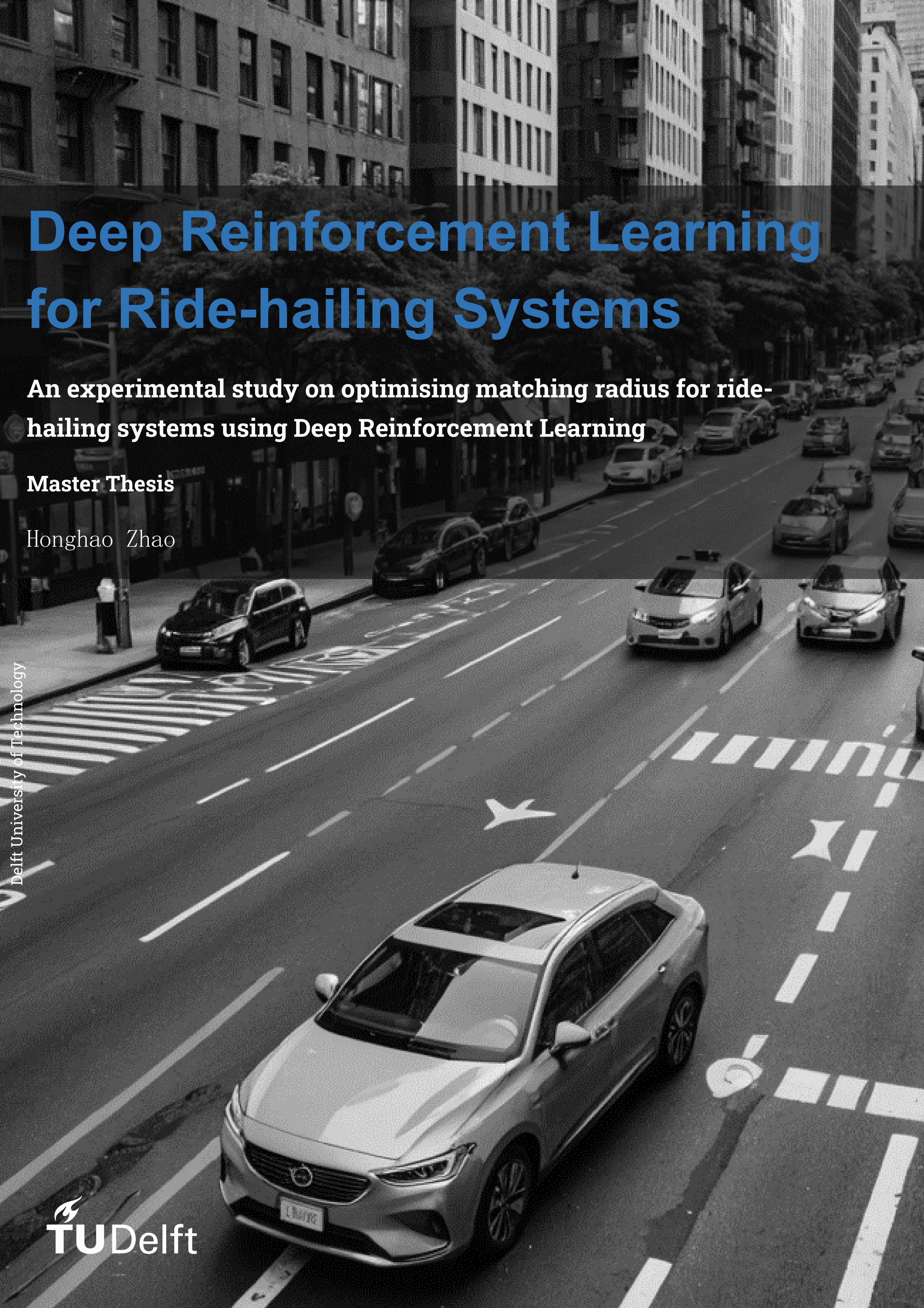


Deep Reinforcement Learning for Ride-hailing Systems

An experimental study on optimising matching radius for ride-hailing systems using Deep Reinforcement Learning

Master Thesis

Honghao Zhao

Delft University of Technology

Deep Reinforcement Learning for Ride-hailing Systems

An experimental study on optimising matching radius for ride-hailing systems using Deep Reinforcement Learning

by

Honghao Zhao

To obtain the degree of

Master of Science

in Civil Engineering
at the Delft University of Technology,
to be defended publicly on Friday September 20, 2024.

Student number: 5735289
Project duration: Nov 14, 2023 – Sep 20, 2024
Thesis committee: Prof.dr. Oded Cats, Delft University of Technology, Chair
Dr. Jie Gao, Delft University of Technology, Main Supervisor
Dr. Jie Yang, Delft University of Technology, Second Supervisor
Mr. Weiming Mai, Delft University of Technology, Daily supervisor

Cover: AI generated image of a city scene, obtained from openai.com/dall-e-3
An electronic version of this thesis is available at http://repository.tudelft.nl/.

Acknowledgements

This thesis brings an end to my master program at the TU Delft. This work and my passion originate from the interest I have in urban developments, especially transportation systems within them. China is undergoing a rapid urbanism development, with various modes of transportation continuously emerging. Over the past two decades, I have witnessed and experienced numerous traffic and transportation changes in China, such as the construction of the high-speed rail network, the rapid development of electric vehicles, and the popularity of ride-hailing services. In the midst of these changes, I began to develop a strong interest in transportation and the knowledge behind it. The Traffic Technology and Transport programme at TU Delft provided a path to knowledge for this passion of mine. Through my master research project and various courses, I am able to explore the field of traffic and transport in depth and further develop my interest. This thesis research project was enjoyable but also challenging. I have encountered many difficulties in this research project, but I can now proudly say that all the hard work has paid off. I could not have completed my research without the help and support from many people, here I would like to take this opportunity to express my deepest gratitude to them.

First of all, I would like to thank my family, especially my parents for their support. It was you who allowed me to study at TU Delft and paved the way for me to explore my interests in the field of transport engineering. This master thesis report is dedicated to my family and I would like to express my deepest gratitude for the unconditional support, encouragement and love you have given me. Without your support and encouragement, I would not be able to achieve what I am today. Your guidance to me has played an irreplaceable role in my life and educational journey, from the beginning of shaping my personality to today's graduation from TU Delft. Thank you for supporting me unconditionally and guiding me on how to achieve my dreams.

I would also like to thank the members of my thesis committee. I am grateful to Professor Oded Cats for your meticulous guidance during my thesis research. Your advice has been invaluable in the research and writing of this thesis. Dr. Jie Gao, thank you for introducing and guiding me to the field of Reinforcement Learning. Your enthusiasm for Reinforcement Learning sparked my interest in exploring its potential applications in transportation. Your daily guidance has helped me conduct my research in the right direction. Weiming, I am grateful for your careful and patient guidance throughout the entire research and master thesis, especially when I was confused about the simulation environment and Reinforcement Learning algorithms. Your advice and guidance have continually assisted me in smoothly conducting my research.

Finally, a million thanks to my buddies in the Netherlands for an amazing two-year time. I am grateful that you are my excellent classmates and friends to help during the two years of master studies. Countless nights and dinners with you have been weekly sources of entertainment, joy, and stress-relief to take me away from the stress of studying and research. Thank you for all the support, happy moments, and advice you have given me. I also want to thank my friends back in China. Your friendship is like that of a family, making me feel the warmth of home even while far away from China. Despite the distance, you can always find ways to help and uplift me.

Honghao Zhao
Delft, July 2024

Abstract

In the field of public transportation, environmentally friendly and convenient transportation modes are the future trends. The ride-hailing services is an important component of them. However, current ride-hailing systems, particularly the matching systems, still have issues related to low system efficiency and bad user experience. Although existing ride-hailing rider-driver matching system can allocate travel demands and drivers to a certain extent, they still have deficiencies in certain scenarios. For example, they cannot ensure effective rider-driver matching during peak hours, or they cannot find a good balance between pick-up distance and matching rate. As Reinforcement learning (RL) has been proven in many studies to be applicable and effective in solving complex and dynamic optimization problems. This study aims to explore how Reinforcement Learning (RL) can be adapted to the ride-hailing matching system to optimize system efficiency and user experience through a dynamic matching radius policy. The research objective of this study is to simulate an actual ride-hailing system and use RL to train a policy. This policy can output an optimized dynamic matching radius in real-time based on real-time rider-driver demand-supply relationship, hence achieving a higher matching rate, a shorter average pick-up distance, and a higher driver utilization rate of the ride-hailing system.

Adapting Reinforcement Learning (RL) to optimize the ride-hailing system's matching radius has several difficulties and challenges due to the uncertainties in the real-world rider-hailing market. Traditional approaches are normally static, solving the matching problem at specific times through mathematical models. However, these methods often perform inconsistently when dealing with fluctuating ride-hailing supply-demand relationships, particularly during peak hours. On the other hand, the dynamics and complexity of the ride-hailing market and the ride-hailing environment also make it difficult to model the ride-hailing system. The ride-hailing market is easily affected by many variables, such as weather conditions and local traffic conditions. When quantitatively optimizing the matching radius of the ride-hailing matching system, it is critical to reasonably control irrelevant variables. To address these challenges, this study models the ride-hailing matching problem as a Markov Decision Process (MDP). Based on the defined MDP, a ride-hailing matching simulator is developed. Some assumptions and simplifications are also made to ensure high realism while reasonably controlling irrelevant variables and uncertainties. Multi-replay-buffer Deep Deterministic Policy Gradient (MDDPG) algorithm is then applied to handle the optimization problem of the ride-hailing matching radius. Through the interactions between the MDDPG agent and the developed simulator, feedback rewards are received for the agent to improve the policy. The proposed method is then validated in a case study showcasing the application of the developed simulator and the RL algorithm in a real-world scenario in Austin, Texas. The case study includes an analysis of the current ride-hailing market in Austin, how to apply the simulator based on it, the implementation details of the RL algorithm, and the resulting performance improvements. The results of the case study show that the actions obtained from the proposed method outperform all the baselines in multiple scenarios, highlighting the benefits of using Reinforcement Learning to improve ride-hailing efficiency and user experience.

To conclude, the optimization method proposed in this study applies an advanced Reinforcement Learning approach to the ride-hailing system, successfully improving overall efficiency and user experience. The results of this research demonstrate the potential of Reinforcement Learning in optimizing ride-hailing matching systems, offering a promising direction for further exploration. This study lays a solid foundation for future research to build upon, encouraging the development of more optimization methods with RL technologies that can enhance the effectiveness and adaptability of ride-hailing system in increasingly complex and dynamic environments.

Contents

List of Figures

List of Figures

List of Tables

Nomenclature

Related abbreviations and their definitions.

Abbreviation	Definition
RHS	Ride-hailing System
ML	Machine Learning
RL	Reinforcement Learning
RQ	Research Question
SRQ	Sub-Research Question
MDP	Markov Decision Process
V-RL	Value-based Reinforcement Learning
P-RL	Policy-based Reinforcement Learning
DRL	Deep Reinforcement Learning
MARL	Multi-Agent Reinforcement Learning
DBRAS	Dynamic Broadcasting Radius Adjustment System
WESM	Weighted Exponential Smoothing Multi-task learning
QL	Table-based Q-Learning
DQN	Deep Q-network Learning
AC	Actor-Critic
A2C	Advantage Actor-Critic
ACER	Actor-Critic with Experience Replay
NN	Artificial Neural Network
DNN	Deep Neural Network
PG	Policy Gradient
DPG	Deterministic Policy Gradient
DDPG	Deep Deterministic Policy Gradient
MDDPG	Multi-replay-buffer Deep Deterministic Policy Gradient
MSE	Mean Squared Exponential
ELU	Exponential Linear Unit
Tanh	Tanh activation function
MR	Matching Rate
APD	Average Pick-up Distance
DUR	Driver Utilization Rate

1

Introduction

1.1 Context

Ride-hailing services have changed urban mobility scenarios across the globe by offering a more convenient and efficient alternative mode of transport for daily commuting than traditional modes. As major cities target 2050 for net-zero emissions, ride-hailing systems, as a type of shared transportation, become increasingly popular in public transportation, accompanied by the continuous expansion of the ride-hailing market *(Wu and Wang, 2022)*. The ride-hailing services not only reduce dependence on private car ownership and simultaneously boost vehicle utilization which can significantly reduce carbon emissions. However, against this backdrop of growth and due to the dynamic and imbalanced characteristics of the ride-hailing market, optimizing the matching efficiency and enhancing the user experience in the same time for the ride-hailing system remains challenging. This section will introduce the context and motivation for optimizing ride-hailing matching systems, highlighting their importance in achieving sustainable development goals, and propose specific research questions for this study. Figure 1.1 shows what a ride-hailing system looks like.

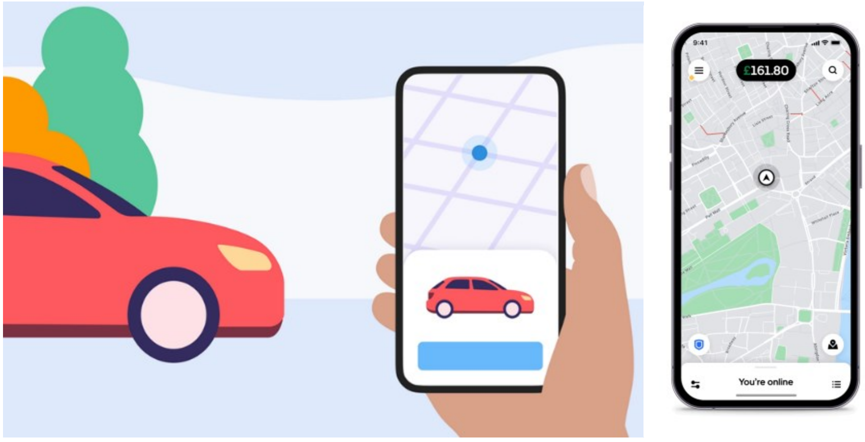


Figure 1.1 An example of a ride-hailing system. [1]

To achieve the goal of net-zero emissions by 2050 as mentioned above, the transportation industry needs transformation, including the adoption of new energy sources or low-emission transport modes *(Khosrowshahi, 2023)*. Ride-hailing services, as a form of shared transportation, play a crucial role in this transition by reducing the frequency of private car use, thus contributing to carbon emission reduction. Moreover, with the increasing process of urbanization, more and more people are willing to use ride-hailing services. According to Uber's data, as shown in Figure 1.2, from the first quarter of 2017 to the first quarter of 2024, the total number of ride-hailing bookings nearly tripled. Several factors have driven this rapid growth in ride-hailing services. There are several reasons that cause the booming growth of the ride-hailing market, according to a survey *(Alejandro and Wesley, 2019)* of ride-hailing users, as listed in Figure 1.3, main reasons include shortage of parking space in cities, growing demand for convenient transport, low transportation cost and lack of car ownership. More specifically, ride-hailing improves the comfort and security of riders for several types of trips and increases mobility for car-free households and for people with physical and cognitive limitations. Ride-hailing has the potential to be more efficient for rider-driver matching than street-hailing. Ride-hailing is expected to reduce parking requirements, shifting attention towards curb management. These days, more and more people are tending to choose ride-hailing services. This has led to a growing market for ride-hailing services, and now ride-hailing has become a large part of citizens' daily travel modes and is still growing.

[1] *: Dara Khosrowshahi, Uber: Making it easy to go green, 2023, https://www.uber.com/newsroom/go-get-zero/*

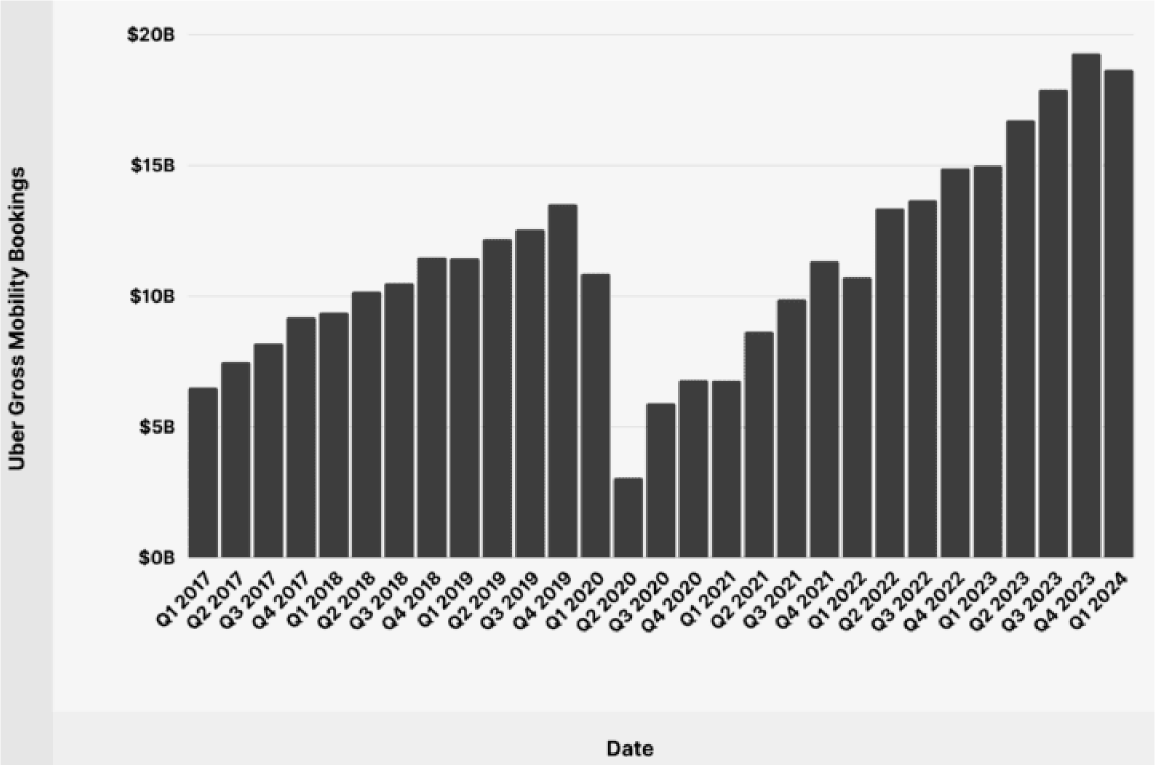


Figure 1.2 Uber's gross ride-hailing bookings from Q1 2017 to Q1 2024. [1]

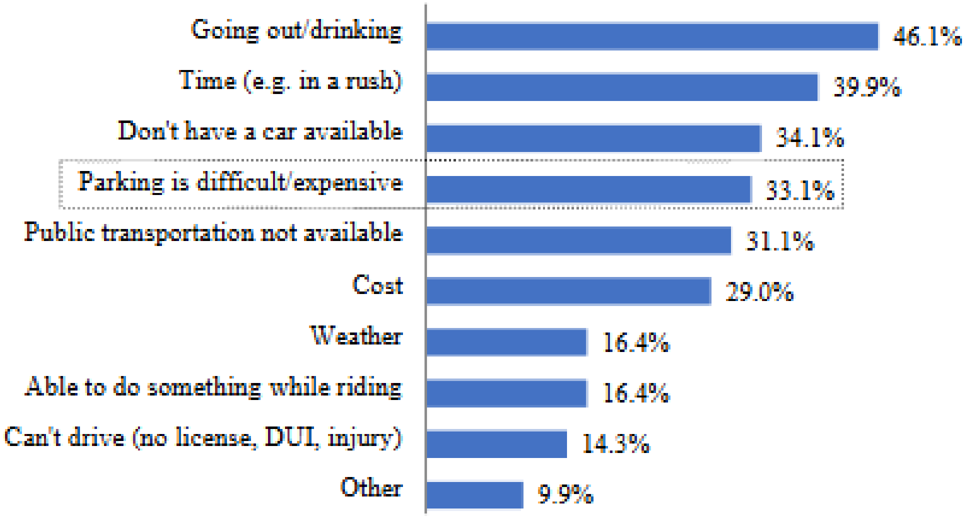


Figure 1.3. Reasons and percentages of people use ride-hailing services. [2]

In recent urban public transportation modes, ride-hailing has become one of the primary choices for urban citizens *(Sikder, 2019)*. However, the supply-demand imbalance between drivers and riders always causes extra stress to the ride-hailing system. It often faces challenges of uneven supply and demand relationships both in time and in space. For example, riders may experience longer waiting times during rush hours. Drivers may also find it difficult to receive ride requests in certain areas like city outskirts or during off-peak hours. This issue is particularly exacerbated in high density urban environments or during rush hours like evening peaks. Such imbalances can lead to decreased system efficiency, reduced user experience for both drivers and passengers, and lower overall revenue for ride-hailing companies. In the long term, it can also hinder the achievement of expected goals in reducing

[1] *: Brian Dean. Uber Statistics: How Many People Ride with Uber. 2024. https://backlinko.com/uber-users.*

[2] *: Alejandro and Wesley. Impacts of Ride-sourcing on VMT, Parking Demand, Transportation Equity, and Travel Behaviour. 2019. https://rosap.ntl.bts.gov/view/dot/42496*

carbon emissions. In this context, ride-hailing systems not only need to adapt to growing demand, but also need to optimize the overall system efficiency to achieve the ultimate goal of environmental sustainability, enhanced user experience, and economic benefits. It forms a multi-task optimizing problem for stakeholders of users, ride-hailing company and municipalities. Users are primarily interested in reduced waiting times and matching rate, while ride-hailing companies focus on maximizing their revenue and maintaining a high level of service efficiency. Municipalities, on the other hand, are concerned with increasing transport efficiency as well as reducing carbon emissions. These interests can be summed up into two major aspects, ride-hailing system efficiency and user experience.

It is beneficial to optimize ride-hailing systems. From the environmental perspective, applying advanced optimizing algorithms and data analysis to deal with the supply-demand relationship can reduce driver's idling times and detours, thereby significantly reducing the carbon footprint of ride-hailing systems. In terms of user experience, optimizing ride-hailing systems can shorten wait times, increase matching success rates, and enhance overall service reliability. From the economic perspective, improving the efficiency of the ride-hailing systems can help lower operational costs for ride-hailing companies. Better resource management in sharing economy and increased system efficiency can boost the number of matched orders and enhance net profit from the ride-hailing system. Additionally, as more consumers are willing to use ride-hailing services, companies that can provide a better user experience are more likely to gain a larger market share. By optimizing these aspects, ride-hailing systems can better meet the evolving market demands, provide a more stable and reliable service for both riders and drivers, and play a more crucial role in achieving the goal of net-zero emissions in 2050.

In previous studies, the ride-hailing system was divided into four main modules, pricing, online ride matching, driver-relocating and navigating with the online matching module being the most influential on the user experience for both riders and drivers. The online matching module directly impacts waiting times and the success matching rate for both riders and drivers. In the online matching module, there are two critical variables that play an important role in determining the scale of the matching pool and the matched pairs, which are the matching time-window and the matching radius.

Many recent studies have explored the optimization of ride-hailing matching systems from various perspectives. Among these studies, many have used Reinforcement Learning (RL) to optimize ride-hailing matching systems from different aspects. Reinforcement learning (RL) is a machine learning paradigm that trains an agent to take optimal actions (measured by total cumulative reward) through interaction with the environment and getting feedback signals. It is a class of optimization methods for solving sequential decision-making problems with a long-term objective in a stochastic environment. Thanks to the rapid advancement in deep learning research and computing power, the integration of deep neural networks and RL has generated explosive progress in solving complex large-scale decision problems attracting huge amount of renewed interests and studies in the recent years. As detailed in Chapter 2, these studies have improved efficiency and user experience for the ride-hailing system in several ways, including adjusting both the matching time-window and matching radius *(Yang et al. 2020)*, and exploring the impact of optimizing the matching time-window solely. The matching time-window is the period during which the system only intakes travel demands and available drivers, and dynamic adjustments to this time-window can increase matching rates and reduce waiting time *(Qin et al. 2023)*. Similarly, optimizing the matching radius has proven beneficial together with the optimized matching time-window. Despite these advancements, there is a noticeable gap in the research: no studies have focused solely on optimizing the matching radius. This study aims to address this knowledge gap by introducing a dynamic matching radius that adjusts in real-time based on the ride-hailing environment's real-time circumstances.

1.2 Research Question

In the current ride-hailing platforms, both matching time-window and matching radius remain fixed over location and time. The optimization direction of this research aims to apply dynamic matching radius to the batched matching system for the ride-hailing platform to increase user experience and system efficiency.

Hence, the main research question is defined as:

RQ: In a ride-hailing system, when the matching time window is fixed, what is the policy to determine the optimal matching radius for riders and drivers with a goal of achieving a higher matching rate, shorter pick-up distance and a higher driver utilization in different urban areas with different supply-demand relationships?

The sub research questions (SRQ) to support the main research question are defined as:

SRQ 1: What are the current methods for determining the matches between riders and drivers in ride-hailing systems, and what challenges do they face?

- **SRQ 1.1:** What strategies are currently utilized for optimizing the matching system?
- **SRQ 1.2:** Under what conditions do these methods encounter issues or limitations?
- **SRQ 1.3:** What are the requirements and constraints for implementing these methods?

SRQ 2: How effective is the ride-hailing simulator in replicating real-world scenarios for testing the RL-based matching radius optimization?

- **SRQ 2.1:** What are the key features and parameters of the ride-hailing simulator, and how accurately do they represent real-world conditions?
- **SRQ 2.2:** How is variability in urban environments handled by the simulator, including different supply-demand relationships in different urban areas and traffic conditions?
- **SRQ 2.3:** What are the metrics used to assess the matching efficiency and the user experience in ride-hailing simulator as well as real ride-hailing environment?

SRQ 3: How can the proposed RL framework be implemented to address the challenges in optimizing the matching radius for ride-hailing systems?

- **SRQ 3.1:** What are the state-of-the-art RL algorithms applicable to this problem?
- **SRQ 3.2:** How to design the RL algorithm and how can the RL algorithm handle the dynamic and complex nature of ride-hailing matching radius optimization?
- **SRQ 3.3:** What are the key parameters in the proposed RL algorithm and to what extend can they affect the performance of the RL algorithm?
- **SRQ 3.4:** How does the proposed RL framework perform in various ride-hailing scenarios compared to traditional methods?

1.3 Objective and Research Contributions

The ultimate goal of this study is to improve the ride-hailing matching system's overall efficiency and user experience by using Deep Reinforcement Learning (DRL) to dynamically optimize the matching radius. This approach can promise better resource management in sharing economy (idling drivers), particularly in urban areas and during rush hours, and ultimately enhancing the overall performance and user experience for the ride-hailing system. In this study, the system manager's decision of determining the matching radius of each cell for the ride-hailing system is modelled as a Markov Decision Process. The overall performance of the system is considered as the reward, this reward will concern the matching rate, the average pick-up distance and driver utilization in the ride-hailing matching system. To determine the optimal matching radius, a virtual learning environment (ride-hailing simulator) is created, allowing agents to engage in online Reinforcement Learning interactions. Using parameters learned from a real-world dataset named RideAustin, such as the idle drive arrival rate and start locations of each ride, the learning is conducted using the map of Austin City in Texas to simulate an online Reinforcement Learning environment that closely resembles the real-world setting.

The expected output of this research is a matching radius strategy (policy) that considers regional ride-hailing supply-demand relationships and the its long-term effect. A policy defines the learning agent's way of behaving at a given time. Roughly speaking, a policy is a mapping from perceived states of the environment to actions to be taken when in those states *(Sutton and Barto, 1998)*. In ride-hailing system, the policy can give the suitable matching radius which can determine the scale of the matching pool, compared with fixed and static matching radius. The goal of this research is to optimize the overall system performance and user experience. As a larger matching radius may cause a longer waiting time, a conflict between system efficiency and user experience is inevitable, so there is a trade-off between larger and smaller matching radius. To further simplify the model, it is also assumed that the demands only vary in different times of the day. For the user experience of both drivers and riders, the research ultimately contributes to the long-term development of ride-hailing systems.

In order to answer the research questions discussed in the last section, this study aims to contribute on three major aspects to optimize the matching radius of the ride-hailing matching system. These contributions of this study are:

- **Ride-hailing matching modelled as a Markov Decision Process (Ride-hailing matching MDP):** The ride-hailing matching problem is modelled as a Markov Decision Process to quantitatively solve the matching radius optimization problem. The state is defined as the real-time supply-demand relationship, the action is the matching radius and the reward function represents the overall system performance.
- **Ride-hailing Matching Simulator:** A realistic ride-hailing matching simulator is designed and developed especially for studying the matching process of the ride-hailing system. It is based on actual ride-hailing data and developed to control irrelevant variables in the ride-hailing market, providing a basis for subsequent research on ride-hailing matching systems.
- **Multi-replay-buffer Deep Deterministic Policy Gradient Algorithm (MDDPG):** This study extended the current DDPG framework with an additional replay buffer to optimize the ride-hailing matching radius. Those extensions are made to improve the performance of the Reinforcement Learning method in solving the ride-hailing matching radius optimization problem. These extensions enable the agent to better handle the complexity and uncertainty of the ride-hailing market, better understand what are the good actions and improve its learning efficiency and overall learning performance. The results indicates that these measure can improve the convergence and the learning performance compared to the original DDPG algorithm.

2

Literature Review

The objective of this literature review includes two parts: reviewing the ride-hailing system and reviewing Reinforcement Learning approaches. The first part is a summary of the existing approaches to optimize the matching radius for the ride-hailing system, focusing on their different objectives, findings and contributions. The second part mainly focuses on the Reinforcement Learning method with a brief introduction to different types of Reinforcement Learning. Reinforcement Learning has already been proven for its ability to handle complex, dynamic optimization problems with its adaptive and continuous learning capabilities, it is well-suited for handling and solving various optimization problems in ride-hailing systems *(Qin and Zhu, 2022)*. The second part of this review also summarizes a number of approaches to solve ride-hailing matching optimization problem with RL, aiming to investigate suitable methods and Reinforcement Learning algorithms. Relevant research papers and literature will be systematically searched and obtained from credible sources such as Google Scholar. The summary of the literature review and the research gap identified addressing the matching radius optimization problem is introduced in the remaining paragraph of this chapter.

2.1 Ride-hailing Systems

2.1.1 Ride-hailing Definition

There are various definitions for ride-hailing services in the literatures, the most common definitions define the ride-hailing services as a common kind of sharing economy. The concept of the sharing economy was first proposed in 2008 and is defined as collaborative consumption through sharing, exchanging, and renting resources without the need to own these items *(Lessig et al., 2008)*. Then it is further referring to the exchange of capital, assets and services between individuals using internet-based platforms for the sharing of underutilised resources, usually with a low transaction cost and the promise of increased efficiency, environmental benefits and economic growth *(Avital et al., 2014)*. In the background of digital economy, the sharing economy framework is shown in Figure 2.1, defined as the interaction between the providers of shared items and users mediated by IT platforms, facilitating access to items instead of ownership *(Karobliene et al., 2021)*. *Fielbaum and Tirachini (2020)* defines the ride-hailing as one of the sharing economy. With ride-hailing, a traveller who wants to take a specific trip is matched through a mobile application with a driver willing to satisfy that demand in a private car. Ride-hailing platforms can remove the exchange of cash and apply basic economics to match supply and demand by dynamically adjusting prices.

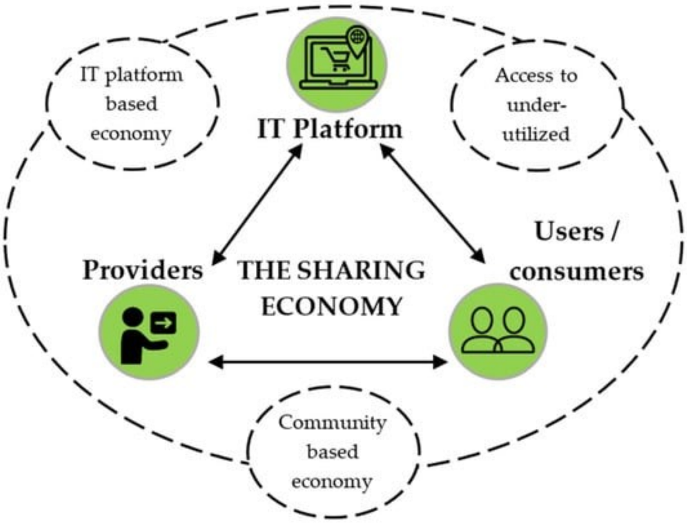


Figure 2.1. Theoretical framework of the sharing economy. [1]

[1]: *Karobliene et al. The Sharing Economy in the Framework of Sustainable Development Goals. 2021. https://www.mdpi.com/2071-1050/13/15/8312*

In this context, the ride-hailing is a form of sharing economy, as shown in Figure 2.2. It matches travel demands (riders) with available vehicles (idle drivers) through a central ride-hailing platform, providing on-demand ride-hailing services. This system leverages the core concept of the sharing economy, which is to achieve better resource utilization through the sharing of available resources (idle cars) without the need for the ride-hailing providers to actually own these resources. The ride-hailing platform matches ride-demands and drivers via a central matching system, allowing riders to ride a vehicle when needed and drivers to take orders during their idle time, thereby making full use of time, vehicles and drivers. Ride-hailing not only improves resource utilization (vehicle vacancy) but also offers travelers a flexible and convenient travel option. It also helps city municipalities to optimize transportation and mitigate environmental pollution. In a word, ride-hailing adapts the concepts from the sharing economy, efficiently utilizes resources, and is able to lower transportation costs for citizens.

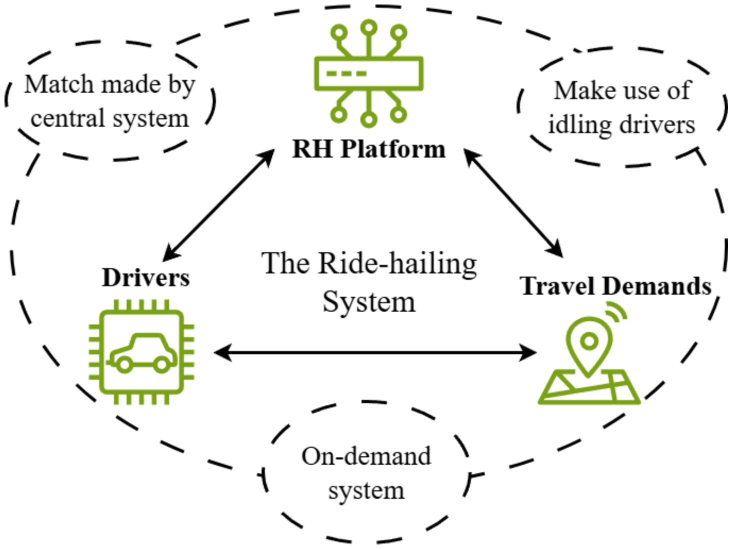


Figure 2.2. Theoretical framework of the ride-haling system.

The ride-hailing system is the core of the ride-hailing service, encompassing various functionalities to ensure the smooth operation of the entire service. In ride-hailing system, functional components are generally classified into four distinct functional modules: the pricing module, online matching module, vehicle reposition module, and navigation module *(Qin and Zhu, 2022)*. When a potential passenger submits a trip request, the pricing module offers a quote, which the passenger either accepts or rejects. Upon acceptance, the matching module attempts to assign the request to an available driver. Depending on driver pool availability, the request may have to wait in the system until a successful match. Pre-match cancellation may happen during this time. The assigned driver then travels to pick up the passenger, during which time post-match cancellation may also occur. The pick-up location is usually where the passenger is making the request or he/she specifies. In some cases, it could be a public designated area, e.g., outside an airport or train station. After the driver successfully transports the passenger to the destination, she receives the trip fare and becomes available again. The repositioning module guides idle vehicles to specific locations in anticipation of fulfilling more requests in the future. Following the reposition recommendations is usually on a voluntary basis unless it is an autonomous ride-hailing setting. Hence, it is common that the platform offers incentives to drivers for completing the repositions. The navigation module provides drivers with optimal travel routes and traffic information to ensure riders reach their destinations safely and on time. In recent studies, the online matching module is the main optimization focus, it is critical for enhancing the ride-hailing system's capability to provide a better user experience for both riders and drivers *(Yan et al., 2019)*. The optimization for each specific module is discussed in section 2.1.3.

2.1.2 Ride-hailing Matching Strategy

In the matching module of the ride-hailing system, the matching strategy determines the efficiency of the matching process. Two main matching schemes are often utilized in ride-sourcing platforms: broadcast and dispatch modes. In broadcast mode, ride-sourcing platforms merely serve as a transaction intermediary; they collect the ride requests from passengers and broadcasts such requests to idle drivers. Each idle driver aims to optimize his/her individual utility (reflected by the pick-up distance, order value, and destination, among others) by selecting one of the broadcasted requests. In dispatch mode, the platforms match idle drivers and waiting passengers to optimize overall system efficiency. This mode is now widely adopted by many ride-sourcing TNCs, such as Didi and Uber. With the help of advanced mobile technologies, online platforms can trace the status of each passenger request and each idle driver, such as real-time location and cumulative waiting/idle time, and detect the current supply–demand conditions, such as the numbers of idle drivers and waiting passengers *(Yang et al. 2020)*. In other studies, *Wang (2018)* and *Yang (2022)* categorize ride-hailing matching strategies into two main kinds: nearest matching (greedy matching) and batched matching (delayed matching). In nearest matching strategy, a rider is immediately matched with the closest available driver *(Shi et al., 2023)*. This approach works well for most riders when demand is low, but it often leads to excessively long wait times for other riders. Across an entire city, these extended wait times can accumulate to extremely high levels, exacerbating issues such as queuing or inadequate driver availability. Therefore, this matching strategy is no longer adopted by most ride-hailing platforms. Instead, the strategy named batched matching is widely adopted by many ride-hailing platforms such as Uber, Didi, and Lyft *(Uber's marketplace website)*. Unlike the nearest matching strategy, which immediately matches riders and drivers at the moment ride requests are created, batched matching allows the system to delay matching for a short period. It groups multiple ride requests and assigns drivers simultaneously. The principle is to model the matching problem between ride demands and available drivers as a weighted bipartite matching problem, solving it to optimize for reduced overall waiting times or shorter pick-up distances *(Xu et al., 2018)*. By considering a broader range of potential matches cross the system, this method can improve overall system efficiency. Figure 2.3 illustrates the operational principles of batched matching.

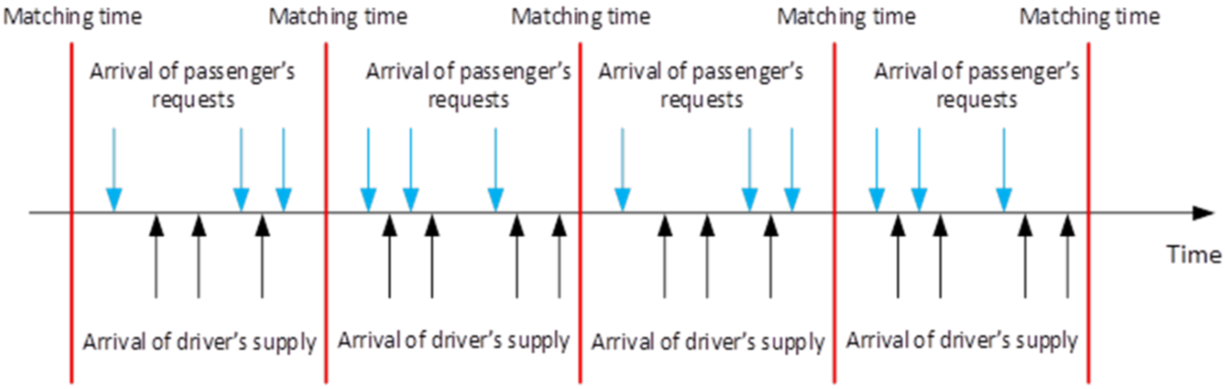

Matching time
Matching time
Matching time
Matching time
Matching time
Arrival of passenger's requests
Arrival of passenger's requests
Arrival of passenger's requests
Arrival of passenger's requests
Time
Arrival of driver's supply
Arrival of driver's supply
Arrival of driver's supply
Arrival of driver's supply

Figure 2.3. The process of batched matching in ride-hailing platform.

In this context, online matching between available drivers and waiting riders is one of the most key components and it is essential to be optimized. The success of these matching strategies is crucial for ensuring platform profitability and influencing customer perceptions of service quality *(Wang & Yang, 2019)*. Additionally, addressing negative externalities such as empty-car cruising *(Wang et al., 2014)*, the matching problem plays a significant role in overall system performance, user experience and profit generation. Another research *(Chen, 2023)* has shown that driver engagement is one of the vital components in a ride-hailing system. Driver engagement can be strongly influenced by driver income, idling time and the number of matched rides. These can be summed up in two factors: the matching rate and the driver utilization rate. However, due to the increasing popularity of ride-hailing services, these two factors are not always satisfactory. This is because high demand for ride-hailing can lead to additional pressure on city traffic, increasing competition between drivers and an imbalance between supply and demand relationship of the ride-hailing market. This could result in a lower matching rate, as well as longer waiting time and lower driver utilization rate, causing a lower reliability of the ride-hailing system. For riders, in the daily use of ride-hailing system, they often encounter situations where the wait time is too long or there are no cars available. This can also be summed up in the matching rate and the average pick-up distance. Due to insufficient vehicles in the matching pool and an imbalanced relationship between riders and drivers, this situation can lead to a poor user experience or inconvenience for people. Therefore, developing and optimizing the ride-hailing matching process, including optimizing variables or developing advanced matching strategies, are therefore essential for improving the efficiency and user experience of ride-hailing services.

Current research is focused on optimizing variables for the batched matching method. The batched matching method strategy involves two key spatiotemporal variables: matching time interval (matching time-window) and matching range (matching radius) *(Yang et al., 2020)*. The scale of the matching pool is determined by these two factors. Figure 2.4 shows how these factors determine the scale of the matching pool. The platform continuously adjusts these variables in real-time to enhance system performance, focusing on optimizing rider wait times and minimizing driver idle times. Several optimized matching strategies and matching time windows have already been developed, while the matching radius is only considered a co-factor with the matching time window. To provide a comprehensive understanding, the next subsection will discuss these studies in detail.

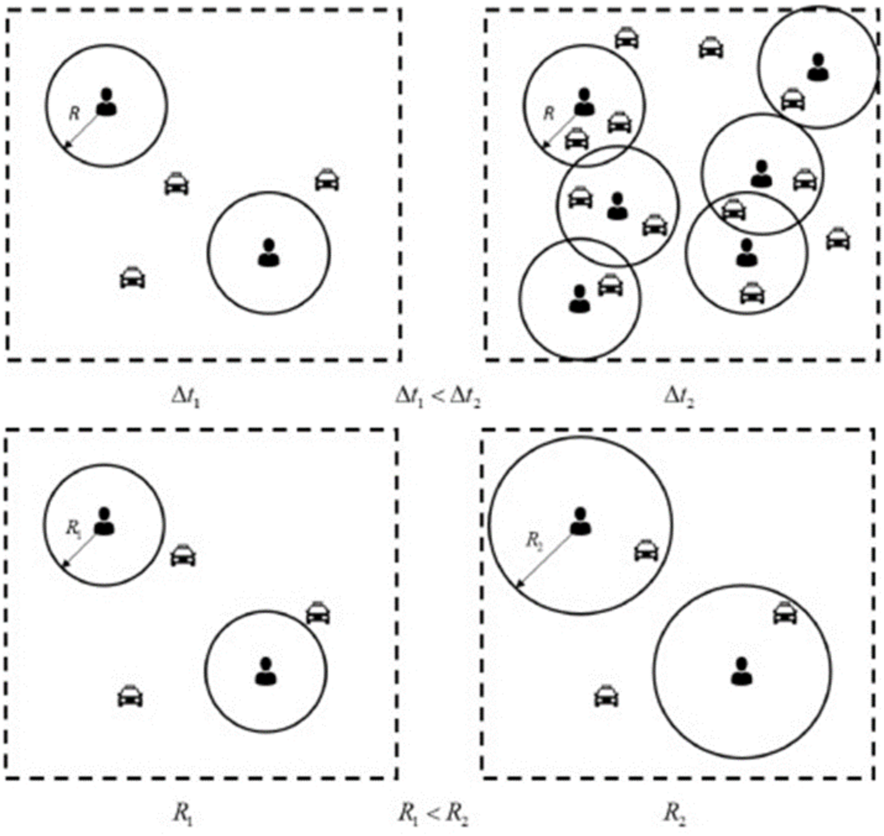


Figure 2.4. The influence of the matching time-window (up) and the radius (below). [1]

[1]: *Yang et al. Optimizing matching time interval and matching radius in on-demand ride-sourcing markets. 2020. https://doi.org/10.1016/j.trb.2019.11.005*

As one of the key components of the matching process, the matching radius directly influences the scale of the matching pool, which can have a decisive effect on the matching rate. However, current ride-hailing platforms use fixed and static matching radius to form the ride-hailing matching pool *(Yang et al., 2020)*, and only dynamic matching time windows have been studied as one of the effective aspects *(Qin et al., 2021)*. No research has been done on the matching radius in ride-hailing systems. In summary, the effect of the matching radius is worthwhile to study in order to determine the optimized matching radius.

2.1.3 Optimization Approaches

In recent studies, many approaches have been proposed to optimize the ride-hailing system. These approaches have explored the possibility of optimization from different angles and have improved the system in various aspects. This sub-section will discuss the differences between these optimization approaches and how they have inspired this study.

Firstly, some studies on the structure and theoretical operating principles of ride-hailing systems have laid the foundation for subsequent optimization approaches. These studies also attempted to model and optimize the ride-hailing system from a theoretical perspective. For the ride-hailing system, in the matching process, the ride-sourcing strategy is the most common strategy. *Zha and Yin (2016)* indicate that the ride-hailing platform serves as an intermediary that matches customers with potential drivers. It assigns order requests to idle drivers, and drivers cannot reject the assignment or they will be punished by the platform. This is the common type that is mostly used by the current existing ride-hailing service providers, such as Didi and Uber. In this type of system, the driver does not need to consider which rider to pick up, and is able to only focus on driving. This type of platform reduces the uncertainty in the system and ensures the system runs smoothly. This type of ride-souring strategy is also used in this study, in order for the agent to better focus on optimizing the matching radius only. Much academic research has explored this type of platform and many research are carried out based on this type of system. The studies by *Liu et al. (2022)* and *Qin (2022)* present a thorough review of these methodologies and tried to optimize the matching problem in this ride-souring market. Their studies give a general overview of how a ride-hailing system works and what the potential variables and components of such a system are. Many optimizations are inspired from their works. In their studies, modelling and solving the batched matching process of the ride-hailing system as a bipartite matching problem is studied, the efficiency of this approach is proved in their results. This bipartite matching approach is very inspiring as many following researches are all based on this approach. This study also uses this approach to model the batched matching problem in developing the ride-hailing matching simulator. The matches are formed by solving the bipartite matching problem in this study.

With those theoretical foundations, different approaches have studied different optimizing problems of the ride-hailing system. Many inspiring methods are used and managed to optimize the ride-hailing system from different aspects. *Xu et al. (2018)* model the order dispatching process as a large-scale sequential decision-making problem. The decision of assigning an order to a driver is determined by a centralized algorithm in a coordinated way. They studied this decision process and developed an order dispatching policy in order to optimize the ride-hailing matching system from the aspect of maximizing drivers' revenue. This method is inspiring as it is useful in dealing with driver and rider's matching problems. With the same settings, *Özkan and Ward (2020)* innovatively divided the study area into a number of cells of the same size, so the large-scale matching problem is simplified into multiple small-scale matching problems within the cells. Their approach is to use hierarchical Reinforcement Learning method to maximize the number of matched pairs. The problem is solved by dividing and hierarchical solving the overall matching problem into several sub-tasks of small-scale matchings. Another approach is to treat the matching process as a Vehicle Repositioning Problem (VRP).

Some studies focus on optimizing the matching time-window of the ride-hailing matching module. It has been demonstrated that dynamic matching time-window can improve the matching rate of the system and reduce the average waiting time *(Qin et al. 2023)*. This means that instead of using a static, fixed time interval for matching, adjusting the time window dynamically based on real-time circumstances, such as the supply-demand relationship, can lead to more efficient matches between riders and drivers. Consequently, this approach not only increases the efficiency of the ride-hailing system but also helps the users to have a better user experience. Another approach is to treat the matching time-window and the matching radius at the same time and try to find a balance between them, *Yang et al. (2020)* studied this topic in their research. They suggest that by appropriately extending the matching time interval, the system can accumulate large numbers of waiting (or unserved) riders and idle drivers and thus match the two pools with a reduced expected pick-up distance. Meanwhile, a short matching radius can reduce the expected pick-up distance but may decrease the matching rate as well. Therefore, the matching time-window and matching radius should be optimized to enhance system efficiency in terms of rider waiting time, vehicle utilization, and matching rate. This approach considered the effect of both the matching time-window and the matching radius, it clearly shows the trade-off between the them to optimize the performance of the ride-hailing system. However, the effect of matching radius solely on the system has not been studied in this paper. Also, the matching radius is studied in this paper as a number of discrete values, and the dynamic continuous matching radius and its impact are not studied. This leads to the knowledge gap addressed in this study.

Other researchers are also exploring the optimization problem of the matching system together with other modules in the ride-hailing system. *Yan (2019)* tried to introduce dynamic pricing and matching time-window into the ride-hailing system. His research shows that the pricing module combined with the matching module can significantly affect the waiting time. This approach adjusts ride fares according to supply-demand patterns in real-time, which encourages drivers to provide services when and where the needs are surged, thereby reducing waiting time. By combining dynamic pricing with matching time-window, the ride-hailing system can better adapt to uncertain supply-demand relationship, and improve overall user experience. *Daniels (2023)* divided the matching area into blocks based on neighbourhoods and introduced a Street Match Model (SMM). This approach combines the matching system with a vehicle repositioning system to optimize the regional aspects of the matching system. By dividing the service area into smaller, more manageable blocks, Daniels aims to improve the overall efficiency of ride-hailing matching. This regional optimization means that riders are more likely to be matched with drivers in a smaller area, thereby reducing waiting time, driver idling time, and improving service reliability. The cell-based matching simulator developed in this study is inspired by his approach. *Miao et al. (2016)* proposed the receding horizon control framework to maximize the matching ratio between supply and demand with minimum idle taxi driving distance. Spatiotemporal passenger demand, real-time GPS locations, and taxi occupancy are incorporated in the framework. Apart from these model-based approaches, model-free approaches, such as the Markov decision process and reinforcement learning, have been recently implemented in taxi and ride-sourcing vehicle dispatch systems *(Xu et al. 2018; Ke et al. 2019; Wang et al. 2018).*

In summary, many researchers have developed various approaches to optimize ride-hailing matching systems from different aspects. These methods improve the matching efficiency and user experience of the ride-hailing system. However, although some approaches have investigated optimizing the ride-hailing matching system by adjusting the matching time-window combined with adjusting the matching radius, as well as optimizing the matching time-window alone, no study has solely focused on optimizing the matching radius and treat it as an independent variable. This research aims to address this knowledge gap and develop a method to apply dynamic matching radius to the ride-hailing system in order to improve its matching efficiency and user experience.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a machine learning method, its principal base on interacting with the environment to gather information and taking actions base on its observations to maximize long-term rewards *(Sutton and Barto, 1998)*. Reinforcement Learning applies its algorithm to the dynamic environment, allowing the agent to learn the policy not only from existing transitions (off-line learning or past experience) but also from its own interactions with the environment. Through learning, the agent can determine which actions are good and which are not. Reinforcement Learning is widely used to solve dynamic optimization and control problems. Through its interactive learning mechanism and policy (strategy) optimization process, Reinforcement Learning is well-suited for solving dynamic optimization and control problems in dynamic systems like cliff walking and lunar landing problems *(Sutton and Barto, 1998)*. This subsection will introduce the basic concepts of Reinforcement Learning, Reinforcement Learning approaches, and their applications in ride-hailing optimization problems.

2.2.1 Reinforcement Learning Basics

Reinforcement learning differs from the more widely studied problem of supervised learning in several ways. The most important difference is that there is no presentation of input/output pairs. Instead, after choosing an action the agent is told the immediate reward and the subsequent state, but is not told which action would have been in its best long-term interests. It is necessary for the agent to gather useful experience about the possible system states, actions, transitions and rewards actively to act optimally. Another difference from supervised learning is that on-line performance is important: the evaluation of the system is often concurrent with learning. *(Kaelbling et al., 1996)*. The object of this kind of learning is for the system to extrapolate, or generalize, its responses so that it acts correctly in situations not present in the training set. The agent adjusts the policy base on feedback from the interactions. Reinforcement learning is also different from what machine learning researchers call unsupervised learning, which is typically about finding structure hidden in collections of unlabelled data. *(Sutton and Barto, 1998)*. The RL algorithm includes two major components: the agent and the environment. The agent is the decision-maker and the learner who is responsible for selecting which action to take (or which policy to employ for actions). The environment, often referred to as the simulator, encompasses everything surrounding the agent and with which it interacts. This includes all entities other than the agent itself, both observable and unobservable within the system the agent operates in *(Sutton and Barto, 1998)*. Figure 2.5 illustrates the relationship between the RL agent and the environment.

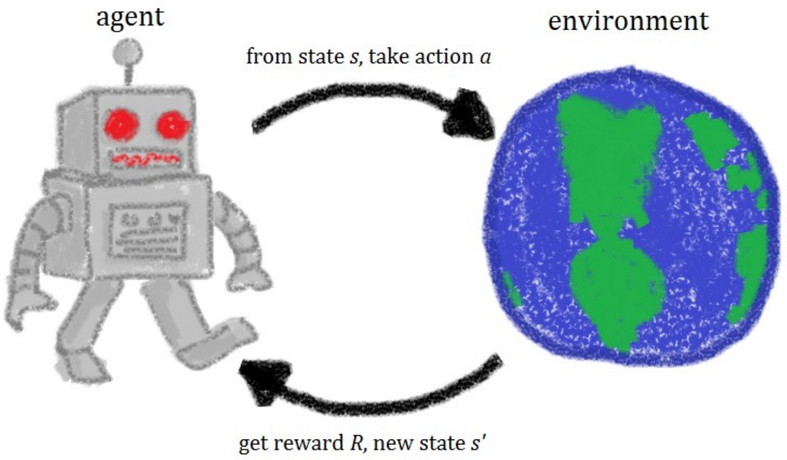

agent
environment
from state s, take action a
get reward R, new state s'

Figure 2.5. RL interaction framework between agent and environment. [1]

[1]: *Swiss Cognitive. What Is Reinforcement Learning. 2019. https://swisscognitive.ch/2019/10/25/what-is-reinforcement-learning/*

In Reinforcement Learning (RL), The environment is typically stated in the form of a Markov decision process (MDP), because many reinforcement learning algorithms for this context use dynamic programming techniques. MDP provides a mathematical framework for describing decision-making processes in RL problems. When using RL to solve dynamic optimization problems, the problem is typically first modelled as an MDP and then the suitable RL algorithm will be selected accordingly.

A Markov Decision Process (MDP) is a discrete-time stochastic control process. It provides a mathematical framework for modelling decision-making situations where outcomes are partly random and partly under the control of a decision-maker. MDPs are very useful for studying optimization problems solved through dynamic programming *(Bellman, 1957)*. Formulating the problem as a Markov decision process assumes the agent directly observes the current environmental state; in this case the problem is said to have full observability. If the agent only has access to a subset of states, or if the observed states are corrupted by noise, the agent is said to have partial observability, and formally the problem must be formulated as a Partially observable Markov decision process. In both cases, the set of actions available to the agent can be restricted. For example, the state of an account balance could be restricted to be positive; if the current value of the state is 3 and the state transition attempts to reduce the value by 4, the transition will not be allowed. When the agent's performance is compared to that of an agent that acts optimally, the difference in performance gives rise to the notion of regret. In order to act near optimally, the agent must reason about the long-term consequences of its actions (i.e., maximize future income), although the immediate reward associated with this might be negative. Thus, reinforcement learning is particularly well-suited to problems that include a long-term versus short-term reward trade-off. It has been applied successfully to various problems, including energy storage operation, robot control, photovoltaic generators dispatch, backgammon, checkers, Go (AlphaGo), and autonomous driving systems. The reward function (R) defines the immediate reward received after taking a particular action in a specific state *(Sutton and Barto, 1998)*. It is crucial in MDP and RL as it is the signal that let the agent know what the good actions are, and guides it to learn a better policy.

The purpose of reinforcement learning is for the agent to learn an optimal, or nearly-optimal, policy that maximizes the "reward function" or other user-provided reinforcement signal that accumulates from the immediate rewards. This is similar to processes that appear to occur in animal psychology. For example, biological brains are hardwired to interpret signals such as pain and hunger as negative reinforcements, and interpret pleasure and food intake as positive reinforcements. In some circumstances, animals can learn to engage in behaviours that optimize these rewards. This suggests that animals are capable of reinforcement learning. As for the machines, through this iteration, the learning agent gradually improves the policy to maximize rewards.

2.2.2 Reinforcement Learning Approaches

In Reinforcement Learning (RL) problems, the agent's goal is to maximize its accumulated reward, which includes both immediate and long-term rewards. Therefore, it must find a policy or strategy that allows it to choose actions maximizing long-term rewards. In Reinforcement Learning, the accumulated reward is denoted as

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \dots = r_t + \gamma G_{t+1} \tag{2.1}$$

where G_t is the accumulated reward at time step t, r_t is the immediate reward, and γ is the discount factor. The discount factor, ranging from 0 to 1, influences the agent's current decision-making. It determines how far the agent can see the future expected rewards *(Sutton and Barto, 1998)*. For example, if $\gamma = 0$, the agent only considers immediate rewards and acts greedily.

The value function is another important part that influence the agent's action, which determine the advantage of being in a specific state (state value function) or taking a specific action in a specific state (action value function, also called state-action value function). The state value function is defined mathematically as

$$V_\pi(s) = E_\pi\{G_t|s_t = s\} = E_\pi\left[\sum_{t=0}^{T-1} \gamma^t r_{t+1}\,|s_t = s\right] \tag{2.2}$$

Similarly, the state-action value function is defined as

$$Q_\pi(s,a) = E_\pi\{G_t|s_t = s, a_t = a\} = E_\pi\left[\sum_{t=0}^{T-1} \gamma^t r_{t+1}\,|s_t = s, a_t = a\right] \tag{2.3}$$

The goal of a Markov Decision Process (MDP) is to take optimal actions to maximize long-term rewards, i.e., the value function. While the rewards for each state and action are typically unknown, achieving maximum value is essentially about selecting actions that lead to the highest rewards, forming a policy. However, since rewards are often unknown, the Bellman equation becomes crucial as it defines necessary conditions for optimality. The Bellman equation reformulates the value function recursively, expressing the reward of a state in terms of possible future rewards of successor states. Its recursive nature allows initially unknown rewards to converge to actual values through iterative refinement *(Sutton and Barto, 1998)*. The Bellman equation is given by:

$$V_\pi(s) = E[G_t|S_t = s] = E[R_t|S_t = s] + \gamma E[G_{t+1}|S_t = s] \tag{2.4}$$

In Reinforcement Learning (RL), methods that estimate value functions are classified into value-based Reinforcement Learning, whereas those that directly optimize the policy without using a value function are referred to as policy-based Reinforcement Learning. Value function approaches attempt to find a policy that maximizes the discounted return by maintaining a set of estimates of expected discounted returns. Classic value-based RL includes DQN and Q-learning, which are both action-value-based approaches. These methods derive a policy from the values of states and actions. The exploration in this type of RL is always an additional step, not embedded in the method itself. An $\varepsilon - greedy$ with random action is the most common exploration method. Value-based RL is considered indirect as it derives an optimal policy from estimating the best value function. In contrast, policy-based RL are more efficient, they directly derive a policy mapping the best actions for the corresponding states. This type of RL includes policy gradient methods like PPO (Proximal Policy Optimization). Exploration is a part of the algorithm in this type of RL, so it is not needed for additional explorations. There are also some RL algorithms that combine ideas from both value-based and policy-based RL, such as DDPG (Deep Deterministic Policy Gradient) and SAC (Soft Actor-Critic). They adapt advantages from both value-based and policy-based RL. This property enables them to address the optimization problems with continuous action space and high-dimensional state space.

The problem with using action-values is that they may need highly precise estimates of the competing action values that can be hard to obtain when the returns are noisy, though this problem is mitigated to some extent by temporal difference methods. Using the so-called compatible function approximation method compromises generality and efficiency. Another difference lies in how the agent applies its policy in Reinforcement Learning problems. On-policy methods evaluate or improve the policy used to generate these responses and update the decision policy accordingly. Therefore, the same policy is used for exploration and exploitation. Off-policy methods evaluate or improve the data collected by other (or previous) policies, which means that the updated policy is different from the one used for data collection, for example, DQN and DDPG *(Fujimoto, Meger et al., 2019)*. An alternative method is to search directly in (some subset of) the policy space, in which case the problem becomes a case of stochastic optimization. The two approaches available are gradient-based and gradient-free methods. Gradient-based methods (policy gradient methods) start with a mapping from a finite-dimensional (parameter) space to the space of policies. Policy search methods may converge slowly given noisy data. For example, this happens in episodic problems when the trajectories are long and the variance of the returns is large. Value-function based methods that rely on temporal differences might help in this case. In recent years, actor–critic methods have been proposed and performed well on various problems.

Reinforcement Learning methods are further divided into online Reinforcement Learning and offline Reinforcement Learning based on how Reinforcement Learning utilizes the environment and data. Online learning is performed in real time, which means that the agent interacts with the environment in real-time to generate transition data and continues to improve the policy with collected data *(Levine et al., 2020)*. The exploration vs. exploitation trade-off has been most thoroughly studied through the multi-armed bandit problem and for finite state space Markov decision processes. Reinforcement learning requires clever exploration mechanisms; randomly selecting actions, without reference to an estimated probability distribution, shows poor performance. The case of (small) finite Markov decision processes is relatively well understood. However, due to the lack of algorithms that scale well with the number of states (or scale to problems with infinite state spaces), simple exploration methods are the most practical.

More generally, RL algorithms can be classified as model-based or model-free, depending on whether they employ a mathematical model for the environment and the function *(Deisenroth and Rasmussen, 2011)*. The model-based RL do not necessarily require prior knowledge of real data or real variables of the environment, which leads to a distinction between model-free methods. Model-based RL excel at using mathematical model or train a model to describe the environment, predicting future states, and analyzing the potential rewards of actions after updating the model parameters. These methods are sample-efficient as the transition probability and value function are both known to the agent, making it possible to be implemented in real-time. However, their performance often depends on the accuracy of the defined model, which can contain varying degrees of imperfections, affecting real-world applicability. Model learning from the actual environment can also be challenging, requiring significant time and effort with no guaranteed payoff. In contrast, model-free methods do not require any form of model, the policy is improved by sampling transitions from interactions between agent and the environment. This type of RL does not attempt to predict how the environment will react to actions. It is generally easier to be implemented and finetuned, but sacrifices the potential sample efficiency than the model-based RL and are primarily suitable for online learning.

The ride-hailing system is highly dynamic and complex, and is affected by real-time changes in travel demands, driver location, traffic conditions and other factors. Optimizing the ride-hailing matching system for drivers and rider is crucial to improve service quality, reduce waiting time, and improve user satisfaction. The application of Reinforcement Learning method, especially the DDPG method, is expected to improve the matching rate, matching system efficiency and user experience of online ride-hailing matching system. The next subsection will review existing research on Reinforcement Learning in online ride-hailing system.

2.2.3 Reinforcement Learning in Ride-hailing Optimizations

A number of optimizations have been done with Reinforcement Learning to address the online matching problem. To study the effect of matching policy to the ride-hailing system, a widely adopted strategy is to optimize matching time-window under a static matching radius policy *(Qin et al., 2021)*. This method accumulates a batch of potential passenger-driver matches and solve bipartite matching problems repeatedly. The efficiency of matching can be improved substantially if the matching is delayed by adaptively adjusting the matching time interval. The optimal delayed matching is subject to the trade-off between the delay penalty and the reduced wait cost and is dependent on the system's supply and demand states. In addition, this work provides a solution to spatial partitioning balance between the state representation error and the optimality gap of asynchronous matching. In this approach, a Markov Decision Process is developed, the agent is the system, the state number of batched request, estimate arrival rates of demand and supply, the action is matching the current batch or delay to the next batch. The reward after taking the action is the negative total matching wait time for all batched requests. Then the RL is applied to find the optimized policy. The optimized objective obtained from the policy is the matching time-window, the method and MDP developed in this literature illustrates how online matching problem can be optimized by adjusting matching time-window. But it only considers the matching time-window and excluded the effect of matching radius. No study has been conducted on considering the matching radius as the only effective factor.

To tackle the optimization problem, other inspiring method are also built with Reinforcement Learning. First approach is to divide the large-scale matching problem into many small and continuous static subproblems with an equal time interval *(Zha et al. 2021)*, to fit the discrete decision MDP framework. In this research, each request is matched with one idle driver by identifying the available vehicles near the pickup point of the passengers within a pre-defined radius. The overall solution approach is proved to be efficient in solving large-scale ride-hailing optimizing problems. The ride-hailing problem is also modelled as a large-scale parallel ranking problem *(Jin and Zhou 2019)*. They studied the joint decision-making task of order dispatching and fleet management in online ride-hailing platforms. To deal with unique challenges with this model, they treat each region zone as an agent and build a multi-agent Reinforcement Learning framework to facilitate a huge number of vehicles to act and learn efficiently and robustly. Researchers in Stanford University has also built a Reinforcement Learning based model to improving taxi revenue for New York City *(Wang and Lampert, 2014)*. They use reinforcement learning in vehicle optimisation of a taxi route to maximised the revenue. Although the optimisation process is set for one vehicle, if extended to a fleet of taxis, the overall fleet's efficiency could be greatly improved (less particles emitted), drivers would minimise their vacancy time and working hours, and passenger could see better Taxi coverage. They employed an RL approach to improve taxi revenue but only from taxi trip data (and not from taxi trajectory data). Due to learning from trip data, they are forced to make many approximations on movement between trip end and trip start events. Another study addressing a block matching system *(Feng et al. 2022)* also did the similar approach. They proposed a block matching system which is a special type of matching mechanism, where the region of interest is partitioned into blocks, and on-demand matching is separately and simultaneously conducted in each block. A simpler single agent Reinforcement Learning based approach is studied by *Liu and Wu (2022)*, they proposed a single-agent deep Reinforcement Learning model for the vehicle dispatching problem called deep dispatching, by reallocating vacant vehicles to regions with a large demand gap in advance. The vehicle dispatching problem is translated in analogy with the load balancing problem in computer networks. The problem of high concurrency of dispatching requests is addressed by sorting the actions as a recommendation list, whereby matching action with requests. These studies demonstrate the feasibility of breaking large-scale problem into small-scale subproblems, and using Reinforcement Learning to address these matching and system optimization problems is also proved by *Wang et al. (2019)* and their study. They showcase the efficiency of Reinforcement Learning in tackling optimization problems that involve considerations of time and long-term impacts of each action taken by the learning agent. These works inspire the use of Reinforcement Learning in this research proposal to address the matching radius optimization problem for the ride-hailing system.

In summary, the crucial role of online matching in ride-hailing systems has been verified, but there are only a few studies on optimizing the matching time window and matching radius using Reinforcement Learning for ride-hailing systems. Table 2.1 summarizes all related approaches using Reinforcement

Learning to address online matching problems in ride-hailing systems. Optimizing the matching radius with Reinforcement Learning can lead to higher efficiency in ride-hailing systems, resulting in a higher matching rate, lower overall idling travel costs, and an improved user experience. However, unlike the well-studied matching time window, there is still a gap in the in-depth optimization of the matching radius as an independent factor in this field. Although the influence of the matching radius on system efficiency and user experience has been demonstrated, questions remain unstudied regarding the extent of its impact, the optimal matching radius, its impact if dynamic compared to a fixed radius, and how to determine it. This research aims to investigate the matching radius as an independent factor in the matching system, employing Reinforcement Learning methods to optimize it and explain how the optimized dynamic matching radius influences the ride-hailing system's overall efficiency and user experience for all stakeholders. The last row of the Table 2.1 shows a comparison of research topics, MDP settings and research methods between this study and other approaches.

Table 2.1. Approaches for solving RH online matching problems with RL

Authors	Research Topic	State (S), Action (A) and Reward(R) of MDP	Method
Feng et al., 2023	Optimize pricing, matching framework	S: supply vertices A: comparative ratio R: match rate	Mathematical
Xu et al. 2018	Optimize single driver's decision in ride-hailing matching process	S: driver and demand's location and time A: accept the ride or refuse the ride R: total net profit for a time period	TD
Wang et al., 2019	Optimize batched matching process and its batch length	S: current batch size, current supply-demand pattern A: batch length R: summed weights of weighted edges in bipartite graph	Q-learning
Wang and Lampert, 2014	Optimize ride-hailing system revenue	S: drop-off zone at a corresponding time A: remain location or reposition for each vehicle R: average trip fair for a pickup zone	SARSA
Liu, Wu et al., 2022	Optimising on-demand vehicle dispatching process	S: current time, past satisfactory and demand locations A: possible dispatching destination for vehicles in each grid R: total net profit made from the system	Single agent DRL
Wang et al., 2018	Optimize single driver's decision in ride-hailing matching process	S: driver and demand's location, supply-demand pattern A: accept the ride or refuse the ride R: total net profit for a time period	DQN
Holler et al., 2019	Optimize drivers' behaviour in ride-hailing matching process	S: locations of all drivers and demands A: match the drivers and demands or reposition R: total system revenue minus reposition cost	DQN
Ke et al., 2020	Optimize batched matching process and its sequence	S: number of supply-demand and arrival rate in each cell A: match supply-demand or delay the match in each cell R: total trip price, pick-up distance and match time-window	A2C, PPO
Qin et al., 2021	Optimize delayed matching process and time-window	S: vehicle number, rider and driver's arrival rate A: cell join the matching pool or delay the match R: total pick-up waiting time	ACER
Jin, Zhou et al., 2019	Optimize fleet management base on hexagonal cells	S: rider and driver amount, trip duration and prices A: match or reposition (cell), sub-goal (manager) R: distance to sub-goal (cell), total net revenue (manager)	Hierarchical MARL
Li et al., 2019	Optimize drivers' behaviour in ride-hailing matching process	S: driver and demand's location, time and availability A: accept the ride or refuse the ride R: total system net profit for a time period	MARL, AC
Chen et al., 2023	Optimize broadcasting radius in ride-hailing DBRAS	S: number of supply-demand in each cell, time, arrival rate A: broadcasting radius for each cell R: matching rate, system performance	WESM RL
Zhao, 2024	Optimize matching radius	S: number of riders and drivers in each cell A: matching radius for each cell R: overall performance and user experience score	MDDPG

3

Problem Formulation

This study aims to optimize the matching radius in ride-hailing systems by modelling the decision-making process as a Markov Decision Process (MDP). The goal is to optimize overall system performance and user experience by balancing the matching rate, pick-up distance, and driver utilization rate. This section will discuss the formulation of this optimization problem, the definition of the ride-hailing matching MDP, and the roles of RL agent and the environment.

3.1 Problem Description

The objective of this study is to maximise the overall system performance by modelling the radius-deciding process in a ride-hailing system as a Markov Decision Process (MDP). The proposed policy must meet real-time adaptability and computational feasibility. That is, the dynamic matching radius policy must be quickly adapted to respond to changes in demand patterns, driver availability, and traffic conditions to maintain high system performance. The solution should be implementable in real-world scenarios, considering computational efficiency and practical deployment in existing ride-hailing systems. The detailed settings of the defined Markov Decision Process (MDP) are discussed in the following subsections, those settings include state, action and reward. As the problem is designed as a model-free approach, the state transitions are simulated by defining a simulator.

Reinforcement Learning has already been proven for its ability to handle complex, dynamic optimization problems due to its adaptive learning capabilities *(Qin and Zhu, 2022)*, it is well-suited for handling and solving the matching radius optimization problem in ride-hailing systems. To achieve this, a virtual learning environment (simulator) was developed for RL, allowing agents to participate in online RL interactions. Using parameters from the RideAustin dataset, such as idle driver arrival rates and ride start locations, simulations were conducted on a map of Austin, Texas, to closely reflect real-world conditions. Figure 3.1 shows the optimization framework developed in this study. It illustrates how the simulator works together with the RL to optimize the real-world ride-hailing matching system. As shown in the figure, the process starts from the actual ride-hailing market at the bottom of Figure 3.1. The simulator simulates the ride-hailing market in reality by fitting the actual driver-rider supply-demand relationship data and the actual driver-rider location data. In the entire RL process, the agent learns the policy through interaction with the simulation environment. Specifically, the agent in this study is the ride-hailing system operator. The operator outputs the matching radius of each cell based on the observed state of the current ride-hailing simulator (supply-demand state). The ride-hailing simulator receives this matching radius set, simulates a step of the matching process, and gives an immediate reward based on the impact of the matching radius on the ride-hailing system (matching rate, average pick-up distance, driver utilization), the overall performance score is used as this immediate reward. At the same time, the simulator will also output the next state to the operator. This transition containing state, action, reward, and next state will be transmitted to the RL optimizer in real-time, as shown in the upper right corner of Figure 3.1. The RL optimizer uses these transitions through the RL algorithm to adjust and update the operator's policy, thereby gradually optimizing its decision-making ability during the continuous training iterations. Finally, after training, the operator will interact with the actual ride-hailing market, as shown in the interactive loop between the operator and the real ride-hailing market in Figure 3.1. The operator will dynamically output the optimized matching radius based on the real-time supply and demand relationship between drivers and riders to improve matching efficiency, improve user experience, and optimize driver utilization. This optimization framework for the ride-hailing matching system developed in this study is one of the first models for improving the ride-hailing system, proving a solid foundation for future studies.

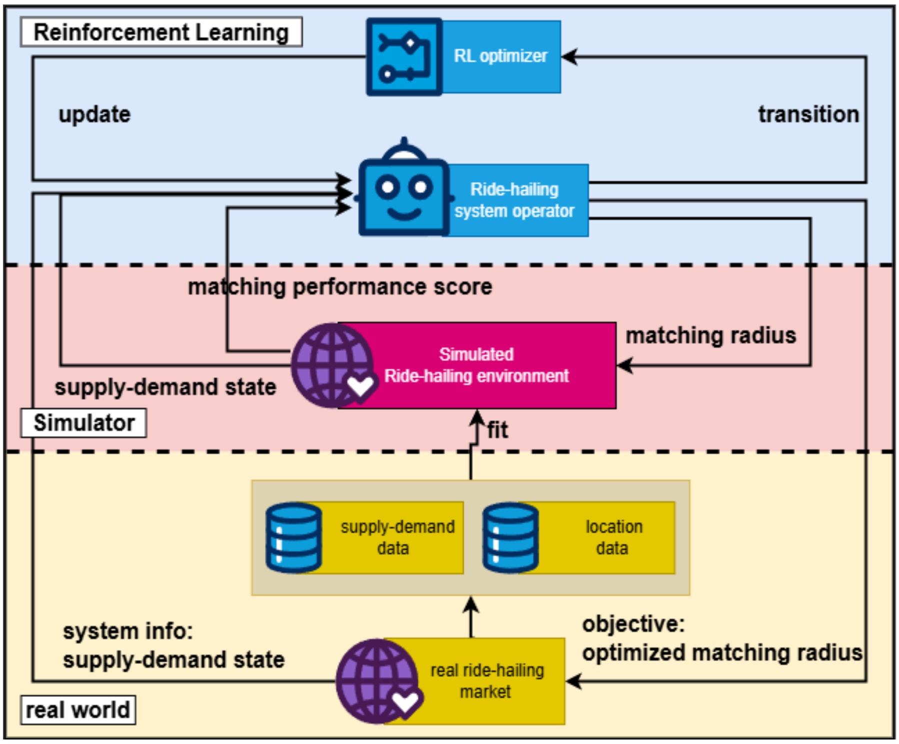


Figure 3.1. Framework of the learning structure in this study.

3.2 Markov Decision Process Definition

State Definition

In this study, the decision-making process of determining the matching radius for each cell in the ride-hailing system by the system agent is modelled as a Markov Decision Process (MDP). In a Markov Decision Process (MDP), the state represents the observable signal from the environment that can reflect the current system status to the agent *(Sutton and Barto, 2018)*. In the ride-hailing system, the system's state is defined by the real-time travel demand and the locations of idle vehicles in each region. The observations in each cell at time t is denoted as s_t. Let A denotes the number of cells of the whole studied area, then the sub-observation of cell i at time t is denoted as $o_{i,t}$. The state at time-step t can be derived as

$$s_t = \{o_{1,t}, o_{2,t}, o_{3,t} \dots o_{i,t} \mid i \in A\} \tag{3.1}$$

The environment is further defined as the place where the agent interacts. It encompasses everything surrounding the agent and with which it interacts. This includes all entities other than the agent itself, both observable and unobservable within the system in which the agent operates *(Sutton and Barto, 1998)*. In the case of the ride-hailing system, the environment is the local ride-hailing market. In the ride-hailing market, the observation the agent can observe from the environment is the real-time supply-demand relationship between drivers and travel requests. The implementation of the observations in this study is discussed in detail in the next chapter.

Action Definition

The action is taken by the system operator, which is the matching radius for each cell. The action is a collection of sub-actions of all cells at time step t. With this set of matching radiuses, a matching is formed. A matching pool refers to a collection of all the riders with available drivers within their matching radius. The matching radius of a matching area determines how many drivers will be included in the matching pool. The larger the matching radius, the bigger the matching pool is. The matching radius can be different across cells.

Let $a_{i,t}$ denotes the matching radius (action) in cell i at time-step t. All the riders in the same cell will have the same matching radius, while riders in different cells can have different matching radius. The action at time step t can be derived as

$$a_t = \{a_{1,t}, a_{2,t}, a_{3,t} \dots a_{i,t} \mid i \in A\} \tag{3.2}$$

Reward Definition

The reward signal is critical in MDP. A reward signal defines the goal in a Reinforcement Learning problem. In each time step, the environment sends to the Reinforcement Learning agent a single number, a reward. The agent's sole objective is to maximize the total reward it receives over the long run. The reward signal indicates what is good in an immediate sense *(Sutton and Barto, 2018)*. In the ride-hailing MDP of this study, the reward signal $R_t(s_t, a_t)$ is an overall system performance score. This performance score at time step t is a weighted sum of three system performance indicators: matching rate r_t^{mr} at with weight w_1, average pick-up distance r_t^{pd} with weight w_2 and driver utilization rate r_t^{du} with weight w_3. The reward is this MDP is then defined as

$$R_t(s_t, a_t) = w_1 r_t^{mr} + w_2 r_t^{pd} + w_3 r_t^{du} \tag{3.3}$$

In the reward function, w_1, w_2, w_3 are the multitask optimization weights for each of the reward components. Firstly, the initial ride-hailing weights are set by the feedbacks from ride-hailing users and other stakeholders and their preferences. After that, a series of experiments with different sets of weights are implemented on the simulator trying to find balance between each reward components, the criteria include reward analyzing and sensitivity analysis.

By introducing a dynamic matching radius policy as well as a realistic data-based ride-hailing simulator, the objective of Reinforcement Learning algorithm based on the defined ride-hailing MDP is to maximize the expected long-term cumulative value $Q_\pi(s, a)$ at each time step. Let G_t denotes the action value of action a at time step t under the state s. The discount factor is denoted as γ, and R_t denotes the immediate reward as derived above. The objective value function is then derived as

$$max\ Q_\pi(s, a) = \max E_\pi\{G_t | s_t = s, a_t = a\} = max\ E_\pi \left[\sum_{t=1}^{T} \gamma^{t-1} R_t \,|s_t = s, a_t = a\right]$$

$$= max\ E_\pi \left[\sum_{t=1}^{T} \gamma^{t-1}\left(w_1 r_t^{mr} + w_2 r_t^{pd} + w_3 r_t^{du}\right) |s_t = s, a_t = a\right] \tag{3.4}$$

The objective of the ride-hailing MDP defined in this study is to use Reinforcement Learning MDDPG algorithm to optimize the policy π in order to get the highest cumulative overall reward as shown in equation 3.4. It is also noted that the reward function used in the ride-hailing matching simulator developed in this study and the final training process of the MDDPG is further developed based on the reward function 3.3. A new method of defining the reward function is used in the final training process, this part is further discussed in Chapter 4.

3.3 Model-free Approach

In this research, a model-free approach is used to model the state transition of the ride-hailing system for capturing the real-world dynamics without defining mathematical models. In reinforcement learning (RL), a model-free algorithm (as opposed to a model-based one) is an algorithm which does not estimate the transition probability distribution (and the reward function) associated with the Markov decision process (MDP) *(Sutton and Barto, 1998)*. The transition probability distribution (or transition model) and the reward function are often collectively called the model of the environment (or MDP), hence the name "model-free". A model-free RL algorithm can be thought of as an explicit trial-and-error algorithm. Typical examples of model-free algorithms include Monte Carlo RL, Sarsa, and Q-learning.

The estimation of value function is critical for model-free RL algorithms. Unlike Monte Carlo (MC) methods, temporal difference (TD) methods learn the value function by reusing existing value estimates. If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal difference. TD has the ability to learn from an incomplete sequence of events without waiting for the final outcome. TD has the ability to approximate the future return as a function of the current state. Similar to MC, TD only uses experience to estimate the value function without knowing any prior knowledge of the environment dynamics. The advantage of TD lies in the fact that it can update the value function based on its current estimate. Therefore, TD learning algorithms can learn from incomplete episodes or continuing tasks in a step-by-step manner, while MC must be implemented in an episode-by-episode fashion *(Sutton and Barto, 1998)*.

In this research, state transition in a Markov Decision Process (MDP) is the process by which the system transits from one state to the next state with only the impact of the actions performed by the agent. In the simulator, state transitions are not expressed as mathematical functions. Instead, a data-driven simulated environment is used. This developed simulator should able to capture different real-world factors like time-varying travel demand, driver availability dynamics and spatial-temporal supply-demand patterns. A description of the settings for the developed simulator can be found in Chapter 4.

In summary, the model-free approach is applied to the ride-hailing matching optimization problem in this research. In the MDP, the transition function with properties that are inherent to real-life ride-hailing matching systems is simulated by developing a simulator due to their complex and dynamic nature. The method makes it possible to more flexibly and realistically simulate the ride-hailing matching environment while allowing for unforeseen changes and uncertain properties of the ride-haling environment in the real world. The next chapter will introduce a ride-hailing simulator built on the MDP defined in this chapter.

4

Simulator Design

In Reinforcement Learning, a simulator is a virtual environment where the agent can interact and learn without the need for costly or impractical real-world experiments. In ride-hailing optimization, the simulator is a deep copy of the ride-hailing market in the real-world. It models travel demands, driver locations, aiding algorithms in testing and refining dynamic matching radius. This chapter describes the settings of the simulator and the details base on the proposed MDP. The pseudo code of the simulator's working flow is included in Appendix A.

4.1 Simulator Description

In this section, a Markov Decision Process (MDP)-based simulator is defined for modelling a realistic ride-hailing market and providing an interactive environment for the Reinforcement Learning agent to generate data for its learning. On the other hand, this simulator is also used to evaluate and optimize the dynamic matching radius policy.

The MDP-based simulator was chosen because it is able to efficiently simulate the process of assigning matching radius to multiple cell-based area by the system manager, as well as the complex interactions between riders and drivers. In the same time, the simulator makes it possible for the agent to capture the dynamic behaviours of the system at a micro level. The reason for defining a simulator is to make the Reinforcement Learning environment more controllable and easier to obtain the data needed for learning, as well as to make it possible to test and validate the trained policy in a controllable environment. The simulator also makes it possible to be able to simulate the complex conditions of the real world and to perform experiments in multiple scenarios without interfering with the actual operation of the ride-hailing system in real world.

In order to simplify the system and effectively control irrelevant variables, this study made several assumptions in the developed ride-hailing simulator:

- **Cell-based map:** It is assumed that the ride-hailing market consists of multiple independent, equally-sized cell areas, each with fixed geographic boundaries. The matching process between travel demands and drivers in each cell is independent. The matches can be made across the boundaries of the cells. This assumption simplifies the matching process and reduces its systematic complexity.
- **Cell-based matching radius:** It is assumed that all riders in the same cell have the same matching radius, while the matching radius between different cells can be different.
- **Simplified ride-hailing behaviours:** It is assumed that riders will not cancel orders after being successfully matched, and it is also assumed that drivers will not refuse to accept orders after being matched with travel demands. This assumption simplifies the complexity of the matching system and makes the optimization goal of the system clearer.
- **Distance-based preference:** It is assumed that the order weight of the matching system is only related to the pick-up distance, regardless of the length of the order trip. This assumption means that all orders have the same value, and the weight is only negatively correlated with the pick-up distance.
- **Controlling variables:** It is assumed that all other environmental factors (such as traffic conditions and weather) remain unchanged during the simulation. This assumption aims to control these complex and irrelevant variables, hence reducing the impact of irrelevant factors on optimizing the ride-hailing matching system.

With the discussed definitions and assumptions, a highly realistic ride-hailing simulator is proposed in this study. The simulator consists of five parts: data, action, state-action interaction, reward, and state transition. All those five parts are discussed in detail in the following subsections.

4.2 Data Usage of the Simulator

The data of this ride-hailing simulator is the most critical data in the simulator and the model-free approach. It is the basis of all changes and internal variables in the simulator's state transition. The quality of the data directly affects the reliability of the simulator in estimating the real-world ride-hailing environment, and also significantly affects the accuracy of the simulator in simulating the state transition of the real ride-hailing environment. Therefore, the data must come from real and feasible ride-hailing operation records in the real world, and appropriate methods must be selected to preprocess it to ensure its quality. When selecting data, the reliability and representativeness of the data source should be considered to ensure that it include information that can accurately reflect the actual operation of ride-hailing service. The preprocessing process is also important, it includes steps such as data cleaning, denoising, and normalization to eliminate outliers and noise and ensure data consistency and accuracy.

Part of the simulator's data is a comprehensive ride-hailing dataset from the study area. This dataset should include detailed locations of idle drivers in the study area, the starting location of ride requests, the ending location of ride requests, the start and end time of ride requests, and other relevant details. This extensive dataset provides a real-world data source that is critical to the accuracy and reliability of the developed ride-hailing simulator. In this ride-hailing simulator, the initial location of drivers and travel requests are sampled from distributions fitted based on this part of the data. This approach ensures that the simulation environment is highly realistic compared to the real-world operations of the ride-hailing system observed in the dataset. Another part of the simulator's base data comes from OpenStreetMap (OSM). OpenStreetMap (OSM) is a free, open geographic database updated and maintained by a community of volunteers via open collaboration. Contributors collect data from surveys, trace from aerial imagery and also import from other freely licensed geodata sources. *(OpenStreetMap official wiki)*. The simulator uses OSM data as the road network data and map data for modelling studied area. The use of this data is crucial for generating cells in the simulator, enabling environment visualization, significantly enhancing the realism of the simulator, and making it easier for simulator users to observe the current state and the transitions in the simulator.

In the ride-hailing simulator developed in this study, the locations of all new travel demands and initial locations of drivers are generated based on location models fitted from the real ride-hailing operating dataset. General fitting methods, such as parametric models, may not capture the complex, multimodal nature of travel demand distributions in urban environments. For instance, fitting a simple two-dimensional gaussian distribution might fail to account for the multiple peaks corresponding to different hotspots in a city, such as transportation hubs, residential areas, and the city center. This study employs an advanced non-parametric fitting method to capture the locational characteristics of travel demands and drivers, which is the Kernel Density Estimation.

Kernel Density Estimation (KDE) is a non-parametric method used to estimate the probability density function for multi-dimensional variables. In statistics, adaptive or variable-bandwidth kernel density estimation is a form of kernel density estimation in which the size of the kernels used in the estimate are varied depending upon either the location of the samples or the location of the test point. It is a particularly effective technique when the sample space is multi-dimensional. Using a fixed filter width may mean that in regions of low density, all samples will fall in the tails of the filter with very low weighting, while regions of high density will find an excessive number of samples in the central region with weighting close to unity. To fix this problem, *Terrell G. R., Scott D. W. (1992)* vary the width of the kernel in different regions of the sample space. There are two methods of doing this: balloon and pointwise estimation. In a balloon estimator, the kernel width is varied depending on the location of the test point. In a pointwise estimator, the kernel width is varied depending on the location of the sample. For example, in a ride-hailing demand distribution, there are often multiple peaks corresponding to different demand hotspots. Using KDE can better reflect these multimodal demand characteristics rather than relying on simple parametric models. Mathematically, a kernel is a positive function $K(y;h)$ controlled by the bandwidth parameter h. Given this kernel form, the density estimates at a point y within a group of points $x_i; i = 1 \dots N$ are provided by

$$\rho_K(y,h) = \frac{1}{Nh}\sum_{i=1}^{N} K\left(\frac{y-x_i}{h}\right) \tag{4.1}$$

The kernel function used in this study is the gaussian kernel. The gaussian kernel is defined as

$$K(y,h) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y-x_i}{h}\right)^2} \tag{4.2}$$

The bandwidth acts as a smoothing parameter, controlling the trade-off between bias and variance in the result. A large bandwidth leads to a very smooth (i.e., high-bias) density distribution, while a small bandwidth leads to an unsmooth (i.e., high-variance) density distribution.

4.3 Input of the Simulator

The action is the input to the simulator, its value is decided by the system operator (agent), which is the matching radius of each cell. The action is a collection of sub-actions of all cells at time step t, it can be written as $a_t = (a_{1,t}, a_{2,t}, a_{3,t} \dots a_{i,t} | i \in A)$, where $a_{i,t}$ is the sub-action in cell i at time-step t. The action has an action space with a lower bound of 50 meters and an upper bound of 3000 meters. All the riders in the same cell will have the same matching radius, while riders in different cells can have different matching radius. A matching pool is a collection of all the riders with available drivers within their matching radius. Available drivers are indicated by drawing a rider-centered matching range with the corresponding matching radius as illustrated in Figure 4.1. The matching radius of a matching area determines how many drivers will be included in the matching pool. The larger the matching radius, the bigger the matching pool is. The matching radius can be different across cells.

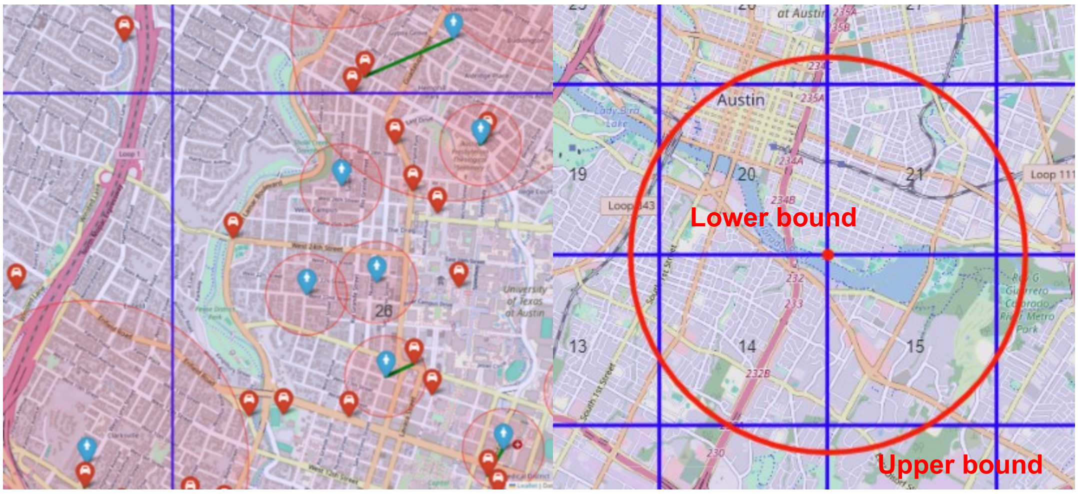


Figure 4.1. An example of the action and the visualized action range.

4.4 Simulation of the Matching Process

The matching process of the ride-hailing simulator is based on the real-time observation and the input matching radius at each time-step t. In the ride-hailing market, the observation the agent can observe from the environment is the real-time supply-demand relationship between drivers and travel requests. In the ride-hailing simulator, as defined in the MDP, the state is defined by the real-time travel demand and the locations of idle vehicles in each region. However, using the specific coordinates of vehicles and travel demand as the state is too cumbersome and not conducive to large-scale computations. Therefore, the real-time supply-demand relationship of traveling demands and idling vehicles in each region is concretized as the number of traveling demands and idling drivers in each cell, to reflect the current system state. Let $N_{i,t}^{r}$ denotes the number of riders (travel demands), and $N_{i,t}^{d}$ denotes the number of idle drivers in cell i at time t. Then, the sub-observation in cell i at time t can be derived as

$$o_{i,t} = \left\{N_{i,t}^{r},\ N_{i,t}^{d} \;\middle|\; i \in A\right\} \tag{4.3}$$

With the definition 4.3 of sub-observations, the state at time-step t can be extended as

$$s_t = \left\{\left\{N_{1,t}^{r},\ N_{1,t}^{d}\right\}, \left\{N_{2,t}^{r},\ N_{2,t}^{d}\right\}, \dots \left\{N_{i,t}^{r},\ N_{i,t}^{d}\right\} \;\middle|\; i \in A\right\} \tag{4.4}$$

With current state s_t and a_t, the state is updated when time-step moves from time interval t to time interval $t + 1$. The state is updated according to the interactions between system operator's action and the environment. After system operator takes the action, all the riders will have a matching radius based on which cell they are in. With the given matching radius, they will each form a matching pool. All the drivers within the matching radius will be included in the corresponding matching pool. To be specific, the matching pool is rider centered, so one matching pool can only have one rider, while one driver could be included in several matching pool, and only one ride can be formed within one matching pool. When all the possible matching pools are formed, the matching process begins. This matching process is a global match, which means riders and idle drivers can be matched across cells. This type of matching problem is solved by abstracting it into a bipartite matching problem, this approach has been proved efficient and feasible in previous studies *(Xu et al., 2018)*. Figure 4.2 illustrates the bipartite graph of this matching problem. In the graph, $o_1, \dots\ o_i$ are riders (travel demands) and $d_1,\ d_2,\ \dots\ d_i$ are available drivers. The solution with the maximum matching rate and the minimum summed edge weights of this bipartite matching problem can be found by using Hungarian Matching Algorithm with weighted edges of pick-up distance based on equation 4.5 with constraints 4.6, 4.7 and 4.8.

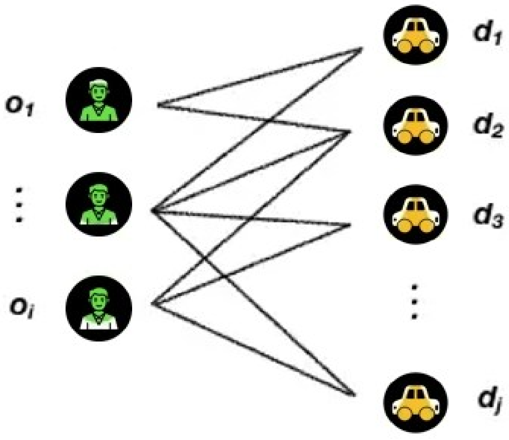


Figure 4.2. A bipartite ride-hailing matching graph.

The Hungarian method is a combinatorial optimization algorithm that solves the assignment problem in polynomial time and which anticipated later primal–dual methods. It was developed and published by Harold Kuhn (1955), who gave it the name Hungarian method because the algorithm was largely based on the earlier works of two Hungarian mathematicians. James Munkres (1957) reviewed the algorithm and observed that it is (strongly) polynomial. Since then, the algorithm has been known also as the Kuhn–Munkres algorithm or Munkres assignment algorithm. In this algorithm, the drivers and riders will be matched according to the basis to achieve the lowest average pick-up distance, this is the principle of batched matching. Let c_{ij} denotes the weight (the pick-up distance) of the match between rider i and driver j, x_{ij} is binary variable, $x_{ij} = 1$ if rider i and driver j are matched, otherwise $x_{ij} = 0$. Let R denotes the rider collection, D denotes the driver collection. Then the objective function is then derived as

$$minimize \sum_{i \in R} \sum_{j \in D} c_{ij} x_{ij} \tag{4.5}$$

This objective function represents the principle of the batched matching method and the bipartite matching method with weighted edges. The goal is to reach the minimum summed weights of edges between all matched pairs, in order to minimize the average pick-up distance in the ride-hailing system. This objective function also has some constraints. The first constraint is that one rider can only be picked-up by one driver. In contrast, the second constraint is that one driver can only pick up one rider at a time step. These constraints are formulated as

$$subject\ to \sum_{j \in D} x_{ij} \leq 1 \ \forall i \in R \tag{4.6}$$

$$\sum_{i \in R} x_{ij} \leq 1 \ \forall j \in D \tag{4.7}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i \in R, \forall j \in D \tag{4.8}$$

4.5 Outputs of the Simulator

The reward and the next state observation are the outputs of the ride-hailing simulator. As defined in the MDP, the reward signal indicates what is good in an immediate sense. In this ride-hailing simulator, the reward signal is combined from two parts: the penalty signal and the performance metrics. This subsection will discuss in detail the reward function of the defined MDP adapted to this simulator, including how it is calculated and how it reflects the performance of the ride-hailing system.

The most important part is the ride-hailing performance metrics, it reflects the optimization objective of this research. In the ride-hailing simulator, a good action is determined when it can lead to a better trade-off between matching rate, average pick-up distance and driver utilization. The matching rate defines the percentage of ride requests that are successfully matched with a driver within the optimal radius. Let N_m denotes the number of matched rides and N_r denotes the number of traveling demands. Then, the matching rate r_t^{mr} at time-step t can be derived as

$$r_t^{mr} = \frac{N_m}{N_r} \tag{4.9}$$

The average pickup distance is the average distance a driver travels to a rider's location, affecting operating costs and user experience. Let c_{ij} denotes the pick-up distance between between rider i and driver j, x_{ij} is a binary variable. Let $x_{ij} = 1$ if rider i and driver j are matched, otherwise $x_{ij} = 0$. Let N_m denotes the number of matched rides. Let a_{max} denotes the upper bound of the action space, which is the maximum pick-up distance, this is used to normalize the average pick-up distance into a scale between [0, 1]. Then the average pick-up distance r_t^{pd} at time-step t is derived as

$$r_t^{pd} = \frac{\left(\frac{\sum_{i\in R}\sum_{j\in D} c_{ij}x_{ij}}{N_m}\right)}{a_{max}} \tag{4.10}$$

The driver utilization rate reflects the efficiency with which drivers respond effectively to ride requests, affecting overall system efficiency and driver satisfaction. Let N_m denotes the number of matched rides, $N_d{}'$ is the number of available drivers within matching radius. The driver utilization rate r_t^{du} at time-step t can be derived as

$$r_t^{du} = \frac{N_m}{N_d{}'} \tag{4.11}$$

The performance metrics at time-step t is represented by a score summed form three weighted reward components. Let w_1, w_2, w_3 denotes the multi-task optimization weights for each of the reward components, the performance score $C_t(s_t, a_t)$ at time-step t with a state s_t and the input matching radius set a_t can be derived as

$$C_t(s_t, a_t) = \mathrm{w}_1\, r_t^{mr} + \mathrm{w}_2\, r_t^{pd} + \mathrm{w}_3\, r_t^{du} \tag{4.12}$$

$$= \mathrm{w}_1\, \frac{N_m}{N_r} + \frac{\mathrm{w}_2 \left(\frac{\sum_{i\in R}\sum_{j\in D} c_{ij}x_{ij}}{N_m}\right)}{a_{max}} + \mathrm{w}_3\, \frac{N_m}{N_d{}'} \tag{4.13}$$

The importance of match rate, average pick-up distance, and driver utilization rate is clear to researchers, but these optimization goals can be difficult for a machine (the agent) to understand. The original reward function straightly includes those performance metrics are tested in this research, the training performances such as cumulative reward convergence and learning speed are not always satisfying. Therefore, reward shaping is employed in this study to help the agent learn more effectively. Reward shaping in reinforcement learning is a technique used to guide an agent toward desirable behaviors by modifying the reward function *(Laud A. D., 2004)*. By carefully designing the shaping function for the rewards, the agent can learn more effectively and efficiently, especially in complex environments such as the ride-hailing matching environment. There are many types of reward shaping approaches. Potential-based reward shaping, for example, is a particular type of reward shaping, it uses the expected value of the immediate reward that the agent will ultimately receive for taking a particular action to reshape the reward function *(Wiewiora, 2003)*.

In this study, a reward shaping method inspired by the potential-based approach is employed. The reward signal is normalized to a binary value to help the agent better understand which matching radius is considered optimal for the system's performance and user experience. A threshold is set to determine the binary value: if the overall system performance score exceeds the threshold, the reward is normalized to 1; otherwise, it is normalized to 0. This threshold is determined through a series of tests called random exploration conducted before the training process. During random exploration, the matching radius is randomly input into the simulator, and all output performance scores are recorded. The threshold is then set at a value higher than 95 percent of all the output rewards, which is 0.6 in this case study.

Let $R_C(s_t, a_t)$ denotes the reshaped reward, it is a binary to demonstrate the quality of the action a_t under the given state s_t. This approach helps the agent to better understand whether the action is good or not. The $R_C(s_t, a_t)$ is explained as

$$R_C(s_t, a_t) = \begin{cases} 1, & C_t(s_t, a_t) \geq 0.6 \\ 0, & C_t(s_t, a_t) < 0.6 \end{cases} \tag{4.14}$$

In reward shaping, it is also important for the reshaped reward function to guide the agent away from actions that are likely to lead to poor rewards *(Miller T., 2022)*. In this study, a penalty is applied to the matching radius value to eliminate poor or deceptive actions that the agent might take during the learning process. While these tricky actions might yield higher rewards in certain steps, they are not beneficial for the overall long-term discounted return, as formulated in Equation 3.4. The objective of this research is to optimize the matching radius. The agent should manage to control the matching rate, pick-up distance and driver utilization rate within a reasonable level. This means that both very large and very small matching radius should be penalized. To achieve this goal, a sigmoid-shaped penalty is added. Let a_{min} denotes the lower bound of the action space (the minimum matching radius), $a_{i,t}$ denotes the matching radius of cell i at time-step t, A is the collection of all the cells. The matching radius penalty $P_t(a_t)$ with action a_t at time-step t can be derived as

$$P_t(a_t) = \left(\sum\left[\left(1 + e^{-0.05(a_{i,t} - a_{min})}\right)\left(1 + e^{0.05(a_{i,t} - a_{max})}\right) - 1\right] \middle| i \in A\right) \tag{4.15}$$

The plot of this penalty function is illustrated in Figure 4.3.

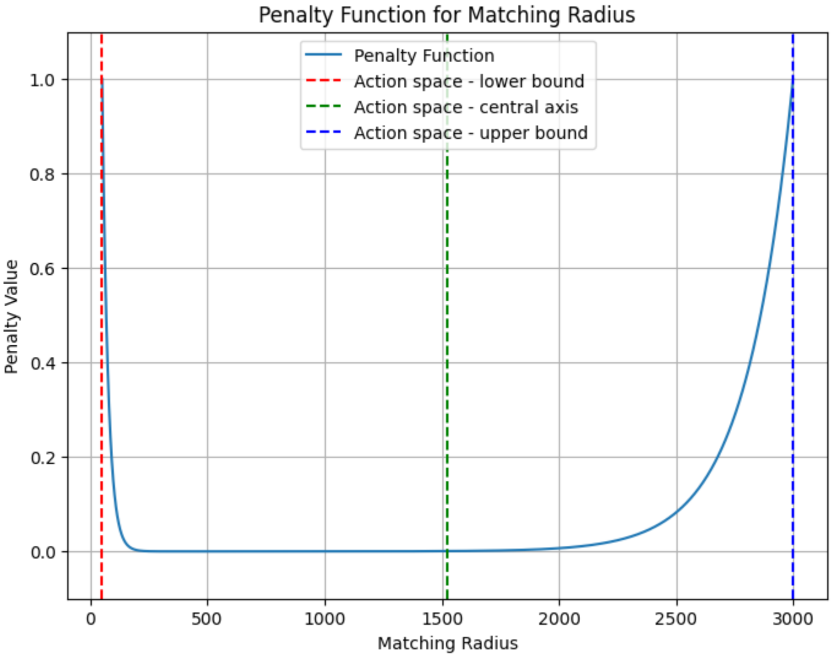


Figure 4.3. Plot of the penalty function for the matching radius.

Let $R_t(s_t, a_t)$ denotes the instant reward of action a_t in state s_t, $R_C(s_t, a_t)$ is the normalized overall performance binary, $P_t(a_t)$ is the penalty given to the action. The overall reward function at time-step t can be derived as

$$R_t(s_t, a_t) = R_C(s_t, a_t) - P_t(a_t)$$

$$= R_C(s_t, a_t) - \left(\sum\left[\left(1 + e^{-0.05(a_{i,t} - a_{min})}\right)\left(1 + e^{0.05(a_{i,t} - a_{max})}\right) - 1\right] \middle| i \in A\right) \tag{4.16}$$

For the Reinforcement Learning agent, the objective is to find a policy that maximizes the long-term cumulative reward, known as the discounted return. This discounted return can be viewed as the sum of the function in Equation 4.16 over all time steps, with each term multiplied by the discount factor at its respective time step, derived as

$$max\ E_{\pi}\left[\sum_{t=1}^{T}\gamma^{t-1}R_t(s_t,a_t)\,|s_t=s,a_t=a\right] \tag{4.17}$$

4.6 State Transition

Those unmatched idle drivers and demands are marked as match failed. When matching process is done, successfully matched idle drivers and demands are marked as successful rides and removed from the map. Unmatched demands will be checked if they exceed the threshold Δt_{max} (tolerance matching time defined as maximum matching time-steps), if so, they will be moved out from the matching pool, if not, they will be kept in the matching pool and join the matching process at the next time-step $t+1$. Those unmatched idle drivers have two possible behaviours of staying in the same cell (idling within 300 meters) or travel to the nearest adjacent cell (relocating, only if already idled for more than 5 minutes), they are included in matching pools of the corresponding cells at the next time-step $t+1$. New demand and idle drivers enter the matching pool at time-step $t+1$ are determined by generating rate $\lambda_r(i,t)$ and $\lambda_d(i,t)$. Let $n_{i,t}^{d,out}$ denotes the number of drivers that are no longer in cell i after time-step t (matched or relocated), $n_{i,t}^{d,in}$ denotes the number of drivers come to cell i from other cells. Similarly, let $n_{i,t}^{r,out}$ denotes the number of riders who left the matching pool in cell i after time-step t (matched or quit the queue). The observations are updated as:

$$N_{i,t+1}^{d} \longleftarrow N_{i,t}^{d} - n_{i,t}^{d,out} + n_{i,t}^{d,in} + \lambda_d(i,t)\Delta t \tag{4.18}$$

$$N_{i,t+1}^{r} \longleftarrow N_{i,t}^{r}, - n_{i,t}^{r,out} + \lambda_r(i,t)\Delta t \tag{4.19}$$

Figure 4.4 and Figure 4.5 shows the difference between the driver idling behaviours and driver relocating behaviours in the state transition process of the ride-hailing simulator.

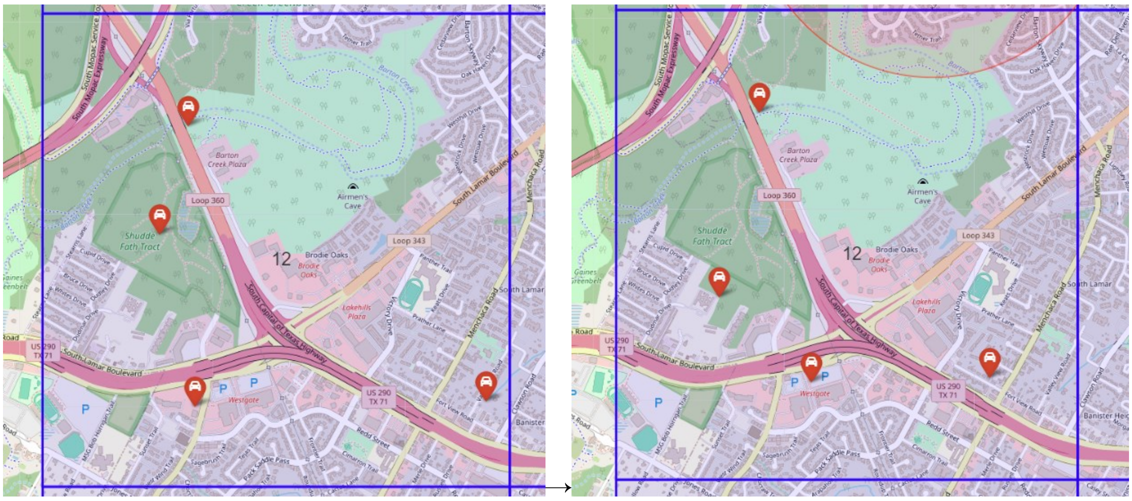


Figure 4.4. An example of idling behaviours of drivers.

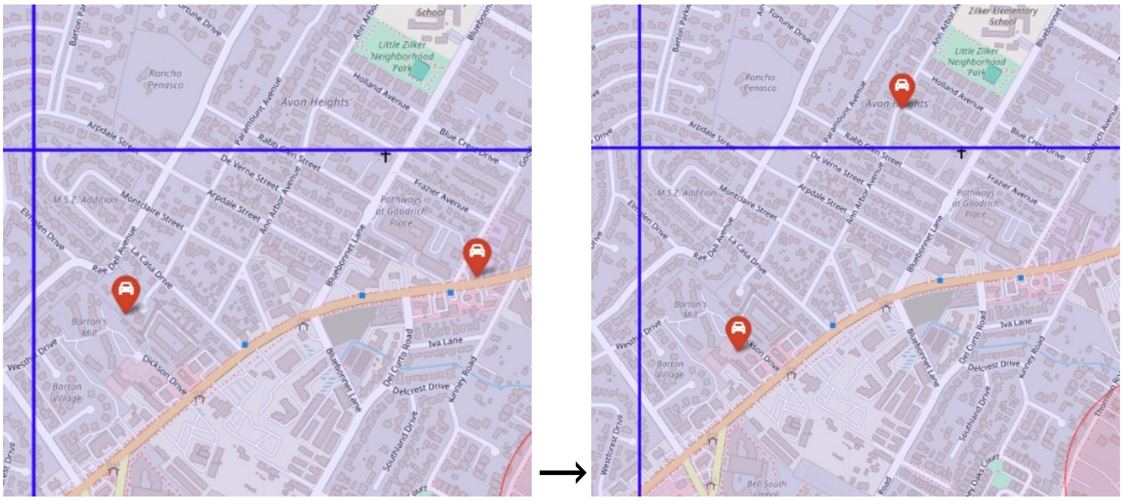


Figure 4.5. An example of relocating behaviours of drivers.

The pseudo codes of this state transition can be found in Appendix A. Although this simulator simplifies driver behaviour for the sake of computational efficiency and optimization effectiveness, it still manages to approximate driver behaviour in the ride-hailing market to a certain extent. The limitations arising from these simplifications and assumptions will be discussed in the final section.

5

MDDPG Algorithm

After building the simulator, it is needed to define a Reinforcement Learning agent to interact with the environment. With the inspiration of human learning behaviour, several Reinforcement Learning algorithms are developed. In this study, a customized DDPG algorithm for the ride-hailing environment is introduced, named MDDPG (Multi-replay-buffer Deep Deterministic Policy Gradient).

5.1 Algorithm Introduction

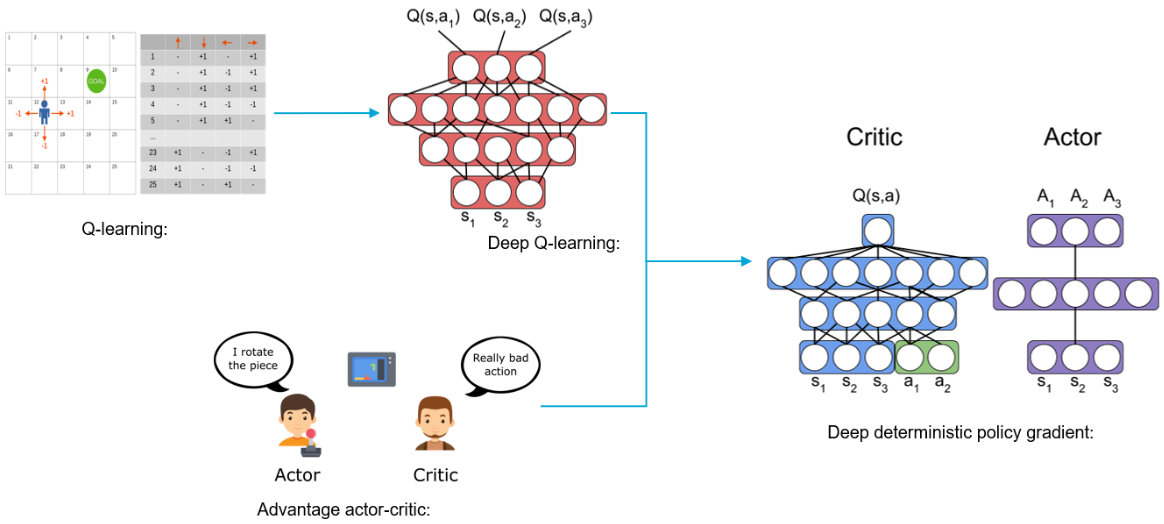
Reinforcement Learning algorithms can be classified into value-based and policy-based methods according to the optimization objects. Value-based RL calculates the estimated action-value or state-action value for each action with the current a state, selecting the highest value action as the next action to take. Q-learning is the simplest one of value-based Reinforcement Learning approaches. It uses a Q-value table to determine the values for all state-action pairs, offering good learning efficiency and convergence. However, it is only suitable for the problems with finite state and observation space and discrete, finite action space, making it unsuitable for the ride-hailing environment.

A more advanced value-based method is Deep Q-Network (DQN) learning. Reinforcement learning is unstable or divergent when a nonlinear function approximator such as a neural network is used to represent Q. This instability comes from the correlations present in the sequence of observations, the fact that small updates to Q may significantly change the policy of the agent and the data distribution, and the correlations between Q and the target values. The method can be used for stochastic search in various domains and applications. The technique used experience replay, a biologically inspired mechanism that uses a random sample of prior actions instead of the most recent action to proceed. This removes correlations in the observation sequence and smooths changes in the data distribution. Iterative updates adjust Q towards target values that are only periodically updated, further reducing correlations with the target *(Li et al. 2023)*. These techniques are critical to the successful of DQN in playing Atari games and many other applications, due to the deadly triad issue *(Sutton and Barto, 2018)* of reinforcement learning when one tries to combine bootstrapping (i.e., TD-learning and Q-learning), off-policy training (i.e., Q-learning), and function approximations (i.e., neural networks), which may lead to instability and divergence. The algorithms introduced so far are all value-based methods, which focus on learning the value function, and the policy is derived from the learned value function. Neural network-based value function approximation is important to ride-hailing applications because the state is often high dimensional. Tabular methods suffer from the curse of dimensionality and are not tractable in this case.

Model-free reinforcement learning algorithms can start from a blank policy candidate and achieve superhuman performance in many complex tasks, including Atari games, StarCraft and Chinese Go. Deep neural networks are responsible for recent artificial intelligence breakthroughs, and they can be combined with reinforcement learning to create something astounding, such as DeepMind’s AlphaGo. Mainstream model-free RL algorithms include Deep Q-Network (DQN), Dueling DQN, Double DQN (DDQN), Trust Region Policy Optimization (TRPO), Proximal Policy Optimization (PPO), Asynchronous Advantage Actor-Critic (A3C), Deep Deterministic Policy Gradient (DDPG), Twin Delayed DDPG (TD3), Soft Actor-Critic (SAC), Distributional Soft Actor-Critic *(Li et al. 2023)*.

MDDPG (Multi-replay-buffer Deep Deterministic Policy Gradient) is the algorithm that is applied in this research based on the DDPG framework. DDPG (Deep Deterministic Policy Gradient) is a model-free off-policy reinforcement learning algorithm for learning continuous actions. It combines ideas from DPG (Deterministic Policy Gradient) and DQN (Deep Q-Network). The development of deep deterministic policy gradient (DDPG) was inspired by the success of DQN and is aimed to improve performance for tasks that requires a continuous action space. DDPG incorporates an actor-critic approach based on DPG. The algorithm uses two neural networks, one for the actor and one for the critic. Deep

Deterministic Policy Gradient (DDPG) is an algorithm which concurrently learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function and uses the Q-function to learn the policy. DDPG also uses target critic and target actor networks to stabilize training by preventing direct updates of network outputs on the network itself. Unlike DQN, DDPG uses a soft update approach for the target networks, gradually adjusting their parameters towards the source networks. The DDPG method has the following advantages, it has high sampling efficiency, it is suitable for multi-dimensional continuous action space and its policy is stable. In this study, an extended DDPG algorithm is developed using two replay buffers (MDDPG) to help the agent to better train the policy during the learning process, this algorithm is discussed in detail in the next subsection. Figure 5.1 illustrates the relationships between different Reinforcement Learning algorithms.



	↑	↓	←	→
1	-	+1	-	+1
2	-	+1	-1	+1
3	-	+1	-1	+1
4	-	+1	-1	-1
5	-	+1	+1	-
...				
23	+1	-	-1	+1
24	+1	-	-1	-1
25	+1	-	+1	-

Figure 5.1. Relationship between different RL algorithms.

The coding environment of the RL algorithm is Python 3.9.13 with packages listed in Appendix B.

5.2 Description of the MDDPG

The DDPG algorithm is suitable for continuous action spaces, working by combining the ideas of deep Q-learning, policy gradients, and the actor-critic method. This section discusses the customised DDPG algorithm proposed in this study named Multi-replay-buffer Deep Deterministic Policy Gradient (MDDPG). Two sets of deep neural networks are used in the MDDPG algorithm, which are source and target critic networks Q_ω and $Q_{\omega'}$, source and target actor networks μ_θ and $\mu_{\theta'}$, respectively.: The actor network decides the actions to take given a state; and the critic network evaluates these actions by estimating the expected reward. To stabilize the training, target networks are employed for both the actor and critic. Additionally, the MDDPG algorithm in this study uses two replay buffers: one for storing all experiences and another specifically for storing high-reward transitions. This ensures higher learning efficiency and accelerates convergence.

The workflow of the algorithm is illustrated in Figure 5.2, with the entire training process starting from the bottom right. In Reinforcement Learning, the main entity that learns and makes decisions is called the agent. The agent interacts with the environment to learn how to take actions base on the policy that maximize a long-term reward. Figure 5.2 shows the process by which the MDDPG agent interacts and learns with the developed ride-hailing simulation environment.

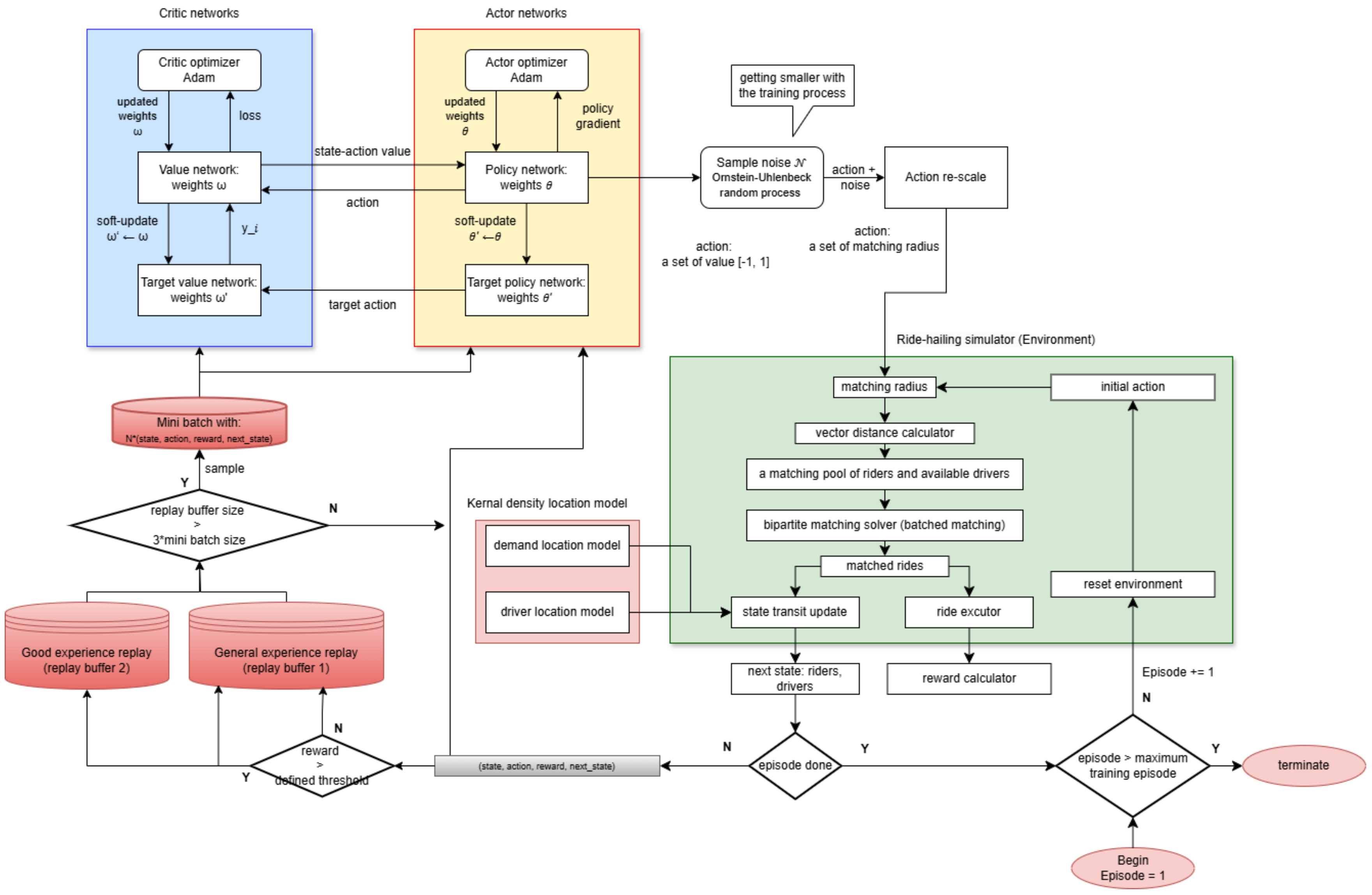

Critic networks
Critic optimizer Adam
updated weights ω
loss
Value network: weights ω
soft-update ω' ← ω
y_i
Target value network: weights ω'
state-action value
action
target action
Actor networks
Actor optimizer Adam
updated weights θ
policy gradient
Policy network: weights θ
soft-update θ' ←θ
Target policy network: weights θ'
getting smaller with the training process
Sample noise 𝒩 Ornstein-Uhlenbeck random process
action + noise
Action re-scale
action: a set of value [-1, 1]
action: a set of matching radius
Ride-hailing simulator (Environment)
matching radius
initial action
vector distance calculator
a matching pool of riders and available drivers
bipartite matching solver (batched matching)
matched rides
state transit update
ride excutor
reset environment
Kernal density location model
demand location model
driver location model
Mini batch with: N*(state, action, reward, next_state)
sample
Y
replay buffer size > 3*mini batch size
N
Good experience replay (replay buffer 2)
General experience replay (replay buffer 1)
N
reward > defined threshold
Y
(state, action, reward, next_state)
next state: riders, drivers
reward calculator
N
episode done
Y
Episode += 1
N
episode > maximum training episode
Y
terminate
Begin Episode = 1

Figure 5.2. The working flow of the MDDPG algorithm in ride-hailing systems.

The first step of the entire training process is initializing the simulation environment (simulator) for each episode. The green boxes represent methods related to the simulation environment within the algorithm. First, the simulator is reset to obtain initial state, including the number of idle drivers and ride requests in each cell. Then, the noise generation function is reset. In each step within an episode, the current state (number of idle drivers and ride requests) is passed to the actor network to obtain an action. This action, after being output by the actor, is added with a noise generated by the noise function to form the final action for the current step. Finally, this action is rescaled (min-max rescale) to the corresponding matching radius range, and the radius is passed to the simulator for interaction. In the simulator, the input matching radius for each cell is used to form a matching pool, then the rides are formed by solving the matching pool as a bipartite matching problem with weighted edges of pick-up distance. The immediate reward is then calculated with the current state-action pair based on the reward function mentioned earlier. Then, the simulator updates the environment, determines the behaviour of unmatched riders and drivers, and generates new ride requests based on the driver-rider location model, thus completing the state transition. The simulator then observes the new state, checks whether the current state meets the episode termination criteria, and normalizes the states before outputting them along with the reward and done status to the agent. This set of information, including the current state, action taken, reward, next state, and done status, is called a state transition. One such interaction process is named as a step in RL training. In Reinforcement Learning, a series of consecutive steps from resetting the environment to the next reset is called an episode. After each step of interaction between the agent and the simulator, the agent first checks whether the current episode has ended based on the done status (done status is True). If the episode ends, the entire environment is reset, and the next episode begins. If the episode has ended, the agent also checks whether the training termination criteria is met. For example, in this study, the training terminates when the number of training episodes reaches the maximum training episode threshold. If not, the above process is repeated for training.

In the MDDPG algorithm, after each step of interaction, the current step's state transition is stored in the replay buffer, as shown in the bottom left corner of Figure 5.2. If the reward in this state transition is higher than a given threshold, this transition is also stored in a separate replay buffer called the good action buffer. Continuing with the MDDPG process in the figure, after storing the state transition, the agent checks the number of state transition data in the replay buffer. Once it exceeds a given threshold (in this study, the threshold is three times the mini-batch size), it indicates that the replay buffer has warmed up, which means the replay buffer has accumulated enough transitions for the agent to learn from. The MDDPG algorithm then begins updating the weights of the neural networks to learn from the replay transitions. Next is the neural network updating part in the MDDPG algorithm. When the agent starts learning, it samples a mini-batch of transitions from both replay buffers, as shown in the bottom left corner of Figure 5.2. After sampling, these data are passed to the critic and actor networks for updating their weights. The specific update process is detailed below, with the process referenced in the upper half of Figure 5.2. The source actor network is then used with the current step's state to output actions for the next interaction, as shown in the top right of the figure. This step's training process cycles continuously until the done status is True. This cycle of exploration with noise, transition storage, and network updates continues until the maximum number of episodes is reached, gradually improving the policy to optimize the ride-hailing system's matching radius. The finally trained source actor network represents the optimal dynamic matching radius policy.

As for the method for updating the weights in the deep neural networks, the optimizer applied to the MDDPG algorithm is the Adam optimizer, it uses an incremental update method to update the weights for the neural networks. The weights of critic network are updated by minimizing the loss

$$L = \frac{1}{N}\sum_{i=1}^{N}\left(q_i^{\omega'} - Q_\omega(\, s_i,\, a_i)\right)^2 \tag{5.1}$$

where $q_i^{\omega'}$ is an expected state-action value, defined as $q_i^{\omega'} = r_i + \gamma Q_{\omega'}(s_{i+1}, \mu_{\theta'}(s_{i+1}))$. The weights of actor network are updated by a policy gradient, its function is to increase the probability of selecting actions with higher values, the policy gradient is calculated as

$$\nabla_\theta J = \sum_{i=1}^{N} \nabla_\theta \, \mu_\theta(s_i) \nabla_a Q_\omega(s_i, a)|_{a=\mu_\theta(s_i)} \tag{5.2}$$

After that, the parameters of target critic network ω and target actor netork are soft-upadted as

$$\theta' = \tau\theta + (1-\tau)\theta', \qquad \omega' = \tau\omega + (1-\tau)\omega' \tag{5.3}$$

where τ is the soft-update rate. The pseudo codes of the MDDPG algorithm are shown below.

```
Algorithm 1 Deep Deterministic Policy Gradient (MDDPG)
for $episode = 1: E$, do
    reset the ride-hailing simulator and get initial observation $N_t^D$, $N_t^R$
    reset done statue to False
    calculate noise factor $p = episode/\text{max\_epiosde}$
    for $t = 1: T$, do
        pass $N_t^D$, $N_t^R$ to actor get primal action $a_t = p * \mu_\theta(s_t) + (1-p) * \mathcal{N}$
        re-scale action $a_t$ to radius set $a_t'$ and pass radius $a_t{}'$ to simulator
        forming a matching pool base on the matching radius set $a_t'$
        execute ride matching by solving the bipartite matching problem
        state transition, determine behaviours for unmatched riders and drivers
        observe next state $N_{t+1}^D$, $N_{t+1}^R$, reward $r_t$, done statue $d_t$
        normalize states $s_t$ and $s_{t+1}$ (state contains two parts, rider and driver)
        store transition $(N_t^D, N_t^R, a_t, r_{t,} N_{t+1}^D, N_{t+1}^R, d_t)$ in replay buffer R
        If $r_t$ > threshold of good action:
            store transition $(N_t^D, N_t^R, a_t, r_{t,} N_{t+1}^D, N_{t+1}^R, d_t)$ in good action replay buffer R'
        If memory > warm up size:
            sample a random minibatch from R and R' as
                        N*$(N_t^D, N_t^R, a_t, r_{t,} N_{t+1}^D, N_{t+1}^R, d_t)|_{i=1,\,2..N}$
            update critic by minimizing target loss
            update actor by calculating the policy gradient
            soft update the target networks:
                        $\theta' = \tau\theta + (1-\tau)\theta', \ \ \omega' = \tau\omega + (1-\tau)\omega'$
        if done:
            break
            episode += 1
            reset random noise $\mathcal{N}$
    end for
end for
```

5.3 DNN Structure and Noise

The applied MDDPG algorithm uses a deep neural network framework developed. The defined neural network is a 4-layer linear fully connected neural network. The input layer and hidden layer use the exponential linear unit (ELU) activation function, and the output layer uses the tanh activation function. ELU is the most common activation function used by most of the NN networks, while the tanh activation function ensures output value is between -1 and 1. This feature can ensure the stability of neural network training, making it easier for the algorithm to be rescaled to the action space.

When optimizing the matching radius for the ride-hailing matching system, the deep neural network structure in the MDDPG is modified to better adapt to the needs and complexity of practical applications of the ride-hailing simulator. Taking the actor network as an example, the specific modifications include splitting the input layer and the first hidden layer into multiple parts, and the output layer and the last hidden layer into multiple parts accordingly, each corresponding to a cell. In this structure, the state of each cell (number of travel demands and drivers) is used as input and first processed by its own first hidden layer. Then, these preliminarily processed inputs are aggregated to the middle fully connected hidden layer. In the fully connected layer, the hidden layer outputs of all cells are processed comprehensively to capture the mutual relationship and overall pattern between different cells. Finally, the processed information is distributed to the independent last hidden layer and output layer of each cell for output, and the matching radius of each cell is obtained. Figure 5.3 shows such a deep neural network structure, in which the middle fully connected hidden layer is simplified. The figure shows only one fully connected layer, and the number of fully connected layers used in actual applications is 4, which constitutes a deep neural network structure.

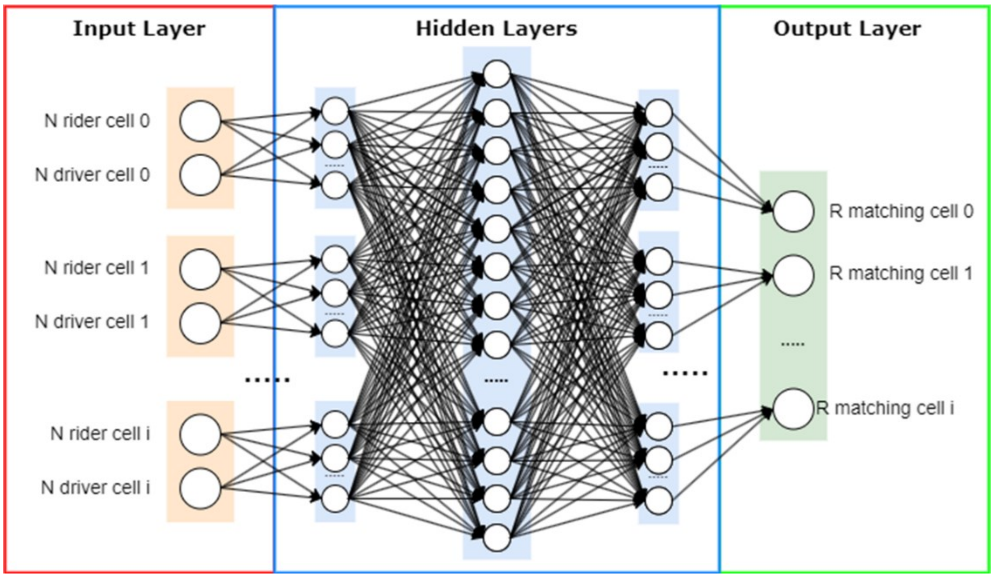


Figure 5.3. Customized structure of the actor network.

By dividing the network's input layer and the first hidden layer into multiple parts, each corresponding to a cell, the state of each cell can be processed separately when input, so that the neural network can better capture the unique supply-demand relationship and characteristics of each cell. The fully connected hidden layer ensures that the relationship between cells can be fully understood and integrated by the neural network, capturing the complex relationship between cells. In this way, the deep neural network can not only process the state and supply-demand relationship within a single cell, but also understand and utilize the interdependence between different cells, thereby optimizing the performance of the overall ride-hailing matching system. This method also effectively reduces the redundant weight updates in the neural network and improves the training efficiency. Since each cell has an independent input and the first hidden layer, this structure avoids redundant calculations of the entire network when processing high-dimensional state and action data. Each cell has its own input and output layers, so that the network can flexibly adapt to changes in supply-demand relationships in different cells. Through such a neural network design, the MDDPG algorithm used in this study can make full use of the unique information of each cell while maintaining efficient calculation, thereby improving the optimization efficiency and optimization effect of the matching radius of the ride-hailing matching system.

The MDDPG algorithm includes a random noise, which is added to the action output by the policy network. This noise is used by the MDDPG agent to explore the action space. Common stochastic processes for generating random noise include Gaussian noise and the Ornstein-Uhlenbeck (OU) process. In this study, we use the Ornstein-Uhlenbeck process. Compared to the commonly used Gaussian noise, the OU process describes a mean-reverting stochastic process. The definition of the OU process is as follows:

$$dX_t = \theta(\mu - X_t)dt + \sigma dW_t \tag{5.4}$$

where X_t is the value of the process at time t, θ is the rate of mean reversion, indicating how quickly the process back to the mean value, μ is the long-term mean, around which the process fluctuates. σdW_t is the random part, where σ is the diffusion coefficient, representing scale of the randomness, dW_t is the increment of a standard Brownian motion (Wiener process). With the definition, it is clear to find that the OU random process is a time-correlated random process, which means the random state has correlation with the previous state.

By using this OU noise, it allows the RL agent to explore more steps in one direction in the ride-hailing matching simulator. This feature can help the agent find more optimal actions and make the training process more stable. Especially because the action in this study is the matching radius. The matching radius has a continuous effect on the time sequence and affects the supply and demand relationship among several time steps. Therefore, applying OU noise to this RL learning process can better help the agent find the optimal matching radius policy, speed up the learning speed of the agent, and enhance the stability of the system. Figure 5.4 shows the different pattern between OU noise and simple Gaussian noise.

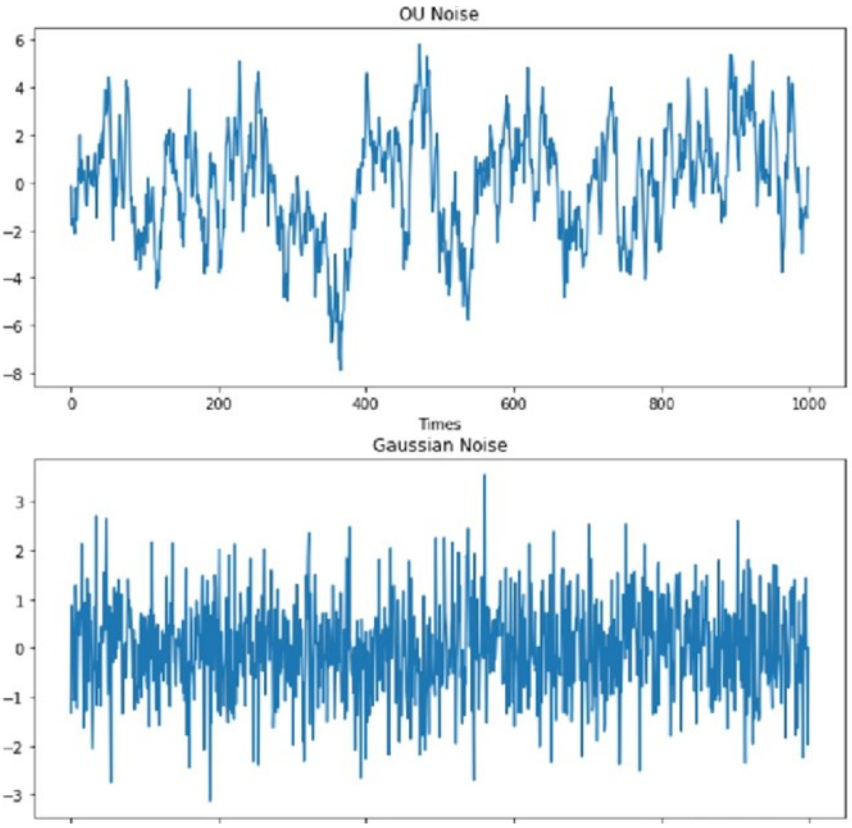


Figure 5.4. Comparison between OU noise and Gaussian noise.

5.4 MDDPG Algorithm Validation

After completing the MDDPG algorithm code, it is necessary to test its effectiveness in a simple environment. The testing environment used in this study is MountainCar-V1, which was developed by the OpenAI team. This environment has been validated under various Reinforcement Learning algorithms as a benchmark testing environment. The goal of the MountainCar-V1 environment is to drive the car to the top of the hill. The observation space and action space are as follows: This is considered as a suitable validation because it has similar setting of the ride-hailing environment but with lower dimension of action and observation space. Figure 5.5 visualizes the basic settings of the MDDPG validation environment MountainCar-V1.

Table 5.1. Validation feature comparison.

Features	MountainCar-V1	Ride-hailing Simulator
Observation Dimension	3	2 * cell numbers
Observation Type	continuous	discrete
Action Dimension	1	1 * cell numbers
Action Type	continuous	continuous
Initialization	coordinates	locations
Termination	reach the flag	reach the time cap

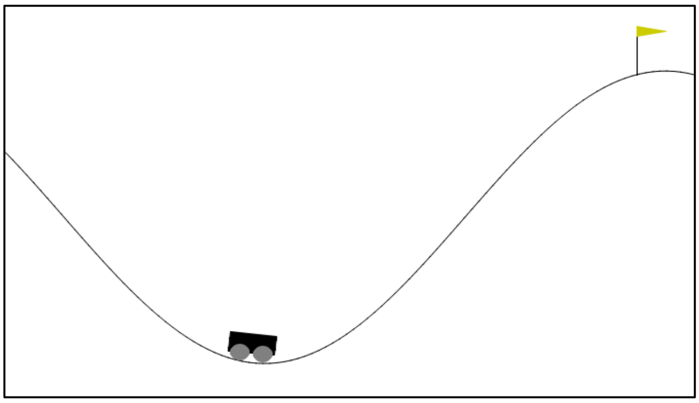
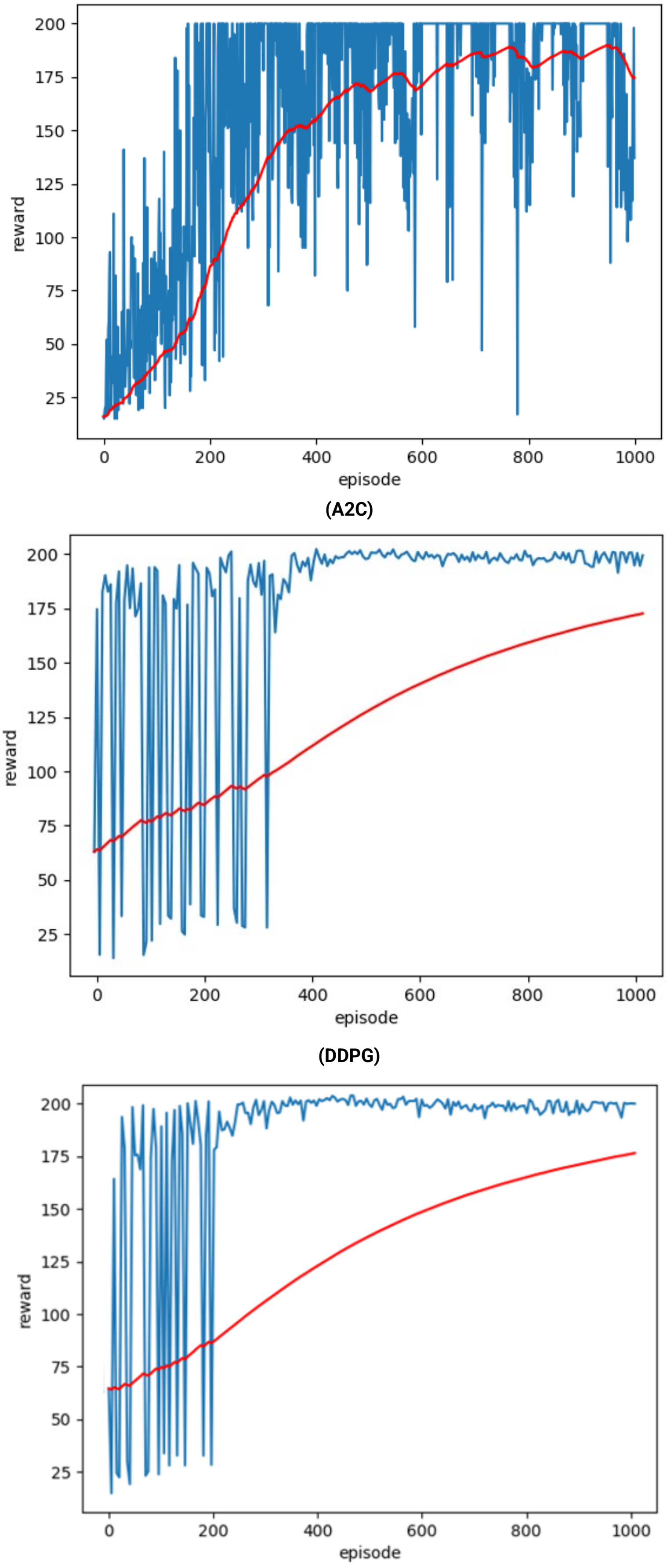


Figure 5.5. MDDPG test environment MountainCar-V1.

Speed of convergence to optimality is used to test the performance between algorithms in this research. Optimality is usually an asymptotic result, and so convergence speed is an ill-defined measure. More practical is the speed of convergence to near-optimality. This measure begs the definition of how near to optimality is sufficient. A related measure is level of performance after a given time, which similarly requires that someone define the given time *(Kaelbling et al., 1996)*. The test results are shown in Figure 5.6. Compared to the A2C algorithm and the original DDPG algorithm, it can be observed that MDDPG is more stable during training and the learning performance is also better. After 200 episodes of training, the reward stabilizes at a higher level, which is faster than the original DDPG algorithm. The results show that the policy trained by MDDPG converges faster than A2C and original DDPG, demonstrating higher training efficiency. The MDDPG algorithm is validated by this experiment because it is based on an already proposed Reinforcement Learning framework and the validation environment has the similar but simpler setting than the ride-haling environment. It can be seen from the result that the trained outcome reaches the same level compared to two validated algorithms (A2C and DDPG) and the trained policy is more stable and also reasonable (straightly push the car to the flag). The training process is repeatable and shows similar characteristics also is evidence showing that the proposed MDDPG algorithm is valid. In summary, the effectiveness of MDDPG in continuous action space environments is validated base on the validation environment.



(A2C)

(DDPG)

(MDDPG)

Figure 5.6. Performance plot between A2C, DDPG and MDDPG algorithms.

6

Case Study

6.1 Introduction of Austin

After developing the ride-hailing simulator and validating the effectiveness of the MDDPG algorithm, it is essential to apply the simulator to a real case and systematically design various scenarios to train the MDDPG agent in order to get the optimal dynamic matching radius policy. The designed scenarios can also be used to assess the effectiveness of the trained policy. This section introduces a case study setup and comprehensively evaluates the impact of various factors on performance metrics of matching rate, average pick-up distance, and driver utilization under various of scenarios with the proposed dynamic matching radius policy. Key independent variables include variations in matching radius adjustment methods (static baseline or dynamic radius), fluctuations in rider demand, and changes in driver availability (reflected in rider-driver demand-supply ratio), while control variables (matching time window etc.) remain constant within each scenario. The decision variable is straightly the matching radius for each cell obtained from the trained dynamic matching radius policy. This approach makes it possible for the researchers to train, test the proposed dynamic matching radius policy and validate the applicability of the dynamic matching radius policy for its application in the real ride-hailing market with a higher level of complexity.

The study area for this case study is Austin, the capital of Texas, USA. The city was chosen as the study area for this case study is because its densely populated downtown area and the well-developed ride-hailing service managed by the municipal authority (the RideAustin Company). The city's rapid population growth, coupled with a fast-growing economy and numerous recreational activities that attract large crowds, has led to a booming developed ride-hailing market. Currently, many online ride-hailing platforms consider Austin as an important market, including Uber, Lyft, and RideAustin. However, this growth has also brought challenges such as traffic congestion, supply-demand imbalance, and inefficient allocation of drivers and travel demands. By applying the developed simulator to the ride-hailing environment in Austin and combining it with the proposed customized MDDPG Deep Reinforcement Learning algorithm, this case study aims to optimize a dynamic matching radius policy suitable for the Austin’s ride-hailing market. The goal of this case study is to verify the effectiveness of the developed simulator and MDDPG RL algorithm. Their feasibility in the actual ride-hailing market is also verified by applying them to the case study subject.

6.2 Simulator Implementation and Validation

In this case study, the first part of the data for the simulator comes from the OpenStreetMap (OSM) project. OSM is a free and open-source global map project created and maintained by a volunteer community *(OSM official website)*. By collecting, filtering and documenting the geographic data of the city of Austin, OSM provides the simulator with a detailed and easily accessible map dataset to generate the basic simulation environment of the study area. The simulator uses OSM data to get useful information about Austin's road network data and map data. The OSM data makes it possible for the built-in algorithms of the developed simulator to generate geographical information for the ride-hailing cells. The locations of travel demand in the simulator are also partly based on this OSM data. On the other hand, the visualization part of the simulator relies heavily on the OSM data. It significantly enhances the realism of environment visualization, helps the users of this simulator to observe the state, action and the state transitions in real-time. Figure 6.1 is a simulation environment of the study area generated based on the OSM map.

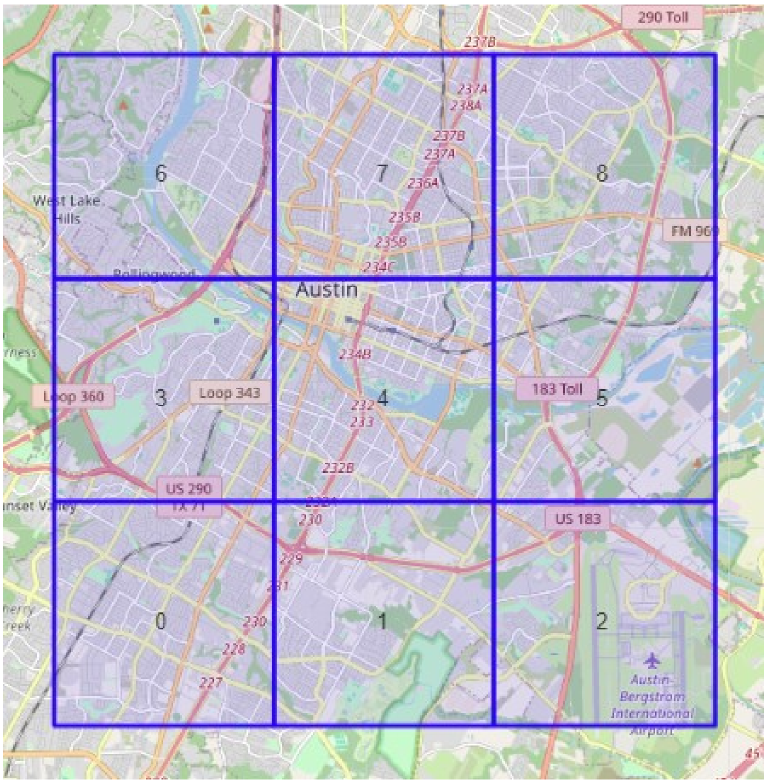


Figure 6.1. The OSM-based simulation environment of Austin.

Another important part of the data used in the developed simulator for this case study comes from the RideAustin dataset. The RideAustin dataset is an open-source dataset that can be retrieved from the internet on a data storage website named Data World. The license is given by the website administrator to the users for any non-commercial use. The dataset contains comprehensive information including locations of idle drivers and travel demands, destinations of matched trips, start and finish time of matched trips, service vehicle types, and other relevant details. This dataset provides a rich source of real-world data that reflects characteristics of the ride-hailing market in the city of Austin. It is critical for the application of the developed simulator to accurately simulate the ride-hailing matching process in the city of Austin. In this simulator, both the initial locations of drivers and travel demands are sampled from distributions fitted based on the RideAustin data. With these fitted location distributions, the simulator can generate realistic and representative locations of driver locations and ride request locations to achieve a realistic simulation. This approach ensures the simulation's stability and reliability, making it very similar to the real ride-hailing market that represented by the RideAustin dataset.

Figure 6.2 shows a heat map of real demand locations sampled from the RideAustin data. In addition, it shows 10,000 demands randomly sampled from one of the fitted distributions, compared to the real demand locations. The heat map highlights areas of high and low demand, visually showing where ride requests are most and least frequent. The randomly sampled demands show how effectively the fitted distributions replicate the spatial patterns of real-world ride requests, thus validating the sampling method. By using this data, the simulator provides the users with a simulation environment of a high degree of realism and accuracy. This, in turn, enables users to run more effective and valid simulations that can be used to test and optimize various ride-hailing strategies under conditions that closely resemble the real world. The generated data of riders and drivers is also mathematically tested using Kolmogorov-Smirnov (KS) test method to further validate the effectiveness of the developed simulator in this case study

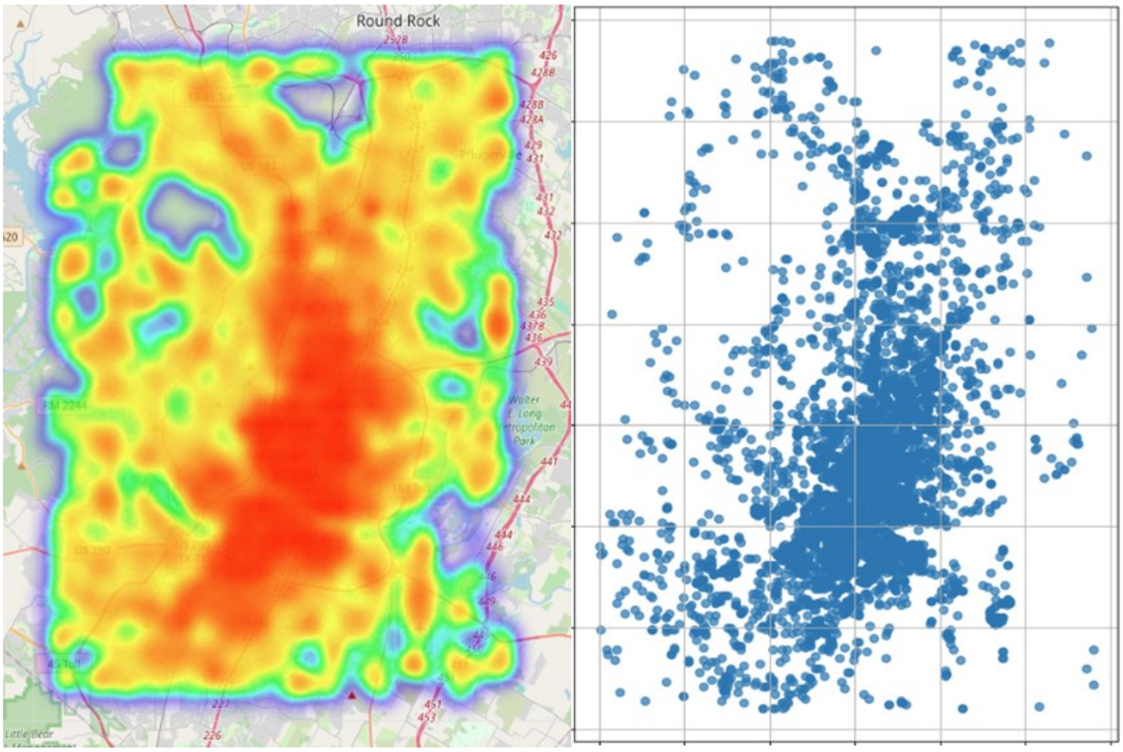


Figure 6.2. Comparison between real demands and samples in the afternoon.

The Kolmogorov-Smirnov (KS) test is a non-parametric test used to compare the differences between two sample distributions or between a sample distribution and a reference distribution (such as the normal distribution). It evaluates whether they come from the same distribution by calculating the maximum difference between two cumulative distribution functions (CDFs), known as the KS statistic. The KS test is particularly suitable for cases where the sample size is large, and the distribution type is unknown. The steps of the KS test include calculating the cumulative distribution function (CDF), computing the maximum difference between the two CDFs (KS statistic), and determining the p-value corresponding to the KS statistic to test the null hypothesis. The simulator developed in this study also uses the KS test to assess the accuracy of its simulation compared to real-world scenarios.

In the ride-hailing simulator, the KS test can be used to evaluate whether the spatial and temporal distributions of riders (travel demand) and drivers (supply) generated by the simulator are consistent with data obtained from real-world operation. This is a crucial step in validating the accuracy and effectiveness of the simulator. For the spatial distribution of riders and drivers, this study tests the longitude and latitude coordinates of the generated data against the coordinates from the original data. The goal is to ensure that the positions of riders and drivers generated by the simulator match the distributions in the actual data. This is vital for the effectiveness of the simulator since the spatial distribution directly impacts whether the simulator can realistically reflect the actual ride-hailing market.

The null hypothesis defined in this experiment is that there is no significant difference between the spatial distribution (or the temporal distribution of demand) of the simulated riders or drivers and that of the actual data (i.e., they come from the same distribution). The alternative hypothesis is that the two samples come from different distributions. In terms of test results, if the p-value is greater than the significance level (set as 0.05 in this study), it means that the null hypothesis cannot be rejected, meaning there is no significant difference between the distributions sampled from simulated ride-hailing data and the real ride-hailing data. This indicates that the data generated by the simulator is consistent with the actual data in terms of distribution, thus validating the simulator. If the p-value is less than the significance level, the null hypothesis is rejected, meaning there is a significant difference between the sample distributions. This indicates that the data generated by the simulator is not consistent with the actual data in terms of distribution, rendering the simulator ineffective. Figure 6.3 shows the KS test of the latitude and longitude of rider location coordinates generated by the simulator from different time-steps compared to the original data. Figure 6.4 shows the KS test of the latitude and longitude of driver location coordinates generated by the simulator compared to the original data.

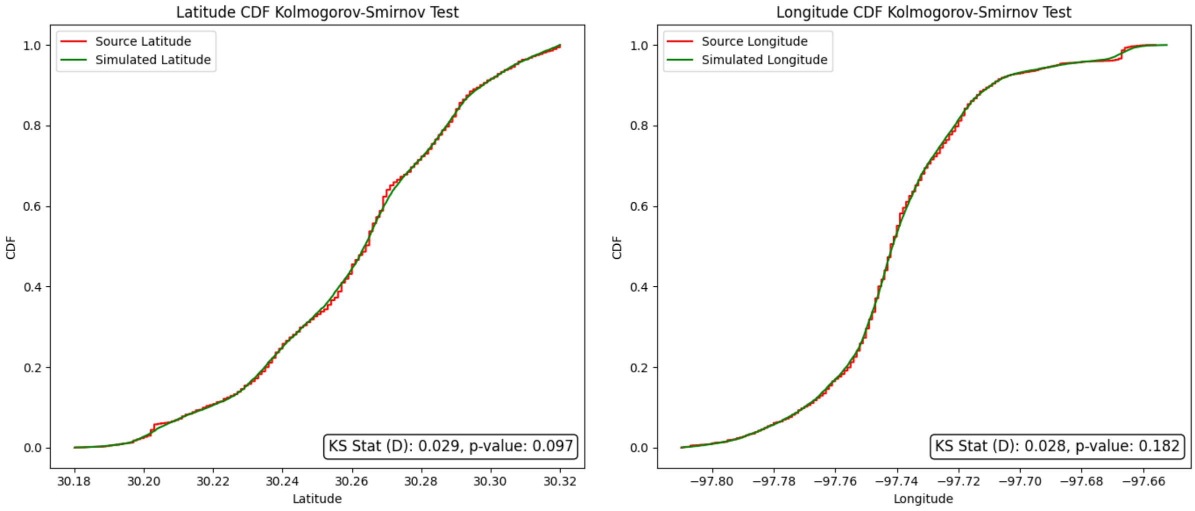


Figure 6.3. KS test for rider locations generated by the simulator.

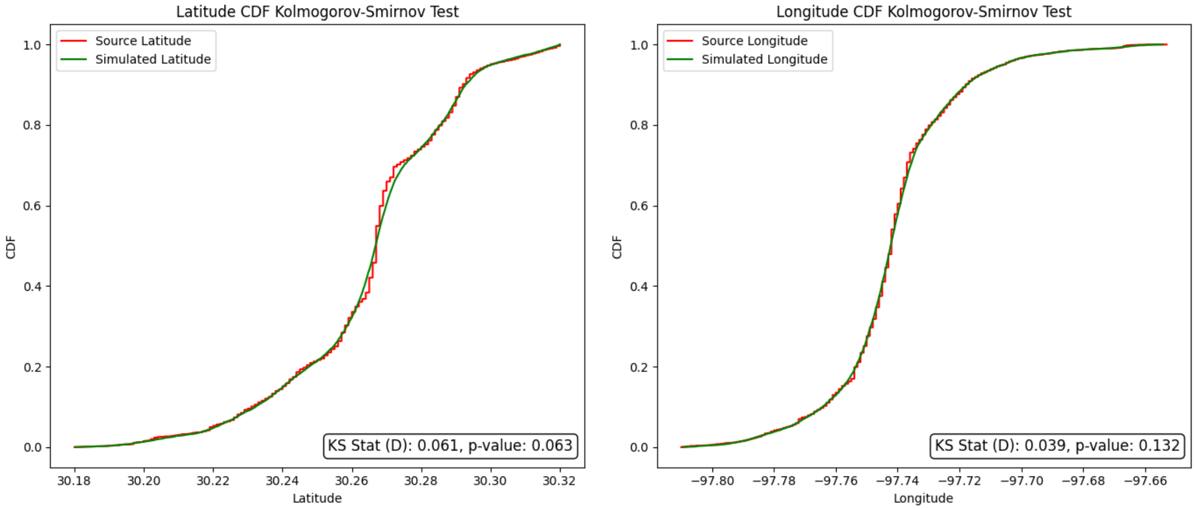


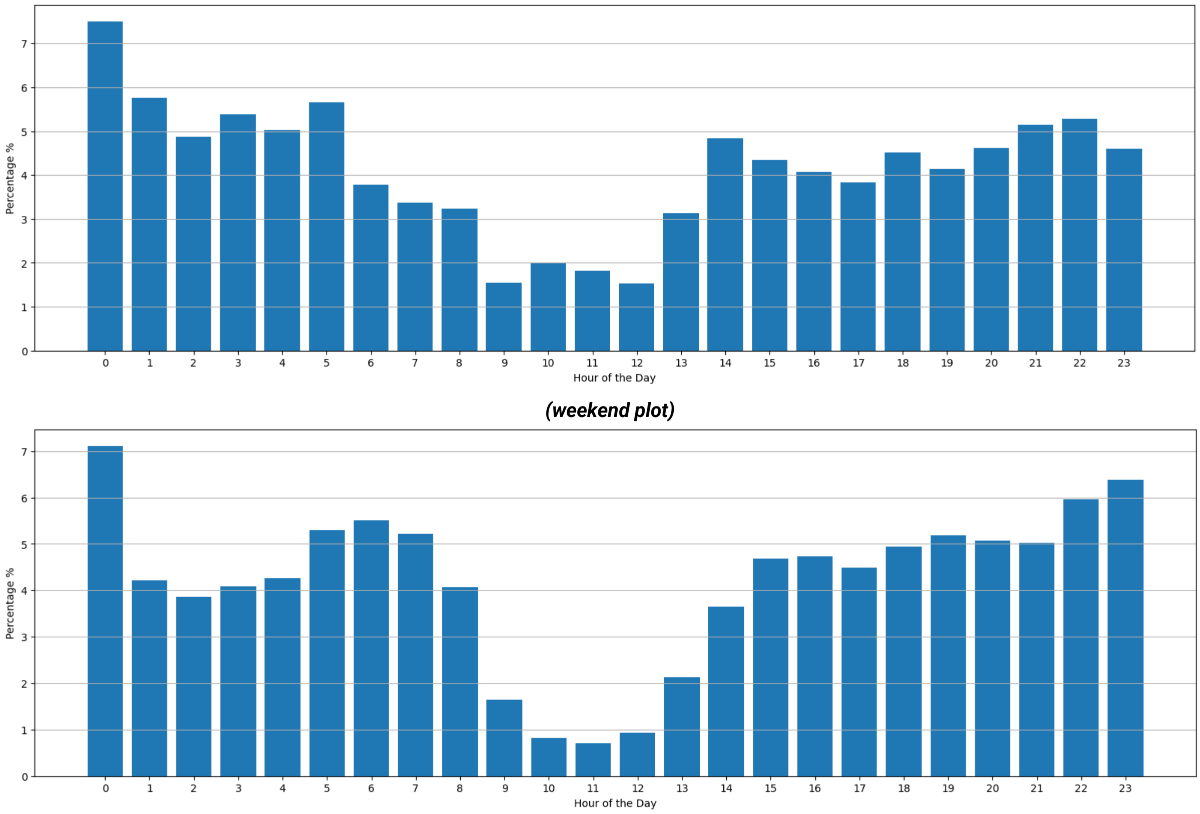
Figure 6.4. KS test for driver locations generated by the simulator.

In the rider location test, the KS statistic for latitude is 0.029 with a p-value of 0.097, and the KS statistic for longitude is 0.028 with a p-value of 0.182. These results indicate that the latitude and longitude distributions of the rider locations generated by the simulator do not significantly differ from the actual data distributions, as the p-values are both greater than 0.05, thus failing to reject the null hypothesis. In the driver location test, the KS statistic for latitude is 0.061 with a p-value of 0.063, and the KS statistic for longitude is 0.039 with a p-value of 0.132. These p-values are also greater than 0.05, so the null hypothesis cannot be rejected as well, meaning the latitude and longitude distributions of the driver locations generated by the simulator do not significantly differ from the actual data's distributions.

Therefore, these results indicate that the data generated by the simulator is consistent with the actual data in terms of spatial distribution, validating the effectiveness of the simulator.

As for the hourly distribution of the number of travel demands. The demand for ride-hailing services in the city of Austin, Texas, shows a distinct pattern related to people's daily activities. The significant differences can also be identified between weekends and weekdays. First of all, high travel demand hours are particularly pronounced on weekends, from 0 to 1 am, 4 to 6 am, and 8 to 10 pm, which are closely associated with dining and entertainment activities. 8 to 10 pm coincides with dinner time, prompting many people to dine out or socialize. Travel demands for ride-hailing services are increased as people are planning to go to the restaurants or bars. From 8 pm to 1 am, nightlife activities (such as going to bars, clubs, and other entertainment venues) come to an end, and a surge in ride-hailing demands appears as those people need a vehicle to drive them home. In contrast, off-peak hours on weekends are generally from 9 am to 12 am. During these hours, people are often having brunch, going shopping, visiting the park, or are more likely staying at home and engaging in leisure activities. It is reasonable that travel demands for ride-hailing services are decreased during that time.

Figure 6.5 shows the distribution of ride-hailing demand on weekends and on weekdays in the city of Austin. The upper figure shows the distribution of ride-hailing demand on weekends, and the lower figure shows the demand on weekdays. The horizontal axis represents the time interval, and the vertical axis represents the proportion of travel demand in the whole day.



(weekend plot)

(weekday plot)

Figure 6.5. Time distribution of travel demands in the city of Austin.

The peak hours on weekdays are significantly different from those on weekends. The peak hours on weekdays are mainly concentrated in the three periods of 5 to 8 am, 3 to 5 pm, and 10 pm to 1 am of the next weekday. The morning peak (5 to 8 am) corresponds to the morning rush hour period when people go to their workplaces. People who do not want to drive themselves or do not have private cars will engage in increasing the demand for ride-hailing services. The afternoon peak (3 to 5 pm) reflects the end of the workday and the time when commuters return home. The late-night peak (10 pm to 1 am) may be due to people's social activities, leisure activities apart from their working hours, or even the need to return home after overtime working. The off-peak hours on weekdays are similar to that on weekends, usually from 9 am to 12 am. Although the off-peak time periods are very close to that on weekends, the reason is different. The reason for the off-peak hours on weekdays is that people are working at their workplaces, and the demand for ride-hailing services is naturally lower.

These trends reflect the varied travel habits and needs of ride-hailing services at different times in Austin. Weekends highlight a preference for evening entertainment, with dining and nightlife as primary drivers of travel. Weekdays show a stronger influence from work and commuting schedules, with fluctuations in demand during lunch and afternoon hours reflecting a blend of work and leisure activities. For the hourly distribution of travel demands, this study also conducted a KS test on the hourly demands generated by the simulator compared to the actual data. The purpose of this test is to verify whether the time distribution of demand generated by the simulator is consistent with the actual demand. The defined null hypothesis is that the time distribution of the simulated demand does not significantly differ from the actual data (i.e., they come from the same distribution), while the alternative hypothesis is that the two samples come from different distributions. Figure 6.6 shows the results of the two KS tests for the hourly demand distributions, ensuring the validity of the test results. The results indicates that the null hypothesis is accepted, meaning that there is no significant difference between numbers of hourly demands generated by the simulator and the real operating data. The effectiveness of the simulator developed for this study is validated.

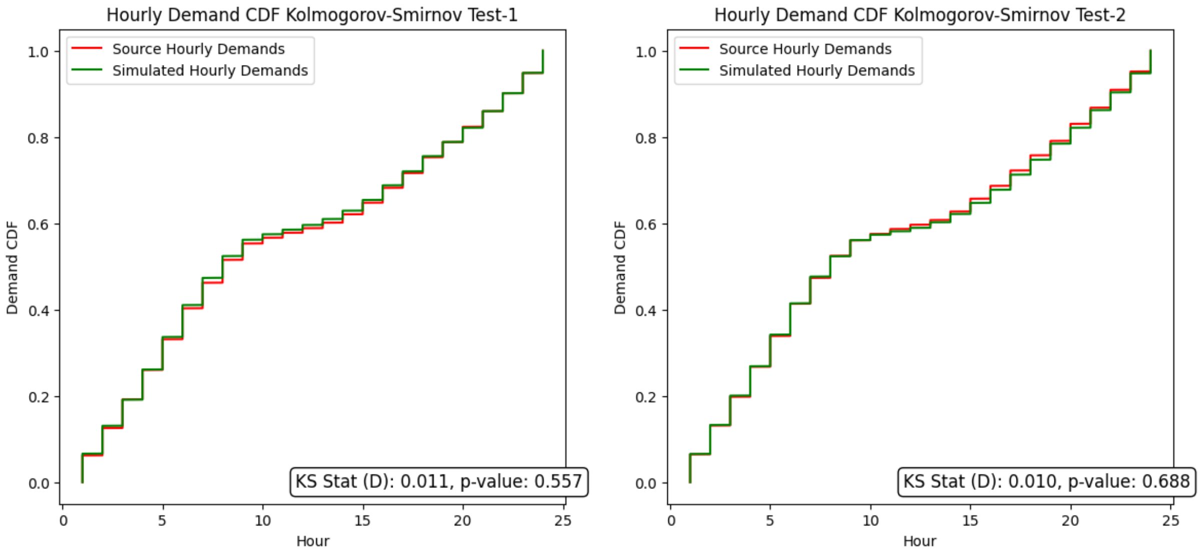


Figure 6.6. KS tests for hourly demands generated by the simulator.

6.3 Experimental Design of Scenarios

In this case study, in order to effectively train the dynamic matching radius policy and evaluate its validation and performance, it is necessary to design a number of experimental scenarios in the developed ride-hailing simulator. These scenarios are designed to simulate the different driver-rider supply-demand relationships mentioned in previous sections in a timely manner. It particularly focuses on the differences between different travel demand patterns, such as oversupplied or undersupplied. There are three distinct scenarios designed for this case study in the city of Austin: a balanced supply-demand scenario, a high-demand scenario, and a high-supply scenario. This subsection will discuss the practical meaning and significance of the settings of the three designed scenarios, as well as specify their differences and distinct variables.

The balanced supply-demand scenario is used as the normal scenario for this case study. The dynamic matching radius policy is trained based on the interactions between the agent and the environment in this scenario. The design of this scenario aims to simulate a normal and typical off-peak operational environment of ride-hailing services in the city of Austin. Therefore, this scenario is then used to train the dynamic matching radius policy and evaluate its effectiveness in handling the daily operational conditions of the ride-hailing market in the city of Austin. In this scenario, the travel demands are generally equal to the number of drivers, with a small scale of fluctuations in the demand-supply relationship. For example, during mid-day in the city of Austin when the travel demand number is more predictable. The environment under this setting can provide the agent with less extreme information for the agent to learn how to effectively change the matching radius. A training set that contains too much extreme data will make the training process more unstable and may even lead to non-convergence, so this scenario is selected to ensure the stability of the training process. On the other hand, the basic performance and validation of the defined baselines are also assessed in this normal scenario in order to test their effectiveness under normal operational conditions.

As for the high-demand scenario, it represents one of the extreme operating conditions for ride-hailing services in the city of Austin, Texas. This scenario simulates a surge in travel demand due to abnormal reasons. For example, during peak hours in metropolitan areas, during the openings of big events like concerts or sports games, or during extreme weather conditions. In this scenario, a significant increase in travel demand can be identified in the supply-demand ratios. As a consequence, there is a relatively shortage of idle vehicles compared to the ride demands. The setting of this high-demand scenario aims to test how the trained policy can manage the surged travel demands and shortage of idle drivers. It can effectively validate the performance of the trained policy under extreme conditions, as it is very important for a policy designed to deal with extreme operational conditions in the ride-hailing system. The third scenario is the high-supply scenario, in contrast to the second scenario, representing another extreme operational condition of the ride-hailing service in Austin. The number of idle drivers is higher than the travel demands. The travel demands in this scenario are far less than those of the high-demand scenario, and even less than the normal scenario. This scenario is designed to simulate the ride-hailing market during holidays or in some regions where there is a surplus of drivers, such as city outskirts. It focuses on evaluating the trained policy's capability to meet demand when there is a relatively high availability of drivers.

Table 6.1 shows the detailed settings of the variables in designed experimental scenarios. Each of these three designed scenarios assesses and validates the trained dynamic matching radius policy from different aspects. The varied operational challenges presented in the designed experimental scenarios mirror the complexity and uncertainty of the real-world ride-hailing system, not only in Austin but also in other major cities around the world. By training the policy under the normal scenario and testing the policy under all scenarios, this case study can gather a comprehensive insight into the robustness, adaptability, and effectiveness of the dynamic matching radius policy.

Table 6.1. Variable settings for defined scenarios

variable	balanced	high demand	high supply
Initial driver number	100	50	150
Initial rider number	100	100	100
Match time window	15 sec	15 sec	15 sec

6.4 MDDPG Training

MDDPG algorithm contains a number of sensitive hyperparameters. The performance of the MDDPG algorithm depends significantly on its hyperparameters. The convergence and final performance of the MDDPG algorithm can be significantly affected by these hyperparameters. While running the MDDPG optimization, it is important to empirically adjust and finetune the hyperparameters. By fine-tuning and empirically selecting the hyperparameters, the MDDPG agent can better learn from the interactions with the simulator. This subsection discusses the process of running the MDDPG algorithm and the impact of each hyperparameters and their values selected for the training process of this case study.

The first parameter is discount factor γ, this hyper-parameter determines the importance of future rewards. It is a factor by which future rewards are multiplied at each time step. A large discount factor makes the agent pay more attention for long-term rewards, whereas a relatively small discount factor makes the agent very short-sighted to the immediate reward. The learning rate λ determines the performance of the actor and critic network. In this research, learning rates for actor and critic are finetuned separately. The learning rate of the actor λ_{ac} is the step size for the gradient descent optimization algorithm used in updating the actor network. The learning rate of the critic λ_{cr} is the step size for the loss minimising method for updating its weights. A higher learning rate can speed up the training but may cause instability, while a lower learning rate can make the training more stable but slower. The soft-update speed τ controls the rate at which the target networks (actor and critic) are updated towards the source networks. In the replay buffer, the memory size M determines how many past transitions can be stored in it. A larger memory size allows the agent to learn from a more extensive history of experiences, which can improve training. The batch size N is the number of samples drawn from the replay buffer to perform a single update of the network. A larger batch size can provide more stable updates but requires more computational resources. In the exploration function, the OU (Ornstein-Uhlenbeck) noise generator, the noise regression speed θ_{noise} determines the rate at which the noise regresses to its mean. A higher value makes the noise revert to the mean faster, affecting the exploration-exploitation balance. The noise variance value σ_{noise} is the scale of the noise. A larger variance leads the agent to do more explorations by adding more significant noise, while smaller σ_{noise} values enable less exploration but can ensure a more stable training process. Additionally, the maximum training episode E_{max} determines the length of the whole training process. It defines the extent of the agent's learning experience and ensures computational efficiency. This hyperparameter defines the quantity of experiences from which the agent learns and the duration of the learning process. If set appropriately, it ensures a good computational efficiency. However, if this hyperparameter is set too small, the agent may not be able to gather enough transitions, failing to learn a good policy. Conversely, if it is set too large, it may, in some cases, lead to a deterioration in the policy or even cause it to become non-convergent again.

In summary, each of the hyperparameters can significantly affect the stability and efficiency of the MDDPG algorithm during training, determiners how well the agent learns. An example of result with convergence issue is shown in Figure 6.7, this problem is tackled by adjusting the hyperparameters.

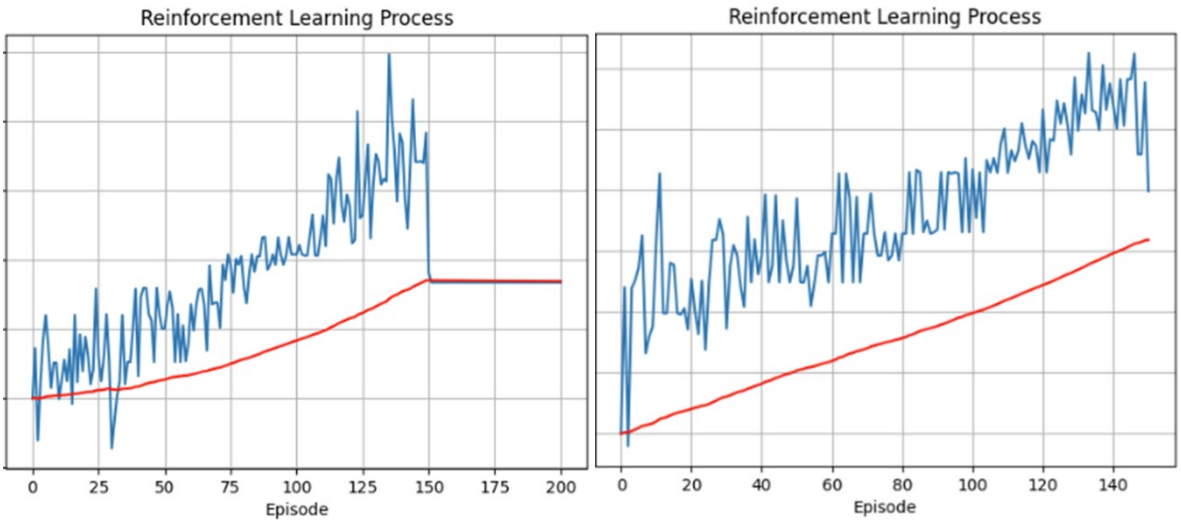


Figure 6.7. Training plot with unsuitable hyperparameters.

In the hyperparameter fine-tuning process of this case study, all hyperparameters are selected based on practical considerations. The values of these hyperparameters were tested and adjusted before the final training. Each hyperparameter's value is determined by its impact on the MDDPG training performance, including convergence, convergence speed, learning speed, result quality, and neural network loss. Figure 6.7 illustrates the training reward plot with unsuitable hyperparameter values. After the fine-tuning process, the suitable hyperparameters for MDDPG algorithm in ride-hailing simulator with a good performance is listed in Table 6.2. Figure 6.8 is the training plot with those hyperparameters.

Table 6.2. Hyperparameters for the MDDPG training

Hyper-parameter	Value
Discount factor γ	0.96
Learning rate actor λ_{ac}	1e-4
Learning rate critic λ_{cr}	8e-4
Soft-update actor τ_{ac}	5e-4
Soft-update critic τ_{ac}	1e-3
Memory size	10000
Batch size N	250
Noise regression speed θ	0.15
Noise variance value σ	0.6
Maximum episodes	800

The ride-hailing multitask optimization weights are also important pre-defined parameters for the training process. The used weight set is selected based on the feedback from the ride-hailing users and other stakeholders. The weights are determined by the percentage of their preferences. After that, a series of analysis with different sets of weights is implemented on the simulator trying to find balance between each component. The weights are selected based on the results of the analyses, the combination with the best overall performance compared to the baselines is used in the training process of the case study. The criteria include a reward analysis and some performance checks compared to the baselines. The weight set selected for the final training of this case study is listed in Table 6.3.

Table 6.3. Weight combination for the MDDPG training

w_1	w_2	w_3
0.4	0.4	0.2

The policy is trained in the normal scenario where the supply number is approximately equal to the demand number. Each episode represents a time period of one hour, the fixed time-window is 15 seconds. The maximum number of steps in one episode is 240 steps. The platform used to train the policy is equipped with AMD Ryzen-9 5900HX CPU @ 3.3 GHz 8 Logical cores, 32 GB RAM and a 16 GB NVIDIA GeForce RTX 3080 GPU. The average computation time is 30.23 seconds per episode. The total training time is 6 hours 43 minutes. The reward plot of the training process is shown in Figure 6.8. From the figure, it can be seen that in the later stages of training, the agent exhibits good convergence. In the figure, the fluctuations observed are partly due to the noise added to the actions and partly due to the uncertainty in the environment, specifically the stochastic nature of the supply and demand locations.

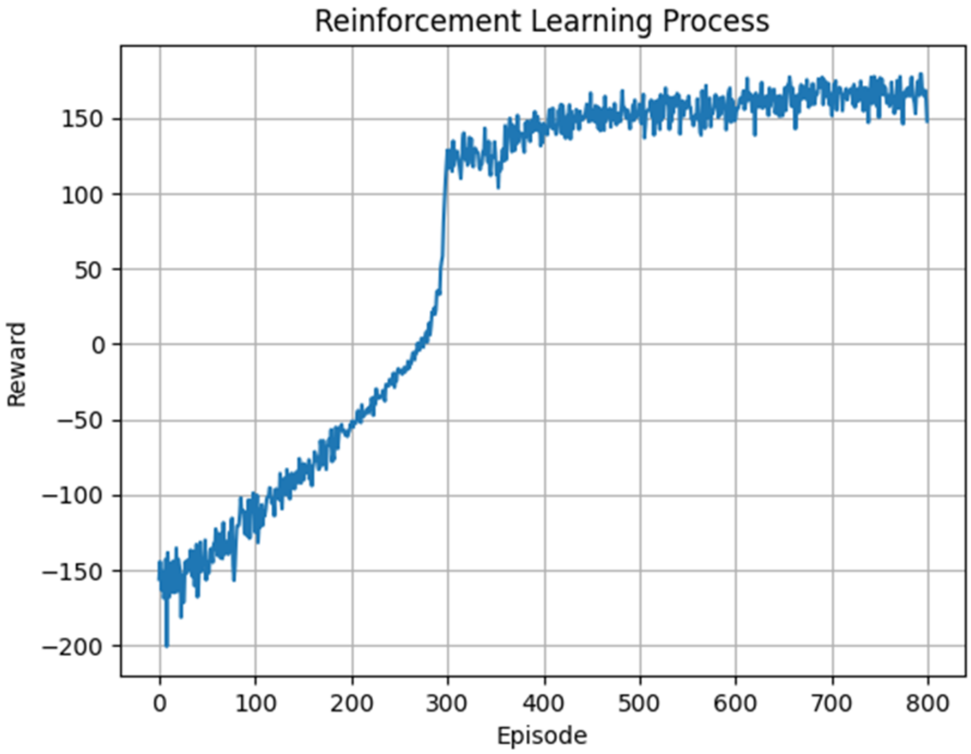


Figure 6.8. Reward plot of the MDDPG training process (equal supply-demand).

In RL, in addition to observing the cumulative reward changes, the numerical changes in the loss function can also be used to evaluate the performance of the RL algorithm. The decrease in the loss function value and its stabilization at a low level indicates that the policy learned by the agent has converged. By observing the changes in the loss function, researchers can evaluate the stability and convergence of the training process. The practical meaning of a lower loss is that the critic network can effectively and accurately estimate the action value of the current state-action pair and can effectively evaluate the output of the actor network. Whether this policy is an optimal policy needs to be analysed in combination with the reward plot and the validation of the results. This will be discussed in detail in the next subsection. It is worth noting that if the loss function value is observed to fluctuate greatly or increase exponentially during the training process, it indicates that the training result is unreliable. When using Gradient Descent and Gradient Backpropagation methods to update weights of DNN, gradient vanishing and gradient explosion are two major problems that can lead to unstable loss values. Gradient vanishing occurs when the gradient becomes zero during updates, leading to an extremely slow update speed for the network. Conversely, gradient explosion refers to the exponential increase in the gradient, which can render the network unreliable and unstable, causing fluctuations in the loss function. Both issues stem from the same underlying cause: the cumulative gradient, which can become either too large or too small due to the chain rule when calculating gradients and during gradient backpropagation when updating network weights. This case study also encountered the problem of gradient explosion during the training process. Upon further investigation, the causes were identified as the input data, unsuitable hyperparameters, and network structure. To address these issues, several measures were taken. First, the input data of transitions was normalized. Data that is not normalized can cause a significant discrepancy between the estimated value and the actual value when updating the weights of the neural network. Second, the soft-update rate of the neural network was decreased. This hyperparameter controls the speed at which the target network updates from the source network; the original value was too high, causing rapid changes in the target network and resulting in unstable weight updates. Lastly, the number of hidden layers in the deep neural network was reduced. Although more hidden layers can improve the critic neural network's ability to estimate the value function, the accumulation effect becomes more pronounced with more neurons that need to be updated through gradient backpropagation and the chain rule. Therefore, the final structure chosen consists of three hidden layers along with an input layer and an output layer. Figure 6.9 shows the changes in the loss function of the critic network during the training process of this case study. It can be observed that as the training progresses, the loss of the critic network gradually decreases and stabilizes at a low level. This indicates that the MDDPG agent has made good progress in learning.

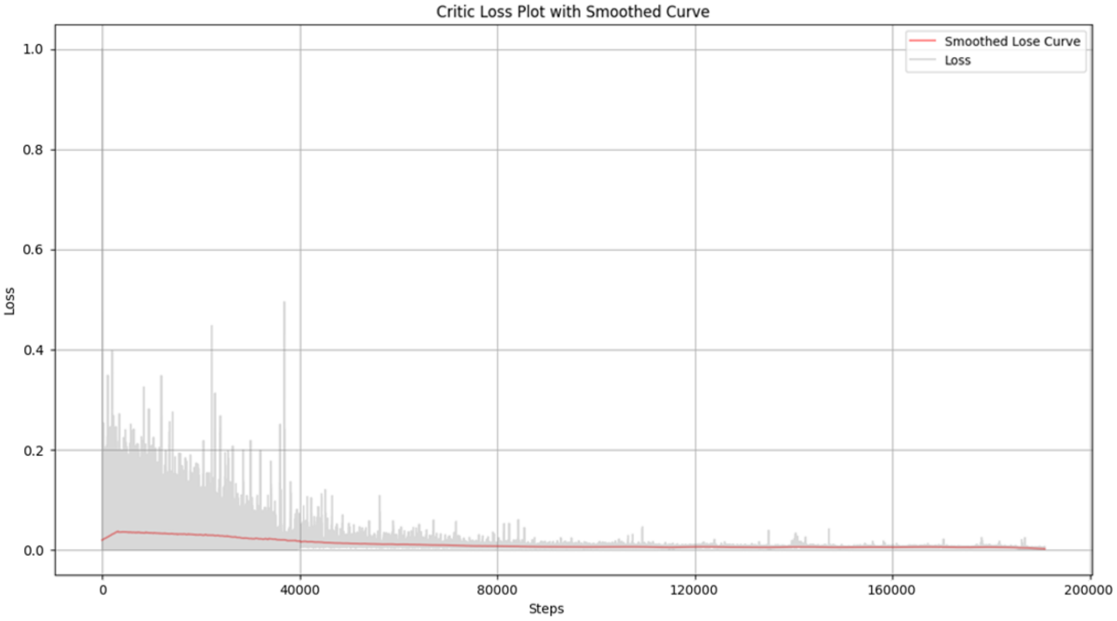
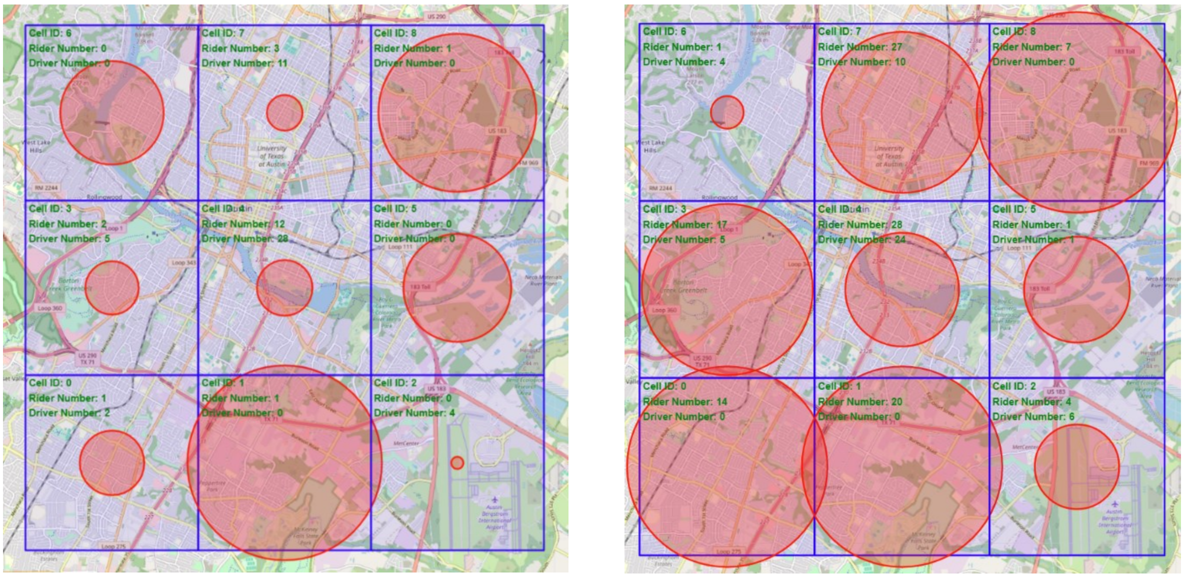


Figure 6.9. Training loss plot of the critic network.

6.5 Results

Figure 6.10 shows the matching radius of each cell output by the trained policy at a certain time-step in the normal scenario. The real-time number of riders (travel demands) and drivers in each cell is marked in the upper left corner of each cell. It can be seen that the trained dynamic matching radius policy can reasonably give the optimal matching radius based on the supply and demand relationship. The trained policy can well handle the changes in the supply-demand relationship between drivers and riders in the ride-hailing matching system, and can manage to adjust the matching radius accordingly.



(over-supplied scenario) **(under-supplied scenario)**

Figure 6.10. Optimised matching radius for each cell at 15:00 PM.

Figure 6.11 shows a real-time matching process with matching radius obtained from the trained policy at one time-step. The additional results showing the matching radius compared with numbers of riders, drivers and supply-demand ratios are illustrated in Appendix C. An example of a series of continuous states and corresponding matching radius can also be found in Appendix C.

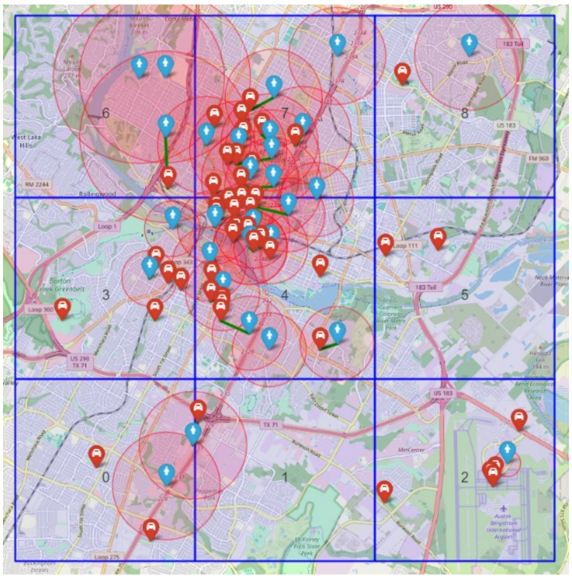


Figure 6.11. Matching process with trained policy.

In the previous research addressing matching radius conducted by other researchers, static and fixed matching radius is used as baseline to evaluate the performance of matching radius policy *(Chen et al., 2023)*. In this case study, this baseline approach is used as a reference to define the baselines. To validate the trained policy, four baselines are designed for this case study, denoted as FR 500, FR 1000, FR 1500 and FR 2000. The "FR" in the baseline code means that the baseline is a static fixed matching radius, the number in the baseline code indicates the value of the respective matching radius. For example, FR 1000 represents a fixed matching radius baseline of 1000 meters. In this subsection, the trained policy is validated in a test environment similar to the training environment. As mentioned earlier, this study designed three different experimental scenarios to evaluate the dynamic matching radius policy trained using the Multi-replay-buffer Deep Deterministic Policy Gradient (DDPG) algorithm. A test environment is built to validate the trained policy. The logic and settings of the test environment are exactly the same as the simulator, and the performance of the trained policy is tested using the overall system performance score $C_t(s_t, a_t)$ as derive in the simulator introduction compared to the baselines. This ensures a better validation of the trained policy compared to the baselines. This subsection tests the trained dynamic matching radius policy against the defined baselines (static matching radius) in the defined experimental scenarios, comparing the performance differences between the baselines and optimized policy in the test environment to validate the effectiveness and robustness of the trained policy.

The balanced scenario represents a normal ride-hailing operational condition where the travel demands are generally equal to the number of drivers, with a small scale of fluctuations in the demand-supply relationship. A comparison of the performances between the trained policy and the baselines among 30 random episodes (steps in an episode is continuous) is illustrated in Figure 6.12. It can be seen from the figure that the trained policy effectively optimizes the matching radius. The score obtained from the trained policy is the highest compared to baselines. This result indicates that the dynamic matching radius policy can improve overall performance of the ride-hailing system in Austin compared to the baselines by balancing the performance metrics. The performance metrics, as mentioned in previous sections, include the matching rate, the average pick-up distance and the driver utilization rate. This demonstrates that the trained policy is valid and effective in the normal scenario.

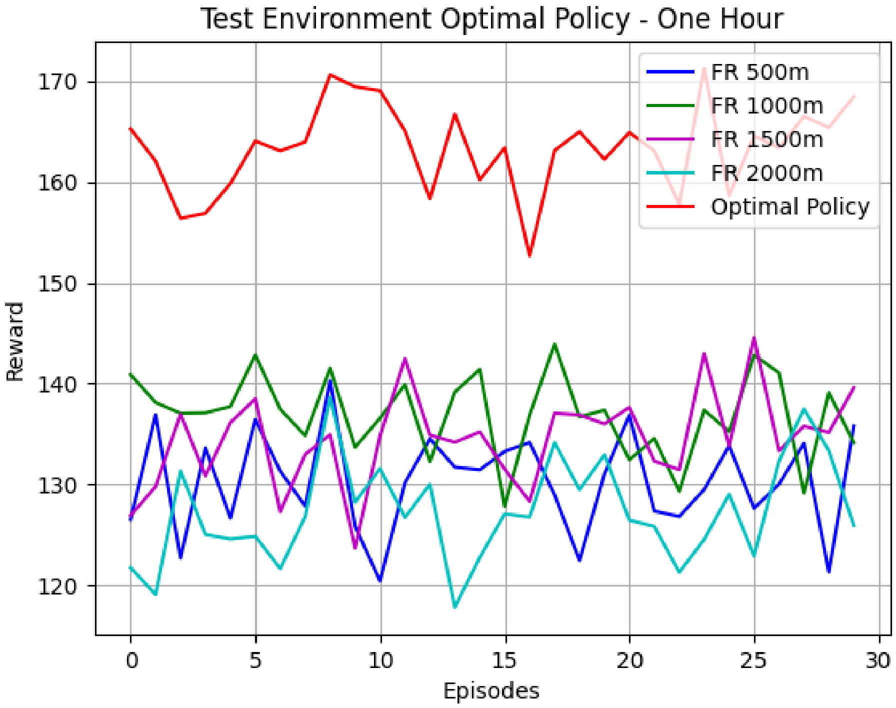


Figure 6.12. Policy reward plot compared to the baselines in Balanced scenario.

In this case study, the effectiveness of the trained policy is evaluated under extreme operational conditions to test its robustness and adaptability. The extreme scenarios discussed in Section 6.3 encompass both high-demand and high-supply environments. Specifically, the high-demand scenario tests the dynamic matching radius policy when the demand for ride-hailing services far exceeds the available drivers. The high-supply scenario tests the policy when there are more drivers available than the number of travel demands. These scenarios are designed to test the policy's performance and adaptability under conditions of extreme demand and supply, respectively.

Figure 6.13 illustrates the results of the policy evaluation in the high-demand scenario. The result shows that the dynamic matching radius policy outperforms all baseline and demonstrates a good stability. Unlike the baselines, which exhibit significant fluctuation in performance under high-demand conditions, the dynamic matching radius policy maintains a more consistent level of overall performance. This reduced fluctuation highlights the policy's capacity to handle fluctuating demand more effectively, providing a more reliable and efficient ride-hailing service. The results underscore that, even in extreme scenarios where traditional static matching radius may have bad performance, the dynamic matching radius policy can still optimize the ride-hailing system's overall performance and user experience. It is important to note that Figure 6.13 focuses on the overall reward comparison between the dynamic policy and baselines. This figure provides a high-level view of performance but does not delve into specific performance metrics. Detailed analyses of each performance metric, including matching rate, average pick-up distance, and driver utilization rate, are listed and discussed in Table 6.4. This comparisons can offer a more comprehensive understanding of how the dynamic matching radius policy influences the overall matching efficiency and user experience in ride-hailing environment.

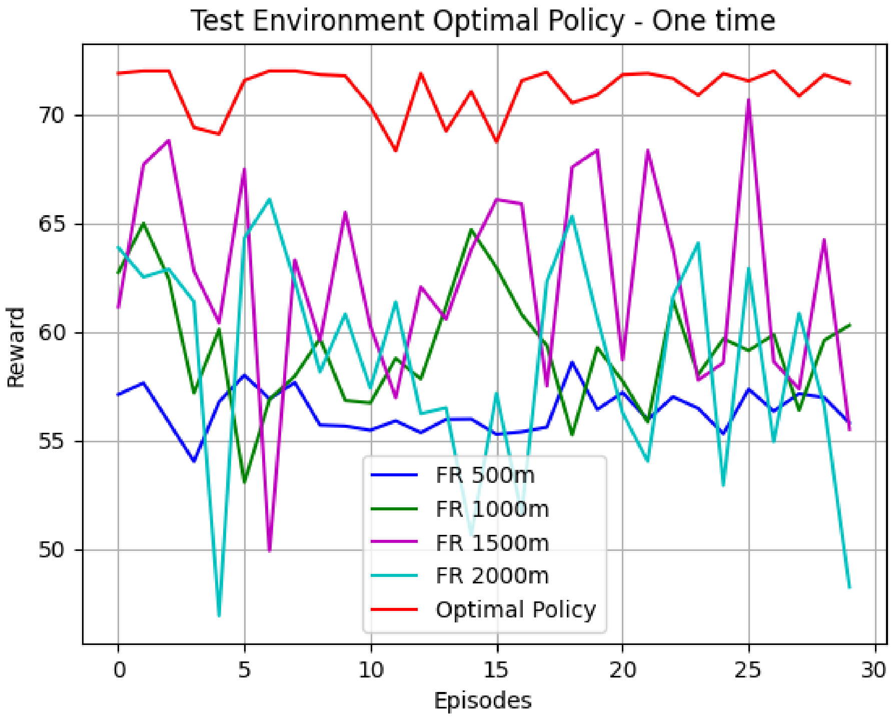


Figure 6.13. Policy reward plot compared to the baselines in high demand scenario.

Some differences were also noted compared to the balanced scenario. Although the trained policy performs well, the overall accumulated reward is lower than that in the balanced scenario. This is due to the bottleneck of available drivers during the undersupplied periods in the high-demand scenario. The reason for this phenomenon is that there are fewer available drivers compared to the balanced scenario, and therefore a much lower overall achievable maximum matching rate. In more detail, increased demands cause available drivers to be matched quickly, leaving fewer idle drivers to satisfy subsequent travel demands. The scarcity of available drivers during periods of high demand inherently forms a bottleneck for the maximum reward. However, it is also noticed that the reward is less volatile than the balanced scenario. This phenomenon can be attributed to several reasons. First, sustained high travel demands make the ride-hailing environment more predictable. As a result of consistently high demand, drivers are matched with riders as soon as they finish the previous ride, keeping match rates at the highest achievable level. This matching phenomenon makes the batched matching system under high demand more similar to the nearest matching method. The constant incoming of ride requests also means that the agent can apply a more consistent (larger) matching radius without the need for large-scale adjustments, resulting in less volatility in the reward. On the other hand, during periods of high demand, the supply of drivers is rapidly matched, resulting in most drivers not being available. This saturation effect reduces fluctuations in driver utilization rate and the corresponding fluctuations in the overall reward. From the perspective of micro-transportation, a natural queuing effect may occur in the high-demand scenario, where riders are matched with drivers as soon as they are available. This makes the ride-hailing matching system even more like the nearest matching system. The reduction in these fluctuations highlights the stability of the trained policy under high levels of travel demand pressure. This observation suggests that the trained dynamic matching radius policy is effective in optimizing the ride-hailing matching system even under extreme conditions. Although some fluctuations can still be found in the reward plot, which indicates that there is still room for improvement in the trained policy, the trained policy can be more flexible and robust to balance the increase in matching rate in the high-demand scenario with the efficiency of the matching system and the user experience, which may lead to even better results. Possible improvements include combining neural networks with real-time data analytics to better predict demand patterns and strategically deploy drivers in anticipation of these patterns (cooperating with the vehicle repositioning module). In addition, defining a more sophisticated reward function that takes into account short-term matching success and long-term system stability may also help to reduce volatility and improve overall performance under such extreme conditions.

In the high-supply scenario, the performance of the dynamic matching radius policy is depicted in Figure 6.14. Compared to the settings in the balanced supply-demand scenario, the dynamic matching radius policy shows better performance in this context. The policy can more effectively utilize the surplus driver resources, adjusting the matching radius to reduce driver idle rate and maintain lower average pick-up distances and higher matching rate. The test result indicates that the dynamic matching radius policy significantly outperforms the baseline policy in the high-supply scenario by successfully balancing the matching rate, pick-up distance and driver utilization rate. This shows that the trained policy is able to adapt to changing supply conditions and achieve the best matching performance of the system under excess supply conditions. By outputting a set of reasonable matching radius, the excess supply of drivers is effectively managed. The dynamic matching radius strategy ensures better utilization of available driver supplies and improves the overall efficiency of the ride-hailing system while maintaining a higher user experience. This further verifies the adaptability and performance of the trained policy under high supply conditions.

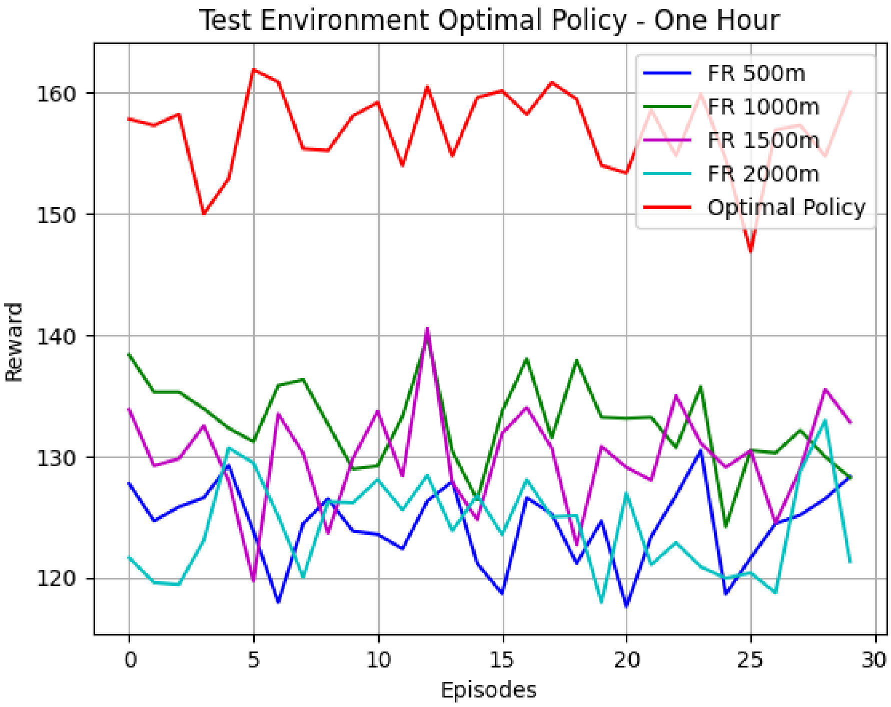


Figure 6.14. Policy reward plot compared to the baselines in high supply scenario.

In summary, through an in-depth test of the trained policy under three experimental scenarios, the results show that the dynamic matching radius policy trained using the Multi-replay-buffer Deep Deterministic Policy Gradient (MDDPG) algorithm outperforms the baselines in all designed scenarios. The results indicate that the trained policy successfully optimizes the ride-hailing matching system to a certain extent under various operational conditions. This trained policy demonstrates a good adaptability and efficiency by dynamically adjusting the matching radius in response to real-time supply and demand fluctuations. Table 6.4 shows an average performance comparison of matching rate $\boldsymbol{R_m}$, average pick-up distance D_P and driver utilization rate R_{ult} between the trained policy and the baselines among 40 random episodes under the balanced scenario. The results show that the trained policy can achieve a good balance between multiple performance indicators and achieve the highest overall performance score. However, it is worth noting that the trained policy does not perform best in each individual system performance indicator. On the contrary, the trained policy finds a good balance among the three system performance indicators, thereby optimizing the overall efficiency of the system and user experience, ensuring the highest overall reward. Specifically, there are many reasons why the trained policy cannot perform best in each system performance indicator. First, expanding the matching radius to improve the matching rate may lead to a longer pick-up distance and may also lead to a decrease in driver utilization rate. Conversely, reducing the matching radius to shorten the pick-up distance may reduce the matching rate. The trained policy can find the best balance between these conflicting goals instead of focusing on optimizing a single indicator. As for driver utilization, maximizing driver utilization may require giving priority to the nearest travel demands to reduce the number of unmatched drivers within the matching radius and improve driver utilization. This will make the system more similar to a nearest matching system rather than a batched matching system, resulting in a significant increase in the average pick-up distance. The results show that the trained policy can maintain a good balance between these system performance indicators. The policy can ensure an acceptable average pick-up distance while maintaining a higher driver utilization rate and a higher matching rate, hence achieve the highest overall system performance score.

Table 6.4. Performance metrics comparison

policy	R_m	$D_p\ (R_P)$	R_{ult}	R
FR-500	0.21	**458.72 (0.85)**	**0.67**	0.558
FR-1000	0.61	896.67 (0.70)	0.52	0.601
FR-1500	0.75	1326.44 (0.56)	0.45	0.614
FR-2000	**0.79**	1678.38 (0.44)	0.27	0.546
Dynamic radius	0.71	993.34 (0.69)	0.63	**0.670**

In addition, Figure 6.15 compares the overall reward of the trained policy with baselines over a 24-hour period in Austin, based on the real supply-demand patterns retrieved from real operating data as illustrated in Figure 6.5. The figure shows that the dynamic matching radius policy outperforms the baselines in overall performance. This indicates that the trained policy successfully finds a balance between performance metrics, leading to improved overall system efficiency and user experience. This balance is achieved by adjusting the matching radius for each cell based on the real-time supply-demand relationship in each cell of the ride-hailing market. The figure also reveals that the performance of the baselines fluctuates over time, suggesting that the performance of the static matching radius is highly unstable when facing different levels of supply-demand imbalances in the ride-hailing system. In contrast, the trained policy not only performs better but also demonstrates better stability compared to the baselines. This highlights the trained policy's ability to reasonably and effectively adjust the matching radius to enhance system efficiency and user experience based on real-time supply-demand relationships. This finding underscores the potential of improving the ride-hailing matching system through the adoption of a dynamic matching radius policy. Additionally, the results affirm the effectiveness of the optimization framework using the ride-hailing matching simulator and the MDDPG algorithm developed in this study. Furthermore, this study's findings provide a strong foundation for future research in applying advanced reinforcement learning techniques to optimize various aspects of ride-hailing systems.

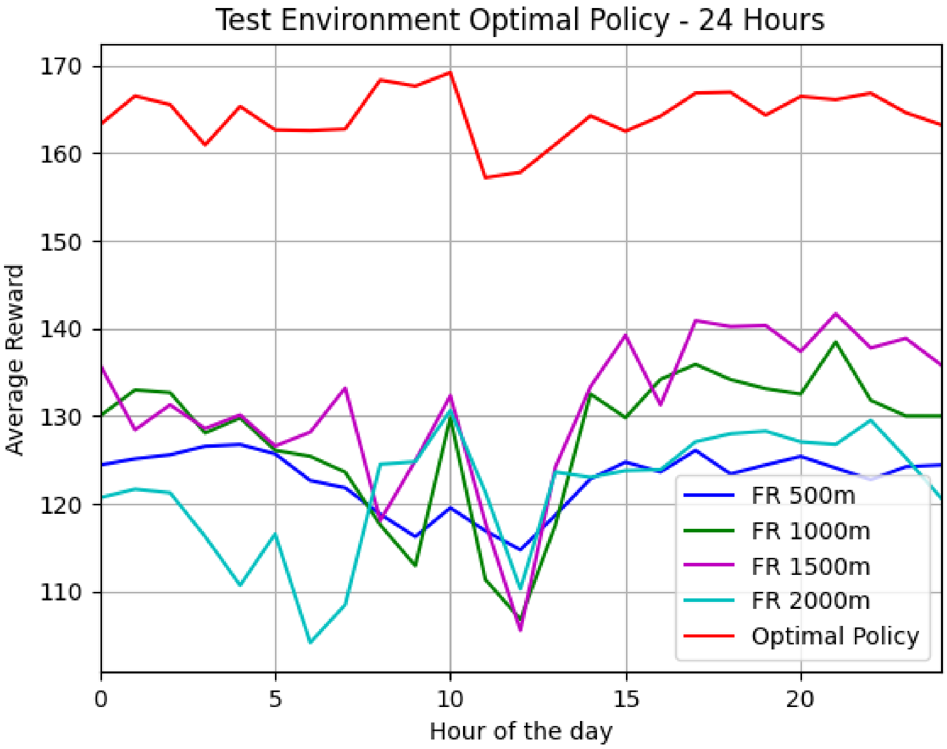


Figure 6.15. Performance plot for the trained policy compared to the baselines.

In conclusion, in the high-demand scenario, characterized by substantial increases in travel demand during events or peak hours, the policy continues to outperform the baselines but with slightly less optimal metrics than in the normal scenario due to system constraints. In the high-supply scenario and the normal scenario, the policy also performs better than the baselines by optimizing matching rate, driver utilization and maintaining low average pick-up distances, effectively using surplus resources (available drivers) and demonstrating good adaptability and efficiency over the baselines. The results in this case study suggest that the dynamic matching radius policy trained with the MDDPG algorithm is highly effective under various ride-hailing operational conditions. However, there is still room for improvement. Future research could focus on enhancing the policy's stability and robustness under more complex conditions that consider more variables in the ride-hailing system, such as the driver's rejection rate and different weather conditions. More advanced technologies could also be involved, such as integrating advanced predictive analytics, in order to better predict demand surges and investigate the possibility to combine the online matching module with the driver relocating module. Additionally, refining the reward function to balance immediate matching reward with long-term system stability could also help to reduce fluctuations and improve overall training performance. By further optimizing these aspects, the trained dynamic matching radius policy can enhance its effectiveness, making it even more reliable and efficient for real-world applications in the ride-hailing market. The continued development and refinement of this policy can help to elevate the operational efficiency and user experience in dynamic and varying supply-demand environments of the ride-hailing services.

6.6 Sensitivity Analysis of Reward Weights

Reward Weight Factors (w_1, w_2, w_3) are weights assigned to the components of the reward function (matching rate, average pick-up distance, driver utilization). The summed value of all weights is always equal to 1. Their different combinations represent different system optimization objectives. In real-world ride-hailing systems, these weights are determined by the goals of various stakeholders. In this experiment, to comprehensively assess the impact of these weight factors on the characteristics and performance of the trained policy, 13 different weight combinations are defined, categorized into balanced weight class and single weight class.

Weight combinations of the balanced weight class are designed to consider multiple performance metrics, testing the system's performance under comprehensive conditions typical of most operational scenarios. Taking three typical balanced weight combinations as an example. The fully balanced weight combination C_1^{bal} contains the averagely balanced weights between the matching rate, the average pick-up distance, and the driver utilization rate. This combination is designed as the normal weight combination. The policy is trained, tested, and validated under the normal scenario using this set of weight combinations. The matching priority combination C_2^{bal} focuses more on optimizing the matching rate. The rider priority combination C_4^{bal} pays more attention to optimizing the average pick-up distance. The weight for the average pick-up distance is slightly increased to hopefully reduce the average pick-up distance to a higher extent. The system priority C_6^{bal} combination adds value to the weight for the driver utilization rate. This causes a bias in the reward function toward the direction of a higher driver utilization rate. The weight combinations C_8^{bal} to C_{10}^{bal} respectively test the policies trained under each two completely balanced weights. The rest of the balanced weight combinations are designed to test how the trained policy is affected by minor changes in those weights.

The full matching rate, full pick-up distance, and full driver utilization weight combinations are classified into the single reward weight class. The performance and behaviour of the MDDPG algorithm to dynamically optimize the ride-hailing matching radius will be tested with these weight combinations. The effectiveness of the MDDPG algorithm and the validation of the trained policy, especially the effectiveness and stability of the algorithm under extreme optimization goals, are tested with this type of reward weight combination. This type of weight combination is an important part of the sensitivity analysis of this case study. In terms of specific weight combinations, the full matching rate combination C_1^{sin} focuses entirely on the matching rate. This extreme combination of weights focuses on improving the matching rate, which is crucial for evaluating the performance of the MDDPG algorithm in improving the overall matching rate during the period of extremely peak demand in Austin. At the same time, this combination of weights also verifies the rationality of the designed reward function. If the reward function is not properly designed, the agent will not be able to learn an effective policy under this extreme reward bias. In contrast, the full driver utilization combination C_3^{sin} gives the full weight on maximizing driver utilization rate. The driver utilization rate refers proportion of successful matched drivers among available drivers. It is essential for enhancing overall system operational efficiency.

The Table 6.5 shows the weight combinations used in this sensitivity analysis for weight factors of three reward components in the reward function. Those weights are specially designed for this case study of Austin.

Table 6.5. Weight combinations

combination	w_1	w_2	w_3
C_1^{bal}	0.4	0.4	0.2
C_2^{bal}	0.5	0.3	0.2
C_3^{bal}	0.5	0.2	0.3
C_4^{bal}	0.3	0.5	0.2
C_5^{bal}	0.2	0.5	0.3
C_6^{bal}	0.2	0.3	0.5
C_7^{bal}	0.3	0.2	0.5
C_8^{bal}	0.5	0.5	0.0
C_9^{bal}	0.5	0.0	0.5
C_{10}^{bal}	0.0	0.5	0.5
C_1^{sin}	1.0	0.0	0.0
C_2^{sin}	0.0	1.0	0.0
C_3^{sin}	0.0	0.0	1.0

The purpose of designing different weight combinations for sensitivity analysis is to test and evaluate the characteristics and performance of the dynamic matching radius policy under various reward weight combinations, and to observe the impact of changes in reward weights on the trained policy. The first goal is to test the flexibility and adaptability of the trained policy. This is measured by observing how the output action of the policy changes under different optimization objectives through various weight combinations. Robustness is another testing objective in the sensitivity analysis. If the algorithm can maintain a high level of performance when the weights change, it indicates that the algorithm has good robustness in optimization performance. In addition, sensitivity analysis also tests the robustness of the training. It is tested by observing whether small perturbations of the weights will lead to significant fluctuations in the learning or whether the trained policy can converge effectively. The generalization robustness of the method proposed in this study will also be verified. It refers to whether the method can maintain effective under different optimization objectives.

Regarding the expected results of the sensitivity test, for the balanced weight class combinations, it is expected that the policy can perform well across multiple reward metrics. Particularly, C_1^{bal} should achieve a good balance among the three optimization objectives, performing well on all three indicators, matching rate, average pick-up distance and driver utilization rate. In the combination C_2^{bal}, $C_4^{bal}$and C_6^{bal}, the policy should perform better in terms of matching rate, average pick-up distance and driver utilization rate, respectively. Compared to C_1^{bal}, the matching radius should increase and decrease appropriately in each of them to achieve higher rewards in the respective metrics. For the rest of the balanced weight combinations, the trained policy should be relatively stable and large fluctuations are not expected to be observed. For the single weight class combinations (C_1^{sin}, C_2^{sin}, C_3^{sin}), the trained policy is expected to perform optimally on the corresponding single reward metric. Although it may sacrifice other metrics, the reward for the metric with a weight of 1.0 should be significantly higher than the corresponding reward in the balanced weight class combinations. The policy is expected to provide matching radius concentrated at boundary values, such as consistently giving the maximum or minimum matching radius.

The training process of the single weight combination shows a slightly different pattern than the balanced weight combinations. By examining the loss plot shown in Figure 6.16, compared to the loss plot of the balanced weight combinations presented in Figure 6.9, it can be seen that the loss of the single weight combination converges to a smaller level much earlier. This is because the agent discovers that the optimal actions are concentrated at the boundary values shortly after training begins. As a result, the agent increasingly outputs boundary value actions during subsequent training.

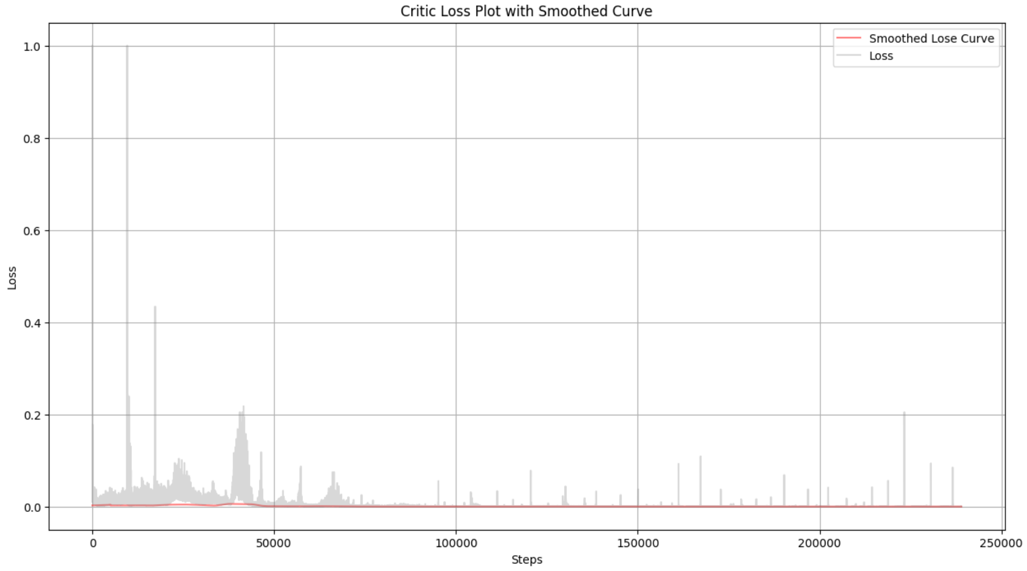


Figure 6.16. Critic loss plot for a single weight combination (matching rate).

The results of sensitivity analysis are shown in Figure 6.17. This figure shows the example of cell 4 (cell of the city center area). It illustrates the average value range (colored area in the plot) and the 90 percent confidence interval for the matching radius (lower and upper bound in the plot) obtained from policies trained with different reward weight combinations. These data were obtained under the normal scenario (balanced supply-demand condition), with each reward weight combination tested over 40 episodes. From the graph, it can be observed that under balanced weight combinations, the matching radius fluctuates within the range of 600-1400 meters. The system's decision in balancing the performance metrics of the matching rate, the average pick-up distance, and the driver utilization can be observed in the figure. For the C_2^{bal} combination, it is the balanced weight combination with a bias toward optimizing the matching rate. An increase in the matching radius can be found compared to that of C_1^{bal}. In order for the agent to enhance the overall matching rate reward, this observation is reasonable as a larger matching radius can bring a bigger matching pool, hence a higher probability for travel demands to be matched with available drivers. As the agent pays less attention to the driver utilization rate, the optimization goal becomes simpler between the matching rate and the average pick-up distance. This is the reason that the fluctuation of the matching radius becomes smaller. In contrast, the C_4^{bal} combination has a larger bias towards the average pick-up distance, and a significant decrease in the matching radius can be found in the plot compared to the previous weight combinations. As for the C_6^{bal} combination, it can be seen that the fluctuation becomes higher. By comparing the result with the single weight combination C_3^{sin}, which only focuses on optimizing the driver utilization rate, it can be concluded that optimizing the driver utilization rate as the single objective is a complex task. The driver utilization rate has a large variance between different ride-hailing supply-demand relationships. Only when it is optimized together with the other two performance indicators, and only when its weight is reasonably set, can a better optimization performance be achieved by the RL agent. The rest of the balanced weight combinations are used to test the impact of slight changes in reward weights on the trained policy. These combinations test the robustness of the policy to small changes in the optimization objective. The results of C_3^{bal}, C_5^{bal} and C_5^{bal} show that slight changes in weights cannot cause large fluctuations in the matching radius output by the trained policy. More specifically, take C_3^{bal} as an example. Compared with C_2^{bal}, the bias on optimizing the average pick-up distance is slightly reduced, the mean matching radius output is slightly increased, and the confidence interval does not change significantly, which is consistent with expectations. For those weight combinations with two completely equal weights of 0.5, the results meet the expectations. C_8^{bal} forms a relatively simple optimization goal, the agent only needs to find the balance between the matching rate and the average pick-up distance without considering the driver utilization rate. As a result, the fluctuation of the matching radius output by the trained policy is significantly reduced, and the confidence interval is narrower. There is a contradiction between the optimization of the matching rate and the driver utilization rate in C_9^{bal}, so the matching radius fluctuation increases and the confidence interval is wider. The two optimization goals of C_{10}^{bal}, the average pick-up distance and the driver utilization rate, have the same optimization direction, so the confidence interval is still relatively narrow. The output matching radius is significantly reduced, as a smaller matching radius ensures a shorter average pick-up distance, as well as a higher driver utilization rate. The results are consistent with expectations, indicating that the method proposed in this study and the RL algorithm used in this study have good robustness and reliability.

As for the weight type of single weight combinations, those weight sets focus solely on one of the ride-hailing performance metrics. The sensitivity analysis result shows that the trained policy with this type of weight combinations outputs a set of matching radiuses concentrated at boundary values to maximize reward in one of these metrics. The policy trained with the weighted combination of C_1^{sin} and C_2^{sin} constantly outputs the maximum matching radius and the minimum matching radius respectively. The single driver utilization combination C_3^{sin}, however, due to its dependency on both the matching radius and the exact locations of drivers and riders, reflects a higher uncertainty in assessing action value under the reward function with a fully biased driver utilization rate and the current observation definition. To deal with this issue, future studies could be carried out on the additional observation from the system, providing the agent with more information to better understand the ride-hailing environment. This topic of possible adjustments and next-step studies is further discussed in Chapter 7.

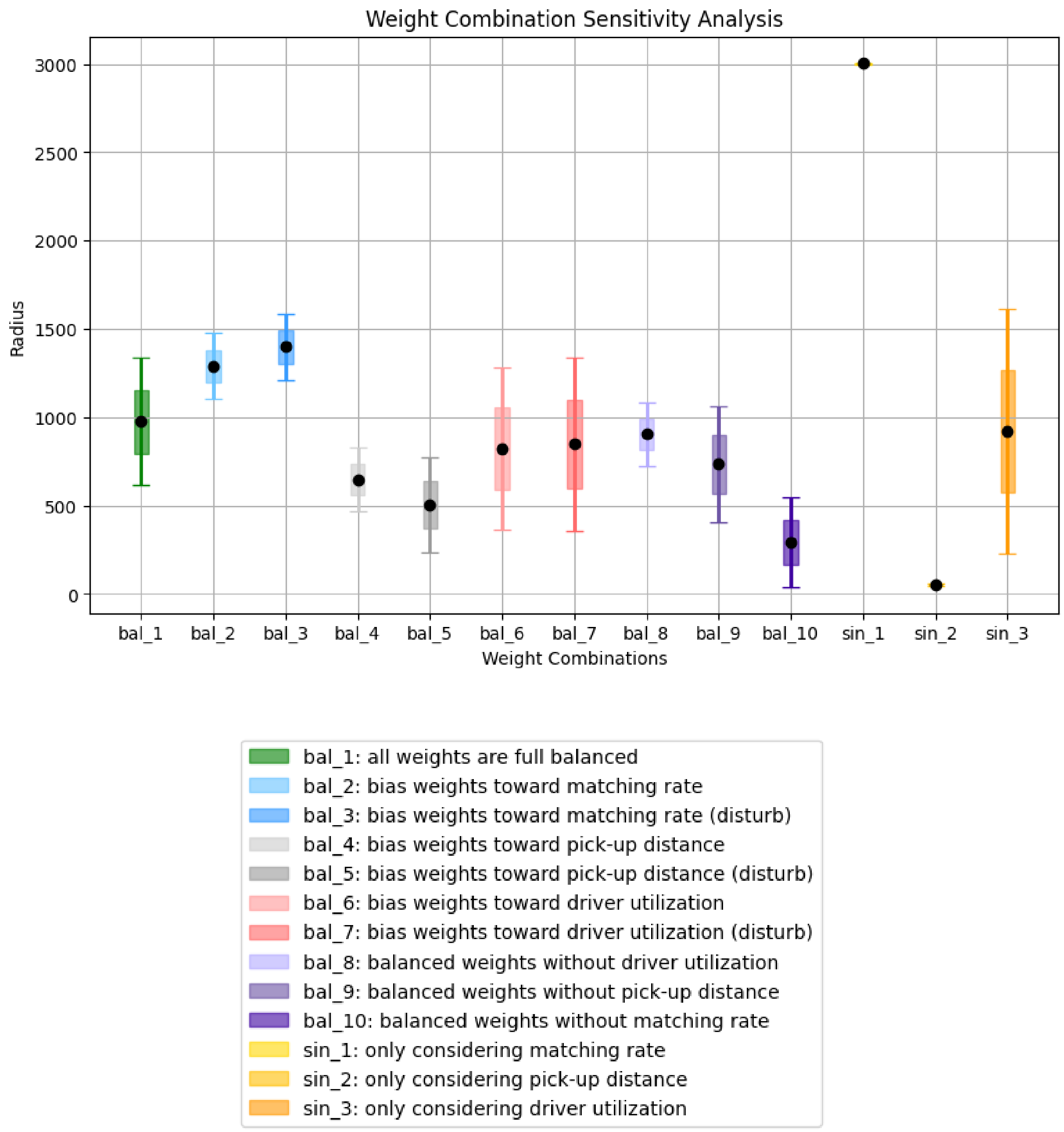


Figure 6.17. Average and 90% trust interval of obtained radius.

The average performance metrics and the average performance score of each weight combination tested over 40 episodes are listed in Table 6.6. The table shows the specific performance metrics of the policies trained under each weight combination. The performance metrics include the matching rate, average pick-up distance (the normalized performance score is in brackets), and driver utilization rate. The last column of the table is the average total weighted performance score. The specific weight of each performance metric is also listed in the table, right after each performance metric. The weight combinations that have zeros actually violate the principle of multitask optimization for the ride-hailing system. It is not reasonable to completely ignore one stakeholder in the ride-hailing system. However, these sets are still tested in order to demonstrate the validation of the proposed DDPG algorithm and the developed simulator under extreme setting.

Table 6.6. Sensitivity analysis results of performance metrics weights.

set	R_m	w_1	D_p (R_P)	w_2	R_{ult}	w_3	R
C_1^{bal}	0.71	0.4	993.34 (0.69)	0.4	0.63	0.2	0.670
C_2^{bal}	0.73	0.5	1131.27 (0.62)	0.3	0.50	0.2	0.651
C_3^{bal}	0.75	0.5	1236.41 (0.59)	0.2	0.48	0.3	0.637
C_4^{bal}	0.42	0.3	638.25 (0.79)	0.5	0.65	0.2	0.651
C_5^{bal}	0.36	0.2	616.32 (0.78)	0.5	0.66	0.3	0.660
C_6^{bal}	0.57	0.2	898.96 (0.70)	0.3	0.68	0.5	0.664
C_7^{bal}	0.59	0.3	911.58 (0.69)	0.2	0.67	0.5	0.649
C_8^{bal}	0.55	0.5	901.19 (0.70)	0.5	0.59	0.0	0.626
C_9^{bal}	0.49	0.5	892.93 (0.70)	0.0	0.65	0.5	0.577
C_{10}^{bal}	0.18	0.0	404.33 (0.86)	0.5	0.79	0.5	0.825
C_1^{sin}	0.80	1.0	1945.31 (0.35)	0.0	0.81	0.0	0.830
C_2^{sin}	0.03	0.0	47.12 (0.98)	1.0	0.87	0.0	0.980
C_3^{sin}	0.73	0.0	1175.90 (0.55)	0.0	0.67	1.0	0.665

The sensitivity analysis results are consistent with the expected results, proving the effectiveness and adaptability of the MDDPG algorithm in optimizing the matching radius of the ride-hailing matching system. The realism and effectiveness of the simulator developed in this study are also verified. First of all, this table demonstrates a validation of the proposed DDPG algorithm, the R is the overall reward of system performance. The reward value all outperform the baselines and can reach a high level. Between each of these weigh sets, it demonstrates the reason why the current weight set is selected. In weight sets that pay attention to all three of the performance metrics, the current weight set has the highest overall score. In the sets that only focus on two of the metrics, two of them performs worse than the selected set. The set C_{10}^{bal} is worth to be mentioned, although the R looks higher, the matching rate is not acceptable, which means this weight set is unreasonable. The reason behind this is that there is no trade-off between optimizing the matching radius and the driver utilization rate, a smaller matching radius can bring shorter distance and higher rate in the same time. So, this trained policy actually is a nearest matching, which is apparently not what this research wanted to study. The reason why this driver utilization rate is still needed is that this performance metric represents an important stakeholder in the ride-hailing system, the system operator. Together with the Figure 6.17, it can be seen that when only optimizing the driver utilization rate, the trained policy has a relatively higher uncertainty in outputting the actions. The matching rate, the average pick-up distance and the driver utilization rate form a solid triangle that has limitation and affection on each other in the ride-hailing system. It is the same as the real ride-hailing system as riders, drivers and system operators are three main stakeholders in the system. To Conclude, for balanced weight combinations, the trained policy manages to maintain good performance on multiple indicators, adjusting the matching radius to optimize the balance between the matching rate, the average pick-up distance, and the driver utilization rate. For a single weight combination, the policy is able to effectively focus on the optimization of the only performance metric and output a set of reasonable matching radius in a targeted manner. By adjusting the matching radius to the boundary value, good rewards were achieved in the ride-hailing simulator of Austin. The sensitivity analysis in this case study confirms that the policy can adapt to different optimization objectives that may exist in the ride-hailing market in Austin, verifying its applicability and effectiveness in a real operating environment. The result C_3^{sin} also meets the expectation, it indicates that optimizing the driver utilization rate as the single objective is a complex task. The driver utilization rate has a large variance between different ride-hailing supply-demand relationships. Only when it is optimized together with the other two performance indicators, a better optimization performance be achieved by the RL agent. The results also show that the method proposed in this study exhibits good robustness and stability. First, the results show that the algorithm can maintain a high level of performance when the weight setting changes. The change of reward weights does not lead to significant fluctuations in the trained policy or overall performance, it indicates that the algorithm has good robustness and stability in optimization performance under small disturbances. In addition, the reinforcement learning process of the method proposed in this study can converge effectively under different weight combinations, indicating that the training process has good robustness. Finally, the performance of the algorithm under different weight combinations is in line with expectations, indicating that the algorithm can be well generalized to a variety of conditions, and its generalization robustness is verified.

In summary, the sensitivity analysis will help researchers comprehensively evaluate the characteristics of the dynamic matching radius policy and verify its effectiveness and flexibility in practical applications. It can also provide readers with deeper insights into the behavioural differences of the obtained policy trained under different optimization objectives and offer valuable insights for the application of ride-hailing systems. On the other hand, this sensitivity analysis also helps readers better observe the performance differences of the MDDPG algorithm in handling dynamic matching radius optimization problems with different objectives and verify its stability and robustness under various conditions.

6.7 Results Discussion

In this case study, the proposed customized MDDPG algorithm was used to optimize the ride-hailing system in the city of Austin, Texas. The purpose of the optimization is to improve the system’s operating efficiency and user experience of the ride-hailing matching system from multiple aspects. In order to achieve this optimization goal, this case study designed several ride-hailing operational scenarios, corresponding to different supply and demand situations in the ride-hailing market in Austin at different times. When using the MDDPG algorithm to train the policy, good convergence was achieved as discussed in subsection 6.5. The trained policy can stably optimize the ride-hailing matching system in Austin in these scenarios. This subsection will discuss the optimization results of the ride-hailing matching system in Austin, as well as the logic of the actions output by the trained policy, and analyse the feasibility and effectiveness of the optimization method proposed in this study in actual scenarios.

The dynamic matching radius policy is trained under the supply-demand relationship of the normal scenario. The reason is that this scenario can provide the agent with relatively stable training data, which is conducive to the updating of the deep neural network (actor and critic). The policy trained in this scenario can also better dynamically adjust the matching radius to adapt to the fluctuating supply and demand relationship between drivers and travel demands in the ride-hailing market in Austin. This is also verified during the actual training process. In the normal scenario, the loss of the critic network changes more evenly and can be reduced to a lower level. The agent can also optimize the cumulative reward of each episode from -150 to about 160 by optimizing the policy. This reflects the good learning ability and learning efficiency of the proposed algorithm in the ride-hailing matching system. The trained matching radius policy also performs well in the test environment. In scenarios with different supply and demand conditions, the trained dynamic matching radius policy shows strong adaptability. Especially under extreme supply and demand relationships, the dynamic matching radius strategy significantly improves the overall performance of the matching system compared to the static baselines. For example, in the high-demand scenario, the strategy can dynamically adjust the matching radius to increase the overall matching rate as much as possible while ensuring an acceptable average pick-up distance. The trained policy improves the overall performance score of the system by about 20% compared to the best performance of the baselines. In the low-demand scenario, the trained matching radius can flexibly adjust the matching radius, reduce the driver's idle driving distance, and improve operational efficiency. This flexible adjustment capability enables the dynamic matching radius strategy to maintain high performance in various supply and demand conditions of the ride-hailing market.

More specifically, the trained policy increases or decreases the matching radius according to the changing trend of the supply and demand relationship between drivers and riders in each cell. When the number of available drivers in a cell is much larger than the number of travel demands, the cell constitutes an oversupplied local ride-hailing market. The corresponding action given by the strategy is a smaller matching radius, as shown in cell 7 in Figure 6.10 and Figure 6.11. A smaller matching radius is reasonable in this case because the probability of riders being successfully matched within a smaller range is higher. At this time, the optimization space for the matching rate is relatively smaller than other situations. By reducing the matching radius and optimizing the average pick-up distance, a higher overall system performance score can be obtained for the agent (system operator). This is also the policy learned by the agent in the learning process by continuously interacting with the ride-hailing matching simulation environment. In the real ride-hailing system, when the number of idle drivers is much larger than that of travel demands, a smaller matching radius will significantly reduce the pick-up waiting time of riders and improve the user experience. When the number of drivers in some cells is less than that of travel demands, an undersupplied ride-hailing market is formed locally in the ride-hailing matching system of Austin. In this case, the trained policy responsively increases the matching radius, as shown in cell 1 and cell 8 in Figure 6.7. By expanding the matching radius, the agent in this case focuses on increasing the probability of riders being successfully matched and achieves a higher overall system performance score by increasing the overall matching rate. In the real ride-hailing market, such a strategy is also reasonable. When the travel demand is far greater than the number of available vehicles, riders are more concerned about whether they can be successfully matched, and are relatively unconcerned about the longer waiting time. This is consistent with the matching radius adopted by the trained policy when demands exceed supplies. It proves that the trained policy can reasonably and effectively optimize the ride-hailing matching system in a targeted manner. A more normal situation is that when the number of drivers in some cells is not much different from that of travel demands, this indicates that these areas are in a relatively balanced local ride-hailing market, that is, a relatively balanced supply and demand relationship. In this case, the matching radius output by the trained policy is relatively maintained at a medium to small level. The specific value of the matching radius also fluctuates with the value of the supply and demand relationship for the agent (system operator) to find a good balance between the matching rate, the average pick-up distance, and the driver utilization rate to achieve a higher system performance score, as shown in cell 4 in Figure 6.11. This policy helps ride-hailing system operator to find a more ideal balance between system efficiency and user experience, ensuring a relatively high matching rate, controlling the pick-up distance within a reasonable range, and maintaining the efficient operation of the ride-hailing matching system. In actual ride-hailing systems, when the number of drivers is close to the number of riders, dynamically adjusting the matching radius can avoid oversupply or undersupply in the long run and maximize the system's operating efficiency.

In summary, the trained dynamic matching radius policy has shown good feasibility, flexibility, and applicability in Austin’s ride-hailing market. Through continuous interactions and learnings with the simulated ride-hailing matching environment, the ability to dynamically adjust the matching radius according to the changing supply-demand relationship between drivers and riders is achieved. In the actual ride-hailing system, the advantages of such a dynamic policy are obvious. In the case of oversupplied drivers, a smaller matching radius can significantly reduce the waiting time of passengers, thereby improving the user experience and system efficiency. On the contrary, when there is an insufficient supply of drivers, an appropriate increase in the matching radius can effectively improve the matching rate of riders, hence improving the user experience. In normal scenarios, this policy can also reasonably optimize the overall system efficiency of the ride-hailing matching system. The shortcomings of the proposed optimization method and its application potential in the ride-hailing markets of other cities are discussed in the next chapter.

7

Conclusion

In recent urban public transportation modes, ride-hailing has become one of the primary choices for urban citizens. However, the supply-demand imbalance between drivers and riders always causes extra stress to the ride-hailing system. It often faces challenges of uneven supply and demand relationships both in time and in space. Therefore, it is necessary to optimize the ride-hailing matching system. Traditional mathematical optimization and static optimization methods have certain limitations in the long-term effects of strategies. Therefore, using Reinforcement Learning to optimize the ride-hailing matching system is a better choice.

The goal of this study is to optimize the overall operating efficiency and user experience of the ride-hailing system. In order to achieve this optimization goal, this study has made contributions in the following three aspects:

- **Markov Decision Process:** Since the ride-hailing matching problem is relatively complex, this study forms it as a Markov Decision Process and conducts following studies based on it.

- **Ride-hailing Simulator:** Due to the complexity and uncertainty of the actual ride-hailing system, it is unlikely to directly use the original operation data from the actual ride-hailing system to train the agent. In order to achieve the optimization purpose, a ride-hailing matching simulator is developed to simulate the ride-hailing matching system in a targeted manner.

- **Deep Reinforming Learning:** RL algorithms have been proven to be effective in optimizing ride-hailing systems. This study selects a suitable Reinforcement Learning method and uses the MDDPG algorithm to optimize the matching radius for the ride-hailing matching system.

In terms of optimization performance, this study designs different experimental scenarios for a case study of the ride-hailing market in Austin, and analyzes the effectiveness and applicability of the proposed method under a variety of different driver-rider supply-demand relationships. The results show that the method proposed in this study can effectively balance the three performance indicators of the ride-hailing matching system: matching rate, average pick-up distance, and driver utilization rate. The policy trained in this study can optimize the overall operating efficiency of the ride-hailing matching system to a certain extent and improve the user experience. However, the method proposed in this study still has some limitations. This chapter will discuss and summarize these limitations and possible future improvements.

7.1 Limitations

The optimization method proposed in this study to optimize the matching radius of the ride-hailing matching system with the MDDPG algorithm has achieved good performance and results. However, there are still some limitations in the implementation of the proposed method in real ride-hailing markets apart from the city of Austin. The limitations of this method mainly come from the simplifications and assumptions made when simulating the actual ride-hailing market to improve the simulation effect, improve the simulation efficiency, and ensure the stability of the Reinforcement Learning algorithm. This section will discuss these assumptions and simplifications, as well as the limitations brought by them of the method proposed in this study.

First of all, this study simplified the ride-hailing market when developing the ride-hailing simulator. It is assumed that it consists of multiple independent, equally-sized grid areas named cells. Each of these cells has a fixed geographical boundary. The matching process of travel demands and drivers in each cell is independent, but the matching can be carried out across the cell boundaries. This assumption simplifies the matching process and reduces the complexity of the system. However, real cities are not composed of equally-sized areas, and the demand and supply relationship of riders and drivers between regions may vary greatly. This assumption may allow the model to perform better in training, but a worse performance can also be caused in actual applications in real ride-hailing markets. At the same time, the simulator defines the matching radius of all ride demands in the same cell as the same, while the matching radius between different cells can be different. The matching time-window in each cell remains the same in all time-steps. The length of each matching time-window in the actual ride-hailing system can be adjusted dynamically, and many studies have already been carried out to discuss this topic. The ideal matching radius also needs to be adjusted according to the specific circumstances of each travel demand. This approach may oversimplify the real dynamics of the actual ride-hailing system. The urban traffic environment is inherently complex, and the assumptions made to optimize the simulator performance may result in the trained policy not being able to fully consider these complex scenarios.

Another assumption is about the behavior of riders and drivers. It is assumed that riders will not cancel orders after successful matching, and drivers will not refuse orders after matching with travel demands. For specific travel demands, the simulator does not consider the end location of each travel demand and assumes that the order weight in the matching system is only related to the pick-up distance, not the length of the ride. This means that all rides have the same price, and their value and weight are only negatively correlated with the pick-up distance. This assumption ignores the difference in importance between long-distance rides and short-distance rides in actual operations, which creates a difference between the simulator and the actual ride-hailing system. It may cause the trained policy to be biased when applied in the actual ride-hailing market. Although the assumption that the ride destination is not considered in the model reduces the complexity of the model and helps reduce the size of generated data, it may deviate from the actual distribution of vehicles in the actual ride-hailing system. Although these assumptions about rider and driver behavior simplify the complexity of the matching system, in actual operation, the behavior of riders and drivers is highly uncertain, which may also lead to model deviations. This assumption also ignores the individual differences between travel demands and driver behavior, which may affect the accuracy of the matching policy and cause certain limitations in the specific implementation of this approach.

When simulating the behavior of unmatched drivers looking for travel demands, a Gaussian distribution (normal distribution) is used for sampling. This method assumes a certain degree of predictability and consistency in driver behavior. However, this assumption can cause a large deviation in predicting driver behaviours in the real world. Theoretically, the Gaussian distribution is a symmetric distribution that is often used to describe fluctuations in natural phenomena or predictable behaviours that are concentrated around a certain value (mean). However, the behavior of drivers is not that predictable. In looking for riders, the driver’s behavior may be affected by many factors and real-time circumstances, such as personal preferences, road conditions, fuel conditions, time, weather conditions, and so on. The variability and unpredictability of driver’s behavior with the impact of these factors are difficult to be fully captured by a Gaussian distribution. Driver behavior in the real world is often more complex and unpredictable. For example, in some cases, drivers may prefer to stay in a specific area in the hope of being matched with travel demands more quickly. This is because these areas tend to have higher travel demand, such as transportation hubs such as airports or train stations. In other cases, drivers may prefer to move to a certain area or certain specific locations. For example, at night, drivers may be more inclined to move to the area where the driver's home is located. Therefore, the use of a Gaussian distribution to capture and simulate the behavior of drivers may be oversimplified, resulting in a deviation between the simulated behavior and the actual situation. This deviation will not only affect the accuracy of the trained policy and reduce the optimization effect of the ride-hailing matching module, but also have an adverse effect on the ride-hailing dispatching system in practical applications.

During the simulation process, in order to reduce the impact of irrelevant variables on the optimization of the ride-hailing matching system, it is assumed that all other environmental factors (such as traffic conditions and weather) remain unchanged during the simulation. This assumption is intended to control these complex and irrelevant variables so that the simulation results can more directly reflect the effects of the input radius to the ride-hailing simulator. However, these factors in the urban traffic environment often change in reality, and this treatment method may reduce the model's applicability in practical applications. First, traffic conditions are changing in real-time at different times and locations, and the efficiency of the ride-hailing matching system will also be affected. For example, the traffic flow during peak hours and off-peak hours varies greatly, and temporary road construction, traffic accidents, and other events can also cause traffic congestion in a short period of time. Ignoring these changes and assuming that traffic conditions remain the same may lead to inaccurate calculations of the model for passenger waiting time and driver pick-up distance, thereby reducing the optimization effect of the method proposed in this study on the ride-hailing matching system. Secondly, weather factors are also important variables that affect travel behavior. Severe weather (such as heavy rain, heavy snow, etc.) will not only lead to a decrease in road capacity but also significantly change passengers' travel needs and drivers' willingness to accept orders. For example, on rainy days, passengers may be more inclined to choose ride-hailing travel, while drivers may reduce the number of trips due to slippery roads. If these weather changes are ignored in the simulation and assumed to be unchanged, the prediction and optimization effects of the model under severe weather conditions will be greatly reduced. Although the influence of irrelevant factors on the simulation results can be reduced by controlling variables, in actual applications, this simplified treatment of assumptions may not accurately reflect the complex and changeable real environment, thereby affecting the actual optimization effect of the ride-hailing matching system. Therefore, future research and model design should consider how to better incorporate these dynamically changing environmental factors to improve the accuracy and practicality of the proposed approach and trained policy. These contents will be discussed in the subsection of recommendations for future research.

In addition, the implementation of the proposed method also faces a series of challenges, which themselves constitute limitations. Most importantly, designing the reward function poses a challenge and a source of limitation. Whether the reinforcement learning model can successfully converge and its learning effect depends largely on the quality of the reward function. In the reward function, the agent obtains the real-time weighted performance score as the immediate reward in the ride-hailing matching system, but this may oversimplify the actual impact of matching decisions on the long-term profitability of the system. The design of the value function also needs to be carefully considered because the design of the value function directly affects the learning effect of the reinforcement learning agent.

7.2 Method Adaptability

The policy trained by the method proposed in this study are highly applicable and can adapt to different supply-demand scenarios and different optimization goals. However, when adapting and applying it in other cities, it is necessary to fully consider the limitations mentioned above and make appropriate adjustments to parameters. Only on the basis of fully understanding the optimization goals of ride-hailing systems in different cities and reasonably selecting parameters to reflect the actual operation of the ride-hailing market, can the proposed optimization method effectively improve the operating efficiency of the actual ride-hailing system and optimize the user experience.

The successful application of this method from Austin to the ride-hailing market in other cities depends on multiple factors. The most important of these is the adjustment of the parameters of the reward function, which plays a vital role in the effectiveness and convergence of the reinforcement learning model. The quality of the reward function directly affects the model's ability to learn and optimize matching decisions in the actual ride-hailing system. Appropriate parameter settings can better reflect the optimization goals of the local ride-hailing system. For example, if the matching success rate of the ride-hailing system in a certain city is low, it is necessary to increase the weight of the matching rate in the overall performance score to focus on optimizing the matching rate.

The geographical specificity of different cities and the different supply-demand relationships in the ride-hailing market mean that in-depth evaluation and adjustment are needed when adapting the proposed method to different cities. For the developed simulator, data needs to be acquired from local real ride-hailing system operation data. The real local ride-hailing system operation data can make the simulator better reflect the actual market and operation environment of the local ride-hailing system, so that the algorithm can optimize it in a targeted manner. The availability and quality of real-world ride-hailing data are crucial to the effectiveness and practicality of the trained policy. If incomplete or biased data is used, the optimization effect of the proposed method on the local ride-hailing matching system may be reduced. On the other hand, the traffic characteristics, population density, traffic flow, and demand-supply relationship of different cities may be significantly different. For example, large cities may have higher demands and supplies, while small cities are the opposite. Therefore, adjusting the supply-demand relationship parameters in the simulator accordingly is also the key to successfully adapting the method proposed in this research to the actual ride-hailing market.

7.3 Answer to the Research Questions

This section answers the research questions raised in this study.

RQ: In a ride-hailing system, when the matching time window is fixed, what is the policy to determine the optimal matching radius for riders and drivers with respect to a higher matching rate, shorter pick-up distance, and a higher driver utilization in different urban areas with different supply-demand relationships?

Answer to RQ: This study develops a dynamic matching radius optimization method for the ride-hailing matching system, which specifically optimizes the matching rate, pick-up distance, and driver utilization. A policy is trained using the MDDPG algorithm. This policy can dynamically output the optimal matching radius based on the real-time driver-rider supply-demand relationships in different urban areas of the ride-hailing system. The effectiveness of this method has been verified in a case study based on the city of Austin. The overall structure of the method is shown in Figure 3.1.

SRQ 1: What are the current methods for determining the matches between riders and drivers in ride-hailing systems, and what challenges do they face?

Answer to SQR 1: The existing methods include the nearest matching method and the batched matching method. At present, the optimization of these matching methods mainly focuses on static planning, optimization based on mathematical models, and optimization of matching time-windows. The challenges encountered by these studies include suboptimal matching due to fixed parameters, inefficiency in handling changing supply-demand dynamics, and scalability issues. At the same time, these methods are usually limited in adapting to the changing supply-demand dynamics, and may not be able to optimize the operating efficiency and user experience of the ride-hailing system at the same time. When optimizing the ride-hailing matching system, the evaluation of an optimization method needs to consider the calculation efficiency, optimization performance, real-time processing capability of the optimization method, and ability to adapt to the dynamic ride-hailing market. A good policy can adjust the matching radius in a targeted manner according to the supply-demand patterns of the ride-hailing market in real-time to achieve the purpose of optimizing the overall system efficiency and user experience.

SRQ 2: How effective is the ride-hailing simulator in replicating real-world scenarios for testing the RL-based matching radius optimization?

Answer to SQR 2: The ride-hailing simulator uses functions such as demand distribution, driver behaviour model, and travel demand generation model to replicate real-world conditions. The main parameters include reward performance weight, travel demand generation rate, and supply-demand relationship ratio. In response to changes in the urban environment, in order to control uncertainties and irrelevant variables in the ride-hailing matching system, a series of simplifications and assumptions were made for the ride-hailing system during simulator development. These simplifications and assumptions and their impact are discussed in Section 7.1. A performance score of the ride-hailing matching system is used to evaluate the operating efficiency and user experience of the ride-hailing system under an optimized matching radius. The evaluation indicators included in this score include matching rate, average pick-up distance, and driver utilization rate. The ride-hailing simulator effectively replicates real-world scenarios by simulating real-time supply and demand relationships, driver behaviour, and simulating the generation of travel demand. It uses matching efficiency and user experience-related indicators to evaluate the effectiveness and performance of the optimization method proposed in this study.

SRQ 3: How can the proposed RL framework be implemented to address the challenges in optimizing the matching radius for ride-hailing systems?

Answer to SQR 3: Reinforcement Learning is suitable for the optimization of the ride-hailing matching systems because it can adapt to dynamically changing supply-demand relationships and optimize long-term benefits in a dynamic system. This advantage is beyond the reach of traditional dynamic planning algorithms and mathematical optimization algorithms. Reinforcement Learning algorithms are able to handle complex decision-making processes and continuously improve the policy through continuous interactive learning and experience replay mechanisms. In addition, Reinforcement Learning can handle continuous action spaces and achieve more refined policy adjustments, thereby improving matching efficiency, shortening pick-up distances, and increasing driver utilization rate in the dynamic ride-hailing environment. Multi-replay-buffer Deep Deterministic Policy Gradient (MDDPG) is the most suitable RL algorithm for this study. This RL algorithm is suitable for problems with continuous action spaces. MDDPG combines policy gradient methods with deep learning to optimize decisions through deterministic strategies. MDDPG uses a deterministic policy network to select the best action in a given state, rather than sampling discrete actions from a distribution like DQN. This algorithm also uses the experience replay buffer in DQN to store past experience to break the correlation between samples and improve learning efficiency. MDDPG can handle continuous action spaces, making it particularly suitable for the matching radius optimization of the ride-hailing system that needs to be adjusted dynamically. It can dynamically adjust the matching radius according to the real-time supply-demand relationships to improve matching efficiency. It can handle the complex and dynamic environment of the ride-hailing system, including the supply-demand relationship in different urban areas. The key parameters of the MDDPG algorithm are mainly the hyperparameters of the algorithm, including learning rate, decay coefficient, experience replay pool size, etc. Their role and impact are discussed in detail in the case study section. The performance of the policy trained by the MDDPG algorithm is better than the baselines in a variety of different supply-demand scenarios. This policy can find a good balance between the system performance indicators to maximize system operation efficiency and user experience.

7.4 Future Research and Outlook

The method proposed in this paper shows its high adaptability under different driver and passenger supply and demand scenarios and optimization objectives through a case study in Austin. However, when applying it to other cities, it is necessary to fully consider the above limitations and adjust the parameters appropriately. In order to achieve computational efficiency and optimization effect, this paper makes some trade-offs in the simulation degree of the model. This study has made a series of simplifications to the actual ride-hailing market and also proposed some targeted assumptions, so that factors such as weather are not considered in this study. In view of the shortcomings and limitations of this study, this section will discuss future research work and prospects from the following specific aspects.

The simulator developed in this study divides the ride-hailing market into cells. The cells used in the case study is relatively rough, and only 9 rectangular cells of equal area are used. Future research needs to increase the number of cells and divide the study area according to practical circumstances. It is suggested to consider using cells of different sizes or different shapes, for example, according to the attributes of the area, such as commercial areas, transportation hub areas and residential areas. This method of dividing cells can make the division of the area more distinctive and better adapt the matching radius to different ride-hailing demand patterns of different type of urban regions. In this way, the method proposed in this study can more accurately capture the supply-demand characteristics of different regions, thereby improving the accuracy and effectiveness of the trained policy. It is helpful to increase the applicability of this method.

On the other hand, this study mainly relies on existing data, which is time-sensitive and may not accurately reflects the future supply-demand relationship of the ride-hailing market. Further research is needed on the better prediction method of future ride-hailing demand-supply relationships. Introducing other advanced technologies, such as deep learning, big data cloud and blockchain, can better predict the supply-demand relationship of the ride-hailing market in time and space. It can improve the realism of the simulator used in this study. By deeply analysing historical data, combined with accurate prediction of future supply-demand patterns, a more accurate supply-demand model can be established. Such a model can provide more reliable data support for the developed simulator and improve the overall optimization effect of the system. In practical applications, real ride-hailing market data is an important guarantee for the effectiveness of optimization strategies. Future research should also focus on obtaining and utilizing higher-quality real-time data to further enhance the realism and applicability of the simulator. Future research can cooperate with ride-hailing platforms to obtain more detailed operational data, so as to more accurately reflect the actual operation of the market and adjust the optimization strategy in a targeted manner.

In addition, more in-depth research on the behaviour of drivers and riders is needed in the future. In this study, the driving behaviour of drivers and the ride-hailing behaviour of riders are simplified, ignoring the randomness and heterogeneity of their behaviour. Future research needs conduct more detailed analysis on the driver's order decision, route selection, and service willingness (rejection probability and rejection reason) and other behaviours. Modelling driver behaviour with a more realistic random model can better simulate the real situation and improve the realism, adaptability and effectiveness of the proposed method in this study. At present, there have been modelling analyses of pedestrian behaviour, and these studies can serve as a reference for future research in this aspect.

In summary, by increasing the sophistication of urban divisions, applying more accurate prediction models, and conducting more in-depth research on behavioural modelling, the optimization effect of the matching radius of the ride-hailing matching system can be further improved. These future studies and improvements will help to apply the methods proposed in this study to different ride-hailing markets, to meet the specific needs of different cities, and to improve the operational efficiency and user experience of ride-hailing systems in a targeted manner. Future research will continue to improve the methods proposed in this study in these directions, in order to achieve a more efficient and intelligent ride-hailing matching system.

References

[1] Sikder S. (2019). Who Uses Ride-Hailing Services in the United States. Transportation Research Record: Journal of the Transportation Research Board, 2673(12), 40-54.
https://doi.org/10.1177/0361198119859302

[2] Levine S., Kumar A., Tucker G. (2020). Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. Computer Science, Education, 2005.01643.
https://arxiv.org/abs/2005.01643

[3] Deisenroth M., Rasmussen C. (2011). PILCO: A Model-Based and Data-Efficient Approach to Policy Search. ICML'11: Proceedings of the 28th International Conference on International Conference on Machine Learning, Pages 465–472.
http://www.icml-2011.org/papers/323_icmlpaper.pdf

[4] Fujimoto S., Meger D., Precup D. (2019). Off-Policy Deep Reinforcement Learning without Exploration. 36th International Conference on Machine Learning, PMLR 97:2052-2062.
http://proceedings.mlr.press/v97/fujimoto19a.html

[5] Wu T., Wang S., Wang L., Tang X. (2022). Contribution of China’s ride-hailing services to its 2050 carbon target: Energy consumption assessment based on the GCAM-SE model, 160, 112714.
https://doi.org/10.1016/j.enpol.2021.112714

[6] Feng Y., Niazadeh R., Saberi A., Saberi A. (2023). Two-Stage Stochastic Matching and Pricing with Applications to Ride Hailing. Operations Research.
https://doi.org/10.1287/opre.2022.2398

[7] Xu Z., Li Z., Guan Q. (2018). Large-Scale Order Dispatch in On-Demand Ride-Hailing Platforms: A Learning and Planning Approach. Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. Pages 905–913.
https://doi-org.tudelft.idm.oclc.org/10.1145/3219819.3219824

[8] Wang Y., Tong Y., Long C., Xu P. (2019). Adaptive dynamic bipartite graph matching: A reinforcement learning approach. 2019 IEEE 35th International Conference on Data Engineering (ICDE)’, IEEE, pp. 1478–1489.
https://ieeexplore-ieee-org.tudelft.idm.oclc.org/8731455

[9] Wang J., Lampert B. (2014). Improving Taxi Revenue with Reinforcement Learning. Stanford University. CS229, Project 2014.
https://cs229.stanford.edu/proj2014/

[10] Mao C., Liu Y., Shen Z. (2020). Dispatch of autonomous vehicles for taxi services: A deep reinforcement learning approach. Transportation Research Part C: Emerging Technologies, 115, 102626.
https://doi.org/10.1016/j.trc.2020.102626

[11] Liu Y., Wu F., Lyu C., Li S., Ye J., Qu X. (2022). Deep dispatching: A deep reinforcement learning approach for vehicle dispatching on online ride-hailing platform. Transportation Research Part E: Logistics and Transportation Review, 161, 102694.
https://doi.org/10.1016/j.tre.2022.102694

[12] Wang Z., Qin Z., Tan X., Ye J., Zhu H. (2018). Deep reinforcement learning with knowledge transfer for online rides order dispatching, in 'International Conference on Data Mining', IEEE.
https://doi-org.tudelft.idm.oclc.org/10.1109/ICDM.2018.00077

[13] Holler J., Vuorio R., Qin Z., Tang X., Jiao Y., Jin T., Singh S., Wang C., Ye J. (2019). Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem. 2019 IEEE International Conference on Data Mining (ICDM)', Institute of Electrical and Electronics Engineers, Washington, DC, pp. 1090–1095.
https://ieeexplore-ieee-org.tudelft.idm.oclc.org/stamp/stamp.jsp?tp=&arnumber=8970873

[14] Ke J., Xiao F., Yang H., Ye J. (2022). Learning to Delay in Ride-Sourcing Systems: A Multi-Agent Deep Reinforcement Learning Framework. IEEE Transactions on Knowledge and Data Engineering, 34(5), 2280-2292.
https://doi.org/10.1109/tkde.2020.3006084

[15] Qin G., Luo Q., Yin Y. (2021). Optimizing matching time intervals for ride-hailing services using Reinforcement Learning. Transportation Research Part C Emerging Technologies. Volume 129, 103239.
https://doi.org/10.1016/j.trc.2021.103239

[16] Jin J., Zhou M., Zhang W., Li M. (2019). Co-Ride: Joint Order Dispatching and Fleet Management for Multi-Scale Ride-Hailing Platforms. 28th ACM International Conference. Pages 1983 – 1992.
https://doi-org.tudelft.idm.oclc.org/10.1145/3357384.3357978

[17] Li M., Qin Z., Jiao Y., Yang Y., Wang J., Wang C., Wu G., Ye J. (2019). Efficient Ride-hailing Order Dispatching with Mean Field Multi-Agent Reinforcement Learning. The World Wide Web Conference (WWW '19). 983–994.
https://doi-org.tudelft.idm.oclc.org/10.1145/3308558.3313433

[18] Chen X., Bai S., Wei Y., Jiang H. (2023). How income satisfaction impacts driver engagement dynamics in ride-hailing services. Transportation Research Part C. Volume 157, 104418.
https://doi-org.tudelft.idm.oclc.org/10.1016/j.trc.2023.104418

[19] Alejandro H., Wesley M. (2019). The impact of ride hailing on parking (and vice versa). Journal of Transport and Land Use, Volume 12, No. 1, 39.
https://www.jstor.org/stable/26911261

[20] Yan C., Zhu H., Kuroko N., Woodard D. (2019). Dynamic pricing and matching in ride-hailing platforms. Naval Research Logistics, Volume 67, Issue 8, Pages 705-724.
https://doi.org/10.1002/nav.21872

[21] Zha L., Yin Y., Yang H. (2016). Economic analysis of ride-sourcing markets. Transportation Research Part C: Emerging Technologies, Volume 71, Pages 249-266.
https://doi-org.tudelft.idm.oclc.org/10.1016/j.trc.2016.07.010

[22] Qin Z. (Tony), Zhu H. (2022). Reinforcement Learning for Ride-hailing: An Extended Survey. Transportation Research Part C: Emerging Technologies, Volume 144, 103852.
https://doi-org.tudelft.idm.oclc.org/10.1016/j.trc.2022.103852

[23] Yang H., Qin X., Ke J., Ye J. (2020). Optimizing matching time interval and matching radius in on-demand ride-sourcing markets. Transportation Research Part B: Methodological, Volume 131, Pages 84-105.
https://doi-org.tudelft.idm.oclc.org/10.1016/j.trb.2019.11.005

[24] Liu Y., Jia R., Ye J., Qu X. (2022). How machine learning informs ride-hailing services: A survey. Communications in Transportation Research, Volume 2, 100075.
https://doi-org.tudelft.idm.oclc.org/10.1016/j.commtr.2022.100075

[25] Özkan E., Ward A.R. (2020). Dynamic Matching for Real-Time Ride Sharing. Stochastic Systems, 10(1), 29-70.
https://doi.org/10.1287/stsy.2019.0037

[26] Gao J., Li X., Wang C., Huang X. (2020). Learning-based open driver guidance and rebalancing for reducing riders' wait time in ride-hailing platforms. In 2020 IEEE International Smart Cities Conference (ISC2), pp. 1-7.
http://dx.doi.org/10.1109/ISC251055.2020.9239059

[27] Zhan X., Szeto W.Y., Shui C.S., Chen X. (2021). Transportation Research Part E: Logistics and Transportation Review. Volume 150, 102124.
https://doi-org.tudelft.idm.oclc.org/10.1016/j.tre.2020.102124

[28] Feng S., Ke J., Xiao F., Yang H. (2022). Approximating a ride-sourcing system with block matching. Transportation Research Part C: Emerging Technologies, Volume 145, 103920.
https://doi-org.tudelft.idm.oclc.org/10.1016/j.trc.2022.103920

[29] Liu Y., Wu F., Lyu C., Li S., Ye J., Qu X. (2022). Deep dispatching: A deep Reinforcement Learning approach for vehicle dispatching on online ride-hailing platform. Transportation Research Part E: Logistics and Transportation Review, 161, 102694.
https://doi-org.tudelft.idm.oclc.org/10.1016/j.tre.2022.102694

[30] Feng G., Kong G., Wang Z. (2021). We Are on the Way: Analysis of On-Demand Ride-Hailing Systems. Manufacturing &Amp; Service Operations Management, 23(5), 1237-1256.
https://doi.org/10.1287/msom.2020.0880

[31] Arulkumaran K., Deisenroth M., Brundage M., Bharath A. (2017). Deep Reinforcement Learning: A Brief Survey. IEEE Signal Processing Magazine, 34(6), 26-38.
https://doi.org/10.1109/msp.2017.2743240

[32] Terrell G. R., Scott D. W. (1992). Variable Kernel Density Estimation. The Annals of Statistics, 20(3), 1236–1265.
http://www.jstor.org/stable/2242011

[33] Wiewiora, Eric (2003). Potential-based shaping and Q-value initialization are equivalent. Journal of Artificial Intelligence Research 19, 205-208.
https://www.jair.org/index.php/jair/article/download/10338/24713/

[34] Laud A. D. (2004). Theory and application of reward shaping in reinforcement learning. Ph.D. Dissertation, University of Illinois, Urbana-Champaign IL, USA.
https://www.ideals.illinois.edu/items/10802

[34] Miller T. (2022). Mastering Reinforcement Learning. Web Educational Book, The University of Queensland.
https://gibberblot.github.io/rl-notes/single-agent/reward-shaping.html

[35] Miao F., Han S., Lin S. (2016). Taxi Dispatch with Real-Time Sensing Data in Metropolitan Areas: A Receding Horizon Control Approach. IEEE Transactions on Automation Science and Engineering, 13(2), 463-478.
https://doi.org/10.1109/tase.2016.2529580

Appendix

Appendix A. Simulator Working Flow

Let T denotes the maximum number of time-steps in an episode. Let S_t denotes the state, A_t denotes the action and R_t denotes the reward at time-step t. Let r_t denotes the matching radius set re-scaled from the action, r_t^i is the matching radius for cell i at time-step t. Let N_t^r denotes the total number of riders (travel demands), N_t^d denotes the number of available drivers (driver supplies) at time-step t. Let d_{ij} denotes the distance between rider i and driver j. The working flow of the ride-hailing simulator developed in this study is shown below.

Algorithm 2 Ride-hailing Simulator

```
Input: the action $A_t$ at time-step $t$
Output: next-state $S_{t+1}$ and reward $R_t$
initial matching pool $P_t$
re-scale the action $A_t$ to matching radius $r_t$
for $rider = 1: N_t^r$, do:
    assign $r_t^i$ to the rider based on which cell the rider is in
    for driver $= 1: N_t^d$, do:
        calculate distance $d_{ij}$ between rider and driver
        If $d_{ij} > r_t^i$:
            continue
        If $d_{ij} \leq r_t^i$:
            store {rider, driver, $d_{ij}$} in the matching pool $P_t$
        end for
end for
use Hungarian algorithm to find matches $M_t$ within $P_t$
calculate matching rate $r_t^{mr}$
calculate average pick-up distance $r_t^{pd}$
calculate driver utilization rate $r_t^{du}$
drop matched riders and drivers
sample new travel demand locations from KDE function
update $N_t^r$, $N_t^d$ to $N_{t+1}^r$, $N_{t+1}^d$
for $rider = 1: N_{t+1}^r$, do:
    if $t_i > t_{max}$, do:
        drop the rider from the rider queue
for driver $= 1: N_{t+1}^d$, do:
    check and update availability
    sample their re-locating behaviours
    update driver's location
get and normalize the observe of the next state $S_{t+1}$
calculate reward $R_t$
```

Appendix B. Developing Environment

Python: Version 3.9.13 available at:

https://www.python.org/downloads/

Here lists all the needed packages for developing and running the Ride-hailing simulator. The distribute source and their versions are also attached alongside the packages.

gym: pip install gym==0.26.2
datetime: already installed with Python from version 2.6
FoliumMap: pip install folium==0.14.0
NumPy: pip install numpy==1.24.2
Pandas: pip install pandas==0.20.3
GeoPandas: pip install geopandas==0.14.0
matplotlib: pip install matplotlib==3.8.2
Scikit-learn: pip install scikit-learn==1.3.2
SciPy: pip install scipy==1.9.1
PyPROJ: pip install pyproj==3.6.1

Here lists all the needed packages for developing and running the MDDPG algorithm with the developed ride-hailing simulator. The distribute source and their versions are also attached alongside the packages.

argparse: already installed with Python from version 3.2
gym: pip install gym==0.26.2
random: already installed with Python from version 3.6
copy: already installed with Python from version 2.6
NumPy: pip install numpy==1.24.2
collections: already installed with Python from version 2.6
Pytorch: pip install torch==2.3.0
matplotlib: pip install matplotlib==3.8.2

The source code of the simulator is available at:

https://github.com/zhh05/RL_ride-hailing_radius

Appendix C. Additional Results

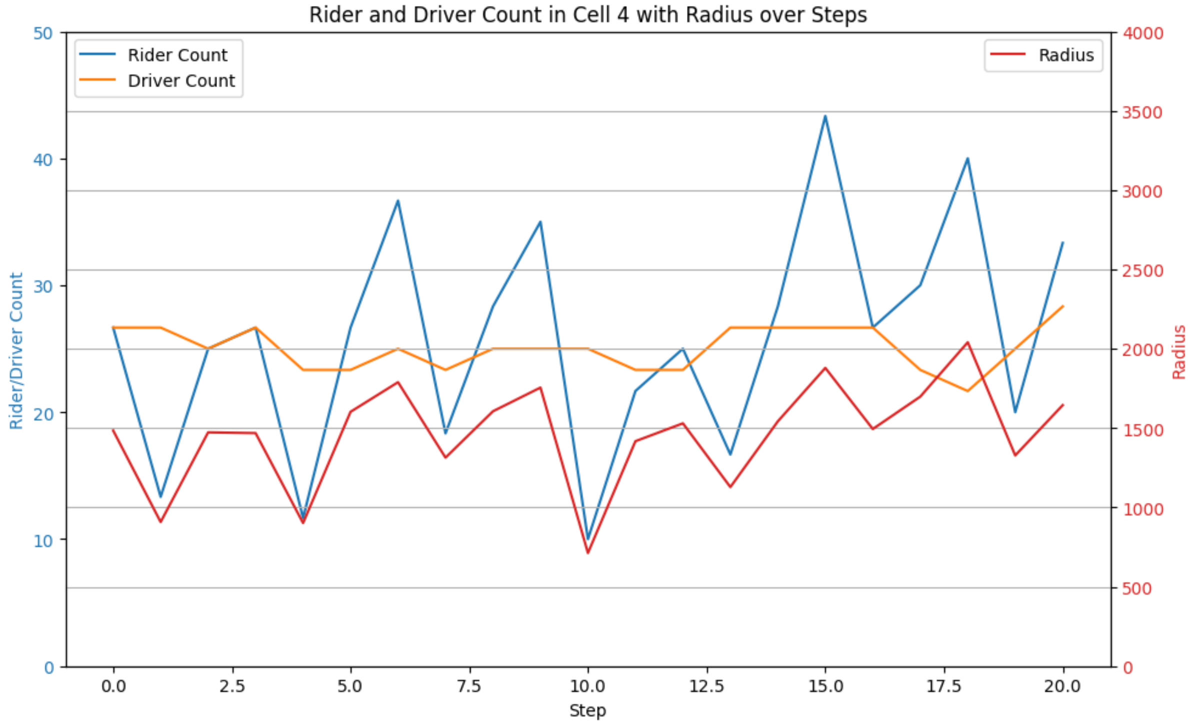


Figure C.1. Radius, rider-driver plot with stable driver numbers.

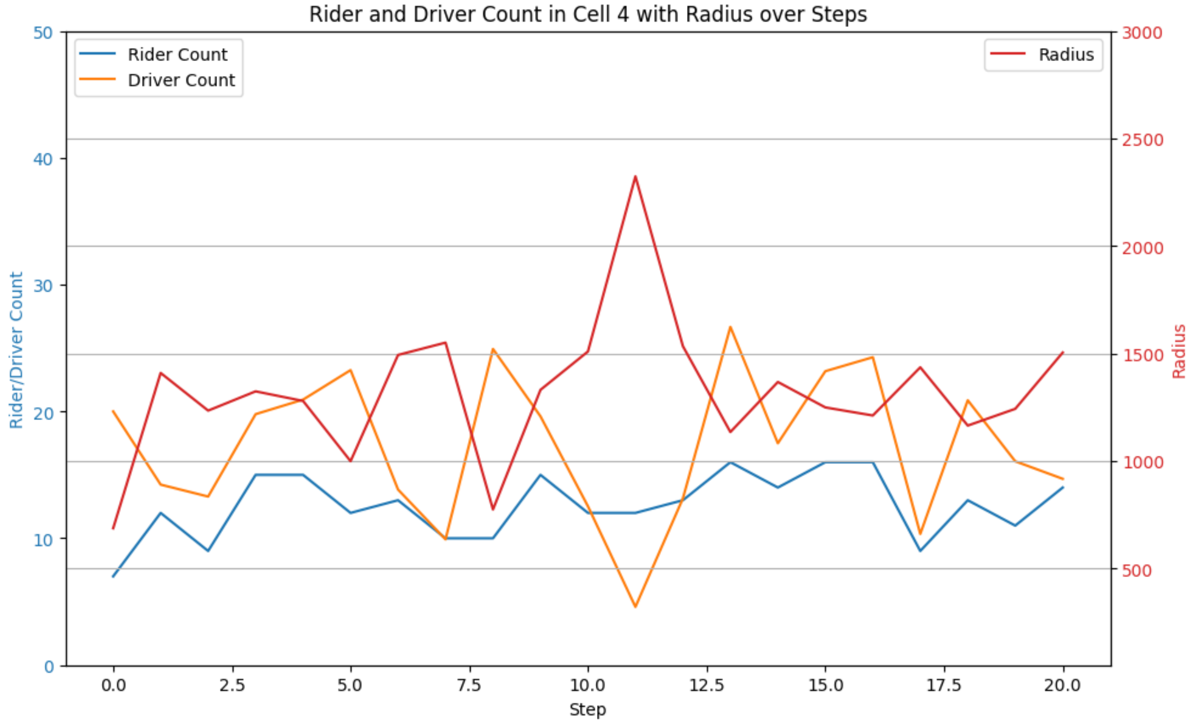
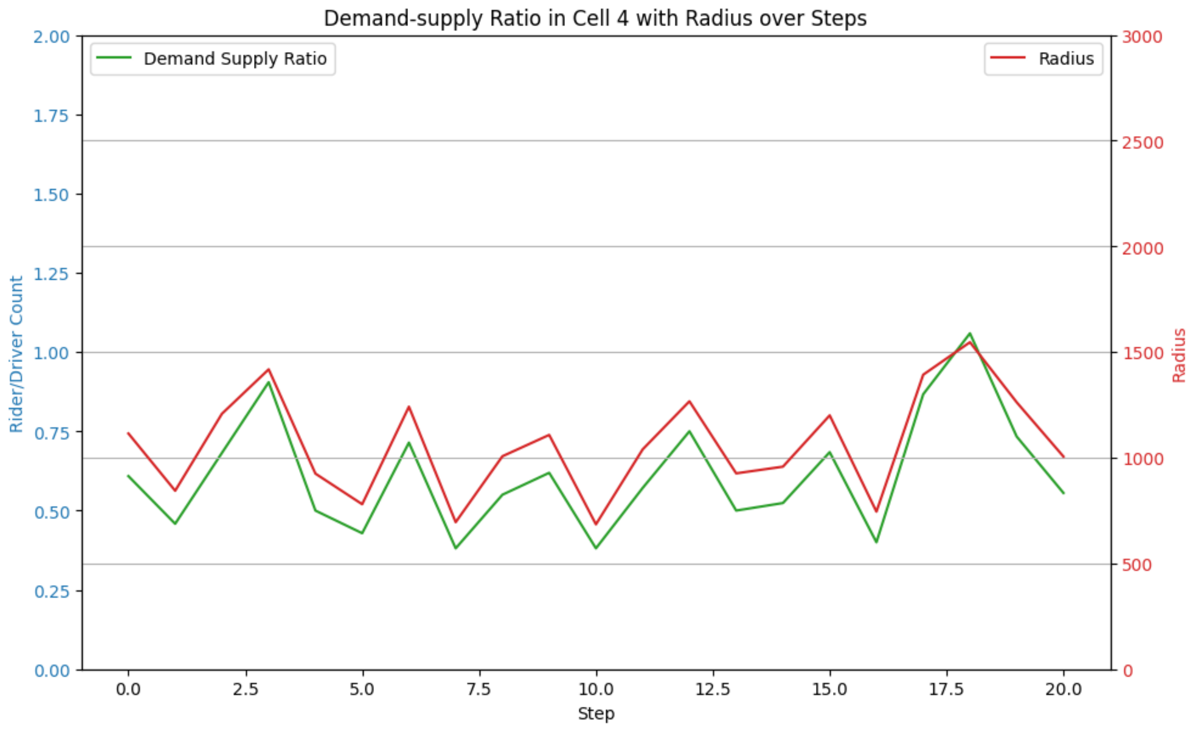
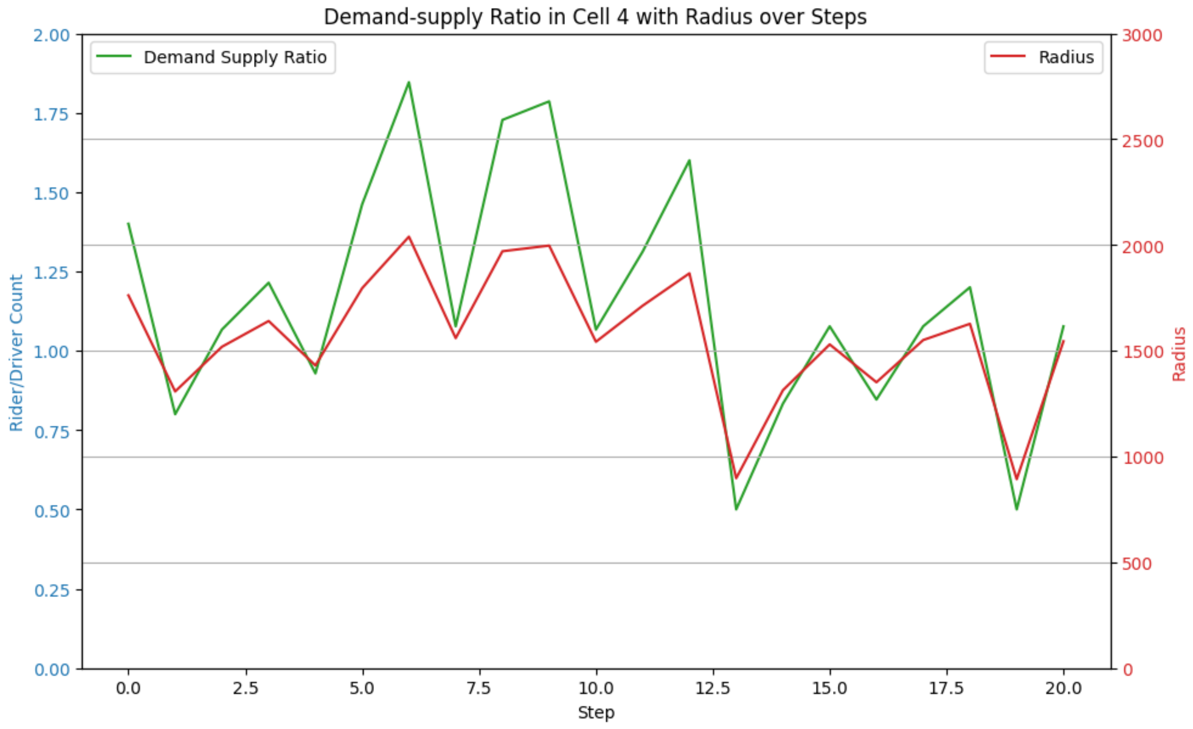


Figure C.2. Radius, rider-driver plot with fluctuating driver numbers.



(over-supplied)



(under-supplied)

Figure C.3. Radius, demand-supply ratio plot.

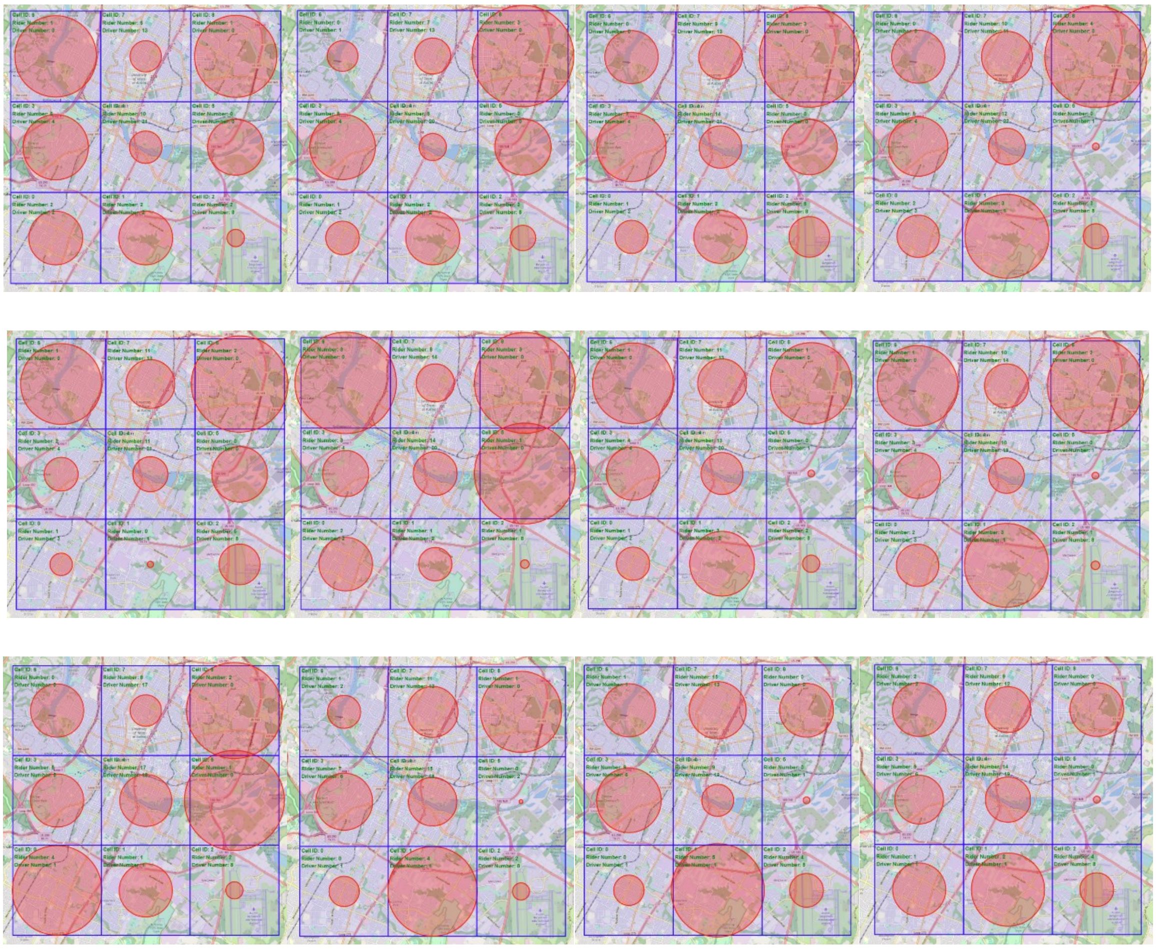


Figure C.4. Radius changing trend over few time-steps.

TU Delft