CIRRELT-2024-04

A Reinforcement Learning Approach for Dynamic Rebalancing in Bike-Sharing Systems

Jiaqi Liang
Sanjay Dominik Jena
Defeng Liu
Andrea Lodi

January 2024

A Reinforcement Learning Approach for Dynamic Rebalancing in Bike-Sharing Systems

Jiaqi Liang[1,2,*], Sanjay Dominik Jena[1,3], Defeng Liu[1,2], Andrea Lodi[1,4]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

[2] Department of Mathematics and Industrial Engineering, Polytechnique Montréal

[3] Analytics, Operations, and Information Technologies Department, School of Management, Université du Québec à Montréal

[4] Cornell Tech and Technion - IIT, 2 West Loop Road, New York, 10044, USA

Abstract. Bike-Sharing Systems provide eco-friendly urban mobility, contributing to the alleviation of traffic congestion and to healthier lifestyles. Efficiently operating such systems and maintaining high customer satisfaction is challenging due to the stochastic nature of trip demand, leading to full or empty stations. Devising effective rebalancing strategies using vehicles to redistribute bikes among stations is therefore of uttermost importance for operators. As a promising alternative to classical mathematical optimization, reinforcement learning is gaining ground to solve sequential decision-making problems. This paper introduces a spatio-temporal reinforcement learning algorithm for the dynamic rebalancing problem with multiple vehicles. We first formulate the problem as a Multi-agent Markov Decision Process in a continuous time framework. This allows for independent and cooperative vehicle rebalancing, eliminating the impractical restriction of time-discretized models where vehicle departures are synchronized. A comprehensive simulator under the first-arrive-first serve rule is then developed to facilitate the learning process by computing immediate rewards under diverse demand scenarios. To estimate the value function and learn the rebalancing policy, various Deep Q-Network configurations are tested, minimizing the lost demand. Experiments are carried out on various datasets generated from historical data, affected by both temporal and weather factors. The proposed algorithms outperform benchmarks, including a multi-period Mixed-Integer Programming model, in terms of lost demand. Once trained, it yields immediate decisions, making it suitable for real-time applications. Our work offers practical insights for operators and enriches the integration of reinforcement learning into dynamic rebalancing problems, paving the way for more intelligent and robust urban mobility solutions.

Keywords: Bike-sharing systems, dynamic rebalancing, Markov decision process, reinforcement learning, Deep Q-Network

---

* Corresponding author: jiaqi.liang@polymtl.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2024

1. Introduction

The blossoming of Bike-Sharing Systems (BSS) in cities worldwide marks a significant stride towards easing traffic congestion and curtailing CO_2 emissions. By 2022, the global landscape boasted over 1,900 operational BSS, collectively fielding close to 9 million bikes [32]. Despite their widespread adoption, optimizing rebalancing strategies remains a critical research area for further exploration and refinement. Due to the stochastic nature of user behaviors and high demand in certain areas during peak hours, the stations are often unbalanced, being either full or empty, which results in user dissatisfaction and lost demand. As a remedy, BSS operators use vehicles to actively redistribute bikes across the station network.

The optimization problems for bike-sharing repositioning can be divided into the Static Bicycle Repositioning Problem (SBRP) and the Dynamic Bicycle Repositioning Problem (DBRP) [36]. The former, also referred to as overnight rebalancing, optimizes the initial inventories of the stations without reacting to their changing inventories throughout the day. In contrast, the DBRP explicitly rebalances intraday, taking into account the synergies with bikes relocated by users. As such, rebalancing operations are performed multiple times during the day such that the unmet demand of bike rentals and returns, especially during demand peak hours, is minimized. We here focus on the DBRP, as it allows for a higher impact on user satisfaction.

Existing works primarily model the DBRP via Mixed-Integer Programming (MIP) or as Markov Decision Process (MDP). The majority of the literature applies MIP [e.g., 13; 28; 48; 46], leveraging its flexibility and frequent application within industrial decision-making processes. However, MIP models typically simplify the planning problem, e.g., by discretizing the planning horizon into multiple time-periods and assuming that each vehicle only visits one station per time-period. Such an assumption is artificially imposed and does not correspond to reality. It may restrict the capability of BSS to effectively coordinate multiple vehicles and hence hinder the system to reach its full potential. In contrast, MDPs present a viable alternative, framing the DBRP as a sequential decision-making problem [1]. Nevertheless, the tractability of MDP models tends to heavily depend on the dimensions of the state and action spaces, confining their applicability to limited problem settings, such as single-vehicle operations [see, e.g., 7; 26; 39], or small station networks [see, e.g., 17; 20; 26]. Although several studies applied dynamic programming or heuristic policies to trackle the MDP [see, e.g., 17; 7; 8, etc], these approaches rely on having a complete and accurate model of the environment and can face challenges in high-dimensional state and action spaces. Reinforcement Learning (RL), on the other hand, learns optimal policies directly from interactions with the environment, obviating the need for an explicit model of the environment dynamics. This makes RL applicable to scenarios where the system dynamics are complex, unknown, or hard to model accurately. Thus, RL techniques offer a potential to solve MDP models for DBRP, an area that remains underexplored.

Our study aims at proposing a spatio-temporal RL-based algorithm for dynamic rebalancing problems in BSS with multiple vehicles under a continuous time framework that does not require time discretization. We formulate the DBRP as a Multi-agent Markov Decision Process (MMDP) whose state and action spaces are designed to capture the dynamics and uncertainties of the system. Utilizing a Deep Q-Network (DQN), which employs Neural Networks to approximate the optimal action-value function, we estimate the total expected rewards of rebalancing actions, targeting to minimize lost demand. A fine-grained simulator is developed to obtain immediate rewards under different demand scenarios. Experiments are conducted on a trip dataset generated based on historical trips, weather conditions, and temporal factors.

The main contributions of this study can be summarized as follows: (1) We propose an MMDP model to address the DBRP, capable of coordinating simultaneous rebalancing operations by multiple vehicles without the need for time discretization. (2) We present a spatio-temporal RL-based algorithm with DQN to solve the model, aiming at minimizing lost demand for networks of up to 60 stations. (3) We introduce a highly realistic simulator to estimate the rewards under the first-arrive-first-serve rule, handling different demand scenarios generated from weather and temporal features. (4) We conduct an experimental evaluation and compare to a MIP baseline model, demonstrating the benefits of our algorithm and its effectiveness in reducing lost demand. The proposed RL approach not only reduces the lost demand by up to 27.12% on average, but also generates planning solutions in a matter of milliseconds, which makes it ideal for dynamic use in practice.

The remainder of the paper is organized as follows. Section 2 reviews related literature on the DBRP for BSS and summarizes the existing MDP models and RL algorithms for this planning problem. Section 3 describes the model formulation based on MDPs under continuous time. Section 4 presents the methodology to estimate the value function for lost demand under certain actions. A simulator considering stochastic demand is discussed to calculate the immediate rewards. Numerical tests and analyses are illustrated in Section 5. This is followed by the conclusions in Section 6.

2. Literature Review

In this section, we first review literature on the DBRP in BSS (Section 2.1), which has predominantly been approached by MIP. We then discuss the existing efforts of using Markov Decision Processes for the DBRP in Section 2.2. Finally, existing RL approaches for planning problems related to the DBRP are summarized in Section 2.3.

2.1. Dynamic bicycle repositioning and MIP models

Dynamic rebalancing strategies for BSS are primarily categorized into two types: user-based rebalancing and operator-based rebalancing [41]. User-based rebalancing entails incentivizing users to rent or return bikes at specific stations, as discussed by [14]. This strategy is predominantly utilized in dockless BSS. In contrast, operator-based rebalancing involves the active participation of a dedicated rebalancing fleet (commonly vehicles) to redistribute bikes, which is prevalent in station-based BSS. According to a recent statistical report by [32], station-based systems continue being more common than dockless systems. Even when addressing rebalancing challenges for the latter, researchers tend to divide the station network into sub-clusters, each of which can be regarded as an individual station [see, e.g., 10; 26; 43; 47]. Given the prevalence and relevance of station-based systems, our study opts to concentrate on operator-based rebalancing in station-based BSS.

The MIP models in DBRP usually employ time discretization, partitioning the temporal dimension into specific periods. Time-periods with equal length is most common in the literature [see, e.g., 9; 48; 24]. To cast the complexity of DBRP planning into tractable MIP models, existing approaches rely on a wide range of modeling assumptions [21]. Indeed, time discretization itself requires to aggregate all rental and return demand for each station occuring within the same time-period. This raises a multitude of issues, including assumptions on the sequence in which rentals, returns and rebalancing operations should be carried out within the same time-period, as well as priority rules to ensure a sufficiently realistic allocation of available bikes to rental demand whenever the latter exceeds the former. Furthermore, it is necessary to constrain each vehicle to rebalance a maximum number of stations (typically, one station) per time-period to ensure that the produced planning solution is time-feasible in practice. Finally, selecting an appropriate length for each time-period is a non-trivial task. Time-periods that are too long may result in idle time of vehicles, which have to wait until the next time-period before starting a new rebalancing operation. They also aggregate over too many rentals and returns, losing accuracy of the original sequences of such user demand. In contrast, time periods that are too short, may result in intractably large optimization models and may not ensure that the vehicles have sufficient time to relocate to stations farther away.

In addition to such challenges, explicitly modeling the stochastic nature of trip demand would require the use of stochastic optimization models, which results in even more complex optimization models [see, e.g., 25; 24], limiting the use of such models to smaller problem instances. While MIP models with discretized time-periods are prevalent in the BSS literature, the above mentioned disadvantages highlight the necessity for models that better align with the dynamic and uncertain nature of rental and return demand, as well as the asynchronous operations of the rebalancing fleet.

2.2. Markov Decision Process

Markov Decision Processes are mathematical models for sequential decision-making in stochastic environments, and have received increasing attention in recent years. The DBRP can be represented as such a sequential decision-making problem [see, e.g., 38], where the agents, represented by the vehicles, interact within a complex environment, here composed of a station network and the stochastic trip demand. The decision-making process involves a series of decision epochs, at each of which the vehicles evaluate the current state of the system and make rebalance decisions, typically with the goal of catering to user demand effectively. Despite its relevance and potential for application to the DBRP, the adoption of MDPs in this application domain has been relatively recent, with a limited but growing body of work in academic research. In DBRP, the state space (representing various configurations of bike availability across stations and vehicles) and the action space (choices of vehicle movements and rebalancing operations) can be vast, leading to exponential growth in computational complexity and requirements [4]. As a result, MDP models for DBRP tend to be limited by the immense complexity of the problem space, often rendering the computation of an optimal policy impractical within a reasonable time [17].

[7] conceptualized the DBRP as an MDP and introduced a dynamic lookahead policy heuristic, albeit limited to scenarios involving a single rebalancing vehicle. Building upon this work, [8] extended the model to incorporate multiple vehicles, proposing a coordinated lookahead policy to simultaneously address inventory and routing decisions. [17] applied MDP to develop the decision-support tool for DBRP, aiming to minimize the rate of unsatisfied users who find their station empty or full. They implemented a one-step policy improvement method, which incrementally refines the strategy by focusing on the immediate next decision rather than a series of future actions, to identify priority stations. However, their approach segments the planning horizon into periods of equal length, which is subject to the same drawbacks as a discretized planning horizon typically engaged in MIP models. [26] designed a policy function approximation algorithm and applied the optimal computing budget allocation method to search for the optimal policy parameters for an MDP model of

dynamic rebalancing with one single vehicle.

As an extension of traditional MDP, Multi-agent Markov Decision Processes allow for concurrently operating multiple decision-making agents. Within an MMDP framework, each agent aims to determine its optimal strategy, while considering both its actions and those of other agents. As multi-agent systems become more prevalent in various domains such as robotics [27; 16], transportation [2; 12], and gaming [18; 29], the importance of developing effective and scalable algorithms for MMDPs continues to grow [44].

The existing literature clearly demonstrates the potential of MDP to more effectively address DBRP planning. Its application in this domain is still evolving and further research is required to overcome challenges, such as their computational complexity, the choice of time discretization, and vehicle synchronization for more accurate and efficient decision-making.

2.3. Reinforcement Learning

Reinforcement learning is a subfield of Machine Learning (ML) that focuses on training agents to make sequential decisions by interacting with an environment and learning optimal strategies through a trial-and-error process [40]. RL has witnessed profound advancements in the realm of multi-agent systems, offering promising avenues for algorithmically learning effective decision-making strategies [31]. This is particularly relevant given the complexity of MDPs encountered in DBRP, where the need for adaptive solutions is paramount. To be specific, RL allows the system to learn from past experiences and adjust its strategies dynamically in response to the changing environment, shaped by the stochastic nature of user demand in BSS.

The success of RL on various optimization problems [5; 6; 23] motivates the use of RL in BSS. [42] further highlighted that rebalancing shares many environmental characteristics (i.e., complex and high dimensional environment, constantly changing system drivers, and closed-loop system) with gaming, finance, and marketing, where RL is typically applied.

In the domain of BSS, the majority of the studies on RL focuses on user-based rebalancing. [33] presented a RL algorithm based on a deep deterministic policy gradient method to offer users monetary incentives and to suggest alternative pick-up/drop-off stations to rebalance the system. Similarly, [11] adapted a deep reinforcement learning framework for an MDP model, concentrating on learning differentiated incentive prices for bike rebalancing. [15] proposed a dynamic incentivization system based on RL, undertaking a comparative analysis of three policy-gradient-based RL methods. In a distinct vein, [37] presented significant findings utilizing a reinforcement-based dynamic incentive mechanism, highlighting the efficacy of RL-based solutions, particularly in terms of the average percentage improvement in service level per hour.

For operator-based rebalancing, [19] proposed a static rebalancing method using a policy gradient-based RL method to enhance user experience and reduce operational costs. As opposed to static rebalancing (mostly during the night), dynamic rebalancing enables efficient resource allocation in response to varying demand during the day. Here, [20] proposed a spatio-temporal RL framework to tackle the rebalancing problem. The authors first proposed a clustering algorithm to group the stations. Then a spatio-temporal RL model is designed for each cluster to learn a rebalancing policy in it. [42] proposed a distributed RL solution with transfer learning to decide how many bikes to rebalance at each station. This approach assigns a unique agent to each station, with each agent tasked with identifying and then combining the best rebalancing strategy for their respective station. The model avoids an exponentially expanding action space but requires a significant processing power to support the parallel learning of all agents. The routing part is not included and needs to further consolidate the rebalancing strategies for the system. Here, the efficient design of the state and action space is crucial, influencing both the complexity of the MDP and the quality of the final solution obtained.

[26] and [39] focused on the DBRP with a single vehicle, while [20] considered multiple vehicles rebalancing in smaller clusters with less than 30 stations. We here consider an algorithm for a station network up to 60 stations with multiple vehicle rebalancing under continuous time framework, aiming to address the challenges associated with a larger problem size and the synchronization of vehicles. The paper of [45] is close to our study, proposing a RL based rebalancing model and generating rebalancing solutions with a DQN, yet under discrete time slots. As a result, a simulator within discretized time was employed, ignoring the order of rentals, returns, and rebalancing operations and resulting in a less realistic reward function.

Nevertheless, the immediate reward of the agents' actions can be, sufficiently realistically, computed within simulators in RL. Given that such simulators can typically be executed within short computing times, they are not subject to the same limitations as optimization models, and can therefore model rebalancing operations and trip behavior much more realistically. This is an important advantage, which we aim at exploiting in this work. Thus, we introduce a fine-grained event-driven simulator designed to more accurately capture system dynamic as in reality, as opposite to the current RL-based studies on DBRP with a relatively rough simulator.

3. Multi-agent Dynamic Rebalancing

In this section, we first define the planning problem in Section 3.1. We then formulate the DBRP as a Multi-agent Markov Decision Process in Section 3.2. Finally, we elaborate on the architecture of the continuous time framework in Section 3.3.

3.1. Problem Definition

We consider the DBRP defined over a finite planning horizon, a BSS with $|N|$ stations and a fleet V of vehicles available to rebalance bikes among the stations. All problem parameters are summarized in Table 1. Each station $n \in N$ has a capacity C_n of docks and each vehicle $v \in V$ has a capacity $\hat{C}_v$. The initial state of the system is defined by the bike inventory d_0^n at each station $n \in N$, as well as the inventory p_0^v and location z_0^v of each vehicle $v \in V$. The distance and transit time between any two stations $i \in N$ and $j \in N$ are given by $D_{i,j}$ and $R_{i,j}$, respectively. Loading or unloading one bike (from the vehicle to the station, or vice-versa) is estimated to take β minutes. The system encounters lost demand when rental or return requests cannot be satisfied due to the absence of bikes or available docks, respectively. The primary goal is to optimize the rebalancing strategies for vehicles across the station network, minimizing the total lost demand. Parameters and variables represented in bold are tuples, matrices, or sets whose elements are tuples or matrices.

Table 1: Input parameters of the BSS rebalancing planning problem

Parameter	Definition
N	The set of stations
V	The set of vehicles
C_n	The capacity of station $n \in N$
$\hat{C}_v$	The capacity of vehicle $v \in V$
$D_{i,j}$	The distance between station $i \in N$ and $j \in N$
$R_{i,j}$	The transit time between station $i \in N$ and $j \in N$
β	The time (in minutes) for loading/unloading one bike
d_0^n	The initial number of bikes in station $n \in N$
p_0^v	The initial number of bikes in vehicle $v \in V$
z_0^v	The initial location (station) of each vehicle $v \in V$

3.2. Multi-agent Markov Decision Process Model

The DBRP can be formulated as an MDP, and more specifically as an MMDP to accommodate multiple rebalancing vehicles within the system. The related notation is summarized in Table 2. An MMDP is characterized by the tuple $(\boldsymbol{S}, \boldsymbol{A}, W, R, \gamma)$, which includes $\boldsymbol{S}$ as the set of system states, $\boldsymbol{A}$ as the set of actions for the agents, W

as the state transition probabilities, and R as the reward function with the numerical output for the cumulative discounted total reward. Discount factor γ is defined to emphasize a higher confidence in short-term rewards as opposed to rewards collected farther into the future. In an MMDP, agents operate through a sequence of decision epochs, each involving action choices leading to new states and rewards based on the chosen actions. The structure and dynamics of these components will be elaborated in the following sections to provide a thorough understanding of the MMDP model applied to DBRP.

Table 2: Notation of MMDP model for rebalancing

Symbol	Definition
K	The sequence of decision epochs
$\boldsymbol{S}$	The set of states
T	The time of each decision epoch, $T = \{t_1, ..., t_{\lvert K\rvert}\}$
$\boldsymbol{d_k}$	The number of bikes available at stations, $\boldsymbol{d_k} = (d_k^n, \forall n \in N)$
b_k^v	The station that vehicle $v \in V$ is or just departed from at time t_k
g_k^v	The station that vehicle $v \in V$ is heading to at time t_k
p_k^v	The inventory of vehicle $v \in V$ at time t_k
m_k^v	The estimated arrival time of vehicle v at the station g_k^v at time t_k
o_k^v	Remaining rebalancing operations for vehicle $v \in V$ at time t_k
$\boldsymbol{A}$	The set of actions
l_k^v	The rebalancing decision at the current station for vehicle $v \in V$ at time t_k
z_k^v	The routing decision for vehicle $v \in V$ at time t_k
Π	The set of policies
γ	Discount factor

3.2.1. State Space

At each decision epoch $k \in K$, there is an associated state $\boldsymbol{S_k} \in \boldsymbol{S}$ that represents a comprehensive system status, including information related to time, stations, and vehicles. The state space encompasses all the potential conditions or configurations that the agent or system can encounter at any given moment. We denote $\boldsymbol{S_k} = (t_k, \boldsymbol{d_k}, \boldsymbol{H_k})$, where t_k is the current time and $\boldsymbol{d_k} = (d_k^n, \forall n \in N)$ represents the inventory of each station. The vehicle status is denoted as $\boldsymbol{H_k} = (b_k^v, g_k^v, p_k^v, m_k^v, o_k^v, \forall v \in V)$, where b_k^v is the current station at which vehicle v is located or has just departed from, g_k^v is the next station that vehicle v is traveling to, p_k^v indicates the current inventory of vehicle v, and m_k^v indicates the estimated time of arrival of vehicle v at station g_k^v. Note that if vehicle v is currently rebalancing at station b_k^v, the number of remaining rebalancing operations (i.e., drop offs or pick ups) for that station is indicated by o_k^v. For the sake of clarity and ease of reference, the here used notation can also be found in Table 2.

3.2.2. Action Space

At each decision epoch $k \in K$, an action is selected, indicating both the rebalancing and routing decisions. Let $\boldsymbol{A}$ denote the set of actions, where $\boldsymbol{a_k} \in \boldsymbol{A}$ is defined by (l_k^v, z_k^v). The rebalancing decision at the current station is indicated by l_k^v. A positive l_k^v value indicates that vehicle v should pick up l_k^v bikes, while a negative l_k^v value implies that vehicle v should drop off $|l_k^v|$ bikes. The routing decision are given by z_k^v, indicating the next station that vehicle $v \in V$ will visit. Note that the DBRP is here formulated to accommodate various operational time frameworks, where an action can be made either for all the vehicles during each time-period in a discrete time framework [45] or only for one specific vehicle at a particular point in time [20], where vehicles operate independently without waiting for each other. In our continuous time framework (see Section 3.3), we opt for the latter alternative, i.e., the action at decision epoch $k \in K$ is exclusively associated with a specific vehicle upon its arrival at the allocated station, allowing vehicles to take action independently from the current state of the other vehicles. The vehicles assigned by actions at different decision epochs vary according to the system status, ensuring that decision-making is dynamic and continuously adapts to status changes of the global system.

3.2.3. Transition Function

At decision epoch $k \in K$, the system transitions from state $\boldsymbol{S_k}$ to the next state $\boldsymbol{S_{k+1}}$ upon taking an action. This state transition is governed by a stochastic transition W_{k+1}, which depends on the current state $\boldsymbol{S_k}$, the chosen action $\mathbf{a}_k$, and the likelihood of transitioning to $\boldsymbol{S_{k+1}}$. Essentially, W_{k+1} determines the probability of moving from $\boldsymbol{S_k}$ to $\boldsymbol{S_{k+1}}$ following action $\mathbf{a}_k$. In the context of the here considered DBRP, these transitions are significantly influenced by user rental and return demand during the time duration between two successive states, introducing uncertainties into the system. To represent these state transitions under various demand scenarios, we utilize a simulator to capture the change of state, which will be further described in Section 4.2.

3.2.4. Reward and Policy

At each decision epoch $k \in K$, an action $\boldsymbol{a_k}$ is taken and the system transits to $\boldsymbol{S_{k+1}}$. Accompanying this transition is the realization of an immediate reward r_{k+1}, which is computed based on $\boldsymbol{S_k}, \boldsymbol{a_k}$, and W_{k+1}. In our case, r_{k+1} is *the negative value of lost demand* across all stations occurring from t_k to t_{k+1}.

The expected cumulative discounted return, $R_k = \mathbb{E}[\sum_{j=0}^{K-k-1} \gamma^j r_{k+j+1})]$, reflects the long-term aggregated reward from a series of actions. The discount factor $\gamma \in [0, 1]$ progressively reduces the value of future rewards as decision epoch j increases, reflecting the idea that immediate rewards are often more valuable than future ones.

In MDPs, a planning solution is encoded through a policy $\pi \in \Pi$, where Π denotes the set of all possible policies (policy space). A policy, represented as $\pi(a|s)$, serves as a strategy dictating the probability of taking an action a given state s. The overarching objective is to identify the optimal policy π^* that maximizes the total expected reward over time. The policy can be further quantified by the Q-value function, denoted as $Q^{\pi}(\boldsymbol{S_k}, \boldsymbol{a_k})$, expressing the expected return of a state-action pair under policy π, as defined in Equation (1).

$$Q^{\pi}(\boldsymbol{S_k}, \boldsymbol{a_k}) = \mathbb{E}[R_k|\boldsymbol{S_k}, \boldsymbol{a_k}]. \tag{1}$$

The goal is to determine the optimal Q-function derived to maximize the total expected reward over time, as in Equation (2). However, directly solving for it can be computationally challenging due to the problem complexity.

$$Q^{*}(\boldsymbol{S_k}, \boldsymbol{a_k}) = \max_{\pi} Q^{\pi}(\boldsymbol{S_k}, \boldsymbol{a_k}). \tag{2}$$

In practice, RL algorithms use iterative methods or deep learning approaches to approximate the optimal Q-function. Once an approximation is obtained, the optimal greedy policy can be employed to select the action with the highest Q-value, as defined in (3). This indicates that the probability of selecting action $\boldsymbol{a_k}$ under the policy π is 1 when $\boldsymbol{a_k}$ is the action that maximizes the Q-value for the given state $\boldsymbol{S_k}$, and 0 for all other actions. This approach simplifies decision-making under uncertainty, guiding actions based on the current knowledge of expected rewards.

$$\pi(\boldsymbol{a_k}|\boldsymbol{S_k}) = \begin{cases} 1 & \text{if } \boldsymbol{a_k} = \arg\max_{\boldsymbol{a_k}} Q^{*}(\boldsymbol{S_k}, \boldsymbol{a_k}) \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

3.3. Continuous Time Planning Framework for MMDP

Based on the above notation, our model framework uses multiple agents to enable fully cooperative rebalancing decision-making tasks. Each vehicle is regarded as an agent and the decisions (i.e., actions) taken by the vehicles affect each other.

We illustrate the states and actions within our continous-time framework in Figure 1. The moment vehicle $v_i \in V$ arrives at station $g_k^{v_i}$, all other vehicles either rebalance bikes at their respective stations or relocate to their next stations. At vehicle v_i reaches its designated station, an action $\mathbf{a}_k = (l_k^{v_i}, z_k^{v_i})$ is instantly generated, dictating the next decision for v_i at decision epoch k. Completing action $\mathbf{a}_k$, the system transits to the next state $\boldsymbol{S_{k+1}}$, and an immediate reward is obtained. As previously mentioned, in our context, the immediate reward is given by the negative value of lost rental and return demand occuring within the time segment $[t_k, t_{k+1}]$. This reward is computed within the simulator described in Section 4.2. The duration between two decision epochs is determined by the global system changes, where the next epoch begins each time a vehicle arrives at a station.

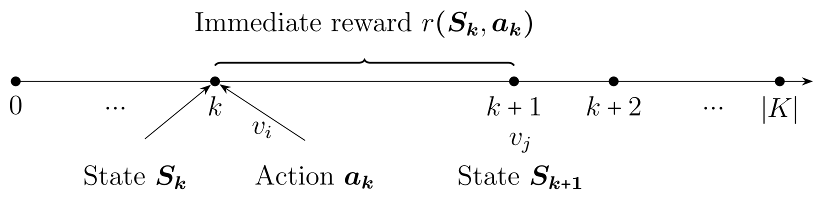
Immediate reward $r(\boldsymbol{S_k}, \boldsymbol{a_k})$
0
···
k
v_i
$k+1$
v_j
$k+2$
···
$|K|$
State $\boldsymbol{S_k}$
Action $\boldsymbol{a_k}$
State $\boldsymbol{S_{k+1}}$

Figure 1: Continuous time planning framework of MMDP

The decision-making process focuses solely on the action for a single vehicle. As a result, the complexity of the action space is significantly reduced compared to the case where actions for all vehicles are taken simultaneously [45]. This framework enables each vehicle to react dynamically and instantaneously to the present state, without the necessity to wait for other vehicles. Additionally, it reflects realistic operational conditions. It is crucial to note that although there is no direct communication or coordination between the vehicles, the status of all vehicles is given within the current state, facilitating an indirect form of situational awareness. In other words, each vehicle operates autonomously, making decisions based solely on the global state of the system (i.e., knowing the status of all the other vehicles) as opposed to engaging in collaborative strategies with the other vehicles.

Adopting a simultaneous action approach for all vehicles would require segmenting the planning horizon into discrete time-periods [45]. In such a system, vehicles that complete their rebalancing tasks before others within a time-period are required to wait until the start of the next period to initiate new actions, which may lead to inefficiencies and not align with real-world operational dynamics. Moreover, the stations that one vehicle can reach within one time-period are limited by the duration of the time-period. This restriction is likely to reduce operational flexibility and responsiveness. Additionally, managing the large action space required for all vehicles simultaneously presents significant computational challenges. The here proposed continuous time planning framework allows for circumventing such limitations, resulting in more adaptive, responsive, and efficient vehicle planning. Next, we will focus on a RL algorithm with DQN, capable of orchestrating a fluid and responsive solution for such an MMDP model.

4. MARL Methodology with Deep Q-Network

Multi-Agent Reinforcement Learning (MARL) is a research field aiming to find high-quality solutions for multiple agents interacting with each other. In this section, we present our Deep Q-Network framework to solve the here-considered MMDP model.

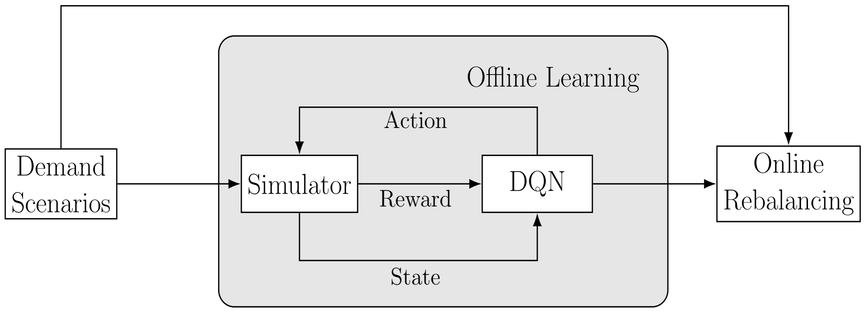

Figure 2: Dynamic Rebalancing Pipeline

The framework for the dynamic rebalancing pipeline is depicted in Figure 2. Aiming at a realistic approximation for the reward function, we design a fine-grained *simulator* as an interactive environment, where rebalancing, rentals, and returns are executed under the first-arrive-first-serve rule. The rebalancing operations are decided by the *actions* that the agents take, depending on the rental and return demand obtained from the various *demand scenarios*. Our simulator allows for computing immediate rewards between two consecutive decision epochs as realistically as possible. The *reward*, along with the system *state*, is then collected for training our *DQN*. Concurrently, the simulator updates the system state, that is then fed back to the agents, forming a critical feedback loop through their interaction with the environment. The DQN is trained through *offline learning*, utilizing a training set of demand scenarios, where user demand is associated to temporal information. Note that, while trip demand, in practice, also depends on weather conditions, our framework does not rely on such information, but rather captures its impact on the demand level implicitly. Once the offline learning is complete, the refined rebalancing policy encapsulated within the trained DQN is applied in the *online rebalancing* phase. This phase involves deploying the policy on a separated test set of demand scenarios, enabling us to assess its performance.

4.1. State and Action Space Design

Building on Section 3.3, we formulate our model within a multi-agent framework. When a vehicle arrives at a new station, it immediately triggers a rebalancing action based on the current state, often referred to as the observation, without waiting the completion of rebalancing from other vehicles. We here demonstrate the details and structure of state and action spaces adapted to our DQN MARL algorithm.

At each decision epoch, the current state encapsulates inventories of all the stations, current time, and the status of all vehicles. The station inventories are encoded as a vector where the size $|N|$ corresponds to the total number of stations, and each element within this vector reflects the current inventory of each station. The current time is denoted as a singular value, encapsulating the precise timestamp of the current state.

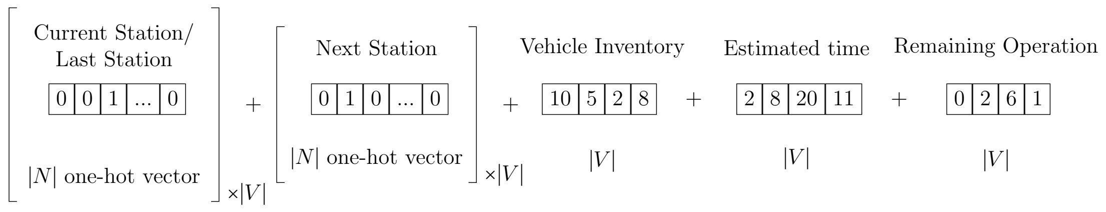

Figure 3: An example for an observation of vehicle fleet status

The status of the vehicle fleet details the current station, the next station, the current inventory, the estimated time of arrival, and the remaining rebalancing operations for each vehicle. An example is illustrated in Figure 3. Note that when a vehicle v arrives at a station, the next station of this vehicle is not determined yet. In our current implementation, we temporarily set the value of this feature to its current station until the routing decision for the next station is made. For vehicles that are still in transit between two stations, we also temporarily fix their current station feature to the station where they left from. Both the current station and next station for each vehicle are represented as one-hot vectors of length $|N|$ to facilitate the representation of data in a format that Neural Networks can easily process and learn from, where $|N|$ is the total number of stations and a singular '1' indicates the active station of the feature. Conversely, the vehicle inventory, estimated time, and remaining operations are encoded in vectors of length $|V|$, with $|V|$ being the total number of vehicles, displaying the current vehicle inventory, the estimated time of arrival at the next station, and the rebalancing operations. All these pieces of information collectively form the comprehensive status of the vehicle fleet (such as depicted in Figure 3).

Moreover, the set of forthcoming trips $I = \{[t_d(i), s_d(i), t_a(i), s_a(i)]\}$ is integrated into the observation in our algorithm, which serves as input to our simulator (see Section 4.2). Each trip $i \in I$ contains origin station $s_d(i)$, departure time $t_d(i)$, destination station $s_a(i)$, and arrival time $t_a(i)$.

The action space is defined as follows. At decision epoch $k \in K$, the vehicle selects decision $\boldsymbol{a_k} = (l_k^v, z_k^v)$, where l_k^v is the loading decision and z_k^v is the routing decision. For the routing decisions z_k^v, an agent can choose any station as its next destination. However, a constraint is applied such that no two vehicles can be directed towards the same station simultaneously. To obtain an action space of tractable size, we only consider three predefined fill levels for the inventories, indicating the proportion of the station capacity: μ_1, μ_2, and μ_3. As such, once the vehicle arrives at station g_k^v, it attempts to rebalance the station inventory either to level $\mu_1 C_{g_k^v}$, $\mu_2 C_{g_k^v}$, or $\mu_3 C_{g_k^v}$. The feasible loading decisions depend on the vehicle's capacity $\hat{C}_v$, the current vehicle inventory p_k^v, and the current station inventory $d_k^{g_k^v}$. Thus, the rebalancing decision variables l_k^v for vehicle v at decision epoch k, representing loading and unloading operations, are defined as follows:

$$(l_k^v)_i = \begin{cases} \min\{\hat{C}_v - p_k^v, d_k^{g_k^v} - \mu_i C_{g_k^v}\} & \text{if} \quad \mu_i C_{g_k^v} < d_k^{g_k^v} \\ \max\{-p_k^v, d_k^{g_k^v} - \mu_i C_{g_k^v}\} & \text{if} \quad \mu_i C_{g_k^v} > d_k^{g_k^v} \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Equation (4) is interpreted as follows. In the first case, $l_k^v > 0$, i.e., the vehicle needs to pick up bikes from the station. In the second case, $l_k^v < 0$, i.e., the vehicle needs to drop off bikes at the station. In the third case, no rebalancing operations are performed.

Once an action is taken, it is evaluated in the simulator detailed in Section 4.2. The methodology and criteria used for selecting the most appropriate action are further elaborated in Section 4.3.

4.2. Simulator of Operating Environment

Upon executing an action $\boldsymbol{a_k}$ under a state $\boldsymbol{S_k}$, an immediate reward is generated, which is quantified as the negative of the total lost demand (rentals and returns) in the duration of t_k to t_{k+1}. We here develop a discrete-event simulator as the environment to compute lost demand under the first-arrive-first-serve rule. Maximizing the long-term reward is equivalent to minimizing the total lost demand. Note that the same simulator is also used to evaluate the policy on the test set.

The pseudo-code for our simulator is given in Algorithm 1. As vehicle v arrives at station g_k^v at t_k, an action $\boldsymbol{a_k}$ will instantly be taken. The system status, such as station inventory and vehicle fleet information, is obtained through the current observation. A queue set W is created to record individual upcoming events such as rentals, returns, and rebalancing operations. Each event within set W is detailed by a tuple $[w_t(m), w_s(m), w_i(m), w_v(m)]$, where m is the index of each event, $w_t(m)$

Algorithm 1: Simulator with a certain action and state

```
Input          : $I$, $D_{s,s'}$, $R_{s,s'}$, $\hat{C}_v$, $C_s$, $\beta$ , $\boldsymbol{a_k}$ , $\boldsymbol{S_k}$
Initialization: $W$; $e_t$; $reward = 0$; $time = w_t(1)$; $s = w_s(1)$;
                $indicator = w_i(1)$; $v = w_v(1)$; $i = 1$;
while $time < e_t$ and $W \neq \varnothing$ do
    $sign = 0$;
    if $indicator = d$ then
        if $d_k^s > 0$ then
            $d_k^s = d_k^s - 1$; $W = W \cup \{[t_a(i), s_a(i), a, 0]\}$;
        else
            $reward = reward - 1$;
        $i = i + 1$;
    else if $indicator = a$ then
        if $C_s - d_k^s > 0$ then
            $d_k^s = d_k^s + 1$;
        else
            $reward = reward - 1$; $s' = \arg\min_{d_k^{s'} < C_{s'}} D_{s,s'}$ ; $d_k^{s'} = d_k^{s'} + 1$;
    else if $indicator = p$ then
        if $d_k^s > 0$ and $p_k^v < \hat{C}_v$ then
            $d_k^s = d_k^s - 1$; $p_k^v = p_k^v + 1$;$o_k^v = o_k^v - 1$ ;
            if $o_k^v = 0$ then
                $sign = 1$;
        else
            Remove $w$ in $W$ whose $w_s = s, w_i = p, w_v = v$; $sign = 1$;
    else
        if $p_k^v > 0$ and $d_k^s < C_s$ then
            $d_k^s = d_k^s + 1$; $p_k^v = p_k^v - 1$; $o_k^v = o_k^v - 1$;
            if $o_k^v = 0$ then
                $sign = 1$;
        else
            Remove $w$ in $W$ whose $w_s = s, w_i = f, w_v = v$; $sign = 1$;
    if $sign = 1$ then
        $m_k^v = time + R_{s,g_k^v}$;
        if $m_k^v < e_t$ then
            $e_t = m_k^v$; Remove elements in $W$ whose $w_i = d$;
    $W = W \setminus \{[time, s, indicator, v]\}$;
    $time, s, indicator, v = w_t(m), w_s(m), w_i(m), w_v(m)$ $(m = \arg\min w_t(m))$ ;
end
Update $\boldsymbol{S_{k+1}}$;
Output         : $reward$ and $\boldsymbol{S_{k+1}}$
```

specifies the time the event occurs, $w_s(m)$ identifies the involved station, $w_i(m)$ denotes the type of event — either a rental demand (‘d’), a return demand (‘a’), a bike pick-up (‘p’), or a bike drop-off (‘f’) — and $w_v(m)$ associates the event with a particular vehicle, if applicable. Note that the value of $w_v(m)$ has no impact on rental or return events and is set to 0. We first initialize W with all rental demands from I (see Section 4.1) and rebalancing operations (pick-ups and drop-offs) from o_k^v occurring within the time interval $[t_k, e_t)$, where e_t is the estimated point in time for the next arrival of a vehicle (i.e., the time of the next decision epoch) derived from m_k^v in Table 2. The corresponding returns of successful rentals are created and added to W in real time. The events from queue W are sorted by event time $w_t(m)$ in ascending order. The simulator then processes the events (i.e., rental and return demand, as well as rebalancing operations) in W in chronological order of time, ensuring a first-come-first-serve execution as it is the case in practice. Once an event is completed, the first element in queue W will be pulled out and executed as the next event.

When a rental demand arises and the station holds at least one available bike, we update the station inventory and add the corresponding return demand at the destination station to queue W. Otherwise, the customer fails to rent a bike when there is no inventory in the station, resulting in a lost rental demand. When a return demand occurs but the station has no available docks, we assume that the customer immediately returns the bike at the nearest station with available docks. Given that the user was not able to return the bike at the desired station, such a failure will be counted as a lost return demand. The station inventories are updated accordingly. For rebalancing events (pick-up/drop-off), we verify whether sufficient bikes/docks at the station and space/bikes within the vehicle are available. When a vehicle completes its rebalancing task or lacks the necessary resources to continue, it departs for the next station. Then, the vehicle’s estimated arrival time at the next station is computed. If this estimated arrival is sooner than the current estimated time point of the next decision epoch e_t (which may concern different vehicles), we update e_t accordingly. As such, e_t will be equivalent to the point in time the next decision epoch will occur.

During the simulation, the state is continuously updated in real time, leading to the subsequent state $\boldsymbol{S_{k+1}}$ and an immediate reward. This reward is quantified as the negative of the total lost demand occurring until the next decision epoch.

4.3. Policy Evaluation and DQN

In this section, we delineate the methodology employed to train a Deep Q-Network to derive an optimal policy for dynamic rebalancing. We present the principles of Q-learning, elucidate the mechanisms of policy evaluation, and then elaborate on the structure of the DQN.

Q-learning is a value-based off-policy Temporal-Difference (TD) RL method. In the realm of value-based RL, the central objective is to learn a value function that predicts the expected return of taking a specific action in a particular state, guiding the agent towards the most rewarding outcomes. Specifically, with a Q-value function, $Q(\boldsymbol{S}, \boldsymbol{a})$ indicates the return after taking an action $\boldsymbol{a}$ in a given state $\boldsymbol{S}$, and a greedy policy π^* that maximizes reward can be obtained from Equation (3). The optimal Q-function obeys the following Bellman equation [3]:

$$Q^*(\boldsymbol{S_k}, \boldsymbol{a_k}) = \mathbb{E}[r_k + \gamma \max_{\boldsymbol{a_{k+1}}} Q^*(\boldsymbol{S_{k+1}}, \boldsymbol{a_{k+1}})], \tag{5}$$

where $\boldsymbol{S_k}$ and $\boldsymbol{a_k}$ are the current state and action, respectively, $\boldsymbol{S_{k+1}}$ and r_k are the next state and immediate reward after taking action $\boldsymbol{a_k}$, and $\boldsymbol{a_{k+1}}$ is the action that achieves maximal Q-value at state $\boldsymbol{S_{k+1}}$. The expectation is computed over the distribution of immediate reward r_k and next state $\boldsymbol{a_{k+1}}$.

Before the Q-value function converges to optimality, the difference between the two sides of the equality is known as the TD error, namely

$$\delta = r + \gamma \max_{\boldsymbol{a_{k+1}}} Q(\boldsymbol{S_{k+1}}, \boldsymbol{a_{k+1}}) - Q(\boldsymbol{S_k}, \boldsymbol{a_k}). \tag{6}$$

The basic idea of Q-learning is to update the Q-value function by minimizing the TD error. The key updating rule is

$$Q(\boldsymbol{S_k}, \boldsymbol{a_k}) \leftarrow Q(\boldsymbol{S_k}, \boldsymbol{a_k}) + \alpha(r_k + \gamma \max_{\boldsymbol{a_{k+1}}} Q(\boldsymbol{S_{k+1}}, \boldsymbol{a_{k+1}}) - Q(\boldsymbol{S_k}, \boldsymbol{a_k})), \tag{7}$$

where α denotes the learning rate.

The Q-learning updating rule exemplifies the TD approach. It adjusts the value (called the Q-value) of a state-action pair based on the actual reward received and the projected value of the next state. This rule essentially allows the agent to forecast future rewards and use this foresight to make informed decisions in the present. Being off-policy, Q-learning enables the agent to gain insights from exploratory actions, which may not immediately appear optimal, thereby enriching the learning process and allowing the policy to evolve beyond the limits of the agent's existing strategy.

Deep Q-Networks are an advancement in RL that combine Q-learning with Deep Neural Networks. In DQN, two neural networks are employed: the prediction network and the target network. The prediction network is responsible for approximating the Q-value of any state-action pair, and it is updated iteratively to reduce the discrepancy between predicted Q-values and target Q-values. The target network serves to stabilize the learning process. Specifically, the target Q-value for a state-action pair is given by $r + \gamma \max_{a'} Q(S', a'; \theta^{target})$, where S' is the subsequent state. The loss function L is given by the square of TD error of the predicted Q-value and the target Q-value

$$L(\theta^{pred}) = \mathbb{E}[((r_k + \gamma \max_{\boldsymbol{a}} Q(\boldsymbol{S_{k+1}}, \boldsymbol{a}; \theta^{target})) - Q(\boldsymbol{S_k}, \boldsymbol{a_k}; \theta^{pred}))^2], \tag{8}$$

where θ^{target} and θ^{pred} are the parameters of the target network and prediction network, respectively.

During the training phase, the agents explore the state-action space by using an ϵ-greedy strategy, where ϵ represents the exploration-exploitation trade-off, indicating the percentage of actions through which the agent takes random actions instead of following the current best policy. Initially, a high ϵ value is set, meaning the agent is more likely to take a random action, ensuring sufficient exploration. As training progresses, ϵ is gradually decreased, shifting the agent's behavior from exploration to exploitation, where the action that maximizes the Q-value from the prediction network has a higher probability to be selected. As the agent interacts with the environment, the accumulated experiences, consisting of state, action, reward and next state are stored in a replay buffer. These experiences are then randomly sampled in mini-batches for training the target network. At each iteration, a mini-batch of training data is sampled from the replay buffer and the training loss is computed as by Equation (8). The parameters of the prediction network are updated via gradient descent to minimize the loss, while the target network's parameters are updated less often, usually by directly copying the prediction network's weights. This delayed update mechanism for the target network ensures the learning process to avoid diverging for instability or erratic behavior, enhancing the reliability of the training.

We employ the same Neural Network architecture for the two Q-value networks. Initially, the input layer is designed to match the dimensions of the state observations defined in Section 4.1. Then, we have two fully connected dense layers followed by a Rectified Linear Unit (ReLU) activation function. The ReLU function is selected for its non-linear properties and its ability to mitigate the vanishing gradient problem, which is especially beneficial during the training of deep networks [30]. Finally, an output layer corresponding to the number of actions allows the network to output a Q-value for each possible action based on a given state.

5. Experiments and Results

In this section, we present the results of our computational investigation. In Section 5.1, we first introduce the dataset and the baselines. Then, in Section 5.2, we demonstrate the training process for the DQN and evaluate the performance of our algorithm and two baseline models on the test set.

5.1. Dataset and Baselines

Dataset. Although real-world trip data is generally available, we opt for synthetic instance generation due to several considerations. The existing real-world data does not account for unobserved demand and may contain inconsistencies or noise. Moreover, data on the rebalancing operations carried out by operators is not available, which highly affects the reliability of reported station inventories. Consequently, we utilize the instance generator introduced by [22] that generates trip data based on different weather conditions and temporal features. The generator first generates synthetic weather based on real-world weather data distributions. Subsequently, hourly rentals are predicted using a linear regression model, trained on real-world weather and trip data. Finally, the trip data is produced based on these hourly rentals, in conjunction with trip distributions and the considered station network (i.e., the station locations and capacities). The trip data therefore covers a diverse set of realistic user behaviors, facilitating the DQN to learn a rebalancing policy that accounts for different demand scenarios via the simulator.

The synthetic dataset is generated based on the weather and temporal data sourced directly from the official website for Canadian weather statistics, where details on temperature, humidity, day, hour, year, and weekday are reported[1]. The real-world trips and station information used as input to the generator are directly obtained from BIXI, a BSS in Montreal. We focus on weekdays from the period of May to September 2019 to ensure that the data is not affected by the COVID-19 pandemic.

We consider two ground-truth datasets, GT1 and GT2, each comprising a 60-station network with varying station distributions. The stations within city centers are equipped with 40 docks, while those outside city centers have 20 docks each. GT1 comprises 9 stations within 1 city center, whereas GT2 is more expansive, including 12 stations spread across 2 city centers. The remaining stations are outside city centers. The configuration of GT2 allows for a more even distribution of work-related trips — where individuals commute between stations in and outside city centers for work — resulting in less stressed trip demand. Details can be found in [22]. Utilizing two datasets is significant for algorithm validation, as it reflects a more realistic variety of urban commuting patterns. In each dataset, we generate 150 days, each of which contains detailed trips (each trip consisting of its origin station, the

[1]https://climate.weather.gc.ca

departure time, its destination station and its arrival time). The first 100 days in the dataset are used for training. The remaining 50 days are used as a test set for evaluation. Four vehicles are available to rebalance the stations, each with a capacity to carry 40 bikes.

Baselines. We benchmark our algorithm with the following two MIP models:

- **Static Rebalancing:** Static rebalancing only occurs during the night with the objective of optimizing the initial inventories of the next day (in our case, starting at 7 a.m.). The planning solution is obtained by solving an MIP model from [21], where the inventories for the first time-period are decision variables that sum to the total number of available bikes in the system. We then simulate all days from the test set without intraday rebalancing starting from the initial inventories gained by static rebalancing model. Note that the inventory solution derived from the static rebalancing model also serves as the initial inventory in all other models.

- **Dynamic Rebalancing:** We apply the multi-period dynamic rebalancing MIP model from [21], with time-periods of 30-minute and 60-minute length. For each station and time-period, the average number of rentals and returns is obtained from the training set, serving as demand estimate used in the MIP model. The produced rebalancing strategy indicates the number of bikes each vehicle should attempt to pick up or drop off at which station for each time-period. The performance of the rebalancing strategy is also evaluated on the test set.

The two baseline models are solved using IBM ILOG CPLEX v20.1.0.0 on 2.70 GHz Intel Xeon Gold 6258R machines with 8 cores. Optimization terminates once the MIP gap reaches 0.01% or the time limit of 24 hours is reached.

5.2. Experimental Results

We consider a planning horizon of 4 hours from 7 a.m to 11 a.m, surrounding the morning demand peak. This period also corresponds to an episode in DQN. The training phase in the DQN is performed on a single GPU Tesla V100-PCIE-32GB.

5.2.1. DQN Training Process

We introduce the parameters used in our DQN algorithm in Table 3. A total of 3,000,000 time steps are chosen to provide the model ample opportunity to learn from a wide range of situations and stabilize its policy. Learning is performed offline, and is therefore not a time-sensitive matter. The moderate learning rate of 2.5e-4 balances speed of convergence with stability of the learning updates. A buffer size of 10,000 allows for storing a diverse set of experiences, enhancing the robustness of the learning by preventing overfitting to recent experiences. The discount factor γ is set to 0.99, i.e., future rewards are considered almost as important as immediate rewards, which is suitable for tasks with long-term strategies. The batch size is 256, a standard choice that provides a good trade-off between the computational efficiency and the statistical robustness of gradient estimates. The exploration rate is annealed from 1 to 0.05 to balance exploration with exploitation as learning progresses. An exploration fraction of 0.5 defines the portion of the total time steps during which the exploration rate is annealed.

Table 3: Parameters in training process

Parameters	Values
Total time step	3,000,000
Learning rate	2.5e-4
Buffer size	10,000
Discount factor γ	0.99
Batch size	256
Exploration rate ϵ	$1 \to 0.05$
Exploration fraction	0.5
1st layer neurons	1,024
2nd layer neurons	512

We first investigate the effects of different action spaces as defined in Section 4.1 for rebalancing decisions. We consider 3 sets of predefined inventory levels of station capacity: a first set with $\mu_1 = 0\%, \mu_2 = 50\%, \mu_3 = 100\%$, a second with $\mu_1 = 10\%, \mu_2 = 50\%, \mu_3 = 90\%$, and a third with $\mu_1 = 15\%, \mu_2 = 50\%, \mu_3 = 85\%$. The standard DQN with the second set of μ is denoted as DQN, while the other two are differentiated by their respective percentages. For a comprehensive understanding of the DQN's performance, we report the episodic return and TD Loss in Figure 4. Each step represents one episode, which here corresponds to 4 hours (7 a.m to 11 a.m). The episodic return measures the cumulative reward obtained per episode, i.e. the negative value of total lost demand, with higher returns suggesting more effective rebalancing strategies. The TD Loss measures the divergence between the predicted and the target Q-values, serving as an indicator of the network's precision in forecasting future rewards. Jointly, the episodic return and TD loss provide a clear picture of both the result of the agent actions and the learning progression over time.

Figure 4 reveals that the episodic return for the DQN with the first action space (i.e., $\mu_1 = 10\%, \mu_2 = 50\%, \mu_3 = 90\%$) shows a more consistent increase compared to

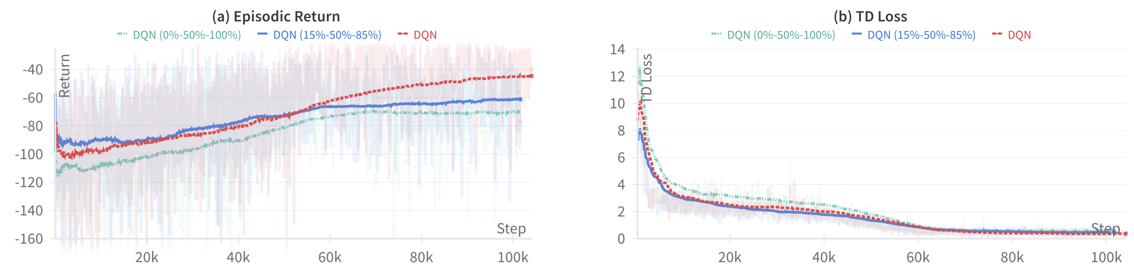

Figure 4: Episodic Return and TD Loss of DQN for GT1 under different action spaces

the other two, indicating a more effective strategy. The TD Loss trends downward for all action spaces, which indicates learning improvement across the board, yet the standard DQN maintains a consistently low TD Loss, suggesting that it learns a more accurate Q-value estimation faster than the others. The converging TD Loss, coupled with the improving episodic returns, strongly points to the conclusion that the DQN model with the μ parameters set at 10%, 50%, and 90% is developing a more refined and potentially more successful policy within the planning horizon, as opposed to the other tested action space parameters.

Based on the best performing action space of the DQN model (i.e., μ_1 = 10%, μ_2 = 50%, and μ_3 = 90%), we next investigate the impact of using distinct activation functions for the output layer on the training performance. While the previous models did not use any activation function in the output layer, we now test Leaky ReLU, Parameterized ReLU (PReLU), and Exponential Linear Unit (ELU) [34]. These functions are specifically chosen for their ability to adeptly handle negative outputs, given that the immediate reward in our DQN model is defined as the negative value of the lost demand. Unlike the conventional ReLU function, which suffers from the zero-gradient problem for negative inputs potentially causing inactive neurons, Leaky ReLU, PReLU, and ELU maintain gradients, preventing neuron death during training [35].

The results of episodic return and TD loss during the training process for GT1 for the resulting models are shown in Figure 5. In Figure 5 (a), the episodic return across all variants exhibits a generally upward trend, indicating an increase in cumulative rewards as learning progresses. The convergence of episodic returns suggests that regardless of the activation function, each variant is improving its policy effectively over time. In terms of TD loss in Figure 5 (b), there is an initial steep decrease in loss across all variants, which then levels off, suggesting a rapid early improvement in predictive accuracy that stabilizes as the models approach optimal policy estimation. Overall, standard DQN slightly stands out, while DQN-ELU lags a bit behind the

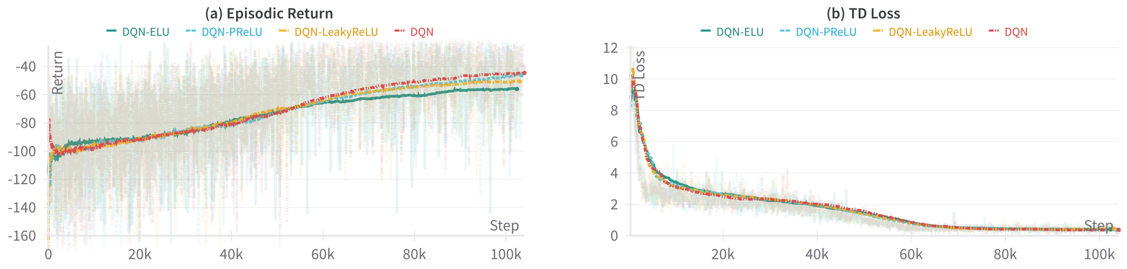

Figure 5: Episodic Return and TD Loss of DQN variants for GT1

other opponents, particularly in episodic return.

We then focus on the best performing model, namely the DQN, in Figure 6 and Figure 7 for GT1. The agents in our algorithm appear to be effectively learning the dynamic rebalancing task in BSS. This is evidenced by the rising Q-values, reducing TD loss (see Figure 6), and improving episodic returns (see Figure 7). Recall that the Q-value represents the expected reward for taking a particular action in a state and following the policy thereafter. The Q-value curve here represents the average of the estimated Q-values over global steps throughout the training episodes. An increase of the Q-value over time is a sign that, as the training phase advances, actions of higher quality are selected, reducing the expected lost demand. As such, it measures the quality of particular actions, with higher values suggesting more favorable actions.

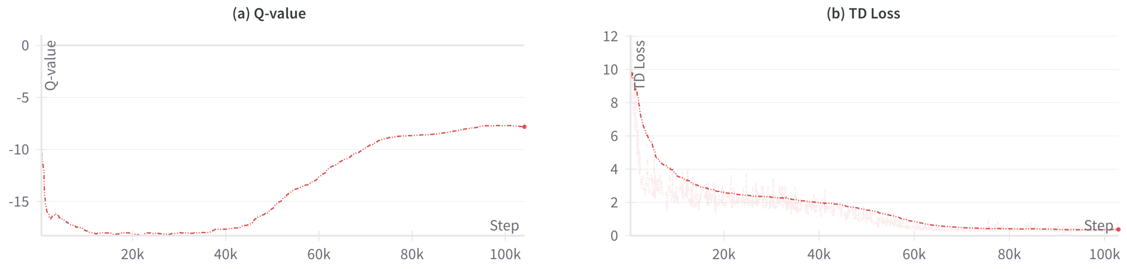

Figure 6: Q-value and TD Loss for DQN (GT1)

The Q-value in Figure 6 (a) decreases at first when agents begin to explore the state-action space with a random policy and encounter highly negative rewards, which is typically characterized by exploration. During this initial stage, the agent learns to track the current policy where the Q-value first converges to the value of the random policy. Once the agents begin to learn from that experiences, the Q-value increases steadily, leading to a better rebalancing policy. The function approximation of the policy's Q-value also improves over time as the agent learns to better predict expected returns from its actions. The decreasing trend in TD Loss observed in Figure 6 (b) suggests that the DQN's predictions are becoming more accurate over time as the agent learns from its experiences. A high TD Loss at the beginning reflects that the DQN starts with random predictions, which is expected to diminish as the network evolves and refines its policy.

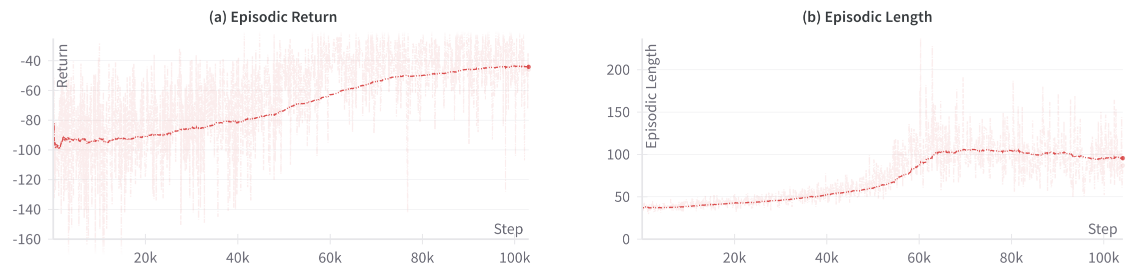

Figure 7: Episodic Return and Episodic Length for DQN (GT1)

The episodic return in Figure 7 (a) suggests that the agent's policy becomes more effective over time at achieving higher rewards, i.e., finding better rebalancing policies. The episodic length is given by the number of steps all agents (i.e., all 4 vehicles) can perform during one episode (i.e., the planning horizon of 4 hours), here reflecting the number of stations visited during one episode, increasing from approximately 50 to 100, as shown in Figure 7 (b). This increasing trend indicates that the agents learn to optimize rebalancing operations by covering more stations to reduce total lost demand in the system.

The illustrated metrics offer a valuable perspective on the models' performance throughout the training phase, capturing the progression and refinement of the learning process. However, it is crucial to investigate the trained models' capabilities on the test set, which is done in Section 5.2.3.

5.2.2. Impacts of routing

In this section, we further investigate the impact of routing decisions within the context of MMDP. We simplify the aforementioned MMDP model, restricting the action space solely to rebalancing decisions. Therefore, upon arrival at the designated station, the DQN policy of vehicles in this MMDP environment only makes loading/unloading decisions. Then, the selection of the next station is determined either randomly or through a predetermined routing heuristic. Specifically, the heuristic rule is to assign the fullest station to the most empty vehicle and vice versa, i.e., selecting the most empty station for the most full vehicle. We conduct experiments to compare the two simplified DQNs against our complete DQN policy that makes both routing and rebalancing decisions. The results are illustrated in Figure 8, where the two new rebalancing algorithms using simplified DQNs are labelled ‘Rebalancing & heuristic routing’ and ‘Rebalancing & random routing’.

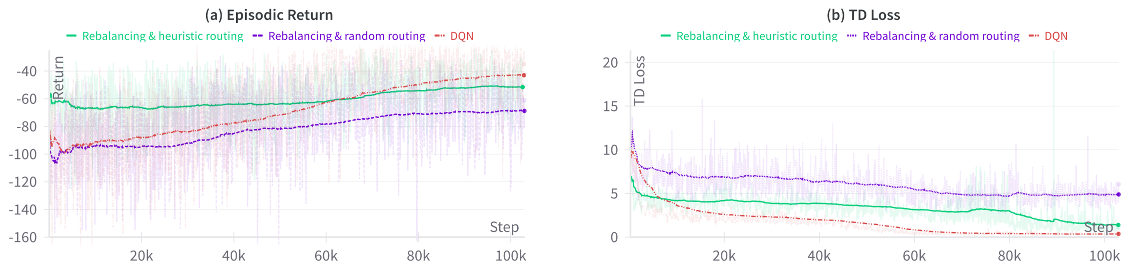

Figure 8: Episodic Return and TD Loss under Different Routing Setting (GT1)

The variant with heuristic routing demonstrates a clear advantage over the one with random routing, both in terms of episodic return in Figure 8(a), which measures the system’s efficiency or profitability, and TD loss in Figure 8(b), which is indicative of a more accurate Q-function approximation. Improvements of both methods are, however, rather modest. The DQN is less effective than the heuristic routing at first, as routing decisions are ineffective at the beginning of the learning process. However, over time, both the episodic return and the TD loss consistently improve and eventually DQN outperforms the variant based on heuristic routing, demonstrating that it is able to learn routing decisions that are superior to those of a heuristic rule. These results emphasize the impact of routing decisions in the planning problem at hand, as well as the relevance of including them in the learning task. Therefore, our subsequent evaluation will concentrate exclusively on the DQN with both rebalancing and routing decisions.

5.2.3. Performance comparison among the various models

We now compare the performance of the various models on the test set to illustrate the practical efficacy and robustness when faced with new and potentially more complex demand scenarios. Tests are carried out for both ground-truth problem instances GT1 and GT2, where the training process for GT2 is the same as for GT1. The training time for the DQN model and its variants has been found to be approximately 14 hours, which is non-critical, given that training is performed offline. In contrast, policy predictions are made within seconds. We here note, again, that each problem instance has been trained on its respective station network (either

GT1 or GT2), which is a realistic assumption, given that one can expect to have available historical data for the specific BSS station network at hand.

Table 4: Total average lost demand on test set for GT1 and GT2

Model	GT1		GT2	
	$\epsilon = 0$	$\epsilon = 0.05$	$\epsilon = 0$	$\epsilon = 0.05$
Static Rebalancing	112.82	-	82.12	-
Dynamic MIP Model (60 mins time-period)	56.78	-	75.32	-
Dynamic MIP Model (30 mins time-period)	44.76	-	36.72	-
DQN-ELU	50.13	51.13	26.76	32.11
DQN-PReLU	36.58	44.21	29.09	29.75
DQN-LeakyReLU	44.53	48.27	27.32	29.22
DQN	35.12	42.69	29.54	31.05

Table 4 reports the average episodic lost rental and return demand on the test set of both our algorithm and the baselines for both GT1 and GT2. For both ground-truths, we report two distinct settings for the DQN with different values of ϵ. Here, we set ϵ at fixed values rather than the ϵ-greedy strategy used during training, which transitions gradually from 1 to 0.05 (see Table 3). This exploration rate is critical in determining the likelihood that a random action is chosen over the predicted best action. When $\epsilon = 0$, the action yielding the maximum Q-value from the trained network is consistently selected. The robustness of the DQN models is evaluated by $\epsilon = 0.05$, which incorporates a degree of randomness in action selection. This exploration can potentially lead to the discovery of more effective strategies that were not captured during training, or it may provide a buffer against overfitting to the training data by occasionally deviating from the deterministic policy. Given that the static rebalancing and the dynamic MIP model do not have a stochastic strategy, their performance are showcased only under the $\epsilon = 0$ condition.

The static rebalancing, i.e., overnight rebalancing without dynamic intraday rebalancing, displays the highest total average lost demand of 112.82 for GT1 and 82.12 for GT2. In contrast, dynamic MIP models with lengths of 30-min and 60-min per time period show a significant reduction in lost demand. This demonstrates the natural advantage of dynamic rebalancing strategies, as they allow to adjust station inventories throughout the planning horizon. Specifically, the 30-min dynamic model (44.76) reduces lost demand over the static model (112.82) by up to a 60.32% (for GT1). Among the two dynamic models, using 30-min instead of 60-min time-periods, the lost demand reduces by up to 51.25%. This is likely explained by the rebalancing frequency allowed by the two models, as each vehicle can only rebalance one station per time-period. Specifically, in the case of the former model, the 4 vehicles can rebalance a total of 32 stations within the period of 4 hours, while in the latter model, only 16 stations can be rebalanced. However, according to Figure 7(b), the number of stations rebalanced by the RL solutions can reach more than 50. The continuous time framework enables the vehicle fleet to rebalance stations more frequently and efficiently since the vehicles do not have to wait for each other.

The DQN variants exhibit commendable adaptability and efficiency in the context of lost demand for both GT1 and GT2. When $\epsilon = 0$, the standard DQN demonstrates the most effective performance, achieving the lowest total average lost demand of 35.12 for GT1, while DQN-ELU outperforms others with a lost demand of 26.76, highlighting the robustness and capability of RL. Particularly, DQN reduces lost demand by 21.5% compared to the Dynamic MIP model with 30-min time-period for GT1 and DQN-ELU achieves an improvement of 27.12% for GT2. The DQN-PReLU and DQN-LeakyReLU also show competitive results, potentially offering superior learning dynamics during the training phase.

Introducing randomness in action selection through $\epsilon = 0.05$ results in an increase in lost demand across all DQN models. In this case, the agents are no longer solely exploiting their learned knowledge but are also exploring potentially suboptimal actions to discover new strategies. Notably, most of the DQN models maintain a superior performance over dynamic MIP model even when exploration is introduced, especially for GT2, which underscores the DQN models' superior ability to generalize and adapt to the test environment despite the introduction of randomness in the decision-making process. The fact that the proposed models exhibit reasonable performance despite random deviations also indicates their capacity to correct mistakes made in previous actions, highlighting the algorithm's adaptability.

6. Conclusions

We address the Dynamic Bicycle Redistribution Problem with multiple vehicles, a planning problem of uttermost importance for dock-based Bike-Sharing Systems to provide high user satisfaction. To provide decision-support, most of the existing literature has focused on MIP approaches. Due to the complexity of this planning problem, most MIP approaches are required to make simplifying assumptions to yield tractable models.

In this work, we circumvent some of such limitations by developing a spatio-temporal RL algorithm to solve the DBRP with a station network of up to 60 stations and enable the coordination of multiple vehicles under a fine-grained simulator. We first formulated the problem as an MDP under a continuous time framework, allowing vehicles to independently and cooperatively rebalance stations without the need to wait for each other, which enhances the realism and efficiency of the rebalancing process. We then developed an advanced simulator, following the first-arrive-first-serve rule, to calculate rewards between two successive states across varied demand scenarios. To solve the MDP model for DBRP, we proposed a DQN framework to learn effective rebalancing strategies with the aim of minimizing lost demand. Our method was evaluated with a diverse trip dataset, responsive to weather conditions and temporal factors.

Our model architecture, as well as our computational experiments on two different station networks with different structures and diverse demand patterns allow for observing the following benefits. First, the proposed DQN models consistently outperform the MIP baseline models (i.e., a static and a multi-period rebalancing model), reducing the lost demand up to 27.12% with respect to the dynamic version of the MIP. Second, the continous-time framework enables the 4 vehicles to visit far more than 50 stations throughout the planning horizon, avoiding idle times. In contrast, the number of rebalanced stations in MIP models (in our experiments, either 16 or 32) depend on the time-discretization and are therefore subject to a delicate trade-off between computational tractability and practical feasibility. Third, our model can be trained offline and the construction of rebalancing policies is instantaneous, whereas MIP models may take significant time to be solved.

This study shows the potential of RL in DBRP, outlining various intriguing avenues for future research. Although our method showcases computational efficiency within a 60-station system, it is extendable to larger systems. The scalability can be further enhanced through the integration of clustering methods, offering effective application on large-scale BSS. Additionally, the extension of our method could also involve additional state parameters such as vehicle initial locations, offering a more granular and responsive approach to rebalancing. Finally, with the understanding that the training time is not a limiting factor, we have the flexibility to emphasize real-time decision-making once the model is trained. This allows for the development and implementation of more detailed and tailored solutions, emphasizing that the focus is on the swift application of the trained model rather than the speed of the training process.

References

[1] O. Alagoz, H. Hsu, A. J. Schaefer, and M. S. Roberts, "Markov decision processes: a tool for sequential decision making under uncertainty," *Medical Decision Making*, vol. 30, no. 4, pp. 474–483, 2010.

[2] A. L. Bazzan and F. Klügl, *Multi-agent systems for traffic and transportation engineering.* IGI Global, 2009.

[3] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, pp. 679–684, 1957.

[4] R. Bellman and R. Kalaba, "On adaptive control processes," *IRE Transactions on Automatic Control*, vol. 4, no. 2, pp. 1–9, 1959.

[5] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016.

[6] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.

[7] J. Brinkmann, M. W. Ulmer, and D. C. Mattfeld, "Dynamic lookahead policies for stochastic-dynamic inventory routing in bike sharing systems," *Computers & Operations Research*, vol. 106, pp. 260–279, 2019.

[8] ——, "The multi-vehicle stochastic-dynamic inventory routing problem for bike sharing systems," *Business Research*, vol. 13, no. 1, pp. 69–92, 2020.

[9] F. Chiariotti, C. Pielli, A. Zanella, and M. Zorzi, "A dynamic approach to rebalancing bike-sharing systems," *Sensors*, vol. 18, no. 2, p. 512, 2018.

[10] Y. Du, F. Deng, and F. Liao, "A model framework for discovering the spatio-temporal usage patterns of public free-floating bike-sharing system," *Transportation Research Part C: Emerging Technologies*, vol. 103, pp. 39–55, 2019.

[11] Y. Duan and J. Wu, "Optimizing rebalance scheme for dock-less bike sharing systems with adaptive user incentive," in *2019 20th IEEE International Conference on Mobile Data Management (MDM).* IEEE, 2019, pp. 176–181.

[12] D. J. Fagnant and K. M. Kockelman, "Dynamic ride-sharing and fleet sizing for a system of shared autonomous vehicles in austin, texas," *Transportation*, vol. 45, pp. 143–158, 2018.

[13] S. Ghosh, J. Y. Koh, and P. Jaillet, "Improving customer satisfaction in bike sharing systems through dynamic repositioning," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, 2019, pp. 5864–5870.

[14] Z. Haider, A. Nikolaev, J. E. Kang, and C. Kwon, "Inventory rebalancing through pricing in public bike sharing systems," *European Journal of Operational Research*, vol. 270, no. 1, pp. 103–117, 2018.

[15] S.-S. Ho, M. Schofield, and N. Wang, "Learning incentivization strategy for resource rebalancing in shared services with a budget constraint," *Journal of Applied and Numerical Optimization*, vol. 3, no. 1, pp. 105–114, 2021.

[16] W. Hönig, T. Kumar, L. Cohen, H. Ma, H. Xu, N. Ayanian, and S. Koenig, "Multi-agent path finding with kinematic constraints," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 26, 2016, pp. 477–485.

[17] B. Legros, "Dynamic repositioning strategy in a bike-sharing system; how to prioritize and how to rebalance a bike station," *European Journal of Operational Research*, vol. 272, no. 2, pp. 740–753, 2019.

[18] J. Z. Leibo, V. Zambaldi, M. Lanctot, J. Marecki, and T. Graepel, "Multi-agent reinforcement learning in sequential social dilemmas," *arXiv preprint arXiv:1702.03037*, 2017.

[19] G. Li, N. Cao, P. Zhu, Y. Zhang, Y. Zhang, L. Li, Q. Li, and Y. Zhang, "Towards smart transportation system: A case study on the rebalancing problem of bike sharing system based on reinforcement learning," *Journal of Organizational and End User Computing (JOEUC)*, vol. 33, no. 3, pp. 35–49, 2021.

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

[21] J. Liang, S. D. Jena, and A. Lodi, "Dynamic rebalancing optimization for bike-sharing systems: A modeling framework and empirical comparison," GERAD, HEC Montréal, Canada, Les Cahiers du GERAD G–2023–47, 2023.

[22] J. Liang, M. C. Martins Silva, D. Aloise, and S. D. Jena, "Dynamic rebalancing for bike-sharing systems under inventory interval and target predictions," CIRRELT, Canada, Research Paper CIRRELT-2023-37, 2023.

[23] D. Liu, M. Fischetti, and A. Lodi, "Learning to search in local branching," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 4, pp. 3796–3803, Jun. 2022.

[24] 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.

[25] C.-C. Lu, "Robust multi-period fleet allocation models for bike-sharing systems," *Networks and Spatial Economics*, vol. 16, no. 1, pp. 61–82, 2016.

[26] X. Luo, L. Li, L. Zhao, and J. Lin, "Dynamic intra-cell repositioning in free-floating bike-sharing systems using approximate dynamic programming," *Transportation Science*, 2022.

[27] A. Mahajan, T. Rashid, M. Samvelyan, and S. Whiteson, "Maven: Multi-agent variational exploration," *Advances in neural information processing systems*, vol. 32, 2019.

[28] K. Mellou and P. Jaillet, "Dynamic resource redistribution and demand estimation: An application to bike sharing systems," *Available at SSRN 3336416*, 2019.

[29] M. Moravčík, M. Schmid, N. Burch, V. Lisỳ, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling, "Deepstack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, vol. 356, no. 6337, pp. 508–513, 2017.

[30] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[31] A. Oroojlooy and D. Hajinezhad, "A review of cooperative multi-agent deep reinforcement learning," *arXiv preprint arXiv:1908.03963*, 2019.

[32] O. O'Brien, P. DeMaio, R. Rabello, S. Chou, and T. Benicchio, "The meddin bike-sharing world map report," https://bikesharingworldmap.com/reports/bswm_mid2022report.pdf, 2022, accessed: 2023-08-04.

[33] 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, no. 01, 2019, pp. 1393–1400.

[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[35] D. Pedamonti, "Comparison of non-linear activation functions for deep neural networks on mnist classification task," *arXiv preprint arXiv:1804.02763*, 2018.

[36] T. Raviv, M. Tzur, and I. A. Forma, "Static repositioning in a bike-sharing system: models and solution approaches," *EURO Journal on Transportation and Logistics*, vol. 2, no. 3, pp. 187–229, 2013.

[37] M. Schofield, S.-S. Ho, and N. Wang, "Handling rebalancing problem in shared mobility services via reinforcement learning-based incentive mechanism," in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2021, pp. 3381–3386.

[38] Y. Seo, "A dynamic rebalancing strategy in public bicycle sharing systems based on real-time dynamic programming and reinforcement learning," Ph.D. dissertation, Doctoral dissertation. Seoul National University, South Korea, 2020.

[39] Y.-H. Seo, D.-K. Kim, S. Kang, Y.-J. Byon, and S.-Y. Kho, "Rebalancing docked bicycle sharing system with approximate dynamic programming and reinforcement learning," *Journal of Advanced Transportation*, vol. 2022, 2022.

[40] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[41] C. M. Vallez, M. Castro, and D. Contreras, "Challenges and opportunities in dock-based bike-sharing rebalancing: a systematic review," *Sustainability*, vol. 13, no. 4, p. 1829, 2021.

[42] I. Xiao, "A distributed reinforcement learning solution with knowledge transfer capability for a bike rebalancing problem," *arXiv preprint arXiv:1810.04058*, 2018.

[43] C. Xu, J. Ji, and P. Liu, "The station-free sharing bike demand forecasting with a deep learning approach and large-scale datasets," *Transportation research part C: emerging technologies*, vol. 95, pp. 47–60, 2018.

[44] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang, "Mean field multi-agent reinforcement learning," in *International conference on machine learning*. PMLR, 2018, pp. 5571–5580.

[45] Z. Yin, Z. Kou, and H. Cai, "A deep reinforcement learning model for large-scale dynamic bike share rebalancing with spatial-temporal context," in *The 12th International Workshop on Urban Computing*, 2023.

[46] D. Zhang, C. Yu, J. Desai, H. Lau, and S. Srivathsan, "A time-space network flow approach to dynamic repositioning in bicycle sharing systems," *Transportation research part B: methodological*, vol. 103, pp. 188–207, 2017.

[47] X. Zhang, H. Yang, R. Zheng, Z. Jin, and B. Zhou, "A dynamic shared bikes rebalancing method based on demand prediction," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 238–244.

[48] X. Zheng, M. Tang, Y. Liu, Z. Xian, and H. H. Zhuo, "Repositioning bikes with carrier vehicles and bike trailers in bike sharing systems," *Applied Sciences*, vol. 11, no. 16, p. 7227, 2021.