Title
Reinforcement Learning for Operational Problems in Transportation Systems with Autonomous Vehicles

Author
Mao, Chao

Publication Date
2019

Peer reviewed|Thesis/dissertation

Reinforcement Learning for Operational Problems in Transportation Systems with Autonomous Vehicles

By

Chao Mao

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Civil and Environmental Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Zuo-Jun Shen, Chair
Professor Michael Cassidy
Professor Mark Hansen
Assistant Professor Paul Grigas

Spring 2019

Reinforcement Learning for Operational Problems in Transportation Systems with Autonomous Vehicles

Abstract

Reinforcement Learning for Operational Problems in Transportation Systems with Autonomous Vehicles

by

Chao Mao

Doctor of Philosophy in Civil and Environmental Engineering

University of California, Berkeley

Professor Zuo-Jun Shen, Chair

In recent years, autonomous driving technologies are developing so fast that we can expect in the near future, fleets of autonomous vehicles will be driving on public roads and providing services to passengers. And the introduction of autonomous vehicles would bring us numerous opportunities to improve the operations of transportation systems, since we can have better real-time control of these vehicles. This dissertation focuses on two representative operational problems for the transportation systems with autonomous vehicles: how to build more effective real-time routing services for these vehicles, and how to better manage the fleet of autonomous vehicles for taxi services.

Most previous works in addressing these two problems are focusing on developing parametric models to reflect the network dynamics and designing efficient algorithms to solve these models. Although there are some clear advantages of these parametric models, there are some major limitations when applying them to real networks. First, in order to allow for efficient solutions, strong assumptions or approximations have to be made in most of the parametric models. However, many of the assumptions can not be validated in reality. Second, some of the proposed parametric models still suffer from the curse of dimensionality, i.e. they can be applied to small networks but can not be incorporated into larger networks. With these considerations, the aim of this dissertation is to explore the use of non-parametric model-free methods for these two problems in transportation networks. More specifically, we incorporate the framework of Reinforcement Learning (RL), which is primarily concerned with how to obtain the optimal policy when the model of the Markov decision process (MDP) is not available.

We first study the problem of adaptive routing in stochastic and time-dependent networks. The problem is formulated as a MDP first, and we present the idea of Q learning to solve it in discrete state space, which is also compared with traditional stochastic dynamic programming method. Our case study on a mid-size network shows that Q learning outperforms the traditional dynamic programming method in most cases, especially during peak demand periods. Then in order to resolve the curse of dimensionality, we introduce an

offline tree-based batch mode reinforcement learning method called fitted Q iteration (FQI), which can work in continuous state space and incorporate any regression algorithm as an approximation for the value function. The case study shows that it can further improve the routing policy and deliver a more flexible strategy during peak hours. Furthermore, the computational study shows that FQI can make more sufficient use of data at hand, since if there are more data fed into the training process of FQI, the resulting routing performance could be improved consistently.

Then we present a deep reinforcement learning approach for the problem of dispatching autonomous vehicles for tax services. We formulate the problem first and present our basic assumptions on the problem. Then we propose an actor-critic framework with neural networks as approximations for both the actor and critic functions, and with adaptations to the output action function. Furthermore, we provide a benchmark by formulating the problem as an integer program in order to maximize the total rewards. From our case study based on the New York yellow taxi data, we find that no matter the system dynamics are deterministic or stochastic, the RL method can always converge to some value close to the true optimal. Furthermore, when we choose to consider user priority and first serve those impatient passengers, we notice that the average waiting time of all passengers would sacrifice, but we add more fairness to the system by making sure there are less extreme long waiting times.

To my mom and dad, who give me endless love and support.

Contents

List of Figures

List of Tables

Acknowledgments

The past five years at Berkeley has been a wonderful and amazing journey for me. I have so many great experiences, and the memories will last forever in my mind. Most importantly, I know it is the great individuals I met along the way that have made this possible.

First, I am so grateful to have Professor Zuo-Jun Max Shen as my research advisor. His broad vision, open-mindness and cheering guidance have made my research work an enjoyable experience. When I feel confused, he is always there to guide me through the darkness and keep me on the right track. And during our weekly group meeting, I can always learn something new and get to know the most popular research topics. My PhD work would be impossible without the help and guidance from him.

I would also like to thank the other members from my dissertation committee. Professor Michael Cassidy has introduced me to the field of transportation operation in my first year and offered great suggestions to this study. Professor Mark Hansen's feedback during the workshop has helped me proceed with my research efficiently and in the right direction. Professor Paul Grigas has also been very supportive in our work and is doing great research in the area of optimization and machine learning. Besides, I am also extremely thankful to Professor Samer Madanat, who advised me during my first year at Berkeley. He guided me through the process of doing independent research work and also provided me with funding support. I really appreciate working with him in my first year.

Meanwhile, my life at Cal is full of great memories with my peers and friends. From my friends within our research group (Wei Qi, Yanqiao Wang, Sheng Liu, Ying Cao, Junyu Cao, Meng Qi etc.), to my friends at McLaughlin Hall (Mogeng Yin, Lei Kang, Yulin Liu, Mengqiao Yu, Lu Dai etc.), we had so much fun together. I will always remember the time we spent hanging out together, trying great food, making jokes of each other, and playing tennis and ping-pong. I would like to thank them and wish them all the best in the future.

Last but not least, I want to thank my parents, for raising me up and providing me with endless love and support.

Chapter 1

Introduction

1.1 Background and Motivation

During the past decade, we have seen considerable developments in automated driving technologies. Actually at present, many vehicles on the road are already considered to be semi-autonomous due to safety features like assisted parking and braking systems, and a few have the capability to drive, steer, brake, and park themselves. While the technology is not perfect yet, it is expected to become more mature and widespread in the near future. According to Chan [19], a lot of major companies are actively involved in developing automated driving technologies, and table 1.1 shows the highlights of news release or announcements from them regarding the projected timeline to have market-ready autonomous driving systems.

As can be seen, the current trend of developments in automated driving moves so fast, which will doubtlessly create a wave of evolutionary transformations and revolutionary disruptions in our society. On the passengers' perspective, this means fewer traffic collisions, more comfort on the road, greater mobility freedom and less ownership of private cars. And on the society's side, this means more environmentally friendly vehicles and infrastructure, less societal losses due to traffic congestion and accidents, and higher productivity from the entire population. Furthermore, the introduction of autonomous vehicles will also bring numerous opportunities to the operation of transportation systems. For example, we can have better management of traffic flows based on the information sharing and smart control of autonomous and connected vehicles, we can introduce more effective real-time navigation and dynamic routing for these vehicles, we can bring more accessible and flexible shared rides for mobility services, and we can also save spaces for better infrastructure planning by smarter planning of parking these vehicles. However, how to achieve such goals when we have autonomous vehicles on the road still remains a challenging problem in the operation of transportation systems. And this dissertation focuses on two representative operational problems for the transportation systems with autonomous vehicles: how to build more effective real-time routing services for these vehicles, and how to better manage the fleet of autonomous vehicles for taxi services.

Organization	Confirmed and predicted product introduction	Predictions of readiness for autonomous vehicles
Audi/VW	2016 Piloted Driving	Full AV by 2021
BMW	2014 traffic jam assist 2014 automated parking	Available by 2021
Bosch	2017 Integrated Highway Assist 2020 Highway Pilot	Auto Pilot by 2025
Continental		Available by 2020
Daimler-Benz	2014 Intelligent Drive	Available by 2020
Ford	2015 fully assisted parking	To mass produce AV in 2021
GM	2017 Super cruise	
Google	2015 Driverless Pod prototype	Available by 2018
Honda		Available by 2020
Hyundai		Available by 2030
Mobile Eye	2016 technology ready for OEMs	
Nissan	2016 traffic jam pilot 2018 multiple lane control	Available by 2020
Tesla	2015 Lane Assist + ACC 2016 highly autonomous	Self-driving 20202025
Toyota	Mid 2010s highly autonomous	
Volvo	2015 traffic jam assist 2017 Drive Me FOT in Sweden	Zero fatality cars by 2020

Table 1.1: Predicted market introduction of automated driving systems

On one hand, in-vehicle navigation system has become an important tool for human drivers nowadays. It can provide drivers with the shortest path from the origin to the quested destination. Usually, a prior shortest path is computed in the beginning, and as the driver travels on the way, the system will recalculate the shortest path based on updated real-time traffic information. However, as for self-driving cars, the navigation system can be totally different. First, we do not need to provide the entire route to any human driver in the beginning. Instead, we only have to tell the vehicle which successor node to travel to when it arrives at some intersection node. And we can make further decisions when later nodes are reached. As we will discuss later, such adaptive routing policy is a better solution when the network traffic condition is stochastic and time-dependent. Second, instead of passively adjusting the route due to the change of traffic conditions on the way, we can make predictive and proactive decisions by looking into the recurrent traffic patterns from day to day. Thus we can try to avoid possible congestion on the road based on our explicit or implicit predictions.

On the other hand, ride sharing services are also becoming more and more popular in people's daily lives. And one major operational challenge for those ride sharing platforms is the imbalance of supply and demand, which will cause vehicles to be clustered in certain regions at certain times of day. And this will make the system inefficient as many travellers will not be served in time. Current ride sharing operators such as Uber and Lyft usually use pricing incentives such as surge pricing to help redistribute vacant vehicles and attract drivers towards regions with high demand. However, this also causes doubts and dissatisfaction among users. While for autonomous taxi fleet, the operator can directly control all the vehicles and thus make centralized decisions on how to dispatch the vacant vehicles at anytime, which will obviously make the system more efficient and cheaper to use. Then the operational problem now is how to optimally control the fleet such that all ride requests can be served in a timely fashion.

If we look into the above two problems, we can find some common characteristics among them. First, they all happen in transportation networks that have recurrent dynamic patterns. As for the adaptive routing problem, the traffic congestion pattern is quite similar from day to day, especially on weekdays. And for the ride sharing services, we know that customers' demand distribution also repeats from day to day on weekdays. For example, most people will travel from the residential area to the CBD area in the morning, and travel back in the evening. Thus we can try to learn such dynamic patterns from historical data either explicitly or implicitly, and then make use of the information in our decision making process. Second, both of these two problems can be modeled as sequential decision problems. In the adaptive routing problem, we are deciding which node to travel to at each intersection. And this decision process goes on step by step until we reach the final destination. Similarly for the taxi fleet control problem, we can make decisions on how to dispatch the vacant vehicles at some discrete times of a day. Notice that in such processes, the decision at each step will affect the following decisions. Thus in the beginning, we should take into account the long-term effect of the current action.

As we will review in Chapter 2, most of the previous works in addressing these problems

have been focusing on developing parametric models to reflect the network dynamics and designing efficient algorithms to solve these models. In these models, some finite set of parameters were used to represent the network characteristics. And when applying them to real networks, these parameters have to be estimated first and the models can be solved with the developed algorithms. There are some clear advantages of these parametric models: first, they are easy to understand and the results are interpretable; second, the parameters can be learned quickly from a small set of data; and third, some of the models can be solved quite efficiently.

However, there are also some major limitations of these models. First, in order to allow for efficient solutions, strong assumptions have to be made in most of the parametric models. However, many of the assumptions can not be validated in real transportation networks. Second, some of the proposed parametric models still suffer from the curse of dimensionality, i.e. they can be applied to small networks but can not be incorporated into larger networks. Moreover, the estimation of the parameters in the parametric models might also be biased due to the outliers of training data, thus resulting in some sub-optimal policies.

With these considerations, the aim of this dissertation is to explore the use of non-parametric model-free methods for these two operational problems in transportation networks. More specifically, we will incorporate the framework of Reinforcement Learning (RL), which is primarily concerned with how to obtain the optimal policy when the model of the Markov decision process (MDP) is not available. Instead of relying on any prior knowledge of the model, the learning agent will gather statistical knowledge of the stochastic transportation network by interacting with it and update the state action values directly, thus eliminating the necessity to estimate the model beforehand. The details of the methods and applications will be discussed further in Chapter 3 and Chapter 4.

1.2 Summary of Contributions

This dissertation focuses on solving two representative operational problems for the transportation systems with autonomous vehicles: the adaptive routing problem in stochastic time-dependent networks, and the problem of dispatching autonomous vehicles for taxi services. As for the adaptive routing problem, our contributions can be summarized in the following aspects:

- First, different from traditional parametric model-based methods, which might be impractical in finding the adaptive routing policies in reality, we show that reinforcement learning can be an effective non-parametric model-free method to solve the problem in stochastic time-dependent networks.
- Second, both the online Q learning method for discrete state space and the offline fitted Q iteration (FQI) method for continuous state space have been presented for the problem, and both methods are compared with the traditional dynamic programming based method.

- Third, a case study based on a mid-sized network is conducted to demonstrate the performances of these different methods. We show that Q learning outperforms the traditional dynamic programming method in most cases, especially during peak demand periods. And the FQI algorithm with the tree-based function approximation further improves the routing policy and delivers a more flexible strategy. Moreover, it can resolve the curse of dimensionality by introducing value function approximations in the continuous state space.

On the other hand, for the problem of dispatching autonomous vehicles for taxi services, our contributions are wrapped up in the following perspectives:

- First, we present a deep reinforcement learning approach for the problem of dispatching autonomous vehicles for tax services. In particular, we propose an actor-critic framework with neural networks as approximations for both the actor and critic functions.
- Second, we also derive the theoretical upper bound of the total costs if we assume the dynamics of the system are deterministic and known to us beforehand, which can serve as a benchmark.
- Third, we implement our RL method and apply it to a simplified network based on the New York yellow taxi services. Our case study shows that no matter the system dynamics are deterministic or stochastic, the RL method can always converge to some value close to the true optimal.
- Fourth, we also investigate the scenario where we have to consider user priority. And the case study shows that this will cause the total waiting time of all passengers to sacrifice, but we add more fairness to the system by making sure there are less extreme long waiting times.

1.3 Dissertation Outline

The dissertation is structured in the following manner:

- Chapter 2 first reviews some of the existing literature on the problem of adaptive routing, most of which is focused on using parametric models to solve the problem. Similarly, current studies on optimal control of taxi fleets for on-demand mobility services are reviewed. Then we provide a through overview on the state of the art of reinforcement learning techniques.
- Chapter 3 studies the problem of adaptive routing in stochastic time-dependent networks. First, we formulate the problem as a MDP, and then we present the basic idea of Q Learning to solve the problem, which is also compared with traditional model-based dynamic programming method. Furthermore, we look into the case of continuous state

space and introduce a batch mode reinforcement learning approach called fitted Q iteration (FQI), while tree-based regression model is chosen as an an approximation of the Q function. And then we conduct a case study, which shows the experiment results and compares the performances of Q learning and FQI with the model-based dynamic programming method.

- Chapter 4 focuses on the problem of dispatching autonomous vehicles for taxi services using deep reinforcement learning approach. First, we formulate the problem and present our basic assumptions. Then we introduce the idea of actor-critic method and the adaptation we make in order for it to work in our problem. Furthermore, as a benchmark, we derive the theoretical upper bound of the total rewards we can get if we assume the dynamics of the system are deterministic and known to us. Also, a different scenario is presented where user priority is taken into account. Moreover, the experimental results and discussion of the RL methods under different cases are presented in the end.

- Chapter 5 provides a comprehensive summary of our research motivation, objective, methodological frameworks, experimental results and corresponding findings. This chapter also focuses on identifying future research directions for more applications of reinforcement learning methods in solving operational problems in transportation systems.

Chapter 2

Literature Review

In this chapter, we will first review some of the existing literature on the problem of adaptive routing in section 2.1, most of which is focused on using parametric models to solve the problem. Similarly, current studies on optimal control of taxi fleets for on-demand mobility services are reviewed in section 2.2. Then we provide a through overview on the state of the art of reinforcement learning techniques in section 2.3.

2.1 Adaptive Routing Problem

In-vehicle route guidance system has become a more and more popular tool in people's daily lives, it can provide drivers with guidance on optimal routes from their current locations to predetermined destinations. Nowadays, many navigation devices can also receive real-time traffic information, which can be incorporated by the system to come up with smarter routing strategies. However, most of the current strategies are in a reactive fashion: when the link travel times are updated base on real-time traffic information, the system will recalculate the shortest path in the current network, and will recommend the new path to the user if it is faster. Since it does not take into account the possible realizations of link travel times in the future, this strategy is only suboptimal.

Different from a deterministic and static network where link travel times are fixed and do not change with time, the real traffic networks are usually stochastic and time-dependent due to the nature of periodicity and volatility of traffic demand. And it has been shown by Hall [38] that the standard shortest path algorithms such as Dijkstra's algorithm and A* search fail to find the minimum expected travel time path in such networks. It was shown that the optimal "route choice" is not a simple path but an adaptive decision rule: an optimal successor node is chosen based on the arrival time of the current node, and further choices are made only when later nodes are reached. A method based on dynamic programming was proposed to find the optimal time-adaptive decision rule. It should be noted that Hall's adaptive routing model is a parametric model since the travel time probability distributions for all the links in the network are assumed to be known and utilized in the algorithm.

Following Hall's work, a large number of studies have been conducted to address the adaptive routing problem in different settings, most of which are based on parametric models and are summarized as follows.

Fu and Rilett [27] extended the shortest path problem in dynamic and stochastic networks to the case where link travel times are defined as continuous-time stochastic processes. A probability-based formula for calculating the mean and variance of the travel time for a given path was developed and a heuristic algorithm based on the k-shortest path algorithm was proposed. Their model required the information on the mean and standard deviation of the link travel time as a function of the time of the day. Similarly, Miller-Hooks and Mahmassani [51] presented two modified label-correcting algorithms for the problem of generating least expected time (LET) paths in stochastic and time-dependent networks. Travel times on the network were represented as random variables with probability distribution functions that vary with time. Fu [26] also examined the adaptive routing problem in networks where link travel times were modeled as random variables with known mean and standard deviation, but the time-dependency of link travel times was handled with an algorithmic scheme. Based on the closed-loop routing policy in Fu [26], Du et al. [24] integrated traveller preferences in terms of travel time and travel time variability into the decision process. And they adopted a discrete distribution updated in real time to describe the dynamic characteristics of the link travel time.

While all the above works assumed that link travel costs are independent from each other, many other works have considered link-wise correlations in the stochastic networks. Waller and Ziliaskopoulos [77] addressed the stochastic shortest path problem with recourse when limited forms of spatial and temporal arc cost dependencies were accounted for. One-step spatial dependence was assumed in such a way that if information from the predecessor arc was given, no further spatial information had an impact on the expected current arc cost. And this relationship was reflected in the conditional probability matrices. Also, Gao and Chabini [28] studied routing policy problems in a general stochastic time-dependent network with both time-wise and link-wise dependency and perfect online information. A joint distribution of link travel times was used to represent the stochastic network, although it was difficult to be estimated in practice. Later, Gao and Huang [29] expanded upon past research by examining the optimal routing problem with partial or no online information. A heuristic instead of an exact algorithm was designed and employed based on a set of necessary conditions for optimality. However, discrete distributions of link travel times were assumed for the convenience of defining routing policies. And the resulting algorithm was strongly polynomial in the number of support points, which might be exponential to the number of links in real networks.

A number of other researchers also attempted to model the stochastic temporal dependence of link costs using Markov chain. Psaraftis and Tsitsiklis [60] examined the shortest path problem in acyclic networks in which arc costs are known functions of certain environment variables, and each of these variables evolves according to an independent Markov process. Azaron and Kianfar [4] applied the stochastic dynamic programming to find the dynamic shortest path in stochastic dynamic networks, in which the arc lengths were inde-

Assumptions	Representative works	Model inputs
No link-wise dependence	Fu and Rilett [27], Miller-Hooks and Mahmassani [51], Fu [26], Du el al. [24]	Mean and standard deviation (or distribution) of link travel time
Link-wise correlation	Waller and Ziliaskopoulos [77], Gao and Chabini [28], Gao and Huang [29]	Conditional probability matrices, discrete joint distribution, finite support points
Markov process	Psaraftis and Tsitsiklis [60], Kim et al. [42], Güner et al. [37]	Markov process parameters, link cost distributions

Table 2.1: Previous works based on parametric models to solve the adaptive routing problem

pendent random variables with exponential distributions. The parameter of the exponential distribution was assumed to be a function of the state of certain environmental variable, which would evolve in accordance with a continuous time Markov process. Later, Kim et al. [41] developed a decision-making procedure for determining the optimal driver attendance time, optimal departure times, and optimal routing policies under time-varying traffic flows based on a Markov decision process formulation. They assumed that each observed link can be in one of two states (congested or uncongested) that determined the travel time distribution used. However, when the number of observed links with real-time traffic information increases, the off-line calculations can be computationally intractable. To address this issue, Kim et al. [42] proposed a procedure for state space reduction. Taking into account the incident induced delays, Guner et al. [37] proposed a stochastic dynamic programming formulation for dynamic routing of vehicles in non-stationary stochastic networks subject to both recurrent and non-recurrent congestion.

In summary, we can notice that most previous studies focus on using parametric models to reflect the network dynamics and then develop algorithms to solve these models. And usually, these models are based on some finite set of parameters or inputs, which need to be calibrated beforehand. And we summarize some representative works in table 2.1.

2.2 Optimal Control of Taxi Fleet

In the past decade, we have seen rapid expansion of ride sharing services and the promising development of self-driving technologies. We believe that over the coming decades, ride sharing companies such as Uber and Lyft may aggressively begin to use shared fleets of electric and self-driving cars that could be summoned to pick up passengers and shuttle them to offices and stores. One major operational challenge such systems might encounter, however, is the imbalance of supply and demand. Peoples travel patterns are asymmetric both spatially and temporally, thus causing the vehicles to be clustered in certain regions at certain times of day, and customer demand may not be satisfied in time. The goal of

our research, therefore, is to come up with an efficient operational strategy to deal with the imbalance issue in such systems.

For traditional taxi fleets, ride sharing operators such as Uber and Lyft usually use pricing incentives (e.g. surge pricing, spatial pricing) to help redistribute vacant vehicles and attract drivers towards regions with high demand. Zha et al. [82] build a model based on a geometric matching framework and investigate the effects of spatial pricing and its regulation in ride-sourcing markets. Banerjee et al. [7] build a queueing-theoretic economic model to study the value of dynamic pricing in ride sharing platforms, and they find that the performance of the system under any dynamic pricing strategy is not better than that under the optimal static pricing policy. However, dynamic pricing is much more robust to fluctuations in the system parameters compared to static pricing. But there are also studies that have questioned the effectiveness of dynamic pricing in real-world ride sourcing market. Based on real data collected from Uber's smartphone app and official API, Chen et al. [20] observe that on a micro-scale, surge prices have a strong negative impact on passenger demand, and a weak positive impact on car supply. The authors also argue that Uber's reliance on discrete surge areas introduces unfairness into its system – two users standing a few meters apart may receive dramatically different surge multipliers.

On the other hand, for autonomous taxi fleet, the operator can directly control all the vehicles and thus make centralized decisions on how to dispatch the vacant vehicles at any time. To optimally control the fleet, lots of model-based methods have been proposed in the open literature. Pavone et al. [57] develop a real-time rebalancing policy that is based on a fluid model of the system, and they assume that all the pickups and drop-offs happen at a set of stations. And they show that under such a policy, every station will reach an equilibrium in which there are excess vehicles and no waiting customers. Similarly in [76], Volkov et al. establish a theoretical Markov-based framework to describe an urban transportation network, in which there are discrete pickup and drop-off locations, and they propose a practical redistribution policy and show that it performs favorably in light of different optimization criteria. Zhang and Pavone [83] propose a queueing theoretical model for the control of autonomous mobility-on-demand (MOD) system, and they show that an optimal open-loop policy can be found by solving a linear program, based on which they develop a closed-loop real-time rebalancing policy and then apply it to a case study of New York.

While the above studies rely on steady-state formulations and their control policies are time-invariant, there is another branch of study that incorporates demand forecasting and utilizes model predictive control (MPC) method. Miao et al. [49] present a receding horizon control (RHC) framework for large-scale taxi dispatching system. In the study, they utilize both historical and real-time GPS and occupancy data to build demand models, and apply predicted models to sensing data to decide dispatch locations for vacant taxis considering multiple objectives. Zhang et al. [84] present a model predictive control (MPC) approach to optimize vehicle scheduling and routing in an autonomous MOD system. At each optimization step, the vehicle scheduling and routing problem is formulated as a mixed integer linear program (MIP), and their case study shows that the MPC method outperforms pre-

vious time-invariant control strategies. While in [50], Miller and How present a predictive positioning algorithm which uses customer arrival rate information to position vehicles at key nodes in a MOD network graph in order to minimize the expected customer waiting time. Later, Iglesias et al. [40] propose a MPC algorithm that leverages short-term demand forecasts based on historical data to compute rebalancing policies, and their algorithm is built on a formulation that computes the optimal rebalancing strategy and the minimum feasible fleet size for a given travel demand. With simulations based on real data from DiDi Chuxing, they show that the approach scales well for large systems.

2.3 Reinforcement Learning

Markov decision problems (MDPs) are problems of sequential decision making in which an action has to be selected in each decision-making state visited by the concerned system. Back in 1950s, Bellman ([9], [10]) showed that the computational burden of a MDP could be dramatically reduced via dynamic programming (DP). However, it was also recognized quite early that classical DP methods like policy iteration (Howard [39]) and value iteration (Bellman [8]) might fail to solve large-scale and complex MDPs. According to Gosavi [32], in problems involving complex systems with several governing random variables, it is usually difficult to compute the values of the transition probabilities. This phenomenon is called the curse of modeling. In problems with a large dimension, storing or manipulating the elements of the so-called value function becomes challenging. This is called the curse of dimensionality. Thus traditional DP methods are quite ineffective for large-scale and complex problems.

On the other hand, the power of reinforcement learning lies in its ability to solve complex and large-scale MDPs to near optimality. In this framework, the learning agent will gather statistical knowledge of the environment by interacting with it directly. At each step, the agent will take some action based on its current knowledge of the environment, and it will receive certain scalar reward as a result, which can give the agent some indication of the quality of that action. And the function that indicates which action to take at each step is called the policy. The goal for the agent is to find the policy that can maximize the total accumulated rewards. By following a given policy, the agent can build estimates of the total rewards. And the function representing this estimated rewards is known as the value function. So once the value function is built based on past experiences, the agent can use it to decide future actions at different steps.

Over the course of time, several types of RL algorithms have been introduced, and they can be divided into three groups (Grondman et al. [34], Konda and Tsitsiklis [44]): actor-only, critic-only, and actor-critic methods, where the words actor and critic are synonyms for the policy and value function, respectively.

Critic-only Methods

Critic-only methods, such as Q-learning (Dayan and Watkins [23], Watkins [80], Bradtke et al. [15]) and SARSA (Rummery and Niranjan [63], Sutton [66]) use a state-action value function and no explicit function for the policy. Instead, a deterministic policy, denoted by π, is calculated by solving an optimization problem over the value function:

$$\pi(s) = \operatorname*{argmax}_a Q(s, a) \tag{2.1}$$

Thus we always choose the action for which the value function indicates that the expected reward is the highest. For discrete state and action spaces, we can store the Q-values for each state-action pair. However, for large-scale problems with lots of state-action pairs, or for continuous state action spaces, it is difficult to store all Q-values. Instead, we can approximate the Q-values using regression functions or neural networks. There are many examples of using neural networks in RL to solve real-world problems (Crites and Barto [21], Tesauro [74], Singh and Bertsekas [65], Das et al. [22], Gosavi et al. [33]).

However, if we use approximated value function when learning in an online setting, there is no guarantee on the near-optimality of the resulting policy. For example, divergence to suboptimality with linear or nonlinear function approximations has been reported (Baird [5], Gordon [31], Boyan and Moore [14], Tsitsiklis and Van Roy [75]). Thus other than the method of function approximation, there is a robust scheme called local regression, which is based on representative states whose Q-values are stored explicitly (Tadepalli and Ok [69]). Then kernel methods, nearest neighbor algorithms, decision trees, or interpolation can be used to estimate the value of any Q-value in the state space.

Actor-only Methods

Policy gradient methods are actor-only and do not make use of any stored value functions. Instead, they work with a parameterized family of policies and optimize the expected total costs over the parameter space directly. At each step, the gradient of the cost function over the parameters in the policy function is calculated, and then the parameters are updated in the direction of this gradient. There are several methods to estimate the gradient, e.g. finite difference methods and likelihood ratio methods.

There are a lot of advantages of policy gradient methods for real world applications. First of all, they can allow policy functions to generate actions in the continuous space, which is hard to achieve with critic-only methods. Second, the policy representations can be task-dependent and can incorporate domain knowledge, thus usually fewer parameters are needed in the learning process than in critic-only methods. Additionally, policy gradient methods can be used either model-free or model-based (Wang and Dietterich [79], Tangkaratt et al. [71]) as they are a generic formulation.

However, there are also some drawbacks of actor-only approach. One major problem is that the estimated gradient may have a large variance, thus leading to slow learning (Riedmiller et al. [62], Papini et al. [56]). In addition, it is on-policy learning method and

needs to forget data very fast in order to avoid the introduction of a bias to the gradient estimator. Thus every gradient is calculated without using any knowledge of past estimates, so the use of sampled data is not very efficient (Peters et al. [58]).

Due to the advantages stated above, policy gradient methods have become particularly useful for a variety of robot learning problems ranging from simple control tasks to complex learning tasks involving many degrees of freedom such as learning of complex motor skills (Gullapalli et al. [36], Mitsunaga et al. [52]) and locomotion (Kohl and Stone [43], Tedrake et al. [73]).

Actor-Critic Methods

Actor-critic methods combine the advantages of actor-only and critic-only methods. Similar to actor-only methods, actor-critic methods can generate continuous actions without the need to optimize over value functions at each step. And by adding a critic to evaluate the current policy produced by the actor, the large variance in the vanilla policy gradient method is reduced. The critic approximates and updates the value function using samples. The value function is then used to update the actors policy parameters in the direction of performance improvement, which can reduce the variance of the gradients. However, the lower variance is traded for a larger bias at the start of learning when the critics estimates are far from accurate (Berenji and Vengerov [13]).

There are also numerous applications of actor-critic methods in different domains. For example, in the field of robotics, some successful results of using actor-critic based methods were shown on a beam setup (Benbrahim et al. [12]), a peg-in-hole insertion task (Gullapalli [35]), and biped locomotion (Benbrahim and Franklin [11]). Moreover, there are also examples of applying actor-critic methods to solve problems in transportation systems. Richter et al. [61] applied the natural actor-critic approach to directly optimize the traffic signals, mapping currently deployed sensor observations to control signals. And Aslani et al. [3] utilized the actor-critic algorithm to design the adaptive traffic signal controllers called actor-critic adaptive traffic signal controllers (A-CATs controllers). And they also compared the performances of different function approximators on the learning of A-CATs controllers.

According to Grondman et al. [34], When applying RL to a certain problem, knowing whether a critic-only, actor-only, or actor-critic algorithm will yield the best control policy is almost impossible. However, there are a few rules of thumb that can help us select the most sensible algorithms to use.

First of all, if the action space is continuous in the problem, then it is better not to use critic only algorithms, since it might require solving some non-convex optimization problem over continuous action space at each control step. However, if the action space is discrete or can be discretized into some finite set, it makes sense to incorporate a critic-only method, since it does not introduce high-variance gradients as in actor-only methods, and it has less tuning parameters than actor-critic methods.

And to choose between actor-only and actor-critic methods, it makes sense to check if the environment (MDP) is stationary or not. If the environment is stationary, actor-critic

methods can provide lower variance in gradients than actor-only methods. However, if the environment is changing with time, a critic would not be able to provide useful information on the changing nature to the actor. Thus we should turn to actor-only methods, which are more resilient to fast changing environments.

Chapter 3

Adaptive Routing Problem in Stochastic Time-dependent Network

3.1 Introduction

As we have discussed in the review in section 2.1, most of the previous work in addressing the adaptive routing problem has been focusing on developing parametric models to reflect the network stochasticity and designing efficient algorithms to solve these models. In their models, some finite set of parameters were used to represent the network characteristics, such as link travel time distributions, link correlations, or Markov processes. When applying them to real networks, we have to first estimate these parameters for the model based on some training data, and then solve for the routing policies based on the developed algorithms. There are many clear benefits of using parametric models: first, they are easy to understand and the results are more interpretable; second, the parameters can be learned quickly from a small set of data; and most importantly, they are more generally applicable, meaning that once the model is established for a certain network, we can solve for the best routing strategy from any origin to any destination quite efficiently. However, there are also some unavoidable limitations of the parametric models:

- First, in most of the parametric adaptive routing models we have seen, strong assumptions have to be made to allow for efficient solutions. However, these assumptions might not be consistent with the cases in real networks. For example, in Azaron and Kianfar [4], link costs were assumed to be independent random variables with known exponential distributions, which might be difficult to validate in real networks since the distributions can vary a lot.

- Second, some of the proposed parametric models still suffer from the curse of dimensionality, i.e. they can be applied to small networks but can not be incorporated into larger networks. For instance, the algorithm in Gao and Huang [29] is polynomial in the number of support points of the discrete joint link travel costs distribution, which

can be exponential to the number of links in the network. Thus in many cases, some approximation have to be applied to these models to allow for tractable solutions, which can lead to suboptimal results.

- Moreover, the estimation of the parameters in the parametric models might be biased due to the outliers of training data, thus resulting in some sub-optimal routing policies.

With these considerations, the aim of this study is to explore the usage of non-parametric methods for the adaptive routing problem in stochastic time-dependent (STD) networks. As in Kim et al. [41], the adaptive routing problem with real-time traffic information in STD network can be modeled as a discrete time, finite horizon Markov decision process (MDP). A MDP is a tuple $\langle S, A, T, R\rangle$ in which S is a finite set of states of the environment, A is the set of possible actions, T is the state transition probability function defined as $T : S \times A \times S \to [0, 1]$, and for each state transition we define the reward function as $R : S \times A \times S \to \mathbb{R}$. Then finding the best adaptive routing strategy in the network turns into computing an optimal policy π^* for the MDP, so for each state $s \in S$ we will know a best possible action $\pi^*(s) \in A$ to take.

In order to solve the MDP, if we assume that the model of the MDP is known beforehand, i.e. we have full information on the transition probability function T and the reward function R, then we can employ various model-based algorithms based on dynamic programming to compute the value functions and optimal policies using the Bellman equation. Two representative dynamic programming methods are value iteration (Bellman [8]) and policy iteration (Howard [39]).

However, for the adaptive routing problem in STD network, the direct application of the parametric model-based algorithms might be infeasible: first, it is almost impossible to estimate a model of the MDP to reflect the dynamics of the real traffic networks perfectly, thus we will get some sub-optimal solutions due to the biased estimation of the parameters; second, in order to improve the performances of the model-based algorithms, we need a more detailed representation of the network, i.e. the state space of the MDP needs to be increased, however, the complexity of those model-based algorithms will grow quickly as the problem size grows. With these two concerns, the main purpose of our study is to examine the application of non-parametric model-free methods for the adaptive routing problem. More specifically, we will incorporate the framework of Reinforcement Learning [67], which is primarily concerned with how to obtain an optimal policy when the model of the MDP is not available. Instead of relying on the prior knowledge of the transition and reward functions of the model, the learning agent will gather statistical knowledge of the stochastic network by interacting with it and update the state action values directly, thus eliminating the necessity to estimate the model beforehand. Moreover, we will also look into the effectiveness of an offline tree-based batch mode reinforcement learning method when we are dealing with continuous state space of the MDP for the network.

The remainder of this chapter is organized as follows. In section 3.2, we formulate the problem as a MDP, and then we present the basic idea of Q Learning to solve the problem

in section 3.3, which is also compared with the model-based dynamic programming method (section 3.4). Furthermore, in section 3.5, we look into the case of continuous state space and introduce a batch mode reinforcement learning approach called fitted Q iteration (FQI), while tree-based regression model is chosen as an an approximation of the Q function. Section 3.6 shows the experiment results and compares the performances of Q learning and FQI with the model-based dynamic programming method. Section 3.7 concludes this chapter.

3.2 Problem Statement

Consider a traffic network $G \equiv (N, E)$, in which N is the finite set of nodes and $E \subseteq N \times N$ is the set of directed arcs. For a pair of nodes n and n', we have $(n, n') \in E$ if and only if there is a road connecting these two nodes such that traffic can flow directly from n to n'. For each node $n \in N$, we can find the set of its successor nodes $\Phi(n)$, i.e. $\Phi(n) \equiv \{n' : (n, n') \in E\}$. Given an origin-destination pair (n_o, n_d), the objective is to find a routing strategy such that the expected total travel time is minimized.

For each link $r \in E$, we define its traffic congestion level at any time t as $w_r(t)$, and it can be in z different states, i.e. $w_r(t) \in \{0, 1, \cdots, z-1\}$. Further, we assume that we have full information on the real-time traffic status of the entire network. So at time t, we know the complete traffic status vector $W(t) = \{w_1(t), w_2(t), \cdots, w_{|E|}(t)\}$. And in order to have a discrete state space, we allow $U = \{0, 1, \cdots, u-1\}$ to be the possible time intervals for t in a typical day. Note that when we have $t > u-1$, we will let $t = t - (u-1)$ since we are assuming a recurrent traffic pattern from day to day.

Whenever we arrive at a certain node $n_k \neq n_d$ along the path, we have to make a decision on which adjacent node to travel to. And the information we have at this decision stage includes the current time t_k and the congestion status vector $W(t_k)$. While the optimal adaptive routing policy requires real-time consideration and projection of the traffic states of the complete network, this approach makes the state space prohibitively large [37]. Actually it is often unnecessary to project the congestion status of the links that are too far away from the current location, because the projected information would be similar to the long run average steady state probabilities of the congestion states. So similar to [37], we can trade off the degree of look ahead (number of arcs to monitor) with the resulting projection accuracy and routing performance. Without loss of generality, we choose to monitor only two arcs ahead of the vehicle's location and model the rest of the arcs' congestion states through their steady state probabilities. For each node $n \in N$, we can define the set of emanating links from it as $\Omega_1(n) = \{(n, n') : (n, n') \in E\}$. And also, the set of emanating links from its successor nodes is defined as $\Omega_2(n) = \{(n', n'') : n' \in \Phi(n), (n', n'') \in E\}$. So when we arrive at node n at time t, the set of arcs we would monitor is $\Omega(n) = \Omega_1(n) \cup \Omega_2(n)$, and the corresponding congestion states vector is $W^{(n)}(t) = \{w_r(t) : r \in \Omega(n)\}$. Thus $W^{(n)}(t) \in H^{(n)} \equiv \{0, 1, \cdots, z-1\}^{|\Omega(n)|}$. The state space of our decision problem can therefore be defined as:

$$\mathbf{S} \equiv \{(n, t, W) : n \in N, t \in U, W \in H^{(n)}\}$$

Each state $s_k = (n_k, t_k, W_k) \in \mathbf{S}$ is characterized by the current node n_k, current time interval of the day t_k and the congestion states vector of the arcs at most two steps away W_k. The corresponding action space for state s_k is defined as $A(s_k)$ and is consisted of the successor nodes of the current node n_k, i.e. $A(s_k) = \Phi(n_k)$. And our goal is to find the optimal policy $\pi : \mathbf{S} \to N$ such that for each state s_k we will know that we should travel to $\pi(s_k) \in \Phi(n_k)$.

In order to find the optimal routing policies, we can solve the MDP problem exactly under the assumption that we have perfect information on the MDP model, i.e. we know the state transition probability function T and the reward (or cost) function R. Then we can employ dynamic programming based methods, which belong to the parametric model-based methods and can serve as the benchmark, and it will be discussed later. However, we can hardly ensure that the MDP model is perfectly known for many applications. So without prior knowledge of the transition and reward models, reinforcement learning methods focus on obtaining the optimal policy by directly interacting with the environment to update the statistical knowledge of it. And one of the most basic and popular RL methods in a model-free fashion is the Q learning algorithm [23] [80].

3.3 Online Method: Q Learning

In the context of MDP, the value function of state s under policy π, denoted as $V^\pi(s)$ is the expected return if we start with state s and follow policy π thereafter. Similarly, the state-action value function $Q^\pi(s, a)$ is defined as the expected return if we start with state s and take action a and then follow policy π thereafter. And for the adaptive routing problem, the expected return is equivalent to the expected negative travel time from the current location to the destination. So our goal is to find the optimal policy π^* such that $V^*(s) = V^{\pi^*}(s) \geq V^\pi(s)$ for any $s \in \mathbf{S}$ and any policy π. Under the optimal policy, we have the following relationship:

$$V^*(s) = \max_a Q^*(s, a) \tag{3.1}$$

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s, a) \tag{3.2}$$

Thus once we know the optimal Q-function, we can find the optimal policy without the need to refer to the model of the MDP any more. And Q learning algorithm is designed to estimate the Q-functions directly based on the feedback of taking actions in the environment. It is a member of the family of temporal difference learning algorithms, which base the update in part on the existing estimates and each step in the interaction can generate a learning sample in accordance to the intermediate reward and observed next state. The interaction and learning mechanism is illustrated in Figure 3.1.

At the tth step in the decision process, the agent just receives a reward r_t and observes that the system is at state s_t. Based on its current knowledge of the environment (the estimated Q function so far), it will choose to take the optimal action a_t for this state (with

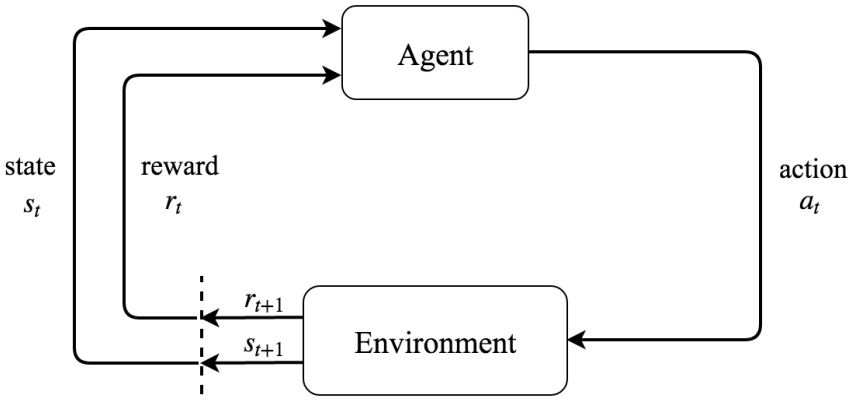


Figure 3.1: Basic idea of Q learning

some chance of exploration). As a result of the action, the environment will turn into a new state s_{t+1} and the agent will receive some reward r_{t+1} as a feedback. And the Q function for the state-action pair (s_t, a_t) will be updated based on the feedback. The entire process does not involve any estimation on parameters of the model such as the transition probability function T or the reward function R. If we choose a learning rate of α and a discount factor of γ, the update rule will be:

$$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a)] \tag{3.3}$$

In the model-free learning scheme, since the model is assumed to be unknown, the agent has to try out different actions in the beginning to see their results. Thus there is a balance between exploration and exploitation: in order to maximize the total rewards, the agent has to exploit its current knowledge and choose the best possible action; but sometimes it also needs to explore some different actions in case that it might be a better choice than the current best action. One of the most basic strategies to force exploration is the ε-greedy method: at each step, the agent will act according to the current best policy with probability $1 - \varepsilon$, and will choose to act randomly with probability ε. And an alternative method is the Boltzmann exploration strategy: the probabilities of taking different actions are weighted by the relative Q values, i.e. the probability of choosing action a at a certain step is

$$P(a) = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum_i e^{\frac{Q(s,a_i)}{\tau}}} \tag{3.4}$$

τ is the temperature and will decrease along the learning process. Higher temperature at the beginning will result in almost random selection of actions, and later lower temperature will produce a more greedy strategy. And this exploration method is employed for the study of our problem.

To solve the adaptive routing problem under the framework of Q learning, we can allow the discount factor γ to be equal to 1 since we are concerned about the expected total travel

time and different link costs are valued with the same weight. Besides, we have to fix the destination of the routing problem since the Q function can only store the expected travel time to a specific destination. Once we arrive at the destination, we do not need to travel any more. Thus the Q value for the destination node should always be zero. The resulting Q learning algorithm is shown in Algorithm 1.

Algorithm 1 Q learning for adaptive routing problem

procedure Q-LEARNING
  Input: $G = (N, E)$, destination n_d, learning rate α
  Output: Q function for routing to n_d
  Initialize: $Q(s, a) \leftarrow 0$, $\forall s \in \mathbf{S}$, $\forall a \in A(s)$
  for each episode (trip) **do**
    $s \leftarrow (n_0, t_0, W_0)$ ▷ s is initialized randomly
    while $s[0] \neq n_d$ **do**
      Choose node $a \in A(s)$ ▷ Boltzmann exploration strategy
      Travel to node $n' = a$
      Observe new state $s' = (n', t', W')$
      Receive link travel time $\delta(n, n')$
      $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[-\delta(n, n') + \max_{a'} Q(s', a')]$
      $s \leftarrow s'$
    end while
  end for
  return Q
end procedure

For the discrete state space, the above Q learning algorithm will converge to the optimal policy if α is decreased gradually and if enough number of trips are conducted, thus each state action pair is visited for enough number of times. And along the learning process, we can evaluate the updated Q function based on the expected travel time of following the resulting routing policy. This will be further discussed in the case study.

3.4 Comparison with Parametric Model-based Method

As a comparison, here we will present a benchmark parametric model-based method to solve the adaptive routing problem by assuming perfect knowledge on the MDP model. Similar to all other model-based methods, we have to make certain assumptions in order for us to estimate and calculate the parameters. These assumption are summarized as follows:

- The congestion state of each link evolves according to a non-stationary Markov chain and the states of different links are independent from each other.

- Link travel time depends only on its congestion state and the arrival time to this link.
- We have estimated the expected travel costs of different links under different states at various time intervals of a day, i.e. we know the cost function $\delta(r, t, w_r(t))$, $\forall r \in E$, $\forall t \in U$ and $\forall w_r(t) \in \{0, 1, \cdots, z-1\}$.
- We have estimated the transition probability matrices for all the links at different time intervals of a day, i.e. we know $\beta^r(t) = [\eta^r_{ij}(t, t+1)]_{ij}$ where $\eta^r_{ij}(t, t+1)$ is the probability for link r to transit from state i at time t to state j at time $t+1$, $\forall i, j \in \{0, 1, \cdots, z-1\}$.

Based on the above assumptions, we can get to know the parameters of the MDP model: the reward function R and the transition probability function T. For a certain state $s_k = (n_k, t_k, W_k) \in \mathbf{S}$, the reward of taking action $a_k \in A(s_k)$ to arrive at a new state s_{k+1} would be:

$$R(s_k, a_k, s_{k+1}) = -\delta((n_k, a_k), t_k, W^r_k) \tag{3.5}$$

Let $P^r(t, t')$ be the transition probability matrix of link $r \in E$ from time t to t', $\forall t, t' \in U$, then we have:

$$P^r(t, t') = \beta^r(t) \times \beta^r(t+1) \times \cdots \times \beta^r(t') \tag{3.6}$$

Thus the probability for link r to transit from state i at time t to state j at time t' would be $P^r_{i,j}(t, t')$. And for state $s_k = (n_k, t_k, W_k)$, if we choose to travel to node $a_k \in A(s_k)$, we will arrive at the new node after time $\delta_k = \delta((n_k, a_k), t_k, W^r_k)$, which is in the unit of the length of one time interval. Then we will arrive at a new state $s_{k+1} = (n_{k+1}, t_{k+1}, W_{k+1})$, in which $n_{k+1} = a_k$, $t_{k+1} = \lfloor t_k + \delta_k \rfloor$ and W_{k+1} is a random vector containing the congestion states of the monitored links around node n_{k+1}, i.e. $\Omega(n_{k+1})$. Since we are not assuming that link travel time is always multiples of the time interval length, thus δ_k might not be integers, $t_{k+1} = \lfloor t_k + \delta_k \rfloor$ might lose some information regarding the exact arrival time and will cause some approximation error. Also we can notice that some of the links in $\Omega(n_{k+1})$ are monitored at node n_k, and we can know its transition probability in this step. While for the other links whose states are unknown at node n_k, we choose to use their steady state probabilities at time t_{k+1} as an approximation, i.e. $\overline{P}^r(t_{k+1})$. Thus the transition probability function will be:

$$\begin{aligned} &T(s_k, a_k, s_{k+1}) = P(W_{k+1} | W_k, t_k, t_{k+1}) \\ &= \prod_{r \in \Omega(n_{k+1}) \cap \Omega(n_k)} P^r_{W^r_k, W^r_{k+1}}(t_k, t_{k+1}) \prod_{r \in \Omega(n_{k+1}) \setminus \Omega(n_k)} \overline{P}^r_{W^r_{k+1}}(t_{k+1}) \end{aligned} \tag{3.7}$$

With the knowledge on both R and T, we know that for any given policy π and state s, we will have:

$$V^\pi(s) = \sum_{s'} T(s, \pi(s), s')(R(s, \pi(s), s') + \gamma V^\pi(s')) \tag{3.8}$$

And with the optimal policy π^*, the V and Q function should satisfy the Bellman optimality equation:

$$V^*(s) = \max_{a \in A(s)} \sum_{s'} T(s, a, s')(R(s, a, s') + \gamma V^*(s')) \tag{3.9}$$

$$Q^*(s, a) = \sum_{s'} T(s, a, s')(R(s, a, s') + \gamma \max_{a'} Q^*(s', a')) \tag{3.10}$$

Based on the above relationship, we can use backward dynamic programming algorithm to solve for $Q^*(s, a)$ for each (s, a) pair. And when we initialize the Q value, we have to let $Q(s, a) = 0$ if $s = (n_d, t, W)$, i.e. if we arrive at a state where the current node is the destination node n_d, we do not have to travel any more and the expected remaining travel time is fixed as 0.

Notice that in order to use the method described here, we have to make sure that the assumptions are met and the required parameters have been estimated beforehand. Later in the case study, we will show how this parametric method performs compared to the non-parametric Q learning method.

3.5 Offline Method: Tree-Based Batch Mode Reinforcement Learning

One major drawback of the above methods is that they require a discretization grid over the state space, which will lead to an exponential increase of the computation cost if we need to consider a larger state space. Either in the parametric dynamic programming method or the Q learning method based on tabular form, we need to estimate the Q value for each state action pair (s, a) in the state and action space $(\mathbf{S}, A)$. However, the size of the state space is proportional to $|N| \times |U| \times z^{|\Omega(n)|}$, where $|N|$ is the number of nodes in the network, $|U|$ is the number of discrete time intervals in a day, z denotes the number of possible congestion states of each link and $|\Omega(n)|$ is the number of monitored links around node n. In order to improve the routing performance, we need a more refined representation of the environment by increasing the value of $|U|$, z and $|\Omega(n)|$ or even using continuous state variables, which will result in a much heavier computational burden and the problem can soon become intractable with the above methods.

One way to ease the curse of dimensionality of Q learning is to introduce value function approximation, so that $Q(s, a)$ is seen as a function to map the state action pairs into real numbers. However, in order to use incremental methods to update the function in an online fashion, we need to use differentiable parametric functions such as linear feature combinations or neural networks. Often they require a time consuming parameter estimation process at each iteration step and are not efficient enough. And there is no guarantee to converge to optimal policy with the function approximation since a small change in the estimated function parameters might have disproportionately large effects on the resulting policy.

Moreover, in the context of adaptive routing in real networks, online learning by directly experimenting on the real system might be impractical. We need a more conservative method since random exploration can be expensive and risky.

With the above considerations, in our work we also incorporate an offline tree-based batch mode reinforcement learning method called fitted Q iteration (FQI). Proposed by Ernst et al. [25], FQI combines the RL idea of learning from experience with the concept of continuous approximation of the value function developed for large-scale dynamic programming [18]. It is an offline batch mode learning method, where the agent is not directly interacting with the environment but receives only a set of four-tuples (s_t, a_t, r_t, s_{t+1}) and is asked to determine a control policy that is close to an optimal policy. By solving a sequence of supervised learning problems, the algorithm can determine an approximation of the Q function all over the state action space, as well as the resulting control policy.

Moreover, the framework makes it possible for us to make use of any regression algorithms in the context of reinforcement learning. As a non-parametric method, tree-based regression algorithms can offer a great modeling flexibility, which is very important since STD networks can vary a lot and the Q functions can be totally unpredictable in shape. And they can also provide high computational efficiency and scalability to high-dimensional spaces.

In this section, we will first introduce the framework of applying fitted Q iteration to solve the adaptive routing problem in an offline fashion. And then we discuss the properties and performances of the tree-based regression method within this framework.

FQI Algorithm

As a reinforcement learning method, fitted Q iteration is also a model-free learning algorithm: it does not rely on any knowledge of the model of the environment, instead the control policy is learned directly from experience. As shown in Figure 3.1, the experience gathered by the agent when interacting with the environment can be represented as a finite dataset $\mathcal{F}$ of four-tuples in the form (s_t, a_t, r_t, s_{t+1}):

$$\mathcal{F} = \{(s_t^l, a_t^l, r_t^l, s_{t+1}^l), l = 1, \cdots, |\mathcal{F}|\}$$

Each four-tuple is an observation of the one-step system transition. Different from the online Q learning method, here we do not have any requirements on the way these four-tuples are generated. They can be corresponding to one single trajectory as well as several independently generated one or multi-step trajectories. As long as the agent is interacting with the environment, the transition samples gathered will reflect the dynamics of the system and can help determine an approximation of the optimal policy. This makes FQI a much more practical method in the real networks, since we can imagine the size of trip data we can collect in real world and all of these random and diverse trips can be used although they might have different origins and destinations and use suboptimal routes.

Different from the previous methods which require a discrete state space, FQI aims at estimating a continuous approximation for the Q function and thus can allow for very large

discrete state space or even continuous state space. For our problem, we can redefine some of the state variables such that they can be continuous and better represent the characteristics of the environment. First, we allow the time variable t to be in range $U' = [0, u']$ such that it represents the time of the day. If it is in the unit of min, then $u' = 1440$ and whenever $t > u'$, we will let $t = t - u'$. Second, the traffic congestion of each link is not restricted to a finite set of levels any more, i.e. $w_r(t) \in \mathbb{R}^+$ can be a continuous variable and represent the instantaneous travel cost of link r at time t. Furthermore, we can also increase the range of monitored links at each node, i.e. we increase $\Omega(n)$. However, in order for us to make comparison with the previous methods, we keep $\Omega(n)$ unchanged. Thus the congestion vector will be $W^{(n)}(t) \in H'^{(n)} \equiv \mathbb{R}^{+|\Omega(n)|}$. Now the new state space of the problem is:

$$\mathbf{S} \equiv \{(n, t, W) : n \in N, t \in U', W \in H'^{(n)}\}$$

With such a continuous representation of state, we will have a continuous state vector (t, W) for each node $n \in N$ in the network, and the dimension of such a vector is $1 + |\Omega(n)|$. Thus if we increase $|\Omega(n)|$, the dimension of the state vector will grow linearly.

To describe the principle of the algorithm, let us start with the deterministic case: the state transition and the associated reward are determined solely by the state and action at time t. Namely, once s_t and a_t are known, $r_t = R(s_t, a_t)$ and $s_{t+1} = f(s_t, a_t)$ will be determined. Then the following sequence of Q_N functions will converge to the optimal Q^* function for the deterministic decision problem [25]:

$$Q_0(s_t, a_t) = 0 \tag{3.11}$$

$$Q_N(s_t, a_t) = R(s_t, a_t) + \gamma \max_{a'} Q_{N-1}(f(s_t, a_t), a') \tag{3.12}$$

Suppose we have already known the function Q_{N-1}, then we can use equation 3.12 to calculate $Q_N(s_t^l, a_t^l)$ for each four-tuple in the dataset $\mathcal{F}$, i.e. $Q_N(s_t^l, a_t^l) = r_t^l + \gamma \max_{a'} Q_{N-1}(s_{t+1}^l, a')$. Thus we have built a training dataset $\mathcal{TS} = \{[(s_t^l, a_t^l), Q_N(s_t^l, a_t^l)], l = 1, \cdots, |\mathcal{F}|\}$, based on which we can fit a function approximator $\hat{Q}_N$ of Q_N over the entire state action space. Using $\hat{Q}_N$ as an substitute for Q_N and based on the same reasoning, we can get $\hat{Q}_{N+1}$, $\hat{Q}_{N+2}$ etc.

If the system transition process is stochastic, then the value we calculate using the right hand side of equation 3.12 will not be $Q_N(s_t, a_t)$ any more. Instead, it should be some realization of a random variable whose expectation is $Q_N(s_t, a_t)$. But we can still use this same update equation to build the training dataset, since the regression algorithm will seek an approximation of the conditional expectation of the output variable given the inputs. Thus the resulting $\hat{Q}_N$ of the regression will still be a good approximation of the true Q_N function.

The fitted Q iteration algorithm for solving the adaptive problem is presented in Algorithm 2, at each iteration step, we build a new training set based on the full set of four-tuples and the estimated Q function from previous step, then we use the given regression method

to compute a new approximate Q function. During this process, it actually yields an approximation of the Q function by iteratively extending the optimization horizon: the Q_1 function produced at the first iteration is the conditional expectation of the instantaneous reward (cost) given the state action pair, while the Q_N function in the Nth step corresponds to an optimization horizon consisting of the rewards of the next N steps. At each iteration, only the output values need to be updated based on the $\hat{Q}$ function obtained in the previous step and the information about the reward and successor state in each tuple.

Algorithm 2 FQI for adaptive routing problem

procedure FQI

Input: $F = \{\langle s_t^l, a_t^l, r_t^l, s_{t+1}^l, \rangle, l = 1, \cdots, |F|\}$, destination n_d, a regression algorithm

Output: Q function for routing to n_d

Initialization: $N = 0$, $\hat{Q}_N = 0$ on $(\mathbf{S}, A)$ space

Iterations: Repeat until stopping conditions are met

$N \leftarrow N + 1$

Build the the training set $\mathcal{TS} = \{(i^l, o^l), l = 1, 2, \cdots, |\mathcal{F}|\}$ based on function $\hat{Q}_{N-1}$ and the set of four-tuples $\mathcal{F}$:

$$i^l = (s_t^l, a_t^l)$$

$$o^l = \begin{cases} r_t^l & \text{if } a_t^l = n_d \\ r_t^l + \max_{a'} \hat{Q}_{N-1}(s_{t+1}^l, a') & \text{if } a_t^l \neq n_d \end{cases}$$

Use the regression algorithm to induce from $\mathcal{TS}$ the function $\hat{Q}_N(\cdot)$

return $\hat{Q}_N(\cdot)$

end procedure

Notice that for a certain state action pair (s, a), we have $\hat{Q}_N(s, a) = \hat{Q}_N((n, t, W), a)$, in which the node variables n and a are discrete and the other variables are continuous. Actually, each (n, a) pair matches a certain link in the network. So in our experiment, we choose to build a $\hat{Q}_N$ function for each link in the network by separately calling the regression method on a sub-sample of the training set which corresponds to the same link. Namely, we will build the function $\hat{Q}_N^{(n,a)}(\cdot)$ for each link (n, a) in the network. In order to calculate $\max_{a'} \hat{Q}_{N-1}(s, a')$, we need to call each function $\hat{Q}_{N-1}^{(n,a')}$ and find the maximum value, i.e.

$$\max_{a'} \hat{Q}_{N-1}(s, a') = \max_{a'} \hat{Q}_{N-1}^{(n,a')}(t, W) \tag{3.13}$$

Also notice that we have to fix the Q value at the destination node as 0. So when we are updating the output values for the training set, we have to check if the next arrival node is n_d. If it is, then the output value should just be the reward(cost) of the last step to the destination.

We can stop the iterations if the difference between $\hat{Q}_{N-1}$ and $\hat{Q}_N$ drops below a predefined threshold. However, with this criterion, we cannot guarantee the convergence of

the algorithm with some function approximators. In the section below, we will discuss in more details about the convergence of the algorithm combined with the tree-based function approximator.

Tree-Based Function Approximator

The FQI algorithm discussed above can be combined with any supervised learning methods, either parametric or non-parametric. However, for some parametric regression methods, we have to preselect the shape of the approximation architecture, which might restrict its modeling flexibility. Especially in the adaptive routing problem, the system dynamics of the STD network can be so complicated that the Q functions for different links can take various shapes. While some other parametric methods such as neural networks can provide modeling flexibility, it comes at a price since a large number of parameters will be included and need to be estimated at each iteration step, thus contributing to a huge computational burden. On the contrary, non-parametric regression methods such as tree-based function approximator can ensure both modeling flexibility and computational efficiency at the same time. And it has been shown to perform much better than the parametric learning methods in some experiments [25]. Thus in our problem, we incorporate the tree-based function approximator as the regression method.

A regression tree aims at partitioning the input space into several regions by starting with a single root and progressively splitting the node into two children nodes with an appropriate splitting rule. This process goes on until the stopping conditions are met. Then each leaf node (region in the space) will contain some elements from the training dataset and a constant prediction value will be assigned to this leaf node by averaging the output values of all these elements.

Different tree-based algorithms differ by the number of trees built (single tree or ensemble of trees), the splitting rule chosen at the splitting nodes, and the stopping conditions of the tree building process. Some representative tree-based methods include KD-Tree, CART algorithm [17], Tree Bagging [16], Extremely Randomized Trees and Totally Randomized Trees [30]. KD-Tree and CART algorithm are designed such that only one single regression tree is built, while the other methods build an ensemble of trees. Moreover, in KD-Tree and Totally Randomized Trees, the splitting rule at each node is chosen in a way that it will not depend on the output values of the training set. While in the other tree-based algorithms, the structure of the tree will be influenced by the output values.

The FQI algorithm can be guaranteed to converge if the supervised learning method can produce a model that satisfies the following conditions [25]:

- $f(i) = \sum_{l=1}^{|\mathcal{TS}|} k_{\mathcal{TS}}(i^l, i) \times o^l$ (3.14)
- $\sum_{l=1}^{|\mathcal{TS}|} |k_{\mathcal{TS}}(i^l, i)| = 1, \forall i$ (3.15)
- $k_{\mathcal{TS}}(i^l, i)$ remains the same from one call to the other within the FQI algorithm.

A regression tree will assign to each leaf node a predicted value by averaging the output values of the elements which belong to this leaf node. Thus it will produce a model in the form of Equation 3.14 with the kernel defined by:

$$k_{\mathcal{TS}}(i^l, i) = \frac{I(i^l, i)}{\sum_{(a,b)\in\mathcal{TS}} I(a, i)} \tag{3.16}$$

in which $I(i^l, i)$ is an indicator function to denote whether i^l and i belong to the same leaf node, i.e. $I(i^l, i) = 1$ if i^l and i are in the same leaf node and 0 otherwise. And if the method builds an ensemble of P different regression trees, an average of the P different predicted values will be used as the final prediction. Thus the kernel function will be:

$$k_{\mathcal{TS}}(i^l, i) = \frac{1}{P} \sum_{m=1}^{P} \frac{I^m(i^l, i)}{\sum_{(a,b)\in\mathcal{TS}_m} I^m(a, i)} \tag{3.17}$$

in which $\mathcal{TS}_m$ is the subset of the training data used to build the mth regression tree, and $I^m(i^l, i)$ is an indicator function to denote whether i^l and i belong to the same leaf node in the mth tree. Clearly, both kernel functions in Equations 3.16 and 3.17 satisfy the normalizing condition in Equation 3.15.

Thus as long as the kernel $k_{\mathcal{TS}}(i^l, i)$ remains the same from one iteration to the other, the FQI algorithm will be ensured to converge. The condition is satisfied if the tree structures can remain unchanged throughout the iterations. Thus if KD-Tree is used with the FQI algorithm, it can ensure convergence. However, it has been demonstrated in [25] that those tree-based methods which can adapt their structure to the new output at each iteration usually provide better results than methods that do not. Especially, Extremely Randomized Trees tends to outperform other tree-based methods in many cases. Although it does not ensure convergence, the policy quality of the sequence was oscillating only moderately around a stable value. And the worst performance was even better than the ones obtained by other tree-based methods ensuring the convergence of the algorithm.

Thus in our case, we adopt the Extremely Randomized Trees as the regression method. If really required, we can still provide convergence in an ad hoc fashion: by freezing the tree structures after a certain number of iterations and then only updating the predicted values at leaf nodes. The algorithm works by building an ensemble of trees, each of which

is based on the complete original training set. To split a node in the tree, K different cut directions are selected at random and for each direction a cut point is also randomly picked. Then it computes a score for each of the K splitting rules and the one with the highest score is chosen to split the node. The algorithm will stop splitting a node if the number of the elements in it is less than the parameter n_{min}. Finally, the average of the predicted values produced by all the trees is used as the regression output.

3.6 Case Study

In order to illustrate the performances of the proposed online and offline reinforcement learning methods, and also to compare with the traditional parametric dynamic programming based method, we carry out a case study on a simulated mid-size network. First, we present the basic setup of the nework and how we design the dynamic congestion pattern from historical traffic data. Then the performances of both Q learning and parametric dynamic programming method are compared based on the average trip costs by following the resulting control policies. Moreover, the results of the FQI algorithm are presented and discussed as well.

Network

The Sioux Falls network (see Figure 3.2) is used as an example to compare the performances of different methods in our work. There are 24 nodes and 38 links in the network, and we assume that we have full real-time traffic information on all the links. In our experiment, node 20 is chosen as the destination.

As shown in [37], they found that two congestion states can better represent arc congestion dynamics based on the Gaussian Mixture Model (GMM) clustering analysis on the historical traffic data from the Michigan highway system. Thus in our case, we also assume that each link can be in two possible states during any time of the day, i.e. "congested" or "uncongested". And under each state, the link travel cost follows a non-stationary distribution that varies with the time of the day.

We further assume the time granularity of a day is 15 min. Thus each link can transit between different congestion states in every 15 min. And under each state, the travel cost distribution (assumed to be Normal distribution) is also renewed every 15 min. The distribution parameters (mean and standard deviation) are employed from the Caltrans Performance Measurement System (PeMS), which contains historical data collected in real-time from nearly 40,000 individual detectors spanning the freeway system across all major metropolitan areas of the State of California. In this dataset, we can check the travel time profiles of some major freeways in California. So for each link in the network of our case study, we randomly choose two similar traffic profiles from the PeMS dataset to represent the "congested" and "uncongested" traffic states. To preclude the possibility that one path might dominate all other paths in the network at all times, we scale the parameters accord-

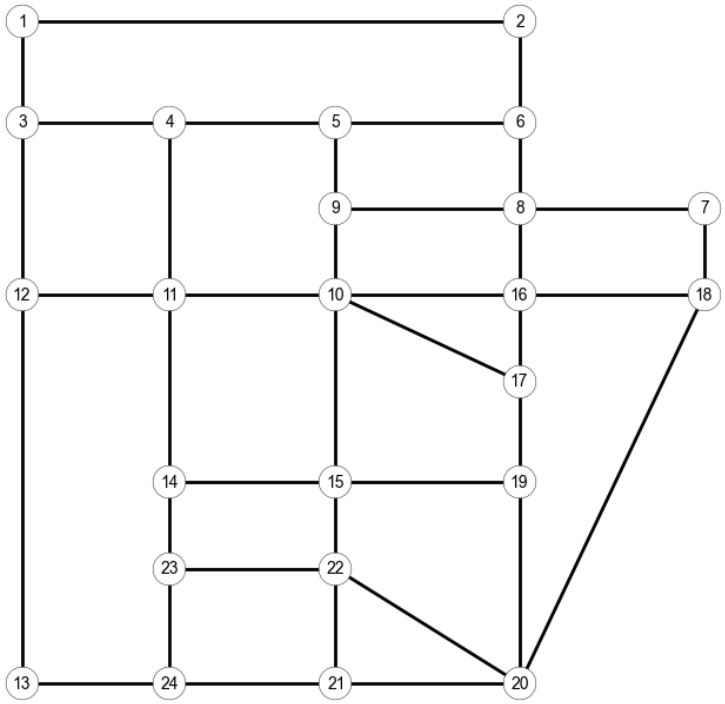
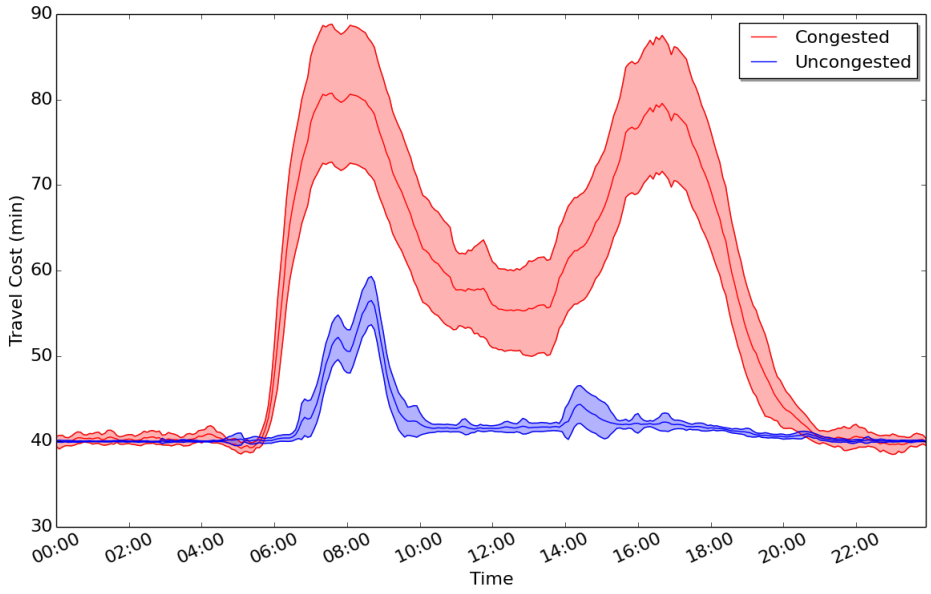


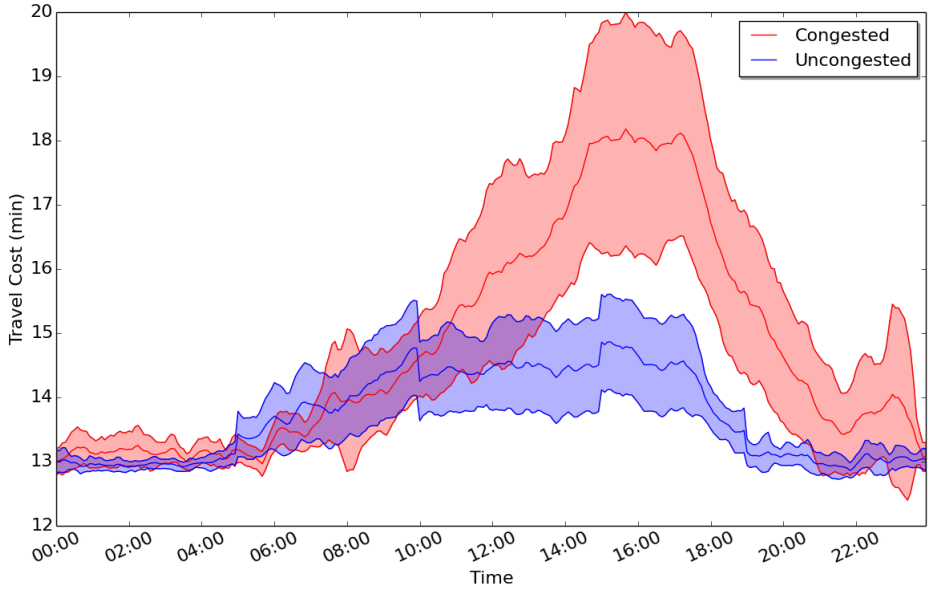
Figure 3.2: Sioux Falls network

ingly so that the travel costs of different routes are close to each other and adaptive routing will be more useful in this case. As an example, Figure 3.3 shows the non-stationary travel time distributions of two representative links under different congestion states, the shaded bands represent one standard deviation away from the mean travel time. Some links have bimodal distributions (two peak hours), while other links have unimodal distributions (only morning peak hour or evening peak hour), which are quite common in real traffic networks.

Moreover, we generate the state transition probabilities of different links based on their travel costs profiles such that the state transitions to same states (i.e., congested to congested or uncongested to uncongested) are more likely during peak demand time periods, which is also a typical finding in [37]. In Figure 3.4, we show the values for $\eta_{1,1}$ (probability of transition from congested to congested) and $\eta_{0,0}$ (probability of transition from uncongested to uncongested) for every 15 min. The congestion states of all the links will follow independent Markov processes based on these non-stationary transition probabilities. And the corresponding travel costs will be generated based on the congestion state and time of the day.

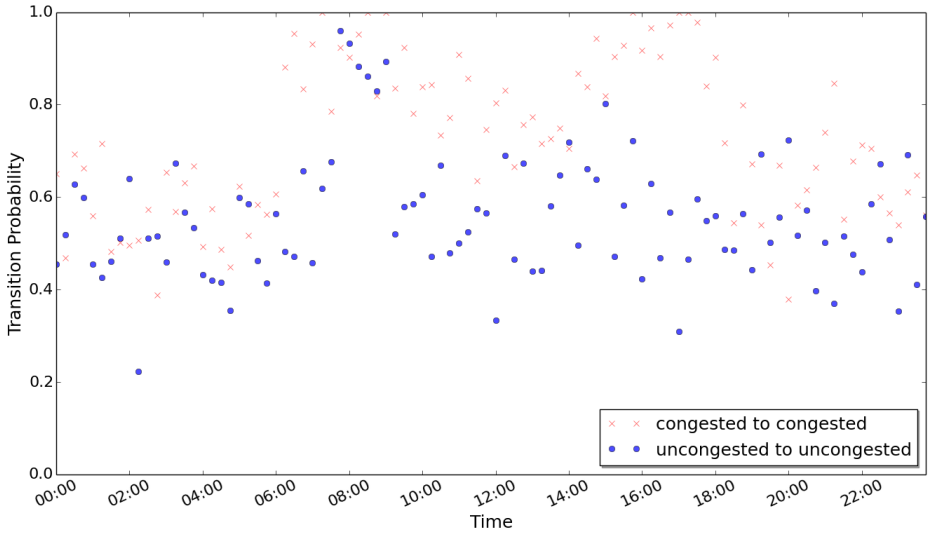


(a) Link (1,2)

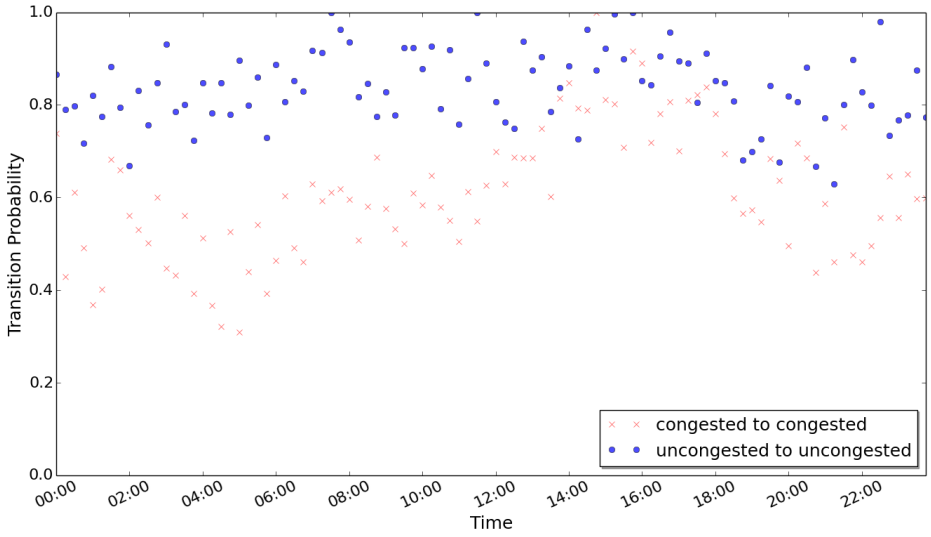


(b) Link (5,9)

Figure 3.3: Travel time distributions of two links under different congestion states



(a) Link (1,2)



(b) Link (5,9)

Figure 3.4: State transition probabilities of two links in a day

Performance of Q Learning

This section highlights the performance of Q learning in solving the adaptive routing problem, and the comparison with the parametric dynamic programming method as well.

In the Q learning algorithm, we first initialize the Q value for each state action pair as 0. Then for each trip, we randomly generate a starting state $s_0 = (n_0, t_0, W_0)$, i.e. we randomly pick a starting node and time, and we also generate a congestion states vector. Then as we travel along in the network, we can observe the feedbacks from the network such as the experienced link travel time and the new congestion states (assuming we can identify the congestion state of each link correctly), with which we can make updates to the Q table. To keep track of the updating process, we can fix the starting state s_0 and record the experienced travel times to the destination. As shown in Figure 3.5, we keep starting the trip from node 1 at 7:00 am with some fixed initial congestion states, as the number of our conducted trips grows, the average travel cost (black line) gradually decreases as a result of the updating on the Q table. It shows that the agent (driver) is becoming smarter and smarter by travelling more and more in the network. Note that in the graph, the shaded band corresponds to one standard deviation of the travel costs away from the mean, and different colors of the points represent different final routes taken by the agent. Since the network congestion dynamics is a stochastic process, the route taken by the agent is also random based on the realization of the network congestion states. Thus even after convergence, there are still different routes being taken.

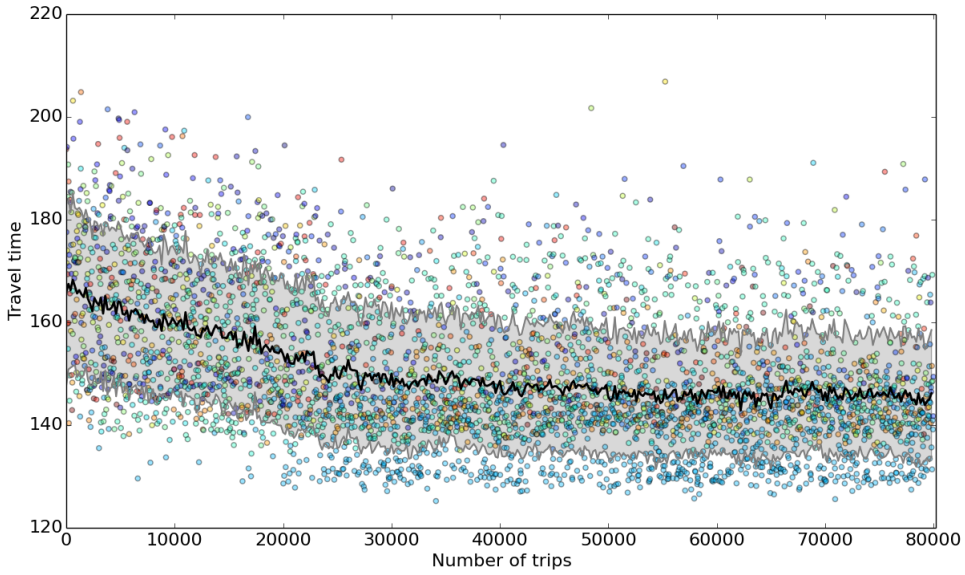


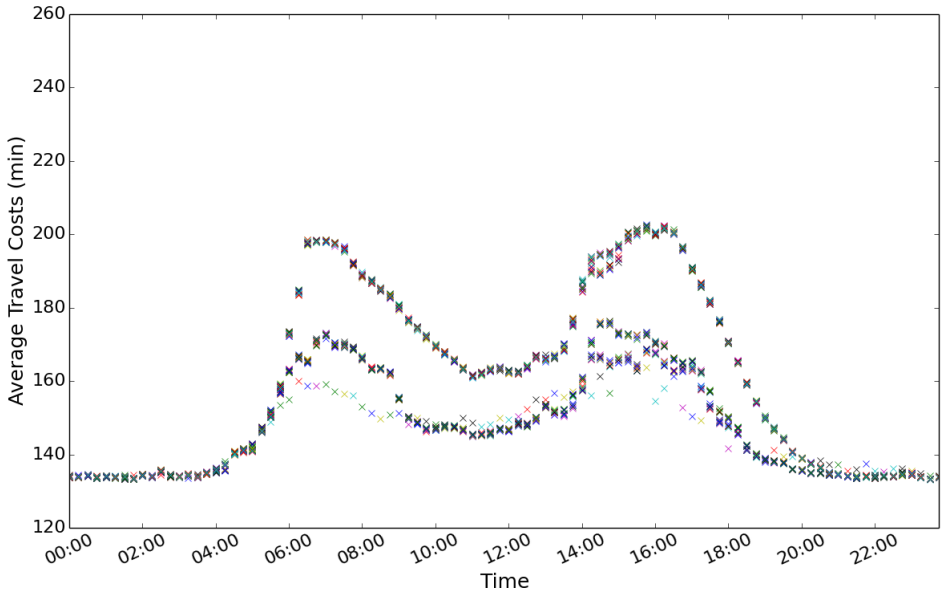
Figure 3.5: Trend of travel costs in Q learning

As a comparison, we also make use of the parametric dynamic programming method

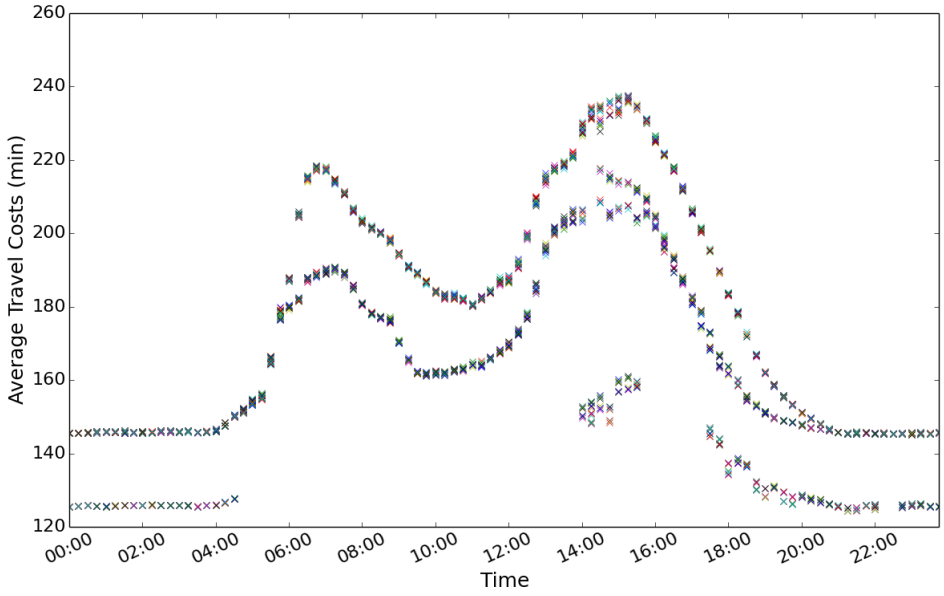
discussed in section II C. While we are updating the Q table online, we also keep a record of the traffic information gathered along the trip, i.e. the experienced link costs and congestion states at different times of a day. With the recorded data, we can estimate the expected travel costs of different links under different congestion states at various times of a day (i.e. $\delta(r, t, w_r(t))$) through the observed average link travel times, as well as the transition probability matrices for all the links at different times (i.e. $\beta^r(t)$). With these parameters, we can also solve for the optimal Q^* value for each state action pair based on the backward dynamic programming method discussed before. Notice that the network is set up in such a way that most of the assumptions made in the dynamic programming method are satisfied, i.e. the states of different links evolve independently and the travel time experienced by the driver is decided by the arriving time to that link and its congestion state. Besides, we also allow the driver to know the true congestion states of the links, i.e. there is no need for us to determine the congestion state based on the link travel cost and we can estimate the cost function and transition probabilities directly.

In order to evaluate the resulting Q tables, we can conduct a number of trips from a fixed starting node (node 1) at various times of a day with different initial congestion states. Then the average trip costs can be an indicator of whether the Q table can provide a successful routing policy in the dynamic network. So we get the updated Q tables from both methods with the same traffic data gathered from almost one million different trips, and then we follow the Q table to find the best route to the destination under different congestion states at various times of a day. The average trip costs are shown in Figure 3.6, in which different colors represent different initial congestion states vector. As can be seen, Q learning outperforms the traditional dynamic programming method in most cases, especially during the two peak demand periods. The savings can be as large as 25% during the peak hours and around 10% during off-peak periods.

Although most of the assumptions made in the dynamic programming method are satisfied in our case study and we provide additional information (congestion states) for the the model parameters estimation as well, there are other causes that might lead to its bad performance here compared with Q learning. First of all, as we have mentioned earlier, in the state vector $s_k = (n_k, t_k, W_k)$, t_k only keeps track of the current discrete time interval of a day. While during this 15-minute time interval, we do not know the exact arrival time to that link since we do not assume that the link travel time is always multiples of 15 min. Thus in the dynamic programming method, when we calculate the arrival time to the next link, the approximation we take by using $t_{k+1} = \lfloor t_k + \delta_k \rfloor$ would introduce some error. Second, since we choose to monitor only two links ahead of the current location, when we attempt to calculate the transition probability, we model the unmonitored links' congestion states through their steady state probabilities. Such an approximation would also cause some error and result in the bad performance. On the other hand, the number of parameters (i.e. $\delta(r, t, w_r(t))$ and $\beta^r(t)$) can be huge and the correct estimation needs a large amount of data. Once there are not sufficient data to support the calibration of the parametric model, the resulting solution can also be influenced by the estimation biases. To further investigate the influence of data availability on the performances of both methods, we vary the size of



(a) Q learning



(b) Parametric dynamic programming

Figure 3.6: Average trip costs based on routing policies from both Q learning and dynamic programming

training data (number of historical trips), and use the average trip cost at 7:00 am as an

indicator of the resulting routing strategy's performance. The results are shown in Figure 3.7.

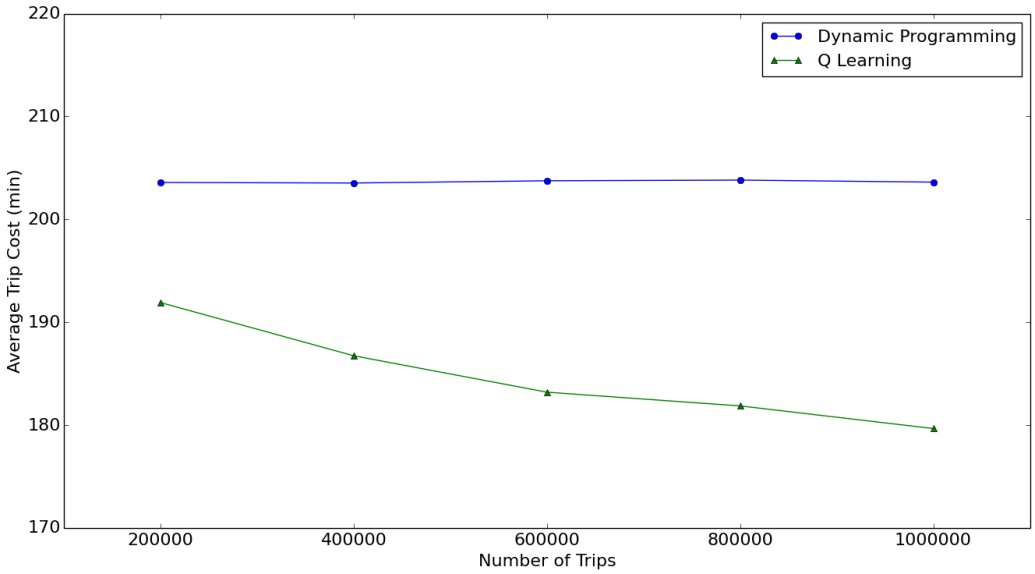


Figure 3.7: Influence of Training Data Size

As can be seen, when we increase the training data size, the performance of dynamic programming almost stays the same while the routing strategy from Q learning gets better and better. It shows that in our case study, the training data size is not the direct cause of the bad performance of dynamic programming. We can have a stable and accurate estimation of the model parameters with the one million trips' record data. Thus it is mainly the approximation we take when solving the model that causes its bad performance. However, in the non-parametric method, we are not restricted by the need to calibrate or solve any model. Instead, we can get the strategy by directly updating the Q values with our travelling information. So data can be utilized more sufficiently without any restrictions from the parametric models.

Nevertheless, we have to keep in mind that one can always mitigate the curse of approximation in the dynamic programming method, either by introducing more refined state space (e.g. using 1-minute time interval and thus tracking more accurate arrival times to links) or having a more idealized network (e.g. letting the travel time of each link be multiples of the time interval length). Since if all our assumptions are valid in the traffic networks and we have enough computing power, dynamic programming method can actually solve the MDP problem to exact optimality. However, the merit of the comparison here is to illustrate the cases when people use parametric models to solve the adaptive routing problem in real traffic networks, many of their assumptions can be violated and they need to take approximations considering the trade-off between efficiency and accuracy. Thus the power of these models

might be limited even if we have abundant data in hand. Learning-based methods, on the other hand, are not restricted by the need to establish or solve any models. Data can be used to update the value functions directly, and thus be utilized more thoroughly. This can be shown in the above experiment where we increase the size of the training data, the performance of model-based method stays the same while Q learning performs better and better with more data.

However, the routing policy obtained from Q learning is still not optimal. We can notice in Figure 3.6 that if we depart from the origin around midnight under certain initial congestion states, we can actually arrive earlier if we follow the routing policy from dynamic programming. This is due to the discrete state space we used in the Q learning method, while the link travel costs are actually continuous variables in our network. Besides, the size of the discrete state space would grow exponentially if we increase the number of congestion states of each link or increase the number of monitored links. Thus it is necessary for us to consider the continuous state space in order to come up with better routing results.

Performance of FQI

In this section, we will show the results we obtained with the offline FQI algorithm combined with the tree-based function approximation.

The network setup is kept as the same, but the training data we used here is in different form. As discussed before, each link traversed by the agent will produce a four-tuple learning sample in the form (s_t, a_t, r_t, s_{t+1}). However, here we employed a continuous state space in this method. Namely in the state variable $s = (n, t, W)$, the time variable t and congestion vector W are stored as continuous variables. Thus we do not need to identify any link as "congested" or "uncongested", we will just record its observed travel cost along the path.

In our experiment, when we built the Extremely Randomized Trees for each link as the Q function approximation, we set the number of trees in the ensemble as 20 and the parameter n_{min} as 100 based on some empirical trials. Similarly, to keep track of the iterating process, we can monitor the average travel cost of a certain trip from a fixed initial state (start from node 1 at 7:00 am) with the updated Q function. The result is shown in Figure 3.8. As can be seen, the average trip time gradually decreases as the iteration goes on, meaning that we are getting a better approximation of the true Q function at each iteration and a smarter routing policy as well. After the 7th iteration, it begins to oscillate slightly around a stable value. As discussed before, although we can not guarantee convergence of the FQI algorithm combined with Extremely Randomized Trees regression method, the policy quality will still remain quite stable after a few iterations. If needed, we can freeze the tree structures after the 7th or 8th iteration and the Q function will finally converge.

Similarly, to evaluate the Q function obtained with the FQI algorithm, we also find the average costs of the simulated trips starting from node 1 at various times of a day under different congestion states. The result is shown in Figure 3.9. As can be seen, the resulting continuous Q function can help improve the routing policy a lot, especially during the peak hours when the network dynamics become more complex and volatile. Compared with Q

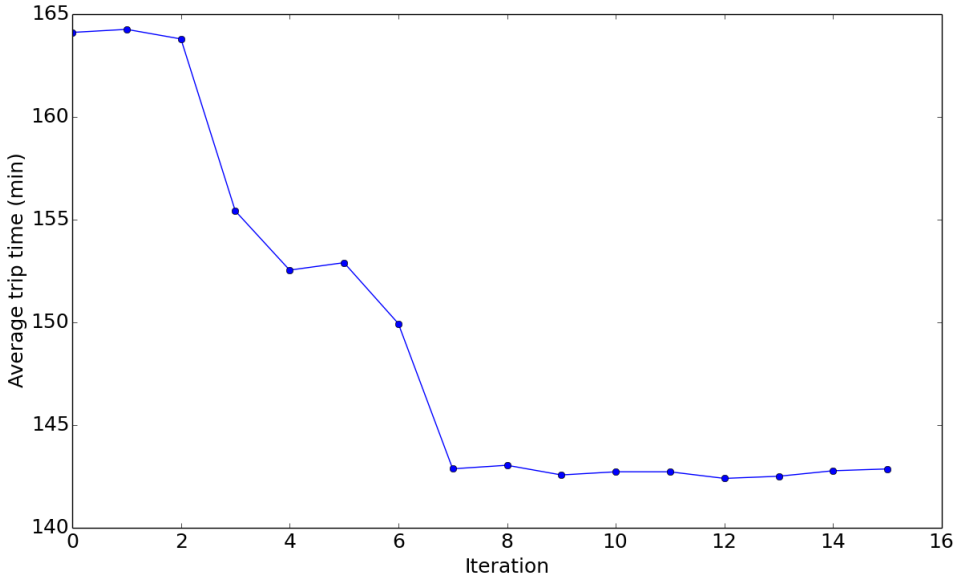


Figure 3.8: Convergence of FQI algorithm

learning, FQI can help save an additional 25% of the trip time during peak hours and around 8% during off-peak periods. And the difference between the trip costs on and off peak demand periods is much smaller, meaning that the new routing policy is more flexible and can resist the sharp changes in congestion to some extent.

One of the major motivations for us to work with continuous state space is to resolve the curse of dimensionality. As in the discrete state space, if we want to improve the performance by having more refined state representation, the problem's complexity will grow sharply. To illustrate the case, we conduct an experiment on the computational performances of both dynamic programming and FQI.

In Table 3.1, we report the CPU time and the resulting average trip cost at 7:00 am of these two methods with respect to different number of monitored links ahead of current node. As for dynamic programming, we do not include the time spent on model calibration (estimating the parameters based on trip data), which depends on the amount of data we use. Thus the reported time only includes the time taken to calculate the value functions based on the estimated parameters. And for FQI, we set the amount of trip data to be 100000, and the parameters for the Extremely Randomized Trees are kept as the same. Then we record the time needed until we see convergence. The results are the average of 5 runs.

As can be seen, if we increase $|\Omega(n)|$, which is the number of monitored links around node n, the size of the discrete state space for dynamic programming will grow exponentially and the problem soon becomes intractable. However, in the continuous state space, the dimension of the continuous state vector grows linearly with $|\Omega(n)|$. And non-parametric regression methods such as regression tree can handle high dimensions quite efficiently. Thus we can

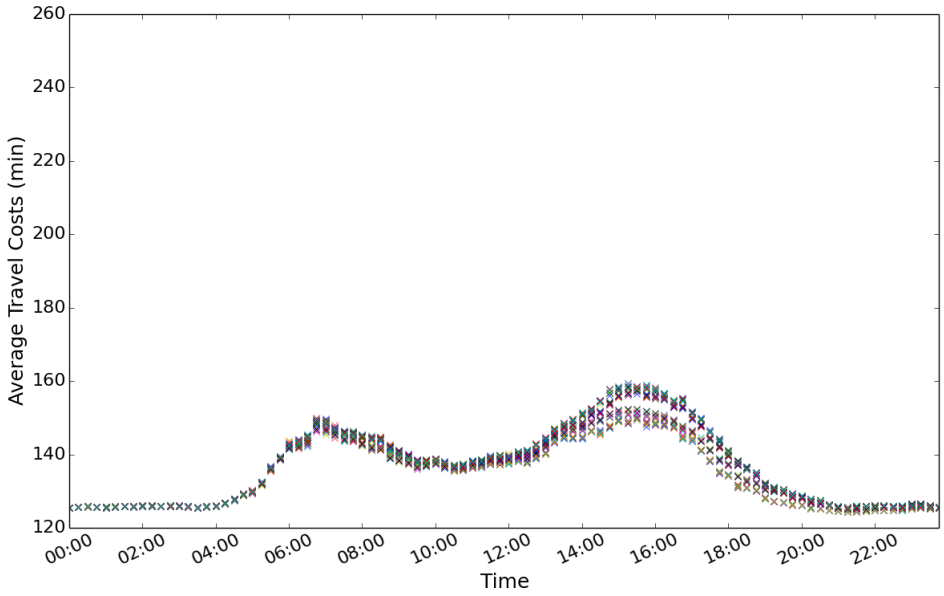


Figure 3.9: Average trip costs based on routing policy from FQI

Number of links ahead to monitor	Dynamic programming		FQI	
	CPU time (s)	Trip cost (min)	CPU time (s)	Trip cost (min)
1	1.8	207.1	8313.2	150.2
2	955.1	203.6	11425.4	142.7
3	$> 1 \times 10^6$	-	12650.7	146.3

Table 3.1: Computational performances of dynamic programming and FQI

see that the running time of FQI does not increase too much. Besides, if we look at the trip costs, monitoring two links ahead can already deliver some satisfying results. Actually, in the experiments on FQI, we can see that monitoring three links ahead causes the trip cost to be worse than monitoring only two links. The reason for this is that we are using the same amount of training data for each experiment, so if we do not provide more training data for higher dimensional space, the regression model might not be fully trained and thus results in worse performance. As for the difference between the trip costs of these two methods, we put off the discussion until next section.

Realizing that the amount of data is also one major factor which can decide the efficiency and accuracy of learning based methods such as FQI, we also look into the sensitivity of FQI's performance with respect to the amount of data. We choose to monitor two links ahead, and the other settings are kept as the same. The results are shown in Table 3.2.

As can be seen, the running time almost grows linearly with the amount of input data, and we see consistent improvements on trip cost with more and more data. This finding

Data size ($\times 10^4$)	CPU time (s)	Trip cost (min)
3	3523.1	149.8
5	5184.2	147.9
7	6991.4	146.4
10	11425.4	142.7

Table 3.2: Computational performances of FQI with respect to data size

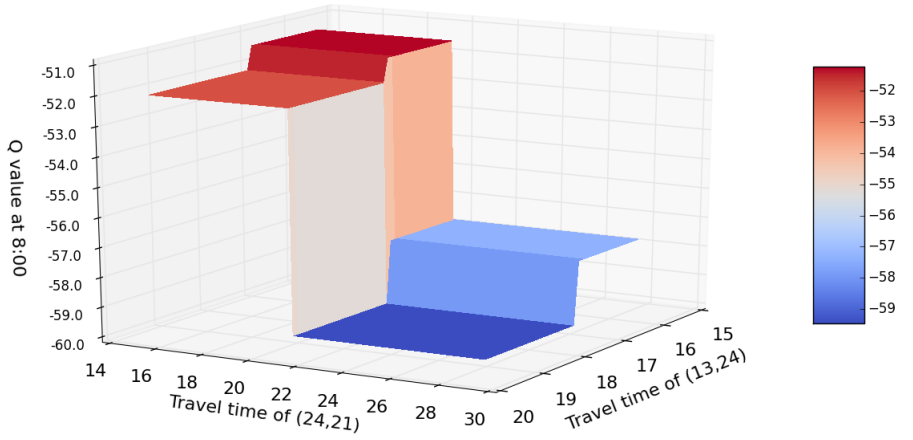
further supports the idea that non-parametric learning based methods are not restricted by model assumptions and data can be utilized more sufficiently. However, by comparing the running time of FQI here with that of dynamic programming for the case of monitoring two links ahead, we can notice that on average FQI takes longer time. This is typical for non-parametric learning based methods, since the advantage of flexibility and powerful performance comes with the cost of requiring more data and slower training process. While for traditional parametric methods, the advantage of speed (for lower dimensional space) comes with the cost of less complexity and poor fit.

Discussion

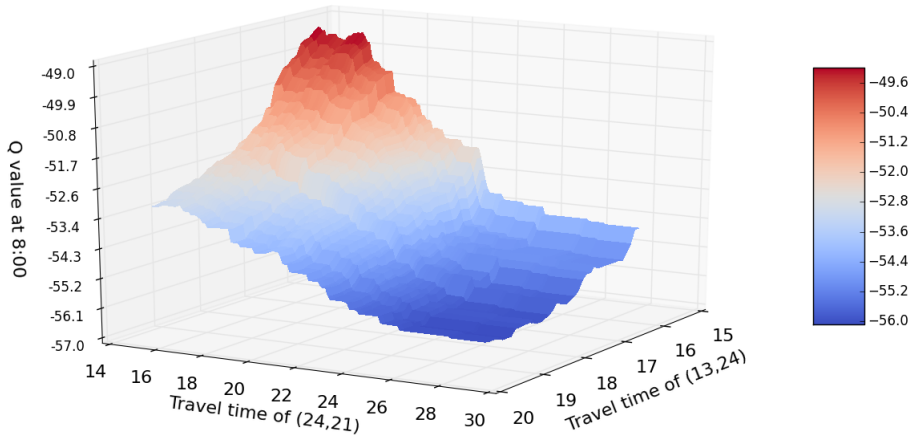
To look into the reason behind the improvement brought by FQI algorithm over Q learning, we can look at the Q functions developed in both methods. As mentioned before, we have built a separate Q function for each link in the network, and the function inputs include the time variable t and the congestion states vector W. However, these variables are discretized in Q learning while are allowed to be continuous in the FQI algorithm. Thus we have developed a group of discrete Q functions in Q learning and continuous Q functions in the offline FQI algorithm. Figure 3.10 shows the Q functions we have developed for link $(13, 24)$ in both methods, note that we have only included two links' travel costs as the inputs in order to visualize the function.

As can be seen, in the continuous state space, the Q function can reflect the influence of congestion on the expected travel cost in a more refined way. But if we choose to discretize the state space, we will certainly lose some information. Suppose the link travel costs have changed for a small value, the Q value in the discrete function might remain unchanged and the routing decision will stay the same as well. However, the continuous Q function can reflect any small change in the input space and the resulting routing strategy will be more responsive to any possible link congestion.

In order to validate the reasoning above, we also examined the routes that have been recommended by both methods. We simulated a number of trips starting from node 1 at the morning peak hour (8:00 am), when most of the links are highly possible to become congested. Since the network is stochastic and random, different routes might be taken during different trips. In Figure 3.11, we have shown the distribution of the final routes traversed by the agents in both methods. Different colors represent different routes and the



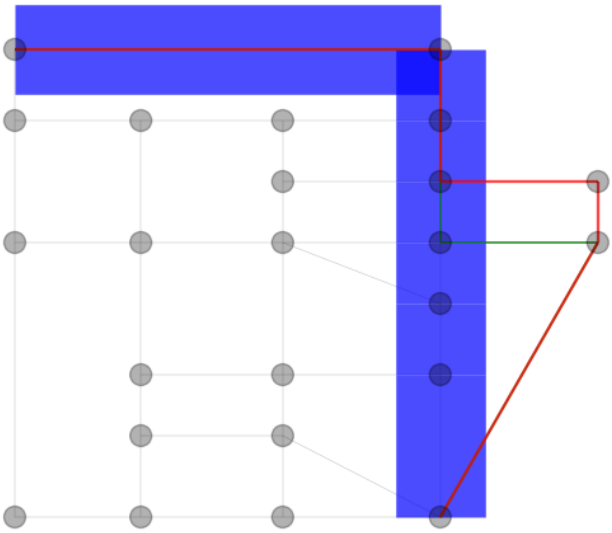
(a) Q learning



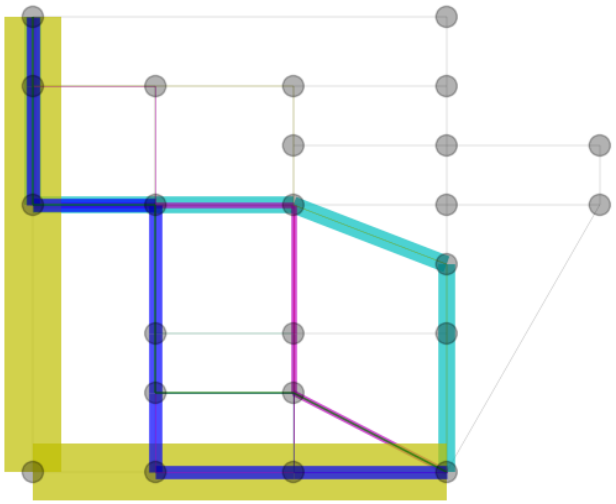
(b) Fitted Q iteration

Figure 3.10: Developed Q functions in Q learning and fitted Q iteration

bandwidths correspond to the frequencies of the paths being taken. First, we can notice that a more variety of routes have been recommended in FQI and their frequencies are more evenly



(a) Q learning



(b) Fitted Q iteration

Figure 3.11: Recommended paths during morning peak hour in Q learning and fitted Q iteration

distributed. While in Q learning, less routes have been used and one path almost dominates the others. The result just proves that the routing policy in Q learning is less sensitive to the changes in the link travel costs, which is due to the coarser representation of the congestion impacts by the discrete Q function. Moreover, we can notice that Q learning and FQI have led the drivers into quite different paths. This is also caused by the discretization of the Q function in Q learning. Since our routing policy is solely based on comparing the Q values

of adjacent links to decide which way to go next, the discrete and rough representation of the Q function might cause the comparison to have a entirely different result.

However, we should be aware that the significant improvement of travel cost brought by FQI method (from almost 200 minutes to 150 minutes during peak hour) might not be realistic in real world. In our simulation, we set 15 minutes as the discrete time interval length. And in order to have adequate state transitions for different links, we set most of the links travel times greater than 15 min. Some link can have travel cost as high as 80 minutes, which can hardly be true in real networks. And having links with such high travel costs might exaggerate the improvement we have shown with the FQI method. Since with the discrete methods, if the driver is misled into such a long link with some inaccurate estimation of the value function, the penalty can be quite high because there is a large difference between congested and uncongested travel time and he cannot switch to other roads within a long period. Although the benefits of FQI method might be amplified in our simulated network compared to real traffic networks, the advantages of using continuous state space and approximate value functions are well supported and should be realized in real world applications.

3.7 Conclusion

For the adaptive routing problem in stochastic and time-dependent network, most previous work has been focusing on developing parametric models for the problem and looking for efficient algorithms to solve these models as well. However, some of the assumptions made in these models can be difficult to validate in real networks and the curse of dimensionality still exists if they are applied in large traffic networks. Thus the parametric model based methods might be impractical in finding the adaptive routing policies in real world. This study provides an encouraging evidence that Reinforcement Learning may be an effective non-parametric model-free method to solve the adaptive routing problem in STD networks. In this method, we do not rely on any prior knowledge or assumptions of the model, instead we will directly update the state action values (Q function) based on the information gathered by travelling in the network. And then we can infer from the Q function to get the best routing policy.

Both the online Q learning method for discrete state space and the offline fitted Q iteration method for continuous state space have been presented for the problem. In the Q learning method, we store the Q value for each state action pair in a tabular form and update the Q values as the agent is travelling in the network. Since no knowledge on the network is assumed to be known at the beginning, we also need to consider the balance between exploration and exploitation. While the offline fitted Q iteration algorithm combines continuous approximation of the value functions with an iterative batch-mode learning process from an offline generated dataset to get the best routing policy. The continuous approximation makes it possible to mitigate the curse of dimensionality and get a more refined representation of the Q function. Also tree-based regression method is adopted as the function approximator

due to its modeling flexibility and computational efficiency. Since there is no requirement on the way the offline training dataset is generated, it is a more feasible method when applying to the real networks.

A small case study on a mid-size network is conducted to demonstrate the performances of different methods. It is shown that Q learning outperforms the traditional dynamic programming method in most cases, especially during peak demand periods. Such an improvement is owing to the fact that Q learning does not rely on the calibration of any parametric models, thus it will not suffer from the approximation taken in the calibration process, and data can be utilized more efficiently by updating the Q values directly. However, it still suffers from the curse of dimensionality and fails to find the optimal routing policy since it is based on the discrete state space while the traffic congestion states can be continuous. And as expected, the FQI algorithm with the tree-based function approximation further improves the routing policy and delivers a more flexible strategy especially during the peak hours. The approximated continuous Q function is more sensitive to the changes of link travel costs, and the resulting routing policy is also more responsive to any possible congestion in the network.

So when applied to real traffic networks, we believe that FQI can be a more suitable method than those methods based on discrete state space. However, we also need to consider some of the drawbacks when we apply Reinforcement Learning to solve the adaptive routing problem. For instance, as a non-parametric method, the learning process in RL can take much more time than traditional parametric methods. Thus it is infeasible to incorporate the RL framework for the real-time query of routing between any random OD pair. Instead, we can solve for the best routing policies to some predetermined popular destinations in advance and use them as guidelines afterwards. Besides, as an offline learning method, it requires us to store all the historical trips as training data, which may take a lot of storage space. Moreover, although the offline FQI algorithm is shown to outperform other methods in delivering more stable routing strategies, it comes at a price of lacking convergence guarantee when combined with the Extremely Randomized Trees regression method. Future research can explore other function approximation methods for FQI to deliver better accuracy/efficiency tradeoff.

Chapter 4

Dispatch of Autonomous Vehicles for Taxi Services

4.1 Introduction

As can be seen from our review in 2.2, for the problem of dispatching autonomous vehicles for taxi services, most previous works focus on model-based methods to solve for the optimal rebalancing strategy. In their models, they use a finite set of parameters to represent the network dynamics, such as the customer demands and travel times. And before solving the model, they have to first estimate these parameters based on historical data. One major drawback of these methods, however, is that strong assumptions have to be made on the network (e.g. Markovian property, constant arriving rate), and usually these assumptions are hard to validate in real networks. Secondly, some of the proposed methods still suffer from the curse of dimensionality and do not scale well in large systems. With these concerns, the aim of our study in this chapter is to explore the use of model free methods to solve for the rebalancing strategy of the autonomous taxi fleet. Specifically, we will incorporate the framework of Reinforcement Learning (RL), which is primarily concerned with how to obtain an optimal policy when the model of Markov decision process (MDP) is not available. Therefore, instead of relying on any prior information of the model, the RL learning agent will interact with the network directly and update the control strategy directly. Recently, there are a lot of studies working on the application of RL methods in the domain of transportation engineering, interested readers may refer to [59], [46], [2] for traffic signal control, [48] for adaptive routing, and [85], [78] for traffic management. Among all RL methods, policy gradient [68] is adopted in our study as described in section 4.3.

The remainder of this chapter is organized as follows. In section 4.2, we formulate the problem and present our basic assumptions. In section 4.3, we introduce the idea of actor-critic method and the adaptation we make, then we derive the theoretical upper bound of the total rewards we can get if we assume the dynamics of the system are deterministic and known to us (section 4.4). Also, a different scenario is presented in section 4.5, where user

priority is taken into account. Moreover, section 4.6 shows the experimental results and discussion of the RL methods under different scenarios. And section 4.7 summarizes the study of this chapter.

4.2 Problem Formulation

In this section, we describe the problem of taxi repositioning, as well as the environment setup in our simulation.

Assume we have a service region discretized into a set $\mathcal{N}$ of disjoint zones, each of which can be represented by a node (like a "station") in a directed graph. Further assume time is represented by discrete time intervals of size Δt. The period of time under consideration is denoted as $\mathcal{T} = [1, 2, \ldots, T]$. Suppose we are providing taxi services to the region. At the end of each time interval t, we have to decide the number of vehicles to be dispatched from zone $i \in \mathcal{N}$ to zone $j \in \mathcal{N}$, which is denoted as x_{ijt}. Suppose the number of waiting passengers who want to go from zone i to zone j at this time t is p_{ijt}, and the number of available (empty) vehicles at zone i is v_{it}.

Thus if we have decided x_{ijt}, we will know the number of passenger calls that can be answered at this time step t. Denote y_{ijt} as the number of passengers that can be served from zone i to j at time t, then we have $y_{ijt} = min(x_{ijt}, p_{ijt})$. Namely, if $x_{ijt} < y_{ijt}$, then the supply is less than the demand, and only a portion of the passenger calls can be answered; and if $x_{ijt} >= y_{ijt}$, we have dispatched more vehicles than the current number of waiting passengers, so all the passengers' requests can be satisfied and we also let some empty vehicles to travel from i to j to meet the future potential demand in zone j.

For simplicity, we assume the discrete time interval Δt is very small, and all the deployment of vehicles happen at the end of each time interval. Therefore, if a waiting passenger is not served at time t, he/she will have to at least wait until $t + 1$ to be served, i.e. no dispatching arrangement will happen between t and $t + 1$. And all serving vehicles will become available again after dropping passengers at the destination zone. And for now, we further assume we will not lose any passengers even if we let them wait for a long time.

The objective of our dispatching system is to provide the optimal vehicle dispatch strategy at the lowest possible operational costs. On the passenger side, we assume there are costs associated with the waiting time experienced by all the passengers. Let δ_{ijt} be the cost of letting one customer who wants to go from i to j wait for one unit of time interval Δt at time t. Thus the waiting time costs from time t to $t + 1$ will be equal to $\sum_{i,j \in \mathcal{N}} (p_{ijt} - y_{ijt}) \cdot \delta_{ijt}$. One the other hand, there are costs resulting from repositioning the empty vehicles between different zones. We assume the cost of repositioning an empty vehicle from zone i to zone j at time t is equal to c_{ijt}. Thus the repositioning costs at time t is $\sum_{i,j \in \mathcal{N}} (x_{ijt} - y_{ijt}) \cdot c_{ijt}$. Since we assume we will not lose any customers, we ignore the rewards and costs of serving these customers' demand.

With the above assumptions and formulations, we then define the state space and action space of the problem under the structure of reinforcement learning. The state space is defined

as:

$$\mathbf{S} := \{(t, P_t, V_t) : t \in \mathcal{T}, P_t \in \mathbb{R}^{+n \times n}, V_t \in \mathbb{R}^{+n}\}$$

Each state $s = (t, P_t, V_t)$ is characterized by the current time interval t, the waiting passenger demand vector $P_t = \{p_{ijt} : i \in \mathcal{N}, j \in \mathcal{N}\}$, and the available vehicle count vector $V_t = \{v_{it} : i \in \mathcal{N}\}$. The corresponding action space for state s is defined as:

$$\mathbf{A}(s) := \{x_{ijt} : \sum_{j \in \mathcal{N}} x_{ijt} = v_{it}, i \in \mathcal{N}, j \in \mathcal{N}, x_{ijt} \in \mathbb{R}^+\}$$

where x_{ijt} is the number of available vehicles to be dispatched from zone i to zone j at the end of time interval t, and is the decision variable of the problem. As we have discussed above, the reward we get from time t to $t+1$ would be the negative of the costs we have incurred, which consists of the waiting time costs and the costs of repositioning empty vehicles, and is formulated below.

$$r_t = -\sum_{i,j \in \mathcal{N}} [(p_{ijt} - y_{ijt})\delta_{ijt} + (x_{ijt} - y_{ijt})c_{ijt}]$$

To solve the above dynamic sequential decision problem, there are several key challenges that we have to tackle. First, to reduce the dimensionality of the state and action space, we allow the state and action vectors to be in continuous forms, i.e. p_{ijt}, v_{it} and x_{ijt} are all continuous variables. With such continuous state and action space, those value-function based reinforcement learning methods such as Q-learning might become intractable. So we need to focus on policy-based reinforcement learning methods. Besides, in our problem, the feasible action space is state-dependent, i.e. the number of vehicles that are dispatched from a certain zone should be equal to the number of available vehicles within that zone.

4.3 Actor-Critic Algorithm

Policy gradient methods are reinforcement learning techniques that rely on optimizing parametrized policies with respect to the expected return (long-term cumulative reward) by gradient descent. They do not suffer from many of the problems that traditional value-based reinforcement learning methods might have, such as the complexity arising from continuous states and actions. The general idea of policy gradient is that, by generating samples of trajectories (sequences of tuples of state, action and reward) from the environment based on the current policy function, we can collect the rewards associated with different trajectories, then we can update our parametrized policy function such that high-reward paths will become more likely and low-reward paths become less likely. One advantage of policy gradient methods is their strong convergence property, which is naturally inherited from gradient descent methods. However, since the sampled rewards usually have very large variances, the estimated gradients can also vary a lot and thus make the vanilla policy gradient method less efficient to learn.

Actor-critic methods combine the advantages of actor-only (policy function only) and critic-only (value function only) methods. Similar to actor-only methods, actor-critic methods are able to deal with continuous action space; but the large variance in the policy gradients of actor-only methods is tackled by adding a critic [34]. The critic approximates and updates the value function using samples. The value function is then used to update the actor's policy parameters in the direction of performance improvement. Since the critic is modeled by a bootstrapping method, it reduces the variance so the learning is more stable than vanilla policy gradient methods.

Figure 4.1 below shows the schematic structure of an actor-critic algorithm. The learning agent consists of a critic agent and an actor agent. The actor is responsible for generating actions based on the states of the environment, and the critic evaluates the value function of the current policy by observing the feedback (states and rewards) from the environment. Then the information of advantage (the improvement when compared with the average value of the current state) is sent to the actor to improve the current policy function unit.

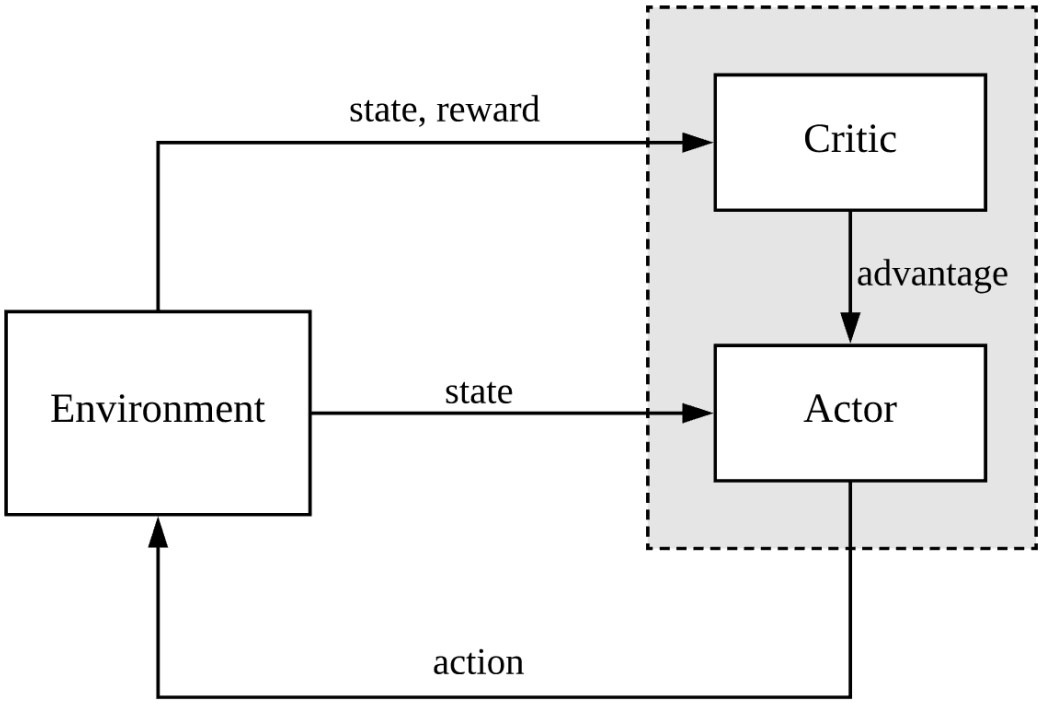


Figure 4.1: Schematic overview of actor-critic algorithm

Suppose the policy function (actor) is denoted as $\pi_\theta(a|s)$, where a is the action, s is the state, and θ represents the actor's parameters to be updated and learned. Further denote the value function (critic) corresponding to policy $\pi_\theta(a|s)$ as $V_\phi^\pi(s)$, which is parametrized by ϕ. Then the actor-critic algorithm works as described in Algorithm 3.

Algorithm 3 Batch actor-critic algorithm

procedure ACTOR-CRITIC
Initialize: $\pi_{\theta_0}(a|s)$
for each episode **do**
1. sample s_i, a_i, r_i, s'_i from $\pi_\theta(a|s)$ (run it in the simulator)
2. fit $\hat{V}^\pi_\phi(s)$ to sampled reward sums
3. evaluate the advantage as $\hat{A}^\pi(s_i, a_i) = r_i + \hat{V}^\pi_\phi(s'_i) - \hat{V}^\pi_\phi(s_i)$
4. calculate $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta log(\pi_\theta(a_i|s_i))\hat{A}^\pi(s_i, a_i)$
5. $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

end for
end procedure

In the algorithm, the advantage $\hat{A}^\pi(s_i, a_i)$ in step 3 represents how much better off the corresponding action is compared with the average value function we have estimated. Therefore, by multiplying the gradient of the policy function with the advantage, the algorithm updates the parameters so that the high rewarded actions become more likely. The critic function $\hat{V}^\pi_\phi(s)$ is estimated by a bootstrapping method, thus it can reduce the variance of the parameter estimation as well.

Although we can choose any parametric functions for the actor function $\pi_\theta(a|s)$ and critic function $V^\pi_\phi(s)$, in this study, we pick neural networks to represent both functions in consideration of its modelling flexibility. However, one key problem is that we have hard constraints on the outputs (actions) of the policy function, i.e. the number of vehicles dispatched from each zone should be non-negative and sum up to the total number of available vehicles in that zone. Thus we have to post-process the output from the action function to make it feasible. As shown in Figure 4.2, suppose the output vector from the policy function $\pi_\theta(\cdot|s)$ is a, in which a_{ij} corresponds to the edge between zone i and zone j. Then we can apply a transformation function for each zone such that

$$x_{ij} = v_i \cdot \frac{|a_{ij}|}{\sum_{k\in\mathcal{N}} |a_{ik}|}$$

Then the resulting actor vector x is a feasible action given the current state vector s.

4.4 Theoretical Bound

In this section, we aim to look at the case where we have full information on the travel demand and the system dynamics, both of which are deterministic and known. We can then formulate the dispatching problem as an optimal control problem to maximize the total rewards and solve the optimal dispatching strategy. Then we compare the resulting rewards with the rewards we obtain from the model-free reinforcement learning method. The definition of the integer program is as follows.

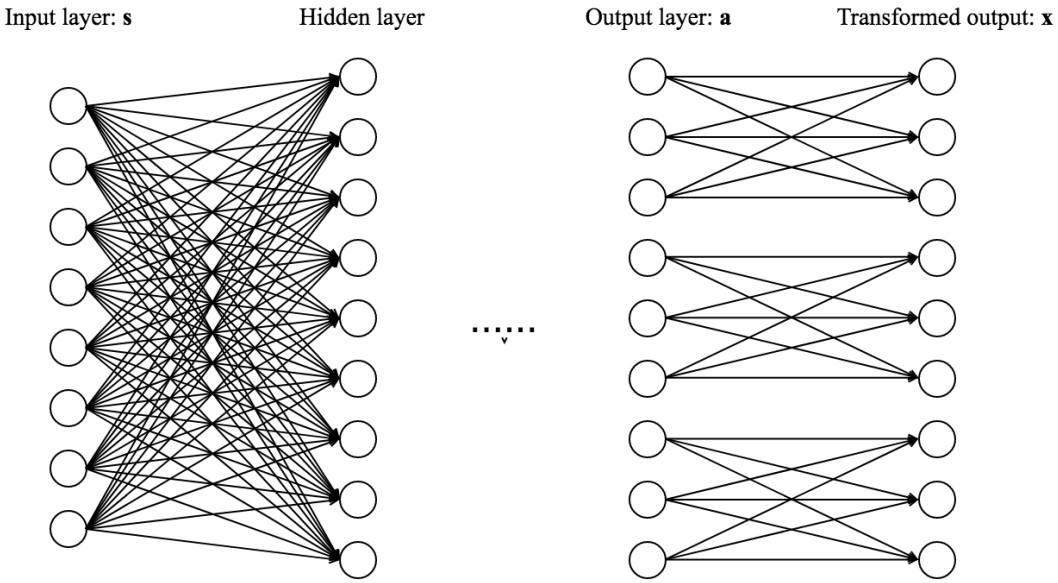


Figure 4.2: Post-processing of action function outputs

Nomenclature

Parameters

δ_{ijt} The cost of letting customer who wants to go from i to j wait for one unit of time at time t

λ_{ijt} Number of ride requests from zone i to zone j at time t

τ_{ijt} The travel time from zone i to zone j at time t

c_{ijt} The cost of repositioning an empty vehicle from zone i to zone j at time t

n_i The initial number of vacant vehicles in zone i

Sets

$\mathcal{A}_{it} = \{(j, t') : t' + \tau_{jit'} = t\}$ Set of departure zones and times that would let vehicle arrive in zone i at time t

Variables

p_{ijt} Number of outstanding passengers that are waiting to go from zone i to zone j at time t

v_{it} Number of vehicles available in zone i at the beginning of time t

x_{ijt} Number of vehicles that are dispatched from zone i to zone j at time t

y_{ijt} Number of passengers going from zone i to zone j that are served at time t

$$\min_{x} \quad \sum_{i,j\in\mathcal{N}} \sum_{t\in\mathcal{T}} (p_{ijt} - y_{ijt})\delta_{ijt} + (x_{ijt} - y_{ijt})c_{ijt}$$

$$\text{s.t.} \qquad y_{ijt} = min(x_{ijt}, p_{ijt}) \qquad \forall i,j \in \mathcal{N}, t \in \mathcal{T} \quad (1)$$

$$p_{ij(t+1)} = p_{ijt} - y_{ijt} + \lambda_{ij(t+1)} \qquad \forall i,j \in \mathcal{N}, t \in \mathcal{T} \quad (2)$$

$$v_{it} = \sum_{j\in\mathcal{N}} x_{ijt} \qquad \forall i \in \mathcal{N}, t \in \mathcal{T} \quad (3)$$

$$v_{i0} = n_i \qquad \forall i \in \mathcal{N} \quad (4)$$

$$v_{it} = \sum_{(j,t')\in\mathcal{A}_{it}} x_{jit'} \qquad \forall t > 0, i \in \mathcal{N} \quad (5)$$

$$p_{ij0} = \lambda_{ij0} \qquad \forall i,j \in \mathcal{N} \quad (6)$$

$$x_{ijt} \in \mathbb{N} \qquad \forall i,j \in \mathcal{N}, t \in \mathcal{T} \quad (7)$$

As can be seen in the above formulation, the objective function consists of the waiting time costs for the passengers and the costs of repositioning empty vehicles. The first constraint states that the number of served customers at any time is either equal to the number of waiting customers (when enough vehicles are dispatched) or the number of dispatched vehicles (when supply of vehicles is less than customers). Constraint (2) ensures the conservativeness of the passenger demand, and constraints (3) and (5) enforce that the number of arriving vehicles must be equal to the number of departing vehicles. We assume that at the beginning of the planning horizon, the initial number of available vehicles and waiting passengers in each zone are given, as shown in (4) and (6). And (7) constrains the decision variables to be non-negative integers.

Notice that in our formulation, we treat idling in the same zone as a special case of repositioning. However, we can alter the parameters δ_{ijt} to reflect any realistic scenarios. For example, parking in the downtown area can be very expensive during daytime, thus we can impose higher δ value to encourage the vacant vehicles to move out of the area during that time.

4.5 Different Scenario: Considering User Priority

One assumption we make in the above problem formulation is that we will not lose any passengers even if we let them wait for a long time. Thus when we decide the optimal dispatch strategy, we do not differentiate between passengers who have already waited for a long time and those who just started waiting. As a result, in our dispatching system, even if there are some areas where passengers' requests have been dismissed for a long period of time, we might still send vehicles to other areas where we can maximize the total rewards of the system.

Such an assumption, however, can hardly be true in the real world and thus, we need to take into account users' tolerance of waiting time. One possible solution is to let the

waiting time cost δ be an increasing function of the actual waiting time ω of any passenger, i.e. for those with a larger ω, the penalty of waiting should be higher. However, if we make the penalty function too refined, we will have to distinguish between every passenger and it might become difficult to aggregate the attributes of passengers of one zone.

Without loss of generality, we let the waiting time penalty function δ be a piecewise constant function. For a customer who is waiting to go from zone i to zone j at time t, if his/her total waiting time is ω, then the cost of letting him/her wait for one more unit of time is

$$\delta_{ijt}(\omega) = \begin{cases} \delta^1_{ijt} & \text{, if } \omega \leq \Omega \\ \delta^2_{ijt} & \text{, if } \omega > \Omega \end{cases}$$

where Ω is a threshold we set beforehand. We can imagine that if $\delta^2_{ijt} \gg \delta^1_{ijt}$, we are imposing a very large penalty if we let customers wait more than Ω units of time, thus we can ensure some level of service with regard to the maximum waiting time.

In order to solve the problem with the above reformulation, we have to redefine the state space and the reward function. For any origin destination pair (i, j) at any time t, the number of waiting passengers p_{ijt} can be divided into two groups: p^1_{ijt} and p^2_{ijt}, where p^1_{ijt} is the number of outstanding passengers who have waited less than Ω units of time and p^2_{ijt} is the number of those who have waited more than Ω. We can call these p^1_{ijt} passengers the "patient passengers", and those p^2_{ijt} passengers the "impatient passengers".

If we have decided x_{ijt}, i.e. the number of vehicles to dispatch from zone i to j at time t, we still have $y_{ijt} = min(x_{ijt}, p_{ijt})$, where y_{ijt} is the number of customers that can be served. Let's further assume that we will always first serve those "impatient passengers", so p^2_{ijt} will be covered first. Thus after we pick up all the y_{ijt} passengers, the number of "impatient passengers" will become as $p^2_{ijt} - min(p^2_{ijt}, y_{ijt})$, and the number of "patient passengers" will turn into $p^1_{ijt} - max(0, y_{ijt} - p^2_{ijt})$. Therefore the associated waiting time cost at this time is: $\sum_{i,j\in\mathcal{N}}[p^2_{ijt} - min(p^2_{ijt}, y_{ijt})] \cdot \delta^2_{ijt} + [p^1_{ijt} - max(0, y_{ijt} - p^2_{ijt})] \cdot \delta^1_{ijt}$, and the repositioning cost stays the same, i.e. $\sum_{i,j\in\mathcal{N}}(x_{ijt} - y_{ijt})c_{ijt}$. Now the new state space is defined as:

$$\mathbf{S} := \{(t, P^1_t, P^2_t, V_t) : t \in \mathcal{T}, P^1_t \in \mathbb{R}^{+n\times n}, P^2_t \in \mathbb{R}^{+n\times n}, V_t \in \mathbb{R}^{+n}\}$$

Each state $s = (t, P^1_t, P^2_t, V_t)$ is characterized by the current time interval t, the demand vector for those "patient passengers" $P^1_t = \{p^1_{ijt} : i \in \mathcal{N}, j \in \mathcal{N}\}$, the demand vector for those "impatient passengers" $P^2_t = \{p^2_{ijt} : i \in \mathcal{N}, j \in \mathcal{N}\}$, and the available vehicle count vector $V_t = \{v_{it} : i \in \mathcal{N}\}$. While the action space stays the same.

4.6 Case Study

In this section, we first present the set up of the case study, then we demonstrate the performance of the above actor-critic method, followed by the comparison with the theoretical upper bound derived by the integer programming problem. Lastly, we discuss the case where we take into account passengers' tolerance of waiting.

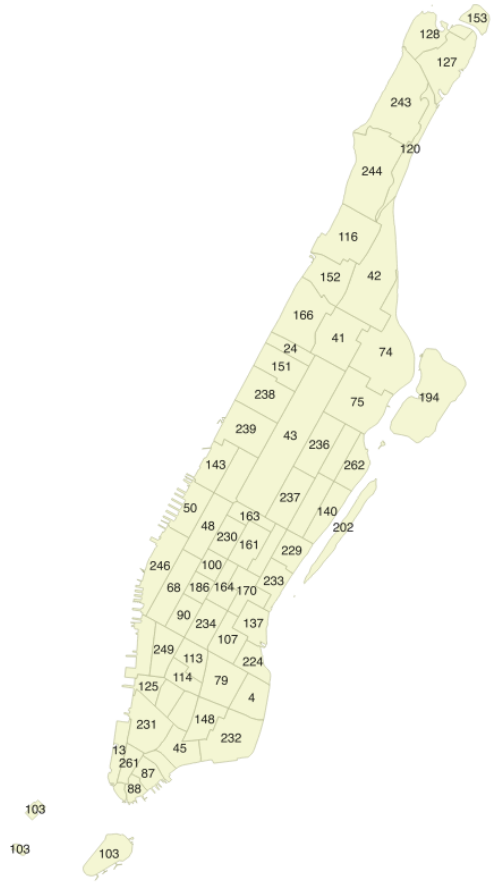


Figure 4.3: Manhattan taxi zone partition

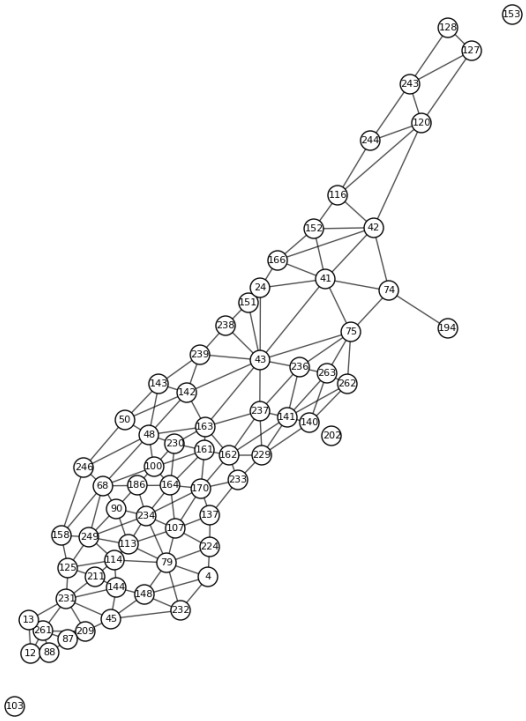


Figure 4.4: Network by representing each zone with a node

Experiment Setup

In our simulation case study, we choose to look at the area of Manhattan and the yellow taxi trip record data in the region to see how the travel demand varies across regions and along the day. As shown in Figure 4.3 and Figure 4.4, the Manhattan area is partitioned into 64 zones by NYC TLC (Taxi & Limousine Commission), and we can represent each service zone as a node.

As for the yellow taxi trip dataset, it includes fields capturing the pick-up and drop-off times/locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts for the taxi trips in the New York City. Based on the data, we obtain the records of June 2016 and estimate the travel demand between each pair of zones in the region along a day. By looking at the trip record data, we can notice the obvious imbalance of demand distribution, as shown in Figure 4.5 and Figure 4.6. First, the travel

demand of zone 161 (which is midtown business area) is much higher than that of zone 116 (which is upper town residential area). Furthermore, for these two regions, there is always a gap between the arrival and departure rates, which are also changing along a day. For business area such as zone 161, we observe higher arrival rate in the morning but higher departure rate in the evening. And for residential area such as zone 116, most of the trips are happening in the evening.

In order to reduce the computational burden of the simulation, we have the following three simplifications. First of all, we aggregate the taxi zones into larger zones such that we get a network of smaller size. In particular, we partition the region into 8 service zones as shown in Figure 4.7. Second, we divide a day into 12 time intervals, and we calculate the average number of ride requests for every time interval on a average day with the data. Notice that in our simulation, the data only helps us come up with the daily demand distribution to reflect "imbalance" in real traffic networks. Third, we assume by the end of each day, all vehicles will return to their origins so the initial number of vehicles in each zone will be the same at the beginning of different days. Note that these simplifications are solely for the purpose of reducing computational time and numerical validation of our approach. Our methods, however, can be generalized to any size of network and time intervals given enough computational power.

Without loss of generality, all other parameters, such as travel times and waiting costs, are set by the authors manually. Notice that the waiting time penalty is much higher than the penalty of repositioning empty vehicles so that we can achieve higher level of service for the customers. And our goal is to decide the number of vehicles to dispatch between each pair of zones at the beginning of each time interval.

Model Performance

In our case study, we choose a fully connected neural network of 4 hidden layers for both the actor function and critic function. And there are 128 units at each hidden layer. After experimenting with different setup of parameters, we choose 5×10^{-5} as the learning rate and 1024 as the trajectory batch size for each iteration.

First, in order to compare the performance of the actor critic algorithm with the theoretical upper bound we have derived in section 4.4, we let the travel demand be deterministic in our simulator, i.e. from day to day there are a fixed number of passengers who need to travel between each pair of zones at a certain time of day. Thus we can solve for the optimal dispatching strategy based on the integer programming (IP) model we have formulated. On the other hand, we can track the converging process of the actor-critic method and then compare the total rewards with the optimal value. And the results are shown in Figure 4.8. As can be seen, the average return (negative total cost) from our RL approach keeps improving at each iteration. And the final converged value is around -1.38×10^5. Compared with the optimal value of -1.33×10^5 we obtain from the IP model, the converged value from RL approach is very close to the actual optimal value.

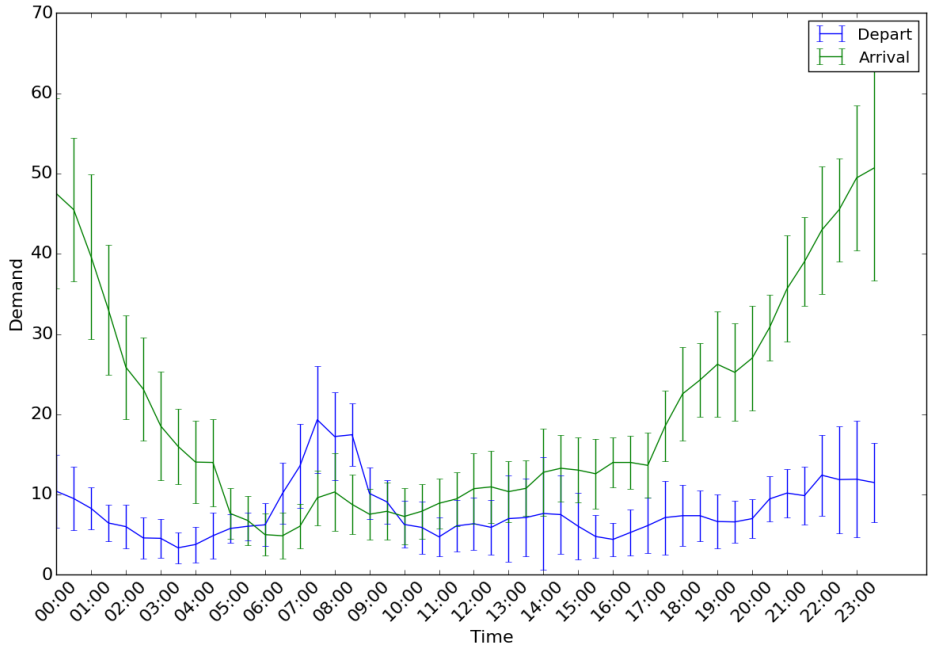


Figure 4.5: Arrival and departure rates for zone 116

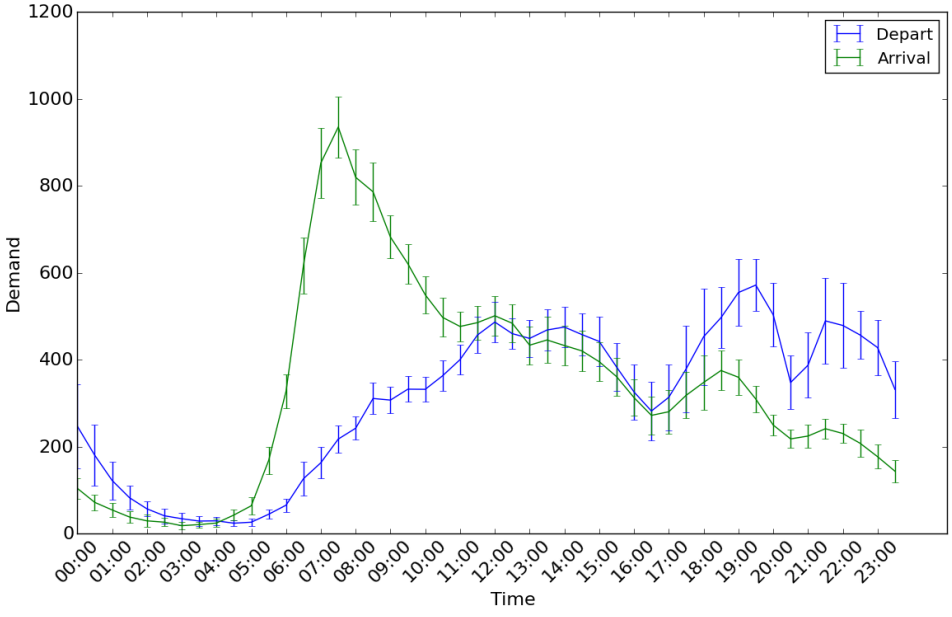


Figure 4.6: Arrival and departure rates for zone 161

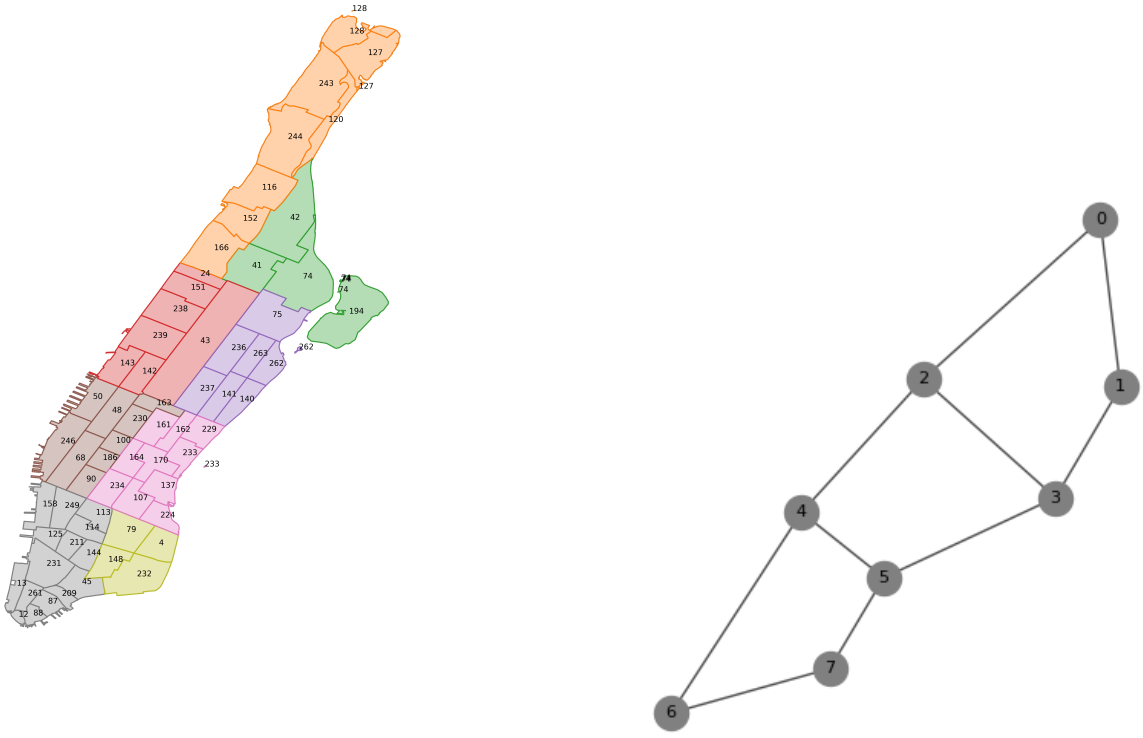


Figure 4.7: Partition of Manhattan into 8 zones

Furthermore, to test the robustness of the performance of our RL method, we can allow some stochasticity in the travel demand realization. In particular, we prepare two different travel demand distributions (e.g. distributions for weekday and weekend), and on each day we randomly pick one travel demand profile for the network. Notice that the total number of trips is the same for these two demand distributions, and the chance that we pick either of them is the same. The RL learner has no idea of this setup and it starts learning without any prior information of the network. At the same time, based on the integer programming method, we solve for the optimal solution to get the upper bound of the total return under these two different travel demand distributions. And the results are shown in Figure 4.9.

As can be seen, the optimal solution we get for weekday demand profile is around -1.33×10^5, and for weekend profile it is around -1.29×10^5. Thus if the real demand distribution is randomly chosen from the two, the optimal value we can achieve should be some value between -1.33×10^5 and -1.29×10^5. With our RL method, the total return converges to about -1.47×10^5, which is close to these two upper bounds. Therefore, when the travel demand is stochastic and unknown to us beforehand, the actor critic method, although may

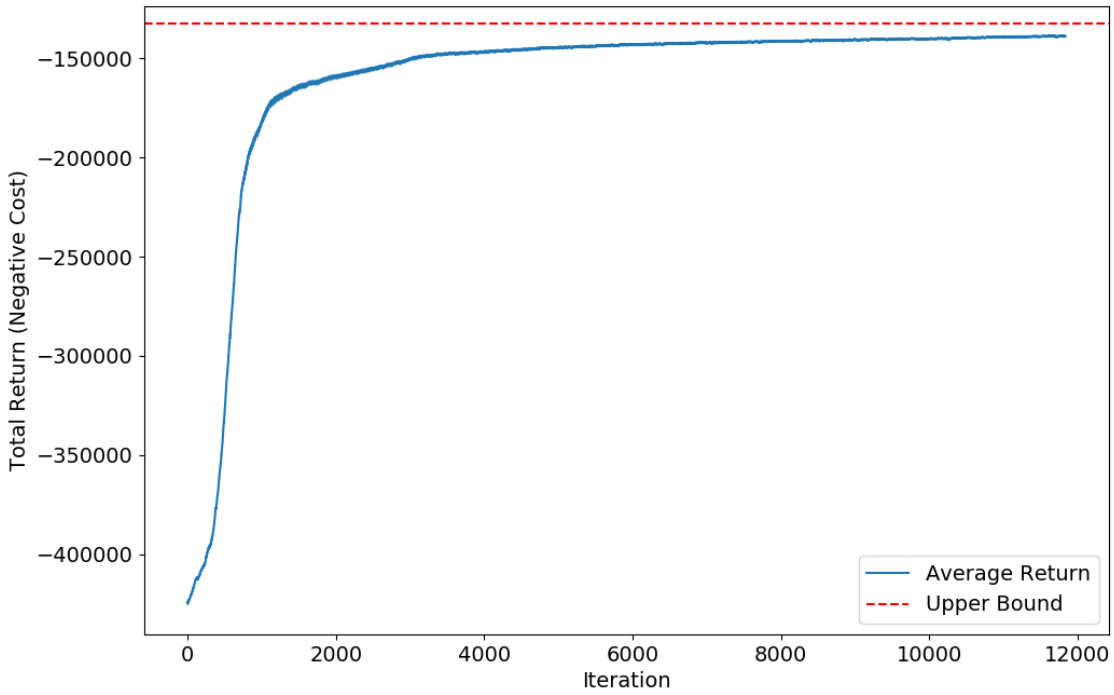


Figure 4.8: Performance of Actor-Critic method Under Deterministic Demand

not give the theoretical optimal, can still provide satisfying results. And we can be sure that in real traffic networks, the network dynamics (demand, travel time etc.) are highly complex and stochastic. And it would be really difficult to establish models to reflect such complicate dynamics. In this case, the proposed model free reinforcement learning method (i.e., actor critic) is an efficient alternative way to solve for reliable and close-to-optimal solutions.

Considering User Priority

As we have discussed in section 4.5, if we need to provide higher level of service to the customers, we need to impose higher penalty on the waiting time of those passengers who have already waited for a long time, i.e. the "impatient passengers". In our case study, we set the threshold $\Omega = 1$, i.e. those who have waited more than one time interval are treated as "impatient passengers" and others are "patient passengers". Furthermore, if we take into account user priority, we set the waiting time penalty as $\delta^2_{ijt} = 5 \times \delta^1_{ijt}$, so the penalty on the waiting time of "impatient passengers" is much higher. Otherwise, if we do not consider user priority, we will let $\delta^2_{ijt} = \delta^1_{ijt}$.

We investigate the two cases in which we either consider user priority or not, and keep

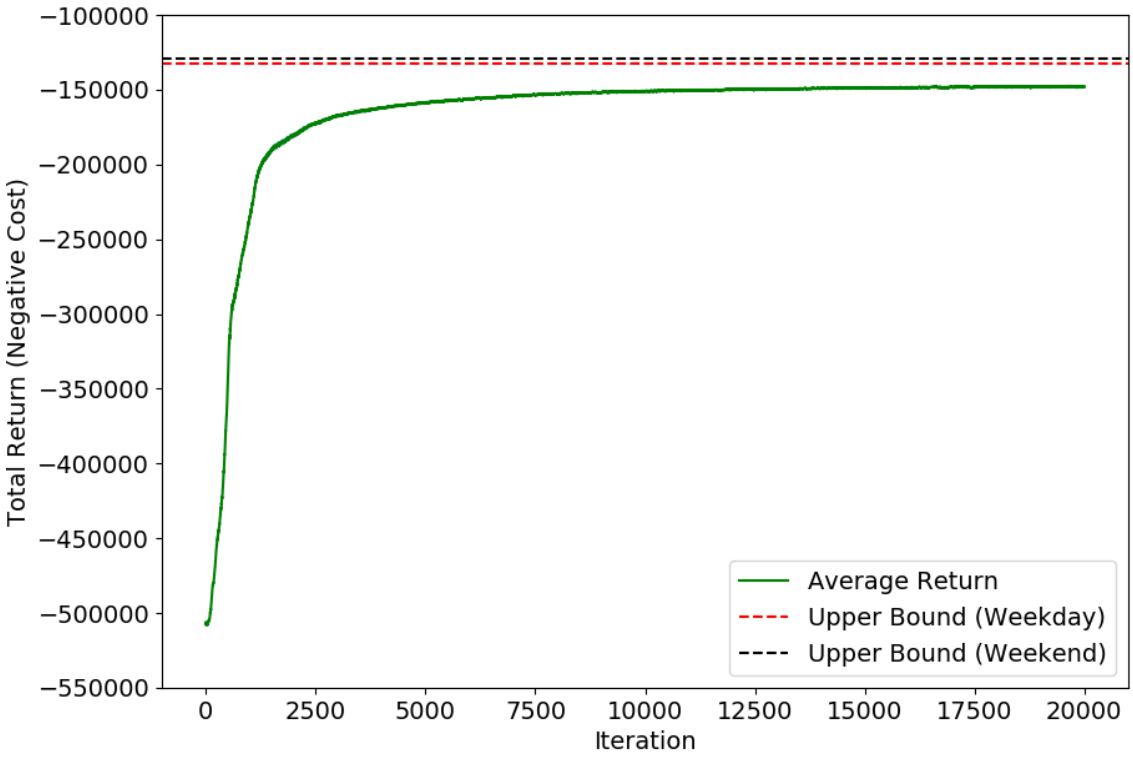


Figure 4.9: Performance of Actor-Critic method Under Stochastic Demand

track of the total waiting time of all the passengers and the waiting time of the "impatient passengers" as well. The results can be found in Figure 4.10 and 4.11. As can be seen, when we consider user priority and choose to first serve those "impatient passengers", the total waiting time of all the passengers would be larger, however, the waiting time of these "impatient passengers" is significantly smaller. This is under our expectation, since if we do not differentiate between passengers based on their waiting time so far, then we can make decisions that can benefit the system the most, although it might let "impatient passengers" wait even longer, and in the end, we can achieve total waiting time that is closer to the optimal solution. But when we want to guarantee level of service and give priority to those "impatient passengers", then we will sacrifice the system optimality but can make sure that there are no extreme long waiting time.

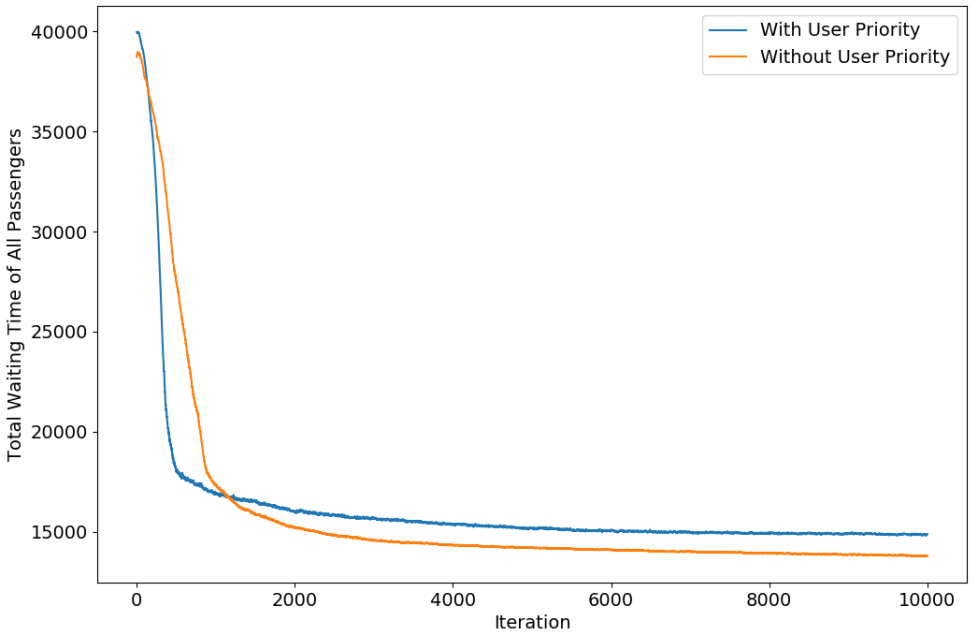


Figure 4.10: Total waiting time of all passengers

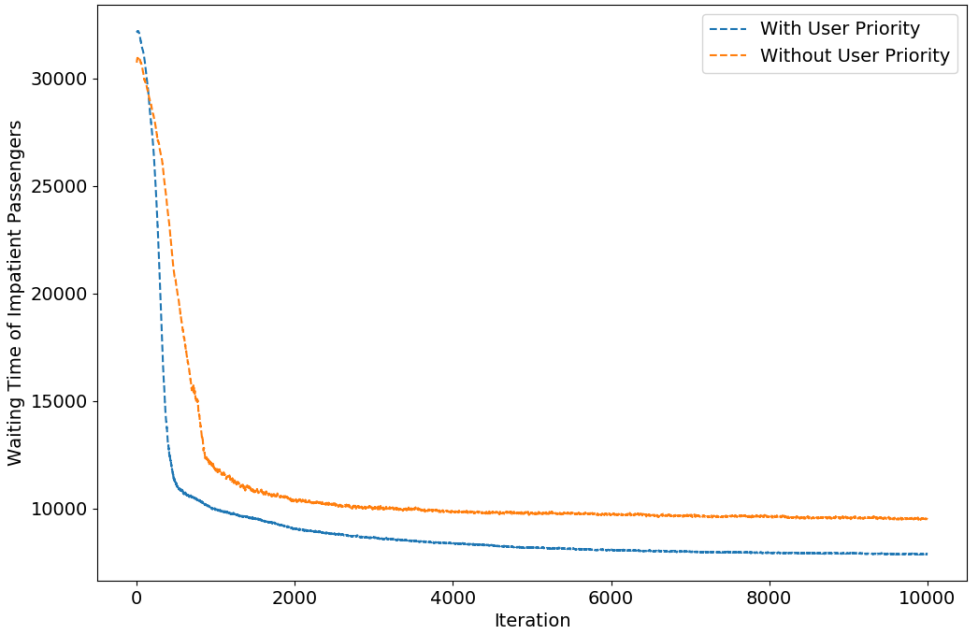


Figure 4.11: Total waiting time of impatient passengers

4.7 Conclusion

In this study, we present a deep reinforcement learning approach for the problem of dispatching autonomous vehicles for tax services. In particular, we propose an actor-critic framework with neural networks as approximations for both the actor and critic functions. Second, we also derive the theoretical upper bound of the total costs if we assume the dynamics of the system are deterministic and known to us beforehand. Third, we implement our RL method and apply it to a simplified network based on the New York yellow taxi services. Our case study shows that no matter the system dynamics are deterministic or stochastic, the RL method can always converge to some value close to the true optimal. In addition, we also investigate the scenario where we have to consider user priority. And the case study shows that this will cause the total waiting time of all passengers to sacrifice, but we add more fairness to the system by making sure there are less extreme long waiting times.

Chapter 5

Conclusions

In recent years, we have seen rapid development of autonomous driving technologies, and it can be expected that in the near future, fleets of autonomous vehicles will be driving on public roads and providing services to passengers. Obviously, this will create numerous revolutions in our society and bring a lot of benefits to the general public, e.g. more safety and convenience, fewer cars on the road and cleaner environment. Moreover, we can also look forward to better operations of transportation systems with the introduction of autonomous vehicles. And two representative operational problems are: how to build more effective real-time routing services for these vehicles, and how to better manage the fleet of autonomous vehicles for taxi services.

The above two problems are similar to each other in the sense that they all originate from transportation networks that have recurrent dynamic traffic patterns, and both of them can be modeled as sequential decision problems. But most previous works in addressing these problems have been focusing on developing parametric models to reflect the network dynamics and designing efficient algorithms to solve these models. Although there are some advantages of these model-based methods, there are some major limitations of these models when applied to real-world transportation networks, e.g. too strong assumptions and the curse of dimensionality.

With these concerns, in this dissertation, we explored the use of non-parametric model-free methods for these two operational problems. In particular, we incorporated the framework of reinforcement learning, and showed the effectiveness of using either value-based or policy-based RL methods in solving these problems. And the main findings and contributions of our work are summarized as follows.

5.1 Findings and Contributions

There are a lot of previous studies on the above two operational problems in transportation systems. In Chapter 2, we have reviewed some of the existing representative works on these problems. On one hand, for the adaptive routing problem in stochastic and time-

dependent networks, most previous works are based on parametric models and focus on developing efficient algorithms to solve these models. We found that for most of these models, strong assumptions have to be made on the networks. For example, some assumed that links' travel costs are independent from each other and follow exponential distributions. Besides, most models still suffer from the curse of dimensionality and cannot be applied to large networks. On the other hand, for the problem of rebalancing autonomous vehicles for taxi services, we noticed similar drawbacks among the current literature. With these considerations, we chose to incorporate the framework of reinforcement learning, which can serve as non-parametric model-free methods to solve these problems. And we reviewed some state of the art of different reinforcement learning techniques, e.g. actor-only, critic-only and actor-critic methods. Different methods were compared and guidelines for choosing the right method were provided.

In Chapter 3, we studied the problem of adaptive routing in stochastic and time-dependent networks. We formulated the problem as a MDP and presented the idea of Q leaning to solve the problem in discrete state space. As a comparison, we also provided a benchmark model-based DP method by assuming perfect knowledge on the MDP model. From our case study on a mid-size network, we noticed that Q learning outperforms the traditional DP method in most cases, especially during peak demand periods. Such an improvement is owing to the fact that Q learning does not rely on the calibration of any parametric models, thus it will not suffer from the approximation taken in the calibration process. However, it still suffered from the curse of dimensionality since it is based on discrete state space. So as a natural extension, we turned to an offline tree-based batch mode reinforcement learning method called fitted Q iteration (FQI), which can work in continuous state space and incorporate any regression algorithm as an approximation for the value function. And our case study showed that it can further improve the routing policy and deliver a more flexible strategy especially during peak hours. Furthermore, by looking into the computational performances of dynamic programming and FQI methods, we found that as the dimension of the state space grows, DP would fail to solve the problem due to the curse of dimensionality, but the running time of FQI did not increase too much. And we also found that as there are more data fed into the training process of FQI, the resulting routing performance could be improved consistently. This further supports the idea that non-parametric learning based methods are not restricted by model assumptions and data can be utilized more sufficiently. Although we believe that FQI is a more suitable method for solving the adaptive routing problem in real traffic networks, there are still some drawbacks of RL methods in general, such as long training process and requirement for large data.

In Chapter 4, we presented a deep reinforcement learning approach for the problem of dispatching autonomous vehicles for tax services. We formulated the problem first and presented our basic assumptions on the problem. Then we proposed an actor-critic framework with neural networks as approximations for both the actor and critic functions, and with adaptations to the output action function. Furthermore, we provided a benchmark by formulating the problem as a integer program in order to maximize the total rewards. From our case study based on the New York yellow taxi data, we found that no matter the system

dynamics are deterministic or stochastic, the RL method can always converge to some value close to the true optimal. Furthermore, when we chose to consider user priority and first serve those impatient passengers, we noticed that the total waiting time of all the passengers would be larger, however, the waiting time of these impatient passengers was significantly smaller. This indicates that considering user priority would cause the average waiting time of all passengers to sacrifice, but we add more fairness to the system by making sure there are less extreme long waiting times.

5.2 Directions for Future Research

There are a lot of directions that the work in this dissertation can be further extended, some examples are listed as follows:

- First of all, for the adaptive routing problem, a natural extension of the current research is to consider the multi-agent routing problem. As can be seen in our work, we assume the agent is "congestion-taker" instead of "congestion-maker", i.e. the routing choice of the agent will not influence the congestion states of the network. However, if more and more drivers begin to adopt the similar routing policy, the network congestion profile will be decided by the aggregate routing choices of all the agents. Thus we can look into the cases of both user equilibrium (all agents behave in the same way) and system optimal (agents behave differently to optimize total travel costs) under the setting of adaptive routing in the future. Currently, there are already a lot of studies on the topic of multi-agent reinforcement learning (Littman [47], Tan [70], Shoham et al. [64]). And we also see many applications of multi-agent reinforcement learning in the field of traffic signal control (Wiering [81], El-Tantawy et al. [72], Abdoos et al. [1]).

- Another direction to extend our work on the routing problem is to consider the stochastic shortest path problem, where the goal is to maximize the probability of arriving at destination before some deadline, instead of finding a path with least expected travel time as in our work. Although there are a lot of studies working on this problem (Nikolova and Karger [54], Nikolova et al. [55], Nie and Wu [53]), most of them still suffer from the curse of dimensionality and too strong assumptions. Thus it would be interesting to see if we can also apply RL methods to solve the problem. And in this case, the Q values learned in the process would have practical meaning as the estimated probabilities of reaching destination before the deadline.

- Furthermore, it is also interesting to test the performances of our proposed routing algorithms in large simulated traffic network that is similar to real-word networks. A very good platform for traffic micro-simulation is the MATSim platform (Balmer et al. [6]), which is a state-of-the-art agent based traffic micro-simulation tool that

performs traffic assignment for a set of agents with pre-defined activity plans. And it can generate daily traffic conditions close to the real-world situation.

- As for the problem of optimal control of autonomous taxi fleets, it also leaves a lot of extensions for future research. For example, we can test the performance of the RL method on larger networks, and compare the results with other model based methods like MPC. Moreover, we can look into the case where there is a mixed fleet of autonomous vehicles and human-driving vehicles. Since in the coming 20-30 years, the vehicle fleet will most likely be made up of a mixture of human-driven vehicles and autonomous vehicles. This complex traffic environment presents an lot of challenges to the control of the mixed fleet. And we will likely need to combine the surge pricing strategy for the human-driving vehicles with the dispatch strategy for the autonomous vehicles (Lei et al. [45]).

- Another extension on the taxi fleet control problem is to consider the effect of ride sharing on our dispatch strategy. Currently we assume that different trips are independent from each other and should be served with different vehicles. However, if we allow ride sharing and trip matching, this would affect our control strategy significantly, since we can dispatch less vehicles to some regions if there are more trips matched together.

Bibliography

[1] Monireh Abdoos, Nasser Mozayani, and Ana LC Bazzan. "Traffic light control in non-stationary environments based on multi agent Q-learning". In: *2011 14th International IEEE conference on intelligent transportation systems (ITSC)*. IEEE. 2011, pp. 1580–1585.

[2] Baher Abdulhai, Rob Pringle, and Grigoris J Karakoulas. "Reinforcement learning for true adaptive traffic signal control". In: *Journal of Transportation Engineering* 129.3 (2003), pp. 278–285.

[3] Mohammad Aslani, Mohammad Saadi Mesgari, and Marco Wiering. "Adaptive traffic signal control with actor-critic methods in a real-world traffic network with different traffic disruption events". In: *Transportation Research Part C: Emerging Technologies* 85 (2017), pp. 732–752.

[4] Amir Azaron and Farhad Kianfar. "Dynamic shortest path in stochastic dynamic networks: Ship routing problem". In: *European Journal of Operational Research* 144.1 (2003), pp. 138–156.

[5] Leemon Baird. "Residual algorithms: Reinforcement learning with function approximation". In: *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 30–37.

[6] Michael Balmer et al. *Agent-based simulation of travel demand: Structure and computational performance of MATSim-T*. ETH, Eidgenössische Technische Hochschule Zürich, IVT Institut für fffdfffdfffd, 2008.

[7] Siddhartha Banerjee, Carlos Riquelme, and Ramesh Johari. "Pricing in ride-share platforms: A queueing-theoretic approach". In: (2015).

[8] R.E. Bellman. *Dynamic Programming*. Dover Books on Computer Science Series. Dover Publications, 2003. ISBN: 9780486428093. URL: https://books.google.com/books?id=fyVtp3EMxasC.

[9] Richard Bellman. "A Markovian decision process". In: *Journal of Mathematics and Mechanics* 6.5 (1957), pp. 679–684.

[10] Richard Bellman et al. "The theory of dynamic programming". In: *Bulletin of the American Mathematical Society* 60.6 (1954), pp. 503–515.

[11] Hamid Benbrahim and Judy A Franklin. "Biped dynamic walking using reinforcement learning". In: *Robotics and Autonomous Systems* 22.3-4 (1997), pp. 283–302.

[12] Hamid Benbrahim et al. "Real-time learning: A ball on a beam". In: *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*. Vol. 1. IEEE. 1992, pp. 98–103.

[13] Hamid R Berenji and David Vengerov. "A convergent actor-critic-based FRL algorithm with application to power management of wireless transmitters". In: *IEEE Transactions on Fuzzy Systems* 11.4 (2003), pp. 478–485.

[14] Justin A Boyan and Andrew W Moore. "Generalization in reinforcement learning: Safely approximating the value function". In: *Advances in neural information processing systems*. 1995, pp. 369–376.

[15] Steven J Bradtke, B Erik Ydstie, and Andrew G Barto. "Adaptive linear quadratic control using policy iteration". In: *Proceedings of 1994 American Control Conference-ACC'94*. Vol. 3. IEEE. 1994, pp. 3475–3479.

[16] Leo Breiman. "Bagging predictors". In: *Machine learning* 24.2 (1996), pp. 123–140.

[17] Leo Breiman et al. *Classification and regression trees*. CRC press, 1984.

[18] A Castelletti et al. "Tree-based reinforcement learning for optimal water reservoir operation". In: *Water Resources Research* 46.9 (2010).

[19] Ching-Yao Chan. "Advancements, prospects, and impacts of automated driving systems". In: *International journal of transportation science and technology* 6.3 (2017), pp. 208–216.

[20] Le Chen, Alan Mislove, and Christo Wilson. "Peeking beneath the hood of uber". In: *Proceedings of the 2015 Internet Measurement Conference*. ACM. 2015, pp. 495–508.

[21] Robert H Crites and Andrew G Barto. "Improving elevator performance using reinforcement learning". In: *Advances in neural information processing systems*. 1996, pp. 1017–1023.

[22] Tapas K Das et al. "Solving semi-Markov decision problems using average reward reinforcement learning". In: *Management Science* 45.4 (1999), pp. 560–574.

[23] Peter Dayan and C Watkins. "Q-learning". In: *Machine learning* 8.3 (1992), pp. 279–292.

[24] Lili Du, Srinivas Peeta, and Yong Kim. "Online stochastic routing incorporating real-time traffic information". In: *Transportation Research Record: Journal of the Transportation Research Board* 2334 (2013), pp. 95–104.

[25] Damien Ernst, Pierre Geurts, and Louis Wehenkel. "Tree-based batch mode reinforcement learning". In: *Journal of Machine Learning Research* 6.Apr (2005), pp. 503–556.

[26] Liping Fu. "An adaptive routing algorithm for in-vehicle route guidance systems with real-time information". In: *Transportation Research Part B: Methodological* 35.8 (2001), pp. 749–765.

[27] Liping Fu and Larry R Rilett. "Expected shortest paths in dynamic and stochastic traffic networks". In: *Transportation Research Part B: Methodological* 32.7 (1998), pp. 499–516.

[28] Song Gao and Ismail Chabini. "Optimal routing policy problems in stochastic time-dependent networks". In: *Transportation Research Part B: Methodological* 40.2 (2006), pp. 93–122.

[29] Song Gao and He Huang. "Real-time traveler information for optimal adaptive routing in stochastic time-dependent networks". In: *Transportation Research Part C: Emerging Technologies* 21.1 (2012), pp. 196–213.

[30] Pierre Geurts, Damien Ernst, and Louis Wehenkel. "Extremely randomized trees". In: *Machine learning* 63.1 (2006), pp. 3–42.

[31] Geoffrey J Gordon. "Stable function approximation in dynamic programming". In: *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 261–268.

[32] Abhijit Gosavi. "Reinforcement learning: A tutorial survey and recent advances". In: *INFORMS Journal on Computing* 21.2 (2009), pp. 178–192.

[33] ABHUIT GOSAVII, Naveen Bandla, and Tapas K Das. "A reinforcement learning approach to a single leg airline revenue management problem with multiple fare classes and overbooking". In: *IIE transactions* 34.9 (2002), pp. 729–742.

[34] Ivo Grondman et al. "A survey of actor-critic reinforcement learning: Standard and natural policy gradients". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6 (2012), pp. 1291–1307.

[35] Vijaykumar Gullapalli. "Learning control under extreme uncertainty". In: *Advances in neural information processing systems*. 1993, pp. 327–334.

[36] Vijaykumar Gullapalli, Judy A Franklin, and Hamid Benbrahim. "Acquiring robot skills via reinforcement learning". In: *IEEE Control Systems Magazine* 14.1 (1994), pp. 13–24.

[37] Ali R Güner, Alper Murat, and Ratna Babu Chinnam. "Dynamic routing under recurrent and non-recurrent congestion using real-time ITS information". In: *Computers & Operations Research* 39.2 (2012), pp. 358–373.

[38] Randolph W Hall. "The fastest path through a network with random time-dependent travel times". In: *Transportation science* 20.3 (1986), pp. 182–188.

[39] Ronald A Howard. "DYNAMIC PROGRAMMING AND MARKOV PROCESSES.." In: (1960).

[40] Ramon Iglesias et al. "Data-driven model predictive control of autonomous mobility-on-demand systems". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1–7.

[41] Seongmoon Kim, Mark E Lewis, and Chelsea C White. "Optimal vehicle routing with real-time traffic information". In: *IEEE Transactions on Intelligent Transportation Systems* 6.2 (2005), pp. 178–188.

[42] Seongmoon Kim, Mark E Lewis, and Chelsea C White. "State space reduction for nonstationary stochastic shortest path problems with real-time traffic information". In: *IEEE Transactions on Intelligent Transportation Systems* 6.3 (2005), pp. 273–284.

[43] Nate Kohl and Peter Stone. "Policy gradient reinforcement learning for fast quadrupedal locomotion". In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004.* Vol. 3. IEEE. 2004, pp. 2619–2624.

[44] Vijay R Konda and John N Tsitsiklis. "Actor-critic algorithms". In: *Advances in neural information processing systems*. 2000, pp. 1008–1014.

[45] Chao Lei, Zhoutong Jiang, and Yanfeng Ouyang. "Path-based dynamic pricing for vehicle allocation in ridesharing systems with fully compliant drivers". In: *Transportation Research Part B: Methodological* (2019).

[46] Li Li, Yisheng Lv, and Fei-Yue Wang. "Traffic signal timing via deep reinforcement learning". In: *IEEE/CAA Journal of Automatica Sinica* 3.3 (2016), pp. 247–254.

[47] Michael L Littman. "Markov games as a framework for multi-agent reinforcement learning". In: *Machine learning proceedings 1994*. Elsevier, 1994, pp. 157–163.

[48] Chao Mao and Zuojun Shen. "A reinforcement learning framework for the adaptive routing problem in stochastic time-dependent network". In: *Transportation Research Part C: Emerging Technologies* 93 (2018), pp. 179–197.

[49] Fei Miao et al. "Taxi dispatch with real-time sensing data in metropolitan areas: A receding horizon control approach". In: *IEEE Transactions on Automation Science and Engineering* 13.2 (2016), pp. 463–478.

[50] Justin Miller and Jonathan P How. "Predictive positioning and quality of service ridesharing for campus mobility on demand systems". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 1402–1408.

[51] Elise D Miller-Hooks and Hani S Mahmassani. "Least expected time paths in stochastic, time-varying transportation networks". In: *Transportation Science* 34.2 (2000), pp. 198–215.

[52] Noriaki Mitsunaga et al. "Robot behavior adaptation for human-robot interaction based on policy gradient reinforcement learning". In: *Journal of the Robotics Society of Japan* 24.7 (2006), pp. 820–829.

[53] Yu Marco Nie and Xing Wu. "Shortest path problem considering on-time arrival probability". In: *Transportation Research Part B: Methodological* 43.6 (2009), pp. 597–613.

[54] Evdokia Nikolova and David R Karger. "Route Planning under Uncertainty: The Canadian Traveller Problem." In: *AAAI*. 2008, pp. 969–974.

[55] Evdokia Nikolova et al. "Stochastic shortest paths via quasi-convex maximization". In: *European Symposium on Algorithms*. Springer. 2006, pp. 552–563.

[56] Matteo Papini et al. "Stochastic variance-reduced policy gradient". In: *arXiv preprint arXiv:1806.05618* (2018).

[57] Marco Pavone et al. "Robotic load balancing for mobility-on-demand systems". In: *The International Journal of Robotics Research* 31.7 (2012), pp. 839–854.

[58] Jan Peters, Katharina Mulling, and Yasemin Altun. "Relative entropy policy search". In: *Twenty-Fourth AAAI Conference on Artificial Intelligence*. 2010.

[59] LA Prashanth and Shalabh Bhatnagar. "Reinforcement learning with function approximation for traffic signal control". In: *IEEE Transactions on Intelligent Transportation Systems* 12.2 (2011), pp. 412–421.

[60] Harilaos N Psaraftis and John N Tsitsiklis. "Dynamic shortest paths in acyclic networks with Markovian arc costs". In: *Operations Research* 41.1 (1993), pp. 91–101.

[61] Silvia Richter, Douglas Aberdeen, and Jin Yu. "Natural actor-critic for road traffic optimisation". In: *Advances in neural information processing systems*. 2007, pp. 1169–1176.

[62] Martin Riedmiller, Jan Peters, and Stefan Schaal. "Evaluation of policy gradient methods and variants on the cart-pole benchmark". In: *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. IEEE. 2007, pp. 254–261.

[63] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, England, 1994.

[64] Yoav Shoham, Rob Powers, and Trond Grenager. "Multi-agent reinforcement learning: a critical survey". In: *Web manuscript* (2003).

[65] Satinder P Singh and Dimitri P Bertsekas. "Reinforcement learning for dynamic channel allocation in cellular telephone systems". In: *Advances in neural information processing systems*. 1997, pp. 974–980.

[66] Richard S Sutton. "Generalization in reinforcement learning: Successful examples using sparse coarse coding". In: *Advances in neural information processing systems*. 1996, pp. 1038–1044.

[67] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998.

[68] Richard S Sutton et al. "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems*. 2000, pp. 1057–1063.

[69] Prasad Tadepalli and DoKyeong Ok. "Model-based average reward reinforcement learning". In: *Artificial intelligence* 100.1-2 (1998), pp. 177–224.

[70] Ming Tan. "Multi-agent reinforcement learning: Independent vs. cooperative agents". In: *Proceedings of the tenth international conference on machine learning*. 1993, pp. 330–337.

[71] Voot Tangkaratt et al. "Model-based policy gradients with parameter-based exploration by least-squares conditional density estimation". In: *Neural networks* 57 (2014), pp. 128–140.

[72] Samah El-Tantawy, Baher Abdulhai, and Hossam Abdelgawad. "Multiagent reinforcement learning for integrated network of adaptive traffic signal controllers (MARLIN-ATSC): methodology and large-scale application on downtown Toronto". In: *IEEE Transactions on Intelligent Transportation Systems* 14.3 (2013), pp. 1140–1150.

[73] Russ Tedrake, Teresa Weirui Zhang, and H Sebastian Seung. "Learning to walk in 20 minutes". In: *Proceedings of the Fourteenth Yale Workshop on Adaptive and Learning Systems*. Vol. 95585. Beijing. 2005, pp. 1939–1412.

[74] Gerald Tesauro. "Practical issues in temporal difference learning". In: *Advances in neural information processing systems*. 1992, pp. 259–266.

[75] John N Tsitsiklis and Benjamin Van Roy. "Analysis of temporal-diffference learning with function approximation". In: *Advances in neural information processing systems*. 1997, pp. 1075–1081.

[76] Mikhail Volkov, Javed Aslam, and Daniela Rus. "Markov-based redistribution policy model for future urban mobility networks". In: *Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on*. IEEE. 2012, pp. 1906–1911.

[77] S Travis Waller and Athanasios K Ziliaskopoulos. "On the online shortest path problem with limited arc cost dependencies". In: *Networks* 40.4 (2002), pp. 216–227.

[78] Erwin Walraven, Matthijs TJ Spaan, and Bram Bakker. "Traffic flow optimization: A reinforcement learning approach". In: *Engineering Applications of Artificial Intelligence* 52 (2016), pp. 203–212.

[79] Xin Wang and Thomas G Dietterich. "Model-based policy gradient reinforcement learning". In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 2003, pp. 776–783.

[80] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". PhD thesis. University of Cambridge England, 1989.

[81] MA Wiering. "Multi-agent reinforcement learning for traffic light control". In: *Machine Learning: Proceedings of the Seventeenth International Conference (ICML'2000)*. 2000, pp. 1151–1158.

[82] Liteng Zha, Yafeng Yin, and Zhengtian Xu. "Geometric matching and spatial pricing in ride-sourcing markets". In: *Transportation Research Part C: Emerging Technologies* 92 (2018), pp. 58–75.

[83] Rick Zhang and Marco Pavone. "Control of robotic mobility-on-demand systems: a queueing-theoretical perspective". In: *The International Journal of Robotics Research* 35.1-3 (2016), pp. 186–203.

[84] Rick Zhang, Federico Rossi, and Marco Pavone. "Model predictive control of autonomous mobility-on-demand systems". In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 1382–1389.

[85] Feng Zhu and Satish V Ukkusuri. "Accounting for dynamic speed limit control in a stochastic traffic environment: A reinforcement learning approach". In: *Transportation research part C: emerging technologies* 41 (2014), pp. 30–47.