instance, the work in [20] designs a spatio-temporal reinforcement learning approach to dynamically reposition bikes in the bike-sharing system. The work in [21] proposes a multi-agent reinforcement learning framework to tackle the fleet management problem, while [18] addresses the order dispatching problem for ride sharing systems using mean field approximation. It has been shown that in those applications, deep reinforcement learning often achieves better performance, e.g., in terms of reducing potential customer loss, or increasing the gross merchandise values, than the traditional rule-based or optimization based approaches, especially when the problem structure is complex.

Infrastructure Deployment and Optimisation. Our work is also related to the problem of infrastructure deployment and optimisation, which has been extensively investigated in various application scenarios, such as determining locations for automotive services [22], optimising locations and qualities of infrastructure [23], [24], [25], and facility planning in healthcare [26]. Particularly in the EV context, most of the existing work focuses on the deployment of charging facilities. For instance, the work in [27] proposes an approach to find locations to deploy chargers using electric taxi trajectory data, while [28] considers an integrated multi-criteria decision making approach for charging stations planning. The work in [29] also considers the cost for charging when optimising charger locations, while [2] proposes two heuristic-based algorithms to deploy EV chargers given their potential social benefits. It also considers the cases of incremental deployment, but

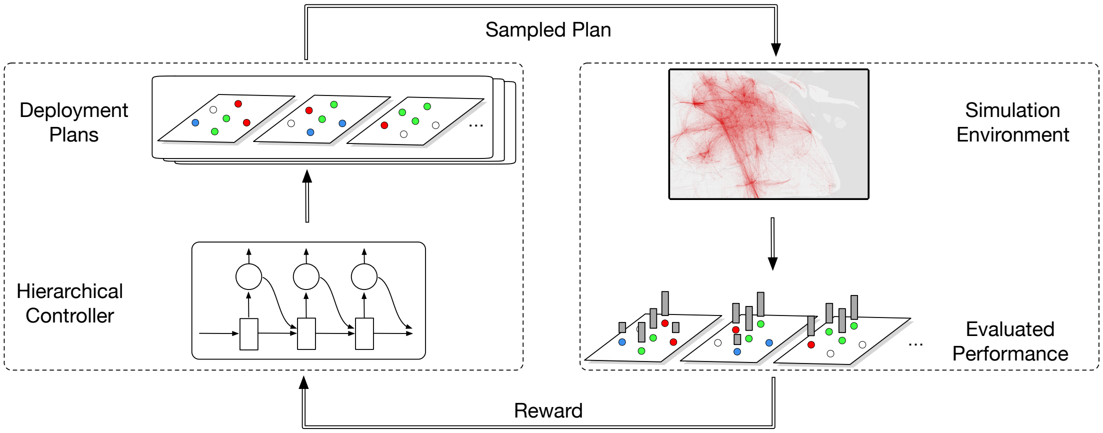


Fig. 1: Workflow of the proposed hierarchical neural search approach. The hierarchical controller generates possible deployment plans, which are then run in the simulation environment to evaluate their performance. The reward signals (e.g., net revenue and service coverage) obtained are used to update the controller parameters with reinforcement learning.

they essentially re-run the proposed algorithms on the new set of candidate locations. Our work is different in that we have different objectives to optimize. These work considers the deployment public chargers across the city, which aims to satisfy as much charging demand as possible with a given budget on charger deployment. In our case besides service coverage, we also care about profit as a private EV sharing service provider. In addition, our approach does not need to work with a predefined budget, which is often difficult to estimate in practice, but just finds the self-sustaining deployment plans that can cover the cost. On the other hand, the work in [1] also studies EV charger planning for private EV sharing platforms, which shares the similar problem with our work. However, it assumes the deployment of chargers is an one-off task, and does not consider dynamic deployment cases as studied in this paper.

Neural Search and Optimization. Recently, there is an emerging interest in using deep neural networks to solve a broad range of optimization problems. Given their structure, it is often possible to formulate the optimization problems as sequential decision progresses and use a single or multiple reinforcement learning agents [30], which learn the heuristics implicitly to find a solution. For instance, the work in [31] proposes a DQN based approach to address the Max-Cut problem, which becomes the new state-of-the-art. In [32] the authors design a general RL approach for combinatorial optimization and show promising results on the Maximum Independent Set Problem. The work in [33] tackles the Graph Coloring Problem for large graphs by using efficient network architectures to speed up optimization. Besides the traditional optimization problems, neural search techniques have also been widely used in the emerging AutoML field, which aims to automate the machine learning pipeline [34], or combining the strengths of different models via imitation learning [35]. Our work bears close resemblance to those existing work, however, we try to solve a different problem in the shared e-mobility context, and propose a novel hierarchical controller architecture to search efficiently, which has not been considered by the existing work.

III. Simulation Design

In this section, we present our design of the high-fidelity simulation environment for shared e-mobility systems at city scale, which builds the playground for the proposed multi-agent neural search algorithm to learn, by trial-and-error, how to optimize deployment with specific goals (e.g., revenue vs. service coverage) in mind. Our simulation environment has two key advantages. Firstly, it is able to simulate the complete operation process of the shared e-mobility system, from users picking up an EV, to returning the EV to destination and charging, at a fine-grained granularity, capturing the unique properties of EV sharing such as range limitations, and the complex interactions between users and the system. Secondly, we calibrate our simulation with multi-modal data collected from the real world, ensuring that it can faithfully capture the key characteristics of the actual system, and thus exhibits similar behaviour as the system operated in practice.

A. *Overview of Shared e-Mobility Systems*

The Shared e-Mobility Model. In this paper we consider *station-based* e-mobility systems, i.e., the users have shared access to the EVs of the system, but can only pick up EVs from and return them to the available stations. Let s be such a station. In our case, s can be represented as a tuple $(\texttt{loc}, \#\texttt{c})$, where $\texttt{loc}$ denotes the location (e.g., latitude and longitude coordinates) of s, and $\#\texttt{c}$ is the total number of charging docks (equivalent to parking spaces) within the station s. In practice, the system may deploy new stations or close existing ones across the city. When a station s is firstly deployed for operation, a number of EVs (denoted by $\#\texttt{v}$) would be assigned to s, where typically $\#\texttt{v} < \#\texttt{c}$. This means at the beginning the station s should have $\#\texttt{v}$ EVs available for pick up and $\#\texttt{c}$ - $\#\texttt{v}$ free spaces for potential EV returns.

In such shared e-mobility systems, the vehicles are of limited range. In this work we assume the fully charged ranges of the EVs for given vehicle models are fixed, and during normal driving the remaining range can be determined by their discharging curves [36]. In practice, we observe that the time for charging of the EVs is much longer than refilling the traditional vehicles, which however can be estimated by the corresponding charging models [36], given the remaining range, battery capacities and charger specifications.

Given the set of its available stations S, the EV sharing system operates as follows. Assume that at time t, a user wants to rent an EV from a certain station $s^{\mathrm{o}} \in S$, and return to the destination station $s^{\mathrm{d}} \in S$. If she finds there is at least one vehicle available at the picking up station s^{o}, that has been sufficiently charged with enough range to cover her planned trip, she will post an order $o_t = (s^{\mathrm{o}}, s^{\mathrm{d}})$ to the EV sharing system. Upon accepting the order, the system allows the user to take over the EV. The order is considered to be completed (i.e., the demand of this user is satisfied) when the EV is returned to the destination station s^{d} and plugged in for charging, which is considered to be ready for the next user. The price for this order is calculated based on her rental duration. If the destination does not have a vacant parking space, we assume the system should reposition the EV to the nearest station which has parking/charging availability within certain radius (3km in our simulation), which is often achieved by a team of dedicated staff in real world operations.

The Shared e-Mobility Data. We have collected a rich set of data from a real-world shared e-mobility system in Shanghai for 12 months to support and calibrate our simulation environment, ensuring its high-fidelity with respect to the systems operating in the real world. More concretely, the dataset contains a) the complete **transactions of orders** in the system, i.e., when and where a user rented/returned a vehicle, and b) information on **station deployment**, i.e., when and where a station was deployed or closed. In the order data, each record contains detailed information about this order, including the anonymous user ID who initiated the order, the ID of the origin/destination stations, the timestamps of pick-up/return, the total duration of the order and the final price etc. In total we have collected >7 million valid order transactions, which were generated by approximately 0.4 million active users during the one year period. For the station deployment data, we have collected the service status at each day during the 12 months period, including station locations, numbers of charging docks in each station (see Fig. 2a for visualization), and when and where stations were deployed/closed. We also collected additional auxiliary data to support our simulation, e.g., to calculate the cost for infrastructure deployment, estimate the potential service coverage, and infer the demand for travel. This includes the POI data of the city [37] (see Fig. 2b for visualization), the average property prices across different regions [38] (see Fig. 2c for visualization), taxi trajectory data during the same period of time [39], and the charging/discharging models of the electric vehicles [36].

B. Simulating Dynamic Deployment of Stations

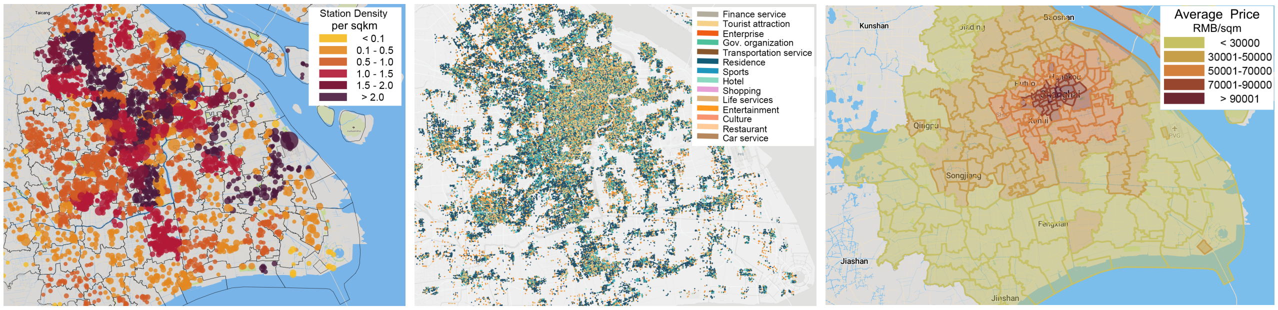
We construct our simulation environment by first simulating the deployment in the stations of the shared e-mobility system. For simplicity, in our simulation we assume that any stations to be deployed are drawn from a candidate pool $\mathbb{S}$, containing all possible candidates of stations in the system. Without loss of generality, we consider a 2D world where a station candidate $s \in \mathbb{S}$ is a point at location $\mathtt{loc}$ and with $\#\mathtt{c}$ charging docks. In our implementation, we generate the candidate pool $\mathbb{S}$ according to the station deployment data collected from the real-world system, i.e. for any station s that has been deployed once in that system, regardless of whether it had been closed or not, we add s to the pool $\mathbb{S}$. In this way we obtain the candidate pool of stations for deployment in simulation, which forms the search space for the proposed approach.

We consider *dynamic deployment* of stations, i.e., stations can be opened or shut down at arbitrary times. Without loss of generality, we assume the time granularity for station deployment is days, i.e., they can be adjusted on daily basis (practicality of this paradigm in the real world discussed later in Sec. VII). Let $S_t \subseteq \mathbb{S}$ be a subset of the candidate pool. We refer to S_t as a *deployment snapshot* at time t, and assume by the end of the timestamp only the stations in S_t will remain active in simulation. The sequence of such deployment snapshots from the beginning to the end of the simulation episode is defined as the *deployment plan* $\boldsymbol{S}$, where $\boldsymbol{S} = \{S_t\}$, $t \in [1, T]$. In essence, $\boldsymbol{S}$ describes the evolving process of the stations in the shared e-mobility system, which as discussed later is the key for performance, such as the revenue and demand satisfaction rate of the service.

We also assume that for each candidate station s, there is a cost $c(s)$ for being active in operation of the system per unit time. In our simulation, this cost is set based on two factors: i) the property price around s; and ii) the size (i.e., number of parking spaces $\#\mathtt{c}$) of s. This is consistent with the cases we observed from the real-world data, where the rental prices of stations depend on their sizes, e.g., those with more parking spaces tend to be more expensive, and also differ in different areas. Therefore, at given time t the cost of running the service depends on the current deployment snapshot S_t, i.e., $c(S_t) = \sum_{s \in S_t} c(s)$. Then for a deployment plan $\boldsymbol{S}$, we define the total cost in this episode $c(\boldsymbol{S})$ as the sum of the costs incurred by its snapshots: $c(\boldsymbol{S}) = \sum_{t=1:T} c(S_t)$. Note that here for simplicity we do not consider the upfront deployment cost such as installing chargers etc., which in practice can be factored into the long-term running cost, and assume there is no overhead of shutting down or re-opening stations.

C. Generating Spatio-temporal User Demand

To better simulate the operation details of the actual shared e-mobility system, we set 10min as one simulation step. Note that this is different from the timestamps considered in station deployment process as discussed above, which is a day (24 hours), containing 144 simulation steps. We define a particular user demand as a tuple $d = (k, \mathtt{o}, \mathtt{d})$, where k is the current simulation step, $\mathtt{o}$ and $\mathtt{d}$ are the origin and destination of this demand respectively. We represent $\mathtt{o}$ and $\mathtt{d}$ with 2D coordinates e.g., GPS locations across the space. This represents the intention of travel from $\mathtt{o}$ to $\mathtt{d}$ of the users, and in our simulation we learn to generate user demand from the real-world data. In practice, the user demand may be affected by the deployment of infrastructure itself, e.g., users may want to try the system when they find there are stations nearby. To mitigate this, in our simulation we consider two independent data modalities, i) the historical order transactions of the shared e-mobility system, as well as ii) the taxi trajectory



(a) (b) (c)

Fig. 2: Visualization of the collected data. (a) The distribution, density and sizes (# of parking spaces/charging docks) of the stations in the real-world shared e-mobility system. (b) POI distribution in the city. (c) Average property price across different regions in the city.

data (cleaned to only have origins and destinations). We use Gaussian Processes (GP) with RBF kernels to fuse the data from both modalities, approximating the genuine "intend to travel" of users at certain point and time. In particular, we first consider the pick up demand (i.e., the origins $\circ$) and learn GP models over space and time, with which at each simulation step k we can obtain a spatial distribution of the potential origins in user demand. This also means for any point in the 2D space, we have an estimate of the numbers of users that intend to travel from that point at step k, with certain confidence interval.

When generating the demand in simulation, at given step k we sample the learned GP over the candidate station pool $\mathbb{S}$, i.e., at each candidate station we obtain the amount of travel demand starting from that location at step k. Here we only sample over $\mathbb{S}$ rather than the entire 2D space because eventually user demand will be satisfied at those stations, while the GP model has already taken the neighbourhood knowledge into account, e.g., nearby taxi demand or EV pick-up demand at stations within certain distance. Therefore in the following text, we consider the user demand only at the candidate stations, i.e., $d = (k, s^o, s^d)$, where $s^o, s^d \in \mathbb{S}$.

To generate the complete demand in pairs of origins and destinations, for a user demand originated from $s^o \in \mathbb{S}$, we use another GP to approximate the spatio-temporal distribution of possible destinations from the historical orders of the e-mobility system as well as taxi data. In essence, at step k for this demand with origin s^o, the GP model generates a 2D distribution over the space, and by sampling from the distribution over the candidate pool $\mathbb{S}$ we can pin down the destination station s^d for this particular demand. The reason why we adopt this two-phase approach rather than sampling independently from the distributions of origins and destinations, is that in practice the origins and destinations of user demand are highly correlated, and the correlations often depend on different times. In addition, unlike existing work [1] which directly uses historical data as demand, we use GPs to i) fuse different sources of information (taxi and the e-mobility data), and ii) generate distributions of user demand rather than scalar values for robustness. As shown later in Sec. VI, this improves the fidelity of our simulation, and helps the proposed algorithm to generalize better.

D. Operating Shared e-Mobility Systems in Simulation

For each simulation episode, let $\boldsymbol{S}$ be a deployment plan of EV sharing stations, e.g., generated by our search algorithm. In essence, $\boldsymbol{S}$ describes which candidate/existing stations should be deployed/removed from the service and when. We assume that this station deployment/adjustment process happens at the beginning of a timestamp t (0am on each day), which updates the active stations according to the corresponding snapshot S_t in the deployment plan $\boldsymbol{S}$. Note that any candidate station to be deployed, we allocate certain number of EVs (fully charged) to this new station for its future operation, according to its maximum capacity (#c) and a probability ratio learned from the historical data. Typically in our simulation we assign 0.5#c to 0.7#c of EVs to the new stations, leaving sufficient parking spaces for incoming EVs from other stations.

Then for each simulation step k (there are 144 steps per time t), our simulator generates user demand as discussed in the previous section. It then operates the system in the same way as discussed in Sec. III-A, i.e., processing the simulated user orders, handling vehicle charging/returns. Then at time t, with the deployment snapshot S_t, we define the total revenue value at t as $\texttt{GMV}(S_t)$, which is calculated over all the satisfied orders of the EV sharing service during that time. The net revenue value $\texttt{NV}(S_t)$ is then defined as GMV subtracts the deployment cost: $\texttt{NV}(S_t) = \texttt{GMV}(S_t) - c(S_t)$. For deployment plan $\boldsymbol{S}$, we define its GMV and net revenue as their summations over T: $\texttt{GMV}(\boldsymbol{S}) = \sum_{t=1:T} \texttt{GMV}(S_t)$, and $\texttt{NV}(\boldsymbol{S}) = \sum_{t=1:T} \texttt{NV}(S_t)$ respectively.

IV. Problem Formulation

A. The Deployment Optimization Problem

Let us assume at time $t = 0$, the simulation world is at the initialization state, i.e., the shared e-mobility system has some stations deployed and is ready to operate. This is reasonable since in practice, such systems tend to deploy their first batch of stations before going live to the public. Let T be the length of one simulation episode, i.e., we run the simulator for T timestamps (days in our case) in one training/testing pass. As discussed above, we assume during the T days the stations of the EV service can be dynamically deployed or removed on a daily basis, i.e., at each t, according to a deployment plan

$\boldsymbol{S} = \{S_t\}$, $t \in [1, T]$, where S_t is the set of stations to be active at t, drawing from the candidate pool $\mathbb{S}$.

Let $\mathcal{P}(\boldsymbol{S})$ be the set of all possible deployment plan of length T for this simulated e-mobility system. Then our deployment optimization problem is to find the optimal deployment plan $\boldsymbol{S}^* \in \mathcal{P}(\boldsymbol{S})$:

$$\begin{aligned} \boldsymbol{S}^* = & \underset{\boldsymbol{S} \in \mathcal{P}(\boldsymbol{S})}{\arg\max}\, \mathtt{SC}(\boldsymbol{S}) + w\mathtt{PM}(\boldsymbol{S}) \\ \text{s.t:} \quad & c(S_t) \leq B(t), \text{ where } S_t \in \boldsymbol{S}, \forall t \in [1 : T]. \end{aligned} \tag{1}$$

Here $\mathtt{PM}(\boldsymbol{S})$ is the *profit margin ratio* of the deployment plan $\boldsymbol{S}$, i.e., the percentage of profit (net revenue $\mathtt{NV}$) with respect to the total income ($\mathtt{GMV}$) if deploying stations as instructed by $\boldsymbol{S}$. $\mathtt{SC}(\boldsymbol{S})$ is the *service coverage ratio* of the deployment plan $\boldsymbol{S}$, which is defined as the normalized sum of two ratios: i) the percentage of satisfied user demand, and ii) the percentage of POIs covered by the service. We assume a POI is covered by a station if it is within $1km$ radius of the station. Therefore, at time t the service coverage of the corresponding deployment snapshot $\mathtt{SC}(S_t)$ depends on both satisfied demand and covered POIs. Similar to $\mathtt{NV}(\boldsymbol{S})$ defined above, we define $\mathtt{SC}(\boldsymbol{S})$ as the mean service coverage of this particular deployment plan over T timestamps, $\mathtt{SC}(\boldsymbol{S}) = T^{-1} \sum_{t=1:T} \mathtt{SC}(S_t)$. In our case, both objectives are in percentage, and thus we can use a weight $w \in [0, 1]$ to balance them, which can be tuned for scenarios with different priorities, e.g., aiming to roll out service to more users vs. obtain more profit. The constraint requests that the instantaneous cost of the deployment plan at t should not exceed the value of a budget function $B(t)$, which can be fixed or varying over time. Unlike existing solutions which typically assume fixed budget (a total sum or evenly spread), in this work we consider a more realistic budget function, $B(t) = \mathtt{GMV}(S_t)$, which requires the system to be *self-sustaining*. In particular, this means that under a given deployment plan S_t, the income $\mathtt{GMV}(S_t)$ of the system at any t should be able to cover the cost of infrastructure deployment, i.e., we want the system to be break-even.

In fact, if we unfold the deployment plans over time, this deployment optimization problem is essentially a constrained combinatorial optimization problem. However, the search space in our case can be prohibitively large, e.g., with $4k$ candidate stations over 30 days period there will be $2^{4k^{30}}$ possible plans to evaluate. In addition as discussed in previous sections, the impact of deploying/removing a particular station is often complicated, i.e., we won't be able to directly estimate the gain of such an action (e.g., increase in $\mathtt{NV}$ and $\mathtt{SC}$) without running the system through. This makes most of the existing heuristics-based optimization approaches [1], [2] infeasible, as they assume the benefits/utility (e.g., of deploying a particular charger/station) are independent and known a priori. In the next section, we show how we formulate this problem as a Multi-agent Reinforcement Learning (MARL) task, which allows us to explore the search space via trial-and-error.

B. Deployment Optimization as a MARL Task

In this paper, we consider a set of autonomous agents, each of which interacts with the common environment to improve its behavior. Comparing to the single agent settings, the multi-agent formulation is more suitable to address our problem, because multiple agents could better exploit the decentralized nature of the station deployment task. By design learning can be more efficient than the single agent case, as the action spaces of the agents are much smaller, and the computation can be accelerated by parallel processing. Given the environment (i.e., EV sharing service), we model the deployment optimization problem as a Markov Game $G = (N, \mathcal{X}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, in which at most N agents interact with the environment, making decisions as to where and when stations should be deployed. $\mathcal{X}$ represents the states of G and $\mathcal{T}$ is the transition function between states. $\mathcal{A}$ is the joint actions of the agents, $\mathcal{R}$ is the reward function, and γ is the discount factor.

Agents. In our multi-agent formulation, we assume an agent controls the deployment of EV stations within a certain geospatial region. Essentially, we delegate the agent to manage the station deployment process of that region, who decides which candidate stations should be deployed or closed and when. We partition the space into regions belong to different agents by clustering candidate stations locations in $\mathbb{S}$. In particular, we cluster the candidate stations based on their pairwise distances, where boundaries of regions are formed by convex hulls of clusters. Let us assume we have obtained N regions as the result of clustering, managed by N agents.

To make sure that the agents face similar learning load, we also require each cluster to have the same amount of M candidate stations. In practice, this may lead to regions with different sizes due to the spatial variation in the density of candidate station locations, e.g., we observe that the obtained regions in city center is much smaller than those at the city edges. However, we found that this won't affect the agents' performance and behaviour, since the candidate station density is also highly correlated with POI density and distribution of potential user demand. This means although managing regions with different sizes, the agents tend to learn to cover similar levels of demand/POIs with its deployment plans, only at different spatial scales. It is also worth pointing out that our approach can also work with the other space partition paradigms, such as hexagonal or rectangular grids, as long as each region/grid is managed by a individual agent.

States and Observations. At time t, for the i-th region, its state x_t^i encodes information about the user demand, EV status and station deployment within the boundary of this region. In particular, we consider both the set of currently online stations, and candidate stations that are not yet deployed. For each of them, we include its location $\mathtt{loc}$, number of available charging docks $\#\mathtt{c}$, the deployment cost, the numbers of EVs parked in the stations and their individual range at each simulation step within t, as well as the potential future rent/return requests (number of EVs) and the average value of potential future orders in the next timestamp. The global state $\boldsymbol{x}_t$ is the combination of states of each region $\boldsymbol{x}_t = \{x_t^i\}$, $i \in [1, N]$. Let agent i manage the i-th region. At time t, we assume the agent observes both its local state $\{x_t^i\}$ and the global state $\boldsymbol{x}_t$. This allows the agents to observe and interact with not just its local environment, but also encourages them to learn to better cooperate with the others that have connections with them, e.g., agents of neighbouring regions, or those with high correlations in terms of user demand (i.e., users may frequently travel from one region to another).

Agent Actions and State Transitions. For agent i, its action a_t^i describes the deployment snapshot of the i-th region, i.e., which stations should be deployed or removed. Therefore at time t, a joint action $\boldsymbol{a}_t \in \mathcal{A}_t^1 \times ... \times \mathcal{A}_t^{N_t}$ all the N agents forms the complete deployment snapshot S_t, as introduced above. Upon performing the joint action at t, the current state $\boldsymbol{x}_t$ will transit to the next state $\boldsymbol{x}_{t+1}$ according to the state transition probabilities $\mathcal{T}$, which are defined as $\mathcal{T}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{a}_t)$. Note that in many MARL cases, it is often not possible to describe the state transitions as functions with analytical forms, and thus in our case we do not attempt to model $\mathcal{T}$ explicitly, but rely on our simulator to capture the state transitions.

Reward Function. For each agent, the reward r_t^i of taking an action at time t is determined by the reward function:

$$\mathcal{R}^i(\boldsymbol{x}_t, \boldsymbol{a}_t) : \mathcal{X} \times \mathcal{A}_t^1 \times ... \times \mathcal{A}_t^{N_t} \to \mathbb{R} \tag{2}$$

As discussed above, in our deployment optimization problem, we would like to maximize both the profit margin (PM) and the service coverage (SC). In practice, we find that the two objectives often diverge, e.g., maximizing PM would often lead to greedy agents that always try to deploy new stations at "hot" locations with high profit. On the other hand, optimizing SC tends to encourage the agents to spread stations across the space, while keeping many less profitable stations active, resulting in decrease in net revenue. In addition, we also require the service to be sustainable, in that the GMV of the deployed stations should be able to cover their cost. To balance such factors, given the agent action a_t^i, which requires the set of stations S_t^i to be active at time t, we design the reward function r_t^i based on our optimization objectives as in Eq. (1):

$$r_t^i = g^{\texttt{SC}}(S_t^i, S_{t-1}^i) + w\, g^{\texttt{PM}}(S_t^i, S_{t-1}^i) + \lambda \min\{\texttt{NV}(S_t^i), 0\} \tag{3}$$

where $g^{\texttt{SC}}(S_t^i, S_{t-1}^i)$ and $g^{\texttt{PM}}(S_t^i, S_{t-1}^i)$ calculates the *improvement rate* in terms of service coverage SC and profit margin PM, given the current deployment snapshot S_t^i and the previous one S_{t-1}^i. w is the weight balancing the two objectives, as in Eq. (1). The term $\lambda \min\{\texttt{NV}(S_t^i), 0\}$ penalizes any S_t^i that produces negative net revenue with a scaling factor λ, which is learned via grid search. Given the reward function, each agent aims to maximize its discounted reward $\mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k}^i]$, where $\gamma \in [0, 1]$ is the discount factor.

V. Hierarchical Deep Neural Search

Under the above MARL formulation, we design a novel hierarchical neural search algorithm, which learns to generate better deployment plans over time. The key idea is to use a controller network to generate various deployment plans (i.e., actions), and evaluate the performance of those plans in our simulation environment. The reward/penalty signals collected from the simulation are then propagate back and used to update the controller with reinforcement learning paradigm, so that it tends to produce improved plans in future runs.

A. Hierarchical Controllers Design

In this paper, we consider a LSTM-based controller, which recursively generates deployment plan S_t at each t for T timestamps. In fact, given the multi-agent setting, for each agent i, we can use a boolean string of length M to represent S_t^i, where M is the total number of candidate stations in the regions. Such encoding has been widely considered in tasks such as searching for neural network architectures. However unlike the existing work, in our case even with multi-agent formulation, directly generating and searching from the 2^M sized encoding space is prohibitively expensive. To tackle this challenges, we design a hierarchical controller which generate S_t^i progressively, as shown in Fig. 3. The key idea is that instead of directly sampling from the distribution of deployment plans at each t, we generate S_t^i in two stages. Concretely, we design a hierarchical structure with two connected controllers: a *high-level* and a *low-level* controllers respectively.

High-level Controller. Given the previous and current states, rather than delving into deployment details, the high-level controller only generates a categorical *deployment prior* for the region i at time t. In particular, we consider a deployment prior as a pair (S_t^{i+}, S_t^{i-}), where S_t^{i+} is a categorical variable indicating the amount of new stations to be made online during time t at region i, e.g., 10% of the total candidates in the region, while S_t^{i+} is for the stations to be made offline. In this case, if the domain of the variables is limited, then for the high-level controller the search space at each timestamp t is manageable. On the other hand, such a deployment prior indicates the coarse volume of the desired deployment plan for the region at time t, e.g., the high-level controller may prefer to deploy more rather than removing in some cases since it predicts demand would surge.

Low-level Controller. Now we explain the design of the low-level controller, which translates the received deployment prior into the concrete deployment plan S_t^i. Essentially, now the low-level controller needs to find two sets of stations, one to deploy and one to close, while the cardinalities of them are known. In this case, one straightforward way is to continue to sample the constrained search space which is already much smaller, and generate the deployment plan S_t^i accordingly. However, we found that in practice this would still lead to many meaningless trials without converging. Therefore, in this work we adopt an efficient approach to generate S_t^i. Concretely, for each candidate station s_j^i within region i, we compute a score which is the combination of two values: i) its service coverage, i.e., the ratio that it could contribute to the total service coverage in this region, and ii) the ratio of expected demand it could satisfied with respect to the total demand in region i. Here we use a Graph Convolutional Neural Network (GCN) (e.g., as in [15]) to predict the expected demand for s_j^i at t, given the previous states in this simulation episode. The GCN is trained with the historical order data, and is able to capture the dynamics caused by station deployment/closure.

Now we obtain the ranked list of the candidate stations in region i based on their scores. Intuitively, we should deploy the best station candidates if they are not yet in operation, and close the poorly performing ones. To generate such S_t^i at t, our low-level controller adopts a ϵ-greedy approach for both deploying and closing stations. In particular, it iteratively chooses the current best/worst station candidates with the probability $1 - \epsilon$, while making random selections for the rest, until the required number is reached. In our experiments,

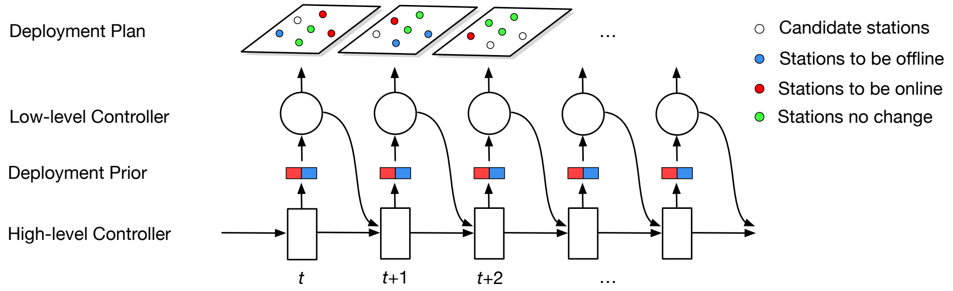


Fig. 3: The architecture of the proposed hierarchical controller.

we set ϵ to 0.1. In this way, the low-level controller balances exploration and exploitation in searching of S_t^i, which makes our algorithm more robust to noises and perturbations.

B. Training with Reinforcement Learning

In this paper, we consider a shared-weights controller for our agents, which is trained with reinforcement learning. As discussed in Sec. IV-B, the generated deployment plan $\boldsymbol{S} = \{S_t\}$ is a sequence of actions $\{\boldsymbol{a}_t\}$ $(t = 1 : T)$ to adjust the EV sharing service. By applying $\boldsymbol{S}$ in the simulation, we could evaluate the performance of $\boldsymbol{S}$, and compute the reward as Eq. (3) and Eq. (2). The reward signal is then fed back to the controller, to help updating its parameters θ. Therefore the updated controller would have high probabilities to propose better deployment plans. Note that here θ is only relevant to the high-level controller, as the parameters of the low-level controller (the GCN) have already been trained. In our case, the reward structure is not differentiable ($\min$ in Eq. (3)), and thus we consider policy gradient approaches to iteratively improve θ. In our case, we train the controller to maximize the performance of its sampled deployment plans $\boldsymbol{S}$. The training objective can be formulated as follows:

$$J(\theta) = \mathbb{E}_{p(\boldsymbol{S};\theta)}[\mathcal{R}(\boldsymbol{S})] \tag{4}$$

where $\mathcal{R}$ is the reward collected from simulation, and the expectation is taken over $p(\boldsymbol{S};\theta)$, i.e., the distribution of deployment plans $\boldsymbol{S}$ given θ. In this paper we use the Proximal Policy Optimization (PPO) approach [40] to optimize $J(\theta)$, which is more sample efficient than the standard REINFORCE [41], and offers faster convergence of the controller. We then iteratively optimize the objective function, until the optimal parameters θ^* of the controller can be found.

C. Accelerating Training with Multi-simulation

As discussed above, during training each update to the controller parameters requires one run of the simulation to collect rewards, i.e., virtually operating the EV sharing system for a certain period of time. This can be time-consuming and expensive to run in practice. To speed up the learning process of the controller, we propose a multi-simulation approach that exploits the benefits of parallelism. Concretely, we set up a job queue, where the controller generates and dispatches deployment plans to this global queue. On the other hand, we employ multiple workers (across multiple threads/processes, or distributed over networked servers), each of which manages its own copy of the simulation environment. During training, a free worker picks up a proposed deployment plan from the job queue, and run the simulation to evaluate the performance of that particular deployment plan. Then the results of different workers are returned to the controller, whose parameters get updated with PPO. The controller then generates another batch of samples and this process continues until a pre-defined number of deployment plans have been evaluated.

VI. Evaluation

A. Experimental Setup

We compare the proposed multi-agent neural search (**MANS**) approach with the following baseline approaches:

Fixed Deployment (FD), which keeps the stations at initialization unchanged, and there is no stations deployed/removed during the simulation.

Human Deployment (HD), which uses the deployment plan obtained from the real shared e-mobility data. Particularly, we use an independent validation dataset which hasn't been considered in simulator calibration, and take the actual deployment plan which had been implemented in the real-world.

Revenue-greedy Deployment (REV), which selects the top/bottom stations to deploy/remove based on their historical averaged net revenue. The amount of stations to be deployed/removed is obtained by sampling a distribution learned from the validation set as in HD, i.e., the scale of changes in stations is similar with HD, but the decisions on deployment are revenue-greedy.

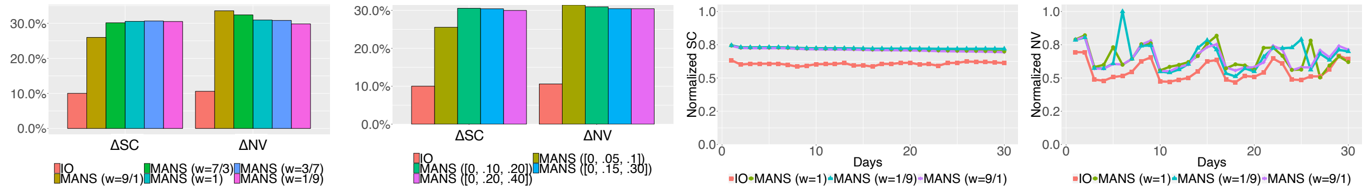
Coverage-greedy Deployment (COV), which is also heuristic-based like REV, but deploys/removes the stations according to their service coverage, i.e., it prefers to deploy stations that tend to satisfy more demand and cover more POIs.

One-time Optimization (OO), which is our implementation of the the state-of-the-art charger location optimization algorithm in [1]. It assumes that the average user demand at each station can be estimated from historical data, and tries to find an one-time optimal station deployment snapshot that maximize both POI coverage and the demand satisfied rate.

Incremental Optimization (IO), which is the incremental version of OO (similar to the work in [2]). Unlike OO which only optimizes station deployment once and uses the solution throughout the episode, this IO performs optimization and finds the best set of stations to maintain at each timestamp.

TABLE I: Overall Performance of deployment optimization by different approaches. ΔSC and ΔNV are obtained with respect to Human Deployment (**HD**). For our MANS, w=1/9 prefers rewards in SC, while w=9/1 reward more on NV. Best performance values are shown in red, and second best in blue.

	FD	**HD**	**REV**	**COV**	**OO**	**IO**	**MANS** (w=1)	**MANS** (w=1/9)	**MANS** (w=9/1)
SC	0.5511	0.5529	0.5680	0.6113	0.6054	0.6063	0.7194	0.7192	0.6942
ΔSC	–	–	3.07%	10.92%	9.85%	10.02%	30.53%	30.50%	25.97%
ΔNV	–	–	11.49%	2.49%	9.86%	10.60%	30.92%	29.82%	33.55%



(a) Impact of Weight w

(b) SC/NV vs. Action Scales

(c) SC over 30 days w.r.t w

(d) NV over 30 days w.r.t w

Fig. 4: Performance of the proposed MANS approach under different parameter settings (weight w and action scale shown in (a) and (b)), and temporal demand distributions (shown in (c) and (d)). Results of **IO** algorithm included as baseline.

All the competing approaches are implemented with TensorFlow 1.14.0, and trained with NVIDIA 2080Ti GPUs. We set the same initialization of deployment snapshot S_0 to all the competing approaches, i.e., they start from the same situation, and try to learn the optimal deployment plan from there. As discussed in Sec. V-B, we consider shared-weights controller for our agents, and train in parallel using the multi-simulation technique to improve efficiency as in Sec. V-C. We evaluate the competing approaches against the following metrics:
Service Coverage (SC) as defined in Sec. IV-A, which is the combination of satisfied demand rate and POI coverage; and **Net Revenue Value (NV)**, which is calculated as the GMV generated by the deployment plan subtracts the cost on deployment, as defined in Sec. III-D.

To be fair, instead of directly comparing SC and NV of the competing approaches, we report the improved percentages of SC and NV with respect to the baselines **HD**, indicating the performance gap between the searched plans and the actual deployment process conducted by human.

B. Experimental Results

Overall Deployment Optimization Performance. This set of experiments study the overall performance of the competing approaches. The results are shown in Table. I. Firstly, we see that there is very little improvement in both SC between **HD** and **FD**, which means that the actual deployment plan does not offer much benefit. This is as expected because the human knowledge is only applicable to that particular case, but can't generalize to unseen scenarios in simulation. We also see the two greedy-based approaches **REV** and **COV** manage to improve NV and SC respectively, but for the other metrics which are not explicitly optimized, the improvements are marginal. On the other hand, the one-time optimization approach **OO** strikes a better balance, offering about 10% improvement in both NV and SC. This is because **OO** uses a demand-aware heuristics, which jointly optimizes the POI coverage and the amount of demand satisfied, leading to better SC while also indirectly improving NV. The incremental optimization approach **IO** further improves the performance, in that it is able to dynamically optimize deployment plan over time, but the gap with **OO** is not significant. This is because that essentially it just re-runs **OO** at each timestamp, while its knowledge about the demand is still estimated from historical data, in the same way as **OO**. Comparing to the state-of-the-art **IO**, our approach **MANS** (with w=1) offers about 20% improvement in NV and in SC. The gain comes from: i) we directly optimize NV in our reward structure, and ii) instead of relying on estimates from historical data which is essentially to optimize on a proxy task, we directly learn by trial-and-error on the target task using deep reinforcement learning.

Deployment Behaviours under Different Rewards. In this set of experiments, we investigate the impact of the weight w between the two terms in our reward structure, as discussed in Eq. (3). Here we vary w in the range [9/1, 7/3, 1, 3/7, 1/9], which gives different importance to the two reward terms g^{SC} and g^{PM}. We train our controller under different w, and report the performance of those variants in Fig. 4a as well as Table. I. We see that as we decrease the weight, NV drops while SC increases. This is expected, as we essentially give more reward on SC, while ignoring actions that lead to high NV. On the other hand, we also see that SC is less sensitive to w than NV. We see that the increase in SC is very small while w decreases from 7/3 to 1/9. This is because that SC cares about the ubiquity of the service, and thus once the stations in the deployment plan can cover the critical mass of the candidate stations, SC tend to be stable. Fig. 4c and Fig. 4d shows the system performance under different w as a function of the demand distribution over temporal domain (for 30 days). In particular, we show the 30 day trend of SC and NV for w = [9/1, 1, 1/9]. We see that under different deployment plans SC remains relatively flat. However, the NV is more sensitive, as shown in Fig. 4d. We see clearly that if we reward g^{PM} heavily, the NV on certain days is much higher. It is possible that under such settings, our controller learns to deploy stations temporarily at hot spots on particular days, serving substantial amount of high-value orders. This may bring extra income but won't necessarily increase the service coverage, since the POIs may already be well covered by neighbouring stations.

Performance vs. Deployment Variability. Finally in this set of experiments, we evaluate the performance of our approach with respect to different level of deployment variability considered in our controller. As discussed in Sec. V-A, our approach uses a hierarchical controller where the high-level controller generates a categorical *deployment prior* (S_t^{i+}, S_t^{i-}), determining roughly how many new/existing stations should be deployed/removed for each category. Here we vary the magnitude of this deployment prior, and consider four *action scales*: [0, 0.05, 0.1], [0, 0.1, 0.2], [0, 0.15, 0.3], and [0, 0.2, 0.4], each of which has three levels. Here 0 means there should be no stations to be deployed or removed. Therefore, the first scale [0, 0.05, 0.1] means that at each timestamp there could be no, 5% or 10% of the total stations removed or deployed. Intuitively, this scale specifies how much our controller can explore in the search space. For instance, if the highest scale [0, 0.2, 0.4] were selected, in the extreme cases, the controller can add 40% of new stations, while removing 40% of the existing ones in one timestamp. Fig. 4b shows the improved SC and NV of different variants of our controller compared with the baseline **HD**. We see that as we increase the scale, i.e., our controller has more room to explore, the NV is relatively stable, but drops a bit as the scale reaches the highest. This means our approach can robustly optimize towards the goal of NV, while at high scales the increased randomness in generated deployment plans may deteriorate NV, e.g., deploying more stations may cost too much. On the other hand, for SC we see that it generally increases, which is also expected, since if more stations are allowed to be altered, the controller is likely to propose plans that quickly fill up the candidate stations to improve the service coverage.

VII. Discussion

Practicality. In practice, the cost of building mobility infrastructure in urban space (e.g., developing parking spaces) can be prohibitively expensive, where dynamical deployment of stations for shared e-mobility systems seems to be an impossible mission. However in the real world, instead of building/owning their own parking/charging infrastructure, those systems typically rent a certain number of parking spaces or charging docks in existing car parks, e.g., those operated in big shopping centres or transportation hubs. Therefore, the shared e-mobility systems can actually adjust their infrastructure in a flexible way without incurring significant overhead. For instance, if some charging stations / parking spaces are not needed during certain days, they will be "returned to" the car park providers, e.g., becoming available to the general public, and thus no extra cost would be imposed to the shared e-mobility systems. For the real-world shared e-mobility system studied in our paper, from the historical data we find that there are already stations which are only rented for a few days per week (e.g., weekends), while during the other days they are labelled as "off-line" in the app, and thus not accessible by the users. This confirms the potential and practicality of our approach, in that if we can discover deployment plans over space and time that reflect such knowledge, e.g., we only need certain stations on weekends, the shared mobility systems could better understand their needs and negotiate a better deal with the car park providers before deploying their infrastructure.

Collaborative vs. Competitive Agents. In typical MARL formulation, the behaviour of agents could be either collaborative or competitive. In our approach, we find that the agents tends to be collaborative, since in our case each agent manages the infrastructure deployment of a local region, while we explicitly allow them to observe both their local and global states. This enables the agents to interact with not just local environment, but also learn to better cooperate with the others that have connections with them. We observe that agents of neighboring regions, or those with high correlations in terms of user demand (i.e., users frequently travel from one region to another) tend to deploy stations together, or within close temporal proximity. This is also because our budget function is more tolerating, i.e., we only requires the deployment cost should not exceed the total income (GMV), encouraging the agents to deploy more stations in seek for better income and service coverage (as shown in Eq. (3)). In cases where the budget is tight (e.g., more penalty in the reward function), the agents may also exhibit competitive behaviour, which would compete for limited resources to obtain more reward.

Limitations. Our approach uses reinforcement learning to search for better infrastructure deployment strategies, which essentially works in the way of trail-and-error. Therefore, unlike the exact or analytical optimization methods such as [1] and [2], our approach is not *complete*, i.e., there is no theoretical guarantee that our approach will always converge to the optimum. However, the advantage of our approach is that it can capture the complex interactions between the users and the system in this infrastructure optimization problem, while the exact or analytical optimization approaches often need to make oversimplified assumptions to make the problem tractable. On the other hand, our approach uses sampling to propose and evaluate potential deployment plans, which is naturally less computationally efficient than the exact approaches that rely on heuristics. This could be a limitation for real-time optimisation, but in our context the deployment plans can be learned offline, i.e., the deployment plans could be precomputed before actually implementing them, and therefore computational efficiency won't be a major limiting factor in practice.

VIII. Conclusion

In this paper, we investigate the deployment optimization problem for shared e-mobility systems, aiming to find the optimal deployment plan with which the system can achieve the desired service coverage, net revenue, and demand satisfied rate while being profitable. We design a high-fidelity simulation environment to capture the key operational details the shared e-mobility systems at fine granularity, and calibrate the environment with rich data collected from a real-world shared e-mobility system over 12 months. To tackle the deployment optimization problem, we propose a novel multi-agent deep neural search algorithm, which employs hierarchical controllers to generate possible deployment plans. The controllers are trained using a multi-simulation paradigm within our simulation environment, which learns to propose better plans in the next iteration. The proposed approach has been evaluated extensively, and experimental results show that: i) our neural search algorithm significantly outperforms both the baseline and state-of-the-art optimization approaches, in various metrics including service coverage and net revenue; ii) by adjusting the weight between objectives, our search algorithm can optimize towards different directions, providing desired deployment plan for different cases; iii) the proposed hierarchical controller architecture reduces the search cost, where tuning the action space of the high-level controller has substantial impact on balancing exploration and exploitation.

Acknowledgment This work was supported in part by The Alan Turing Institute under the EPSRC grant EP/N510129/1.

References

[1] B. Du, Y. Tong, Z. Zhou, Q. Tao, and W. Zhou, "Demand-aware charger planning for electric vehicle sharing," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1330–1338.

[2] Q. Liu, Y. Zeng, L. Chen, and X. Zheng, "Social-aware optimal electric vehicle charger deployment on road network," in *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2019, pp. 398–407.

[3] Y. Tong, Y. Chen, Z. Zhou, L. Chen, J. Wang, Q. Yang, J. Ye, and W. Lv, "The simpler the better: a unified approach to predicting original taxi demands based on large-scale online platforms," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 1653–1662.

[4] M. Lowalekar, P. Varakantham, S. Ghosh, S. D. Jena, and P. Jaillet, "Online repositioning in bike sharing systems," in *Twenty-seventh international conference on automated planning and scheduling*, 2017.

[5] V. Nanda, P. Xu, K. A. Sankararaman, J. P. Dickerson, and A. Srinivasan, "Balancing the tradeoff between profit and fairness in rideshare platforms during high-demand hours," in *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, 2020, pp. 131–131.

[6] T. Pamuła and W. Pamuła, "Estimation of the energy consumption of battery electric buses for public transport networks using real-world data and deep learning," *Energies*, vol. 13, no. 9, p. 2340, 2020.

[7] G. Wang, X. Xie, F. Zhang, Y. Liu, and D. Zhang, "bcharge: Data-driven real-time charging scheduling for large-scale electric bus fleets," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, Dec 2018, pp. 45–55.

[8] Y. Yuan, D. Zhang, F. Miao, J. Chen, T. He, and S. Lin, "p^2 charging: Proactive partial charging for electric taxi systems," in *ICDCS*, 2019.

[9] A. Sarker, H. Shen, and J. A. Stankovic, "Morp: Data-driven multi-objective route planning and optimization for electric vehicles," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 1, no. 4, pp. 162:1–162:35, Jan. 2018.

[10] G. Wang, X. Chen, F. Zhang, Y. Wang, and D. Zhang, "Experience: Understanding long-term evolving patterns of shared electric vehicle networks," in *MobiCOM*, 2019.

[11] F. Soriguera, V. Casado, and E. Jiménez, "A simulation model for public bike-sharing systems," *Transportation Research Procedia*, vol. 33, pp. 139–146, 2018.

[12] M. Maciejewski and K. Nagel, "Simulation and dynamic optimization of taxi services in matsim," *Transportation Science*, pp. 000–000, 2013.

[13] K. T. Seow, N. H. Dang, and D.-H. Lee, "A collaborative multiagent taxi-dispatch system," *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 3, pp. 607–616, 2009.

[14] C. Wei, Y. Wang, X. Yan, and C. Shao, "Look-ahead insertion policy for a shared-taxi system based on reinforcement learning," *IEEE Access*, vol. 6, pp. 5716–5726, 2017.

[15] M. Luo, B. Du, K. Klemmer, H. Zhu, H. Ferhatosmanoglu, and H. Wen, "D3p: Data-driven demand prediction for fast expanding electric vehicle sharing systems," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 1, pp. 1–21, 2020.

[16] L. Pan, Q. Cai, Z. Fang, P. Tang, and L. Huang, "A deep reinforcement learning framework for rebalancing dockless bike sharing systems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 1393–1400.

[17] G. Pan, L. Fu, Q. Chen, M. Yu, and M. Muresan, "Road safety performance function analysis with visual feature importance of deep neural nets," *IEEE/CAA Journal of Automatica Sinica*, vol. 7, no. 3, pp. 735–744, 2020.

[18] M. Li, Z. Qin, Y. Jiao, Y. Yang, J. Wang, C. Wang, G. Wu, and J. Ye, "Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning," in *The World Wide Web Conference*, 2019, pp. 983–994.

[19] M. Luo, W. Zhang, T. Song, K. Li, H. Zhu, B. Du, and H. Wen, "Rebalancing expanding ev sharing systems with deep reinforcement learning," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*, 2020, pp. 1338–1344.

[20] Y. Li, Y. Zheng, and Q. Yang, "Dynamic bike reposition: A spatiotemporal reinforcement learning approach," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 1724–1733.

[21] K. Lin, R. Zhao, Z. Xu, and J. Zhou, "Efficient large-scale fleet management via multi-agent deep reinforcement learning," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1774–1783.

[22] P. Wu, C.-H. Yang, F. Chu, M. Zhou, K. Sedraoui, and F. S. Al Sokhiry, "Cost-profit trade-off for optimally locating automotive service firms under uncertainty," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 2, pp. 1014–1025, 2020.

[23] P. Wu, F. Chu, N. Saidani, H. Chen, and M. Zhou, "Optimizing locations and qualities of multiple facilities with competition via intelligent search," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[24] P. Wu, F. Chu, N. Saidani, H. Chen, and W. Zhou, "Iot-based location and quality decision-making in emerging shared parking facilities with competition," *Decision Support Systems*, vol. 134, p. 113301, 2020.

[25] L. P. Meng, Q. Kang, C. F. Han, and M. C. Zhou, "Determining the optimal location of terror response facilities under the risk of disruption," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 2, pp. 476–486, 2017.

[26] X. Zuo, B. Li, X. Huang, M. Zhou, C. Cheng, X. Zhao, and Z. Liu, "Optimizing hospital emergency department layout via multiobjective tabu search," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 3, pp. 1137–1147, 2019.

[27] Y. Li, J. Luo, C.-Y. Chow, K.-L. Chan, Y. Ding, and F. Zhang, "Growing the charging station network for electric vehicles with trajectory data analytics," in *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 2015, pp. 1376–1387.

[28] H.-C. Liu, M. Yang, M. Zhou, and G. Tian, "An integrated multi-criteria decision making approach to location planning of electric vehicle charging stations," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 1, pp. 362–373, 2018.

[29] Y. Xiong, J. Gan, B. An, C. Miao, and A. L. Bazzan, "Optimal electric vehicle fast charging station placement based on game theoretical framework," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 8, pp. 2493–2504, 2017.

[30] D. Bertsekas, "Multiagent reinforcement learning: Rollout and policy iteration," *IEEE/CAA Journal of Automatica Sinica*, vol. 8, no. 2, pp. 249–272, 2021.

[31] T. D. Barrett, W. R. Clements, J. N. Foerster, and A. I. Lvovsky, "Exploratory combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1909.04063*, 2019.

[32] Q. Cappart, E. Goutierre, D. Bergman, and L.-M. Rousseau, "Improving optimization bounds using machine learning: Decision diagrams meet deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 1443–1451.

[33] J. Huang, M. Patwary, and G. Diamos, "Coloring big graphs with alphagozero," *arXiv preprint arXiv:1902.10162*, 2019.

[34] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[35] P. M. Kebria, A. Khosravi, S. M. Salaken, and S. Nahavandi, "Deep imitation learning for autonomous vehicles based on convolutional neural networks," *IEEE/CAA Journal of Automatica Sinica*, vol. 7, no. 1, pp. 82–95, 2019.

[36] O. Tremblay and L.-A. Dessaint, "Experimental validation of a battery dynamic model for ev applications," *World electric vehicle journal*, vol. 3, no. 2, pp. 289–298, 2009.

[37] "Amap api," https://lbs.amap.com/api, 2020, accessed: 2020-04-29.

[38] "Lianjia," https://sh.lianjia.com, 2020, accessed: 2020-05-12.

[39] "Shanghai qiangsheng taxi data," https://sodachallenges.com/datasets/taxi-gps/, 2020, accessed: 2020-04-29.

[40] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

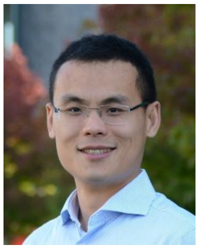
[41] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

Dr. Man Luo is a Lecturer in the Department of Computer Science, University of Exeter, UK. Previously she was a research associate in the Urban Analytics Theme at the Alan Turing Institute, studying potential policy impact in the COVID-19 pandemic and building resilience against future shocks. She received both her PhD and B.Sc (first class) in Computer Science from the University of Warwick. Her research interests lie in urban analytics, data systems and spatio-temporal deep learning.

Bowen Du received his B.Eng and M.Eng degrees in software engineering from Tongji University, Shanghai, China, in 2013 and 2016 respectively. He is currently working toward his PhD degree in Computer Science at the University of Warwick, Coventry, UK. His research interests include cyber-physical systems, mobile computing, and artificial intelligence in sensor systems.

Konstantin Klemmer holds a Bachelors degree in Economics from the University of Freiburg and a Masters in Transportation from Imperial College London and University College London. He currently pursues a PhD in Computer Science at the University of Warwick. His research focuses on machine learning for geospatial data with applications to urban areas and earth systems.

Dr. Hongming Zhu is an Associate Professor in the School of Software Engineering, Tongji University. His research interests include knowledge graph and remote sensing data analysis. He has published more than 40 academic papers in international conferences and various core journals, and obtained 5 invention patents.



Dr. Hongkai Wen is an Associate Professor in the Department of Computer Science, University of Warwick. He obtained his D.Phil in Computer Science from Keble College, the University of Oxford in 2014, and later worked as a post-doctoral researcher in Oxford Computer Science and Robotics Institute. He holds joint appointment as a Senior Research Scientist at Samsung AI Center Cambridge, and is a Fellow of the Alan Turing Institute. His research interests include scalable AI systems, on-device machine learning and pervasive data science.