PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
HIGHER SCHOOL OF COMPUTER SCIENCE AND DIGITAL TECHNOLOGIES - BEJAIA

Dissertation Submitted to the Department Of Computer Science
in Partial Fulfillment of the Requirements for
Master's Degree in Computer Science
Specialty: Artificial Intelligence and Data Science

Traffic Management Optimization in the Presence of Automated Vehicles

Submitted By:

Cheima MEZDOUR

Supervised By:

Pr. Nadir Farhi

Dr. Meriem Bouali

Members of Jury:

Dr. MEDJOUDJ Rafik	President	ESTIN
Mrs. HADJOUT Siham	Examiner	ESTIN
Dr. ALKAMA Lynda	Examiner	ESTIN
Dr. LEKEHALI Soumia	Examiner	ESTIN

Academic Year: 2023/2024

DEDICATION

I would like to express my deepest gratitude to my parents, who have been my greatest source of support throughout my academic journey. Their unwavering belief in me and constant encouragement have been invaluable. Without them, I would not be who I am today.

To my family, especially my siblings, thank you for your understanding, patience, sacrifices, and love. Your support has given me the strength and motivation to persevere.

To my best friend, Amani,
who accompanied me every step of the way during this journey and throughout my academic career, thank you for always being there for me.

Thank you to everyone who supported me with companionship, encouragement, and love. You have made this journey enjoyable and have been a source of great strength.

ACKNOWLEDGEMENTS

I want to thank the Almighty God, ALLAH, who has been my unwavering companion and guide throughout my academic career. His guidance has been a constant source of strength for me.

A heartfelt thank you to my family, whose support during these challenging six months has been nothing short of indispensable. Your encouragement, patience, and love have been vital to my perseverance and success. I couldn't have done it without you.

I am incredibly grateful to my Supervisor, Pr. Nadir Farhi, for his invaluable guidance, expertise, and contributions. His advice and support have played a crucial role in the success of this thesis, and I truly appreciate all his efforts.

Lastly, I want to acknowledge the dedication and hard work of all the school teachers. Your instruction and support have laid a strong foundation for my academic journey. Your efforts have been crucial in shaping my educational path, and I am deeply thankful for all you have done.

ABSTRACT

Traffic congestion on freeways is a significant issue today due to the high volume of vehicles. This congestion impacts traffic flow and can lead to severe delays. Effective traffic management methods and solutions are essential to mitigate this problem and enhance safety for all road users.

One specific issue in freeway traffic congestion is the problem of ramp metering. This occurs when vehicles merge from an entrance ramp onto the main freeway. Any malfunction or inefficiency in the traffic light system at these ramps can cause significant congestion and even lead to accidents.

By implementing a sophisticated decision-making system that follows a real-world network, we can develop better policies to address this issue. This approach can optimize traffic flow, reduce congestion, and improve overall road safety.

We proposed a deep q-learning algorithm to address this problem, which guarantees better learning policies and adapt to changing traffic conditions in real-time, learning optimal actions to take at different states of traffic flow in complex environments.

Deep Q-Networks can be used across multiple ramps, so this can improve the scalability of the model. So during this study, we will present the performance of this algorithm compared to traditional systems in improving the efficiency of traffic control.

Keywords : Traffic congestion, Freeways, Traffic flow, Ramp metering, Traffic management, Deep Q-Network, Decision-making system, Real-world network, Traffic control, Optimization, Scalability, Road safety, Adaptive policies, Traffic conditions

RESUME

La congestion du trafic sur les autoroutes est un problème majeur aujourd'hui en raison du volume élevé de véhicules. Cette congestion impacte la fluidité du trafic et peut entraîner des retards importants. Des méthodes et des solutions de gestion du trafic efficaces sont essentielles pour atténuer ce problème et améliorer la sécurité de tous les usagers de la route.

Un problème spécifique lié à la congestion du trafic sur les autoroutes est celui de la régulation des accès (ramp metering). Cela se produit lorsque des véhicules s'engagent depuis une rampe d'entrée sur l'autoroute principale. Toute défaillance ou inefficacité du système de feux tricolores à ces rampes peut provoquer une congestion importante et même conduire à des accidents.

En mettant en œuvre un système de prise de décision sophistiqué qui suit un réseau réel, nous pouvons développer de meilleures politiques pour traiter ce problème. Cette approche peut optimiser la fluidité du trafic, réduire la congestion et améliorer la sécurité globale sur les routes.

Nous avons proposé un algorithme de deep Q-learning pour aborder ce problème, qui garantit de meilleures politiques d'apprentissage et s'adapte aux conditions de trafic changeantes en temps réel, en apprenant les actions optimales à entreprendre à différents états du flux de trafic dans des environnements complexes.

Les réseaux de neurones profonds (Deep Q-Networks) peuvent être utilisés sur plusieurs rampes, ce qui améliore la scalabilité du modèle. Ainsi, au cours de cette étude, nous présenterons les performances de cet algorithme par rapport aux systèmes traditionnels pour améliorer l'efficacité du contrôle du trafic.

Mots-clés : Congestion du trafic, Autoroutes, Flux de trafic, Régulation des accès, Gestion du trafic, Réseau de neurones profond, Système de prise de décision, Réseau réel, Optimisation, Scalabilité, Sécurité routière, Politiques adaptatives, Conditions de trafic.

ملخص

تُعَدّ ازدحامات المرور على الطرق السريعة مشكلةً رئيسية اليوم بسبب العدد الكبير من المركبات. هذا الازدحام يؤثر على سلاسة حركة المرور وقد يتسبب في تأخيرات كبيرة. إن طرق وأساليب إدارة المرور الفعالة ضرورية للتخفيف من هذه المشكلة وتحسين السلامة لجميع مستخدمي الطريق.

مشكلة معينة تتعلق بازدحامات المرور على الطرق السريعة هي مسألة تنظيم الدخول (التحكم في الإشارات الضوئية عند المنحدرات). يحدث ذلك عندما تدخل المركبات من منحدر الدخول إلى الطريق السريع الرئيسي. أي فشل أو عدم كفاءة في نظام الإشارات الضوئية عند هذه المنحدرات قد يؤدي إلى ازدحام كبير وقد يتسبب حتى في حوادث.

من خلال تنفيذ نظام اتخاذ قرار متقدم يتابع شبكة مرورية حقيقية، يمكننا تطوير سياسات أفضل للتعامل مع هذه المشكلة. هذه الطريقة يمكن أن تُحَسِّن من سلاسة حركة المرور، وتقلل الازدحام، وتحسن السلامة العامة على الطرق.

لقد اقترحنا خوارزمية التعلم العميق لمعالجة هذه المشكلة، والتي تضمن سياسات تعلم أفضل وتتأقلم مع ظروف المرور المتغيرة في الوقت الحقيقي، من خلال تعلم الإجراءات المثلى التي يجب اتخاذها في حالات تدفق حركة المرور المختلفة في البيئات المعقدة.

يمكن استخدام الشبكات العصبية العميقة على عدة منحدرات، مما يُحَسِّن من قابلية توسيع النموذج. وبالتالي، سنعرض في هذه الدراسة أداء هذه الخوارزمية مقارنةً بالأنظمة التقليدية لتحسين كفاءة التحكم في المرور.

الكلمات المفتاحية: ازدحام المرور، الطرق السريعة، تدفق حركة المرور، تنظيم الدخول، إدارة المرور، الشبكات العصبية العميقة، نظام اتخاذ القرار، الشبكة الحقيقية، التحسين، قابلية التوسع، السلامة على الطرق، السياسات التكيفية، ظروف المرور.

ACRONYMS

A2C	Advantage Actor-Critic
AD	Autonomous Driving
AI	Artificial Intelligence
ALINEA	Asservissement Linéaire d'Entrée Autoroutière
AMOC	Adaptive Multi-Objective Control
ANCONA	Adaptive Nonlinear Control Algorithm
ANOVA	Analysis Of Variance
AV	Automated Vehicle
C-ITS	Cooperative Intelligent Transportation System
CACC	Cooperative Adaptive Cruise Control
CAV	Connected and Automated Vehicle
CNN	Convolutional Neural Network
CV	Connected Vehicle
DDQN	Double Deep Q-learning
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
DSRC	Dedicated Short-Range Communications
GA	Genetic Algorithm
HERO	Heuristic Ramp Metering Coordination
IoT	Internet of Things
ITS	Intelligent Transportation System
KPI	Key Performance Indicator
LSTM	Long Short-Term Memory
MADRL	Multi-Agent Deep Reinforcement Learning
MCTS	Monte Carlo Tree Search
ML	Machine Learning

MORL	Multi-Objective Reinforcement Learning
MPC	Model Predictive Control
PPO	Proximal Policy Optimization
PSO	Particle Swarm Optimization
Q-Learning	Quality-Learning
RNN	Recurrent Neural Network
RL	Reinforcement Learning
RL-Agent	Reinforcement Learning Agent
RL-TL	Reinforcement Learning for Traffic Light Control
SARSA	State-Action-Reward-State-Action
SOTA	State Of The Art
SUMO	Simulation of Urban Mobility
SZM	Stochastic Zone Model
TD	Temporal Difference
TL	Traffic Light
TL	Transfer Learning
TraCI	Traffic Control Interface
V2I	Vehicle-to-Infrastructure Communication
V2V	Vehicle-to-Vehicle Communication
V2X	Vehicle-to-Everything Communication
VSCode	Visual Studio Code
XML	Extensible Markup Language

Contents

List of Figures

List of Tables

General Introduction

Traffic congestion remains one of the most pressing urban challenges faced by cities worldwide, disrupting the regular flow of vehicles due to overcrowding on roadways. This issue has far-reaching economic implications, leading to billions of dollars lost annually through wasted time and increased fuel consumption. Additionally, the productivity of individuals and businesses suffers as time spent in traffic reduces overall efficiency.

From an environmental perspective, the consequences of traffic congestion are even more severe, with increased vehicle emissions contributing to air pollution, climate change, and global warming. Cities notorious for heavy traffic experience these environmental challenges on a daily basis.

Moreover, studies have suggested that prolonged exposure to heavy traffic can elevate adrenaline levels, resulting in heightened stress among drivers. This stress can have a cascading effect, diminishing productivity and encroaching on time that could otherwise be spent on personal activities or rest.

A critical factor that aggravates traffic congestion is the bottleneck that occurs when vehicles merge onto freeways from ramps. This merging process disrupts the steady flow of traffic, causing slowdowns and increasing traffic density.

When vehicles enter a freeway from an on-ramp, they must synchronize their speed with the existing traffic. This process often leads to disturbances in the traffic flow, causing drivers already on the freeway to brake and adjust their speeds. The resulting ripple effect extends beyond the merge point, leading to stop-and-go traffic that worsens congestion.

This merging congestion is especially problematic during peak hours, when traffic volumes are at their highest. The surge of vehicles attempting to enter the freeway can overwhelm the merge lanes, leading to extended queues on the ramps and further exacerbating freeway congestion.

The impact of such congestion is considerable. Drivers face longer travel times and

are frequently forced into stop-and-go driving patterns, which not only increases travel time but also raises the likelihood of accidents near merge points. Additionally, efforts to alleviate freeway congestion can cause backups on the ramps, where vehicles queue while waiting to merge, potentially leading to congestion and accidents in other parts of the traffic network.

The primary objective of this research is to develop an intelligent solution to effectively manage traffic flow, ensuring smooth transitions between freeways and ramps. Our goal is to minimize travel time and reduce congestion on both freeways and ramps, thereby improving overall traffic efficiency.

To achieve this, we plan to utilize advanced traffic management techniques and technologies. These may include adaptive traffic signal control, ramp metering, and real-time traffic monitoring systems. The objective is to dynamically adjust traffic flow based on current conditions, preventing bottlenecks and ensuring that vehicles can merge onto freeways without causing significant disruptions.

By optimizing the coordination between freeway traffic and ramp entries, we anticipate significant improvements in traffic efficiency. This will not only reduce travel times but also enhance safety by minimizing the risks associated with sudden braking and frequent stops. Ultimately, our solution aims to contribute to a more efficient and safer transportation network, benefiting all road users.

This approach emphasizes the integration of various technologies to create a responsive and adaptive traffic management system. It highlights the critical role of real-time data in making informed decisions to regulate traffic flow. Through the implementation of these strategies, we aim to develop a sustainable and effective solution to the ongoing problem of traffic congestion in urban environments.

The remainder of this dissertation is structured as follows:

- **Chapter 1: General Concepts and Related Works**

  This chapter examines the foundational principles of reinforcement learning and deep Q-learning, providing a comprehensive literature review of the methodologies that have evolved over time. It contextualizes these approaches within existing

research to highlight their relevance and application to this project.

- **Chapter 2: Proposed Approach**

  This chapter presents our proposed solution for traffic regulation for ramp metering problem. It discusses the methodology and the use of reinforcement learning algorithms, specifically focusing on the application of Deep Q-learning and its variations for managing freeway congestion.

- **Chapter 3: Experimental Studies and Results**

  The results of our experiments are detailed in this chapter. We evaluate the performance of the proposed approach using traffic simulations and compare it against existing methods to demonstrate its effectiveness.

General Concepts and Related Works

1.1 Introduction

This chapter serves as a comprehensive introduction to the core concepts and current advancements that underpin the research presented in this thesis. It begins by outlining the fundamental principles of Reinforcement Learning (RL), a pivotal area in the broader field of machine learning that focuses on how agents can learn optimal behaviors through interactions with their environment. The discussion highlights the theoretical foundations of RL, including key algorithms and their applications across various domains, emphasizing the significance of RL in solving complex, sequential decision-making problems.

Building upon this foundation, the chapter then explores Deep Reinforcement Learning (DRL), a significant evolution of RL that leverages deep neural networks to handle high-dimensional input spaces and learn more sophisticated policies. This section explores the integration of deep learning with RL, discussing prominent DRL architectures, such as Deep Q-Networks (DQN) and policy gradient methods, and their transformative impact on tasks ranging from game playing to autonomous systems. The potential and limitations of DRL are critically examined, setting the stage for understanding the challenges that drive ongoing research in this area.

The chapter further introduces MetaNet, an innovative architecture designed to enhance the flexibility and generalization capabilities of learning agents. MetaNet represents a step forward in the development of meta-learning frameworks, where the goal is to create models that can quickly adapt to new tasks with minimal data. This section

reviews the architecture, underlying concepts, and its role in addressing the limitations of traditional RL and DRL approaches, particularly in environments where adaptability and efficiency are paramount.

In addition to these conceptual foundations, the chapter provides a detailed review of the state of the art in RL, DRL, and MetaNet, encompassing the latest research trends, significant breakthroughs, and the challenges that remain. By critically analyzing contemporary studies, this section identifies key research gaps and discusses how current methods are being refined to address these gaps. The review not only situates this thesis within the broader research landscape but also highlights the contributions this work aims to make to the field.

Finally, the chapter concludes with an analysis and discussion that synthesizes the insights gained from the literature review, setting the stage for the detailed investigations and experimental studies presented in subsequent chapters. This conclusion ensures a logical flow of ideas and prepares the reader for the more specialized discussions that follow, establishing a clear connection between the theoretical concepts explored in this chapter and their application in advancing the field.

1.2 Reinforcement Learning

Reinforcement Learning (RL) is a subfield of machine learning focused on how an autonomous decision-making agent learns to perform tasks within an environment based on its experiences. Without prior knowledge, the agent begins by taking actions that result in rewards or penalties, observing the outcomes of these decisions, as shown in Figure 1.1. Over time, through repeated interactions with the environment, the agent refines its strategy to maximize cumulative rewards, ultimately converging on an optimal behavior for solving the problem at hand. The total set of possible conditions the agent might encounter is known as the state space, while the range of actions it can take constitutes the action space. The environment, which encompasses everything external to the agent, operates according to a set of governing rules and can represent any complex

system. The agent and environment engage in a stepwise interaction at discrete time intervals (e.g., $t+0$, $t+1$, $t+2$, …), where each transition between successive states marks a new step in the process. A complete sequence of these interactions, starting from an initial state s_{t+0} to a terminal state s_{t+f}, is referred to as an episode or epoch. The learning process comprises many such episodes, allowing the agent to progressively improve its decision-making capabilities [2] [3] [4].

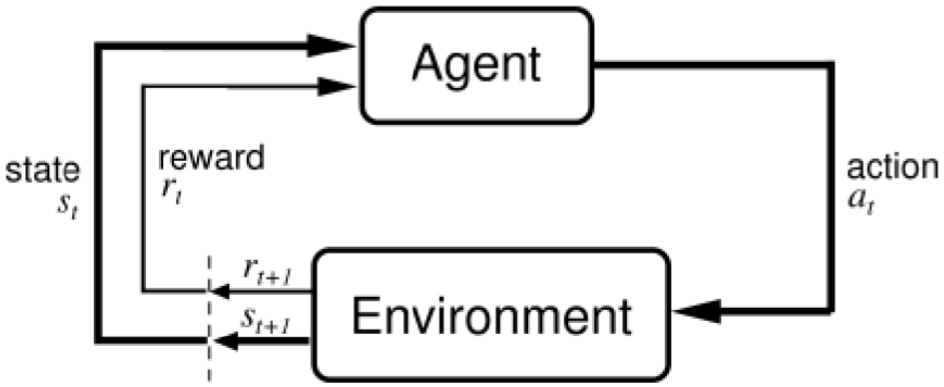


Figure 1.1: Reinforcement learning feedback framework.

1.2.1 Markov Decision Process

In control theory, the decision-making challenge faced by a Reinforcement Learning (RL) agent is termed an optimal control problem and is typically modeled as a Markov Decision Process (MDP). An MDP is a discrete-time stochastic control framework used to represent decision-making scenarios where the controller interacts with a dynamic environment at each discrete timestep t in a sequence. Formally, an MDP is defined by the tuple $\langle S, A, P, R \rangle$, where S represents the set of possible states in the environment, A denotes the set of actions available to the agent, $P(s_{t+1}|s_t, a_t)$ describes the transition dynamics, and $R(s_t, a_t, s_{t+1})$ specifies the reward function. A key assumption of an MDP is that it satisfies the Markov property, meaning the next state s_{t+1} depends solely on the current state s_t and the action a_t taken by the agent, rendering the process memoryless and independent of prior states.

The sequential decision-making process in an MDP proceeds as follows at each timestep

t:

1. The agent observes the environment ε through input signals (e.g., sensors) and estimates the current state s_t using a state-observation preprocessing function ϕ, where $\phi(s_t) = \phi(x_t) \approx s_t$ in the continuous state space S.

2. The agent selects an action a_t from the set of possible actions in the discrete action space $A = \{a_1, a_2, \ldots, a_k\}$, in response to the observed state s_t.

3. The environment transitions from state s_t to state s_{t+1} as a consequence of action a_t, following the internal dynamics represented by the finite probability transition matrix $P(s_{t+1}|s_t, a_t)$.

4. This transition (s_t, a_t, s_{t+1}) generates an immediate reward r_{t+1}, which reflects the instantaneous return of taking action a_t in state s_t, according to the reward function $R(s_t, a_t, s_{t+1})$.

The objective of learning in an MDP is to identify a policy $\pi(a, s) = p(a|s)$, which maps states to actions in a way that maximizes the long-term cumulative reward, i.e., the sum of rewards received after the current time t. As tasks are executed by completing a finite sequence of actions, a discount factor $\gamma \in (0, 1)$ is used to prioritize immediate rewards over those received in the future within a finite time horizon T. Even though the MDP may be arbitrarily large, the finite time horizon T ensures that the process remains finite. The future discounted reward at time t is given by:

$$\begin{aligned} G_t &= \sum_{k=t}^{t+T} \gamma^{(k-t)} \cdot r_{k+1} \\ &= r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \gamma^3 \cdot r_{t+4} + \ldots \\ &= r_{t+1} + \gamma \cdot (r_{t+2} + \gamma \cdot r_{t+3} + \gamma^2 \cdot r_{t+4} + \ldots) \\ &= r_{t+1} + \gamma \cdot G_{t+1} \end{aligned} \tag{1.1}$$

This formulation highlights the recursive nature of the future reward G_t, where each immediate reward is combined with the discounted future reward.

1.2.2 Optimal action-value Q-function

In an environment ε, the transition $(s_t, a_t, s_{t+1}, r_{t+1})$ is stochastic rather than deterministic, meaning the future discounted reward can vary between different rollouts due to the inherent uncertainty of outcomes. When following a policy $\pi(a, s)$, the value function returns the expected discounted reward for being in state s_t across all possible trajectories:

$$V^{\pi}(s) = \mathbb{E}^{\pi}[G_t \mid s_t = s, \pi] = \mathbb{E}^{\pi}[r_{t+1} + \gamma \cdot G_{t+1} \mid s_t = s, \pi] \tag{1.2}$$

With probabilistic state transition dynamics, the expected reward is defined as:

$$r(s, a) = \mathbb{E}[r_t \mid s_{t-1} = s, a_{t-1} = a] = \sum_{r} \sum_{s'} p(s', r \mid s, a) \cdot r \tag{1.3}$$

Thus, the value function can be expressed as:

$$\begin{aligned} V^{\pi}(s) &= \sum_{a} \pi(a, s) \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma \cdot \mathbb{E}^{\pi}[G_{t+1} \mid s_{t+1} = s'] \right] \\ &= \sum_{a} \pi(a, s) \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma \cdot V^{\pi}(s') \right] \end{aligned} \tag{1.4}$$

This recursive relationship between successive value functions is known as the Bellman equation. It describes how the value of a current state s_t is equal to the expected value of the next state s_{t+1}, discounted by γ and incremented by the reward r_{t+1}.

From the value function and with a fixed action a_t at timestep t, the action-value Q-function, which represents the value of taking action a_t in state s_t while following policy $\pi(a, s)$, is defined as:

$$\begin{aligned} Q^{\pi}(s, a) &= \mathbb{E}^{\pi}[G_t \mid s_t = s, a_t = a, \pi] = \mathbb{E}^{\pi}[r_{t+1} + \gamma \cdot G_{t+1} \mid s_t = s, a_t = a, \pi] \\ &= \mathbb{E}^{\pi}[r_{t+1} + \gamma \cdot V^{\pi}(s_{t+1}) \mid s_t = s, a_t = a, \pi] \\ &= \sum_{s', r} p(s', r \mid s, a)[r + \gamma \cdot V^{\pi}(s')] \end{aligned} \tag{1.5}$$

The Q-function assesses the quality of policies based on the value function and allows the agent to rank the expected discounted rewards over different states and actions. By maximizing the Q-function, the agent can derive the policy that yields the greatest expected return, known as the optimal policy $\pi^*(a, s)$. The Bellman optimality equation describes the relationship between the optimal value function and the optimal action-value Q-function:

$$\begin{aligned} V^{\pi^*}(s) &= \max_a Q^{\pi^*}(s, a) \\ &= \max_a \mathbb{E}^{\pi}[r_{t+1} + \gamma \cdot V^{\pi^*}(s_{t+1}) \mid s_t = s, a_t = a, \pi = \pi^*] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma \cdot V^{\pi^*}(s')] \end{aligned} \tag{1.6}$$

Consequently, the optimal action-value Q-function is given by:

$$\begin{aligned} Q^{\pi^*}(s, a) &= \mathbb{E}^{\pi}[r_{t+1} + \gamma \cdot \max_{a'} Q^{\pi^*}(s_{t+1}, a') \mid s_t = s, a_t = a, \pi = \pi^*] \\ &= \sum_{s',r} p(s', r \mid s, a)[r + \gamma \cdot \max_{a'} Q^{\pi^*}(s', a')] \end{aligned} \tag{1.7}$$

1.2.3 Q-Learning Algorithm

1.2.3.1 Q-Learning and Value-Based Reinforcement Learning

Q-learning is a reinforcement learning algorithm that focuses on learning the optimal policy $\pi^*(s, a)$ by estimating the action-values of state-action pairs in an unknown environment ε with stochastic transitions (s, a, s', r'). The algorithm maps from a continuous state space S to a discrete action space A by optimizing the Q-function $Q^{\pi}(s, a)$. It iteratively approximates the optimal Q-function $Q^{\pi^*}(s, a)$ by selecting the action that maximizes the expected Q-value $\arg\max_a Q^{\pi}(s, a)$ for the current state, thereby learning the expected reward associated with each state-action pair. For any finite Markov Decision Process (MDP) and given infinite learning time, exploration, and exploitation, Q-learning can converge to the optimal policy:

$$\pi^*(s,a) = \arg\max_a Q^{\pi^*}(s,a) \tag{1.8}$$

Since the agent learns the optimal policy through value functions, Q-learning is classified under the broader category of value-based reinforcement learning methods.

1.2.3.2 Off-Policy Learning and the Epsilon-Greedy Policy

While there is no definitive rule for addressing the explore-exploit trade-off, Q-learning employs an action-selection strategy known as epsilon-greedy (ε-greedy). This approach is considered off-policy because the action-selection policy differs from the learned policy π by introducing an epsilon ε threshold for random action selection. At each timestep, the agent either explores by taking a random action with probability ε or exploits by taking the best known action for the given state with probability $1-\varepsilon$, following the learned behavior policy π:

$$\varepsilon\text{-greedy policy:} \quad a = \begin{cases} \arg\max_a Q^{\pi}(s,a) & \text{with probability } 1-\varepsilon \\ \text{rand } a \in A & \text{with probability } \varepsilon \end{cases} \tag{1.9}$$

There are various ways to define ε, with the most common being linear decay. At the beginning of the learning process, $\varepsilon = 1$ since there is no knowledge of the environment to exploit. As the agent accumulates knowledge, the need for exploration diminishes, and exploitation becomes more critical, leading to a linear decay of ε to a lower bound ε_{min} over a period ε_{dec}, following an interpolation function:

$$\text{Linear decay: } \varepsilon = \begin{cases} \frac{t\cdot(\varepsilon_{\text{min}}-1)+1}{\varepsilon_{\text{dec}}} & \text{if } t < \varepsilon_{\text{dec}} \\ \varepsilon_{\text{min}} & \text{otherwise} \end{cases} \tag{1.10}$$

The probability of exploration should never reach zero, as a continuous state space can never be fully explored. Consequently, Q-learning converges in the long term to the best local optimum discovered, though it is not guaranteed to find the global optimum.

1.2.3.3 Temporal difference learning and target update

Q-learning updates the optimal Q-function using a process called temporal difference (TD) learning, where Q-values are iteratively adjusted based on the discrepancy between new and previous estimates for a given state-action pair. This adjustment mechanism, known as bootstrapping, involves generating new estimates from previous samples and is performed online, meaning it occurs at each iteration. The temporal difference is calculated as the difference between the TD target—a projected Q-value for the subsequent state, derived from the expected optimal action-value—and the current Q-value estimate for the present state.

The Q-function is then updated by adding the temporal difference, scaled by a learning rate α, to the current Q-value:

$$\begin{aligned} Q^{\pi}_{t+1}(s_t, a_t) &= \mathbb{E}^{\pi}\left[r_{t+1} + \gamma \cdot \max_{a'} Q^{\pi}_{t}(s_{t+1}, a') \mid s_t = s, a_t = a, \pi\right] \\ &= Q^{\pi}_{t}(s_t, a_t) + \alpha \cdot \Big(r_{t+1} + \gamma \cdot \max_{a'} Q^{\pi}_{t}(s_{t+1}, a') - Q^{\pi}_{t}(s_t, a_t)\Big) \end{aligned} \tag{1.11}$$

This iterative process ensures that the Q-function gradually converges to the optimal Q-values as the agent continues to interact with the environment.

The learning rate parameter α determines the degree to which new estimates influence the update, effectively governing how quickly the agent learns. In other words, it controls the weight given to recent experiences versus past knowledge. With an infinite number of temporal difference updates, Q-learning gradually converges to the optimal Q-function:

$$Q \to Q^{*} \text{ as } t \to \infty \tag{1.12}$$

1.2.3.4 Tabular Q-learning algorithm

The Q-learning algorithm operates by iteratively updating a Q-value table, which represents the expected utility (or Q-value) of taking a specific action a in a specific state s. The Q-value $Q(s, a)$ is a real number that estimates the total expected future rewards

when action a is taken in state s, and the agent subsequently follows the optimal policy. The Q-values are updated using the Bellman equation, which is central to dynamic programming and reinforcement learning methods.

At each timestep t, the agent observes the current state s_t and selects an action a_t based on an action-selection policy, often using an ε-greedy strategy to balance exploration and exploitation. After executing the action, the agent receives a reward r_{t+1} and observes the next state s_{t+1}. The Q-value for the state-action pair (s_t, a_t) is then updated using the temporal difference (TD) learning rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \tag{1.13}$$

Here, α is the learning rate, which controls how much the new information overrides the old information, and γ is the discount factor, which determines the importance of future rewards. The term $\max_{a'} Q(s_{t+1}, a')$ represents the maximum expected future reward for the next state s_{t+1}, considering all possible actions a' [5].

1.3 Deep Q-learning

In this section, we introduce deep Q-learning, a variation of the traditional Q-learning algorithm that integrates deep learning techniques. Deep Q-learning, often referred to as DQN, encompasses a range of algorithms, with the most prominent being the seven key methods collectively known as Rainbow DQN (Hessel et al., 2017) [4, 6].

1.3.1 DQN: Deep Q-Network

1.3.1.1 Deep Q-Network (DQN) Architecture

The Deep Q-Network (DQN) architecture, introduced by Mnih et al. [7], represents a significant advancement in the field of deep reinforcement learning. This architecture combines the well-established Q-learning algorithm with deep neural networks, allowing for the approximation of the Q-value function, which is crucial for decision-making in

high-dimensional state spaces. The success of DQN, particularly in achieving human-level performance on Atari 2600 games, marked a pivotal moment in the development of AI systems capable of handling complex environments.

1.3.1.2 Overview of the DQN Architecture

The primary goal of the DQN architecture is to learn an optimal policy for an agent interacting with an environment by approximating the Q-value function $Q(s, a)$. This function predicts the cumulative future rewards that the agent expects to receive by taking action a in state s. The agent then selects actions based on these Q-values to maximize its total expected reward over time.

1.3.1.3 Key Components of the DQN Architecture

1. **Neural Network Structure:**

   The DQN employs a deep neural network to approximate the Q-value function. The input to this network is the current state s of the environment, typically represented as a stack of consecutive frames or a high-dimensional vector. The network outputs a set of Q-values, one for each possible action the agent can take in that state.

   In the context of Atari games, the input is a sequence of game frames. The network architecture usually consists of several convolutional layers followed by fully connected layers. The convolutional layers extract spatial features from the input frames, while the fully connected layers map these features to Q-values corresponding to different actions.

   A simplified schematic of the DQN architecture is shown in Figure 1.2 [1].

Input (State) → Convolutional Layers → Fully Connected Layers → Output (Q-values for actions)

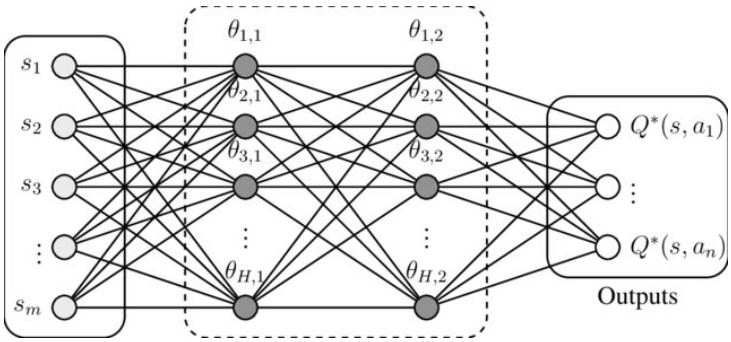


Figure 1.2: Structure of the neural network used for the Deep Q-learning Network implementation with two hidden layers, each of dimension [1].

2. **Experience Replay:**

   One of the key innovations in DQN is the use of experience replay to stabilize the training process. Instead of updating the network after every single experience, DQN stores experiences as tuples (s, a, r, s') in a replay buffer. Here, s represents the current state, a the action taken, r the reward received, and s' the resulting next state.

   During training, the algorithm randomly samples mini-batches from this buffer to update the network. This random sampling breaks the correlation between consecutive experiences, which reduces variance and helps stabilize the learning process.

3. **Target Network:**

   To further stabilize training, DQN employs a target network, which is a copy of the Q-network used to compute the target Q-values during training. Unlike the main Q-network, the target network's parameters θ^- are updated less frequently—typically every few thousand training steps—by copying the parameters from the main Q-network.

   This approach mitigates issues such as oscillations or divergence in the Q-value estimates, which can occur when the network is constantly learning from its own predictions. By keeping the target Q-values stable for a period, the training process

becomes more stable and is less likely to make large, destabilizing updates. The target Q-value for a given state-action pair is computed as:

$$Y_t = r + \gamma \max_{a'} Q_{\text{target}}(s', a'; \theta^-) \tag{1.14}$$

where θ^- represents the parameters of the target network. This separation between the evaluation and the target networks improves stability and convergence during training.

4. **Q-Learning with DQN:**

   The DQN algorithm updates its Q-values using a modified version of the Bellman equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q_{\text{target}}(s', a') - Q(s, a) \right) \tag{1.15}$$

   where:

   - r is the reward received after taking action a in state s,
   - s' is the subsequent state,
   - α is the learning rate,
   - γ is the discount factor, and
   - $Q_{\text{target}}(s', a')$ is the Q-value from the target network.

   The use of a separate target network helps provide more stable targets for the Q-learning updates, thereby improving the stability and convergence of the learning process.

1.3.1.4 DQN Algorithm

In the Vanilla DQN algorithm, the Q-network serves as a function approximator, replacing the traditional Q-table as shown in Table 1.1. The algorithm operates as follows:

The online Q-network Q is initialized with random weights Θ_0, and the target Q-network $\hat{Q}$ is initialized with identical weights $\Theta_0^- = \Theta_0$. The replay memory buffer D is initialized with a capacity N, and it is pre-filled with a minimum number of transitions $N_{\min} \leq N$ obtained by taking random actions.

The training process is conducted over E episodes, each consisting of multiple steps t, beginning from an initial state and progressing to a terminal state. At each step, the agent observes the current state $s_t = \phi(x_t)$, where x_t is the raw observation processed by the function ϕ. The agent selects an action a_t based on the Q-values $Q_{\pi,t}(s_t, a; \Theta_t)$ generated by forward propagating through the online Q-network Q and following an ε-greedy policy. The environment then transitions to the next state s_{t+1} and yields a reward r_{t+1}.

The transition $(s_t, a_t, s_{t+1}, r_{t+1})$ is stored in the replay memory buffer D. A random mini-batch $U(D)$ of size M is sampled from the replay memory to update the Q-network.

For each sampled transition $(s, a, s', r')^{(m)}$ in the batch, the temporal difference target $y_t^{(m)}$ is computed by forward propagating through the target network $\hat{Q}$, and the predicted Q-value $h_t^{(m)}$ is obtained by forward propagating through the online Q-network Q:

$$y_t^{(m)} = r'^{(m)} + \gamma \cdot \max_{a'} \hat{Q}(s'^{(m)}, a'; \Theta_t^-), \tag{1.16}$$

$$h_t^{(m)} = Q_{\pi,t}(s^{(m)}, a^{(m)}; \Theta_t) \tag{1.17}$$

The online Q-network Q is then updated by performing backpropagation using a gradient descent step on the Mean Square Error (MSE) loss over the temporal differences $\delta_t^{(m)} = y_t^{(m)} - h_t^{(m)}$:

$$L(y_t, h_t) = \frac{1}{2M} \sum_{m=1}^{M} \left(y_t^{(m)} - h_t^{(m)} \right)^2 \tag{1.18}$$

We can replace $y_t^{(m)} - h_t^{(m)}$ as follows:

$$L(y_t, h_t) = \frac{1}{2M} \sum_{m=1}^{M} \left(\delta_t^{(m)} \right)^2 \tag{1.19}$$

Finally, every C steps, the weights Θ_t^- of the target network $\hat{Q}$ are updated by copying the weights Θ_t from the online Q-network Q:

$$\Theta_t^- = \Theta_t \quad \text{if } t \equiv 0 \pmod{C} \tag{1.20}$$

Algorithm 1 DQN Algorithm

Initialize online Q-network Q with random weights Θ_0
Initialize target Q-network $\hat{Q}$ with weights $\Theta_0^- = \Theta_0$
Initialize replay memory D with capacity N
Pre-fill replay memory D with $N_{\min}$ random transitions
for each episode $e = 1, \ldots, E$ **do**
    Initialize state s_0
    for each step $t = 0, 1, \ldots, T$ **do**
        Observe state s_t
        Select action a_t based on $Q(s_t, a; \Theta_t)$ using ε-greedy policy
        Execute action a_t and observe reward r_{t+1} and next state s_{t+1}
        Store transition $(s_t, a_t, s_{t+1}, r_{t+1})$ in replay memory D
        Sample random mini-batch $U(D)$ of size M from D
        for each transition $(s, a, s', r')^{(m)}$ in $U(D)$ **do**
            Compute temporal difference target $y_t^{(m)}$
            $y_t^{(m)} = r'^{(m)} + \gamma \cdot \max_{a'} \hat{Q}(s'^{(m)}, a'; \Theta_t^-)$
            Compute predicted Q-value $h_t^{(m)}$
            $h_t^{(m)} = Q(s^{(m)}, a^{(m)}; \Theta_t)$
            Compute loss $L(y_t, h_t)$ and perform gradient descent
            $L(y_t, h_t) = \frac{1}{2M} \sum_{m=1}^{M} \left(y_t^{(m)} - h_t^{(m)} \right)^2$
        end for
        Update Q-network weights Θ_t using backpropagation
        if $t \equiv 0 \pmod{C}$ **then**
            Update target network $\hat{Q}$ weights: $\Theta_t^- = \Theta_t$
        end if
    end for
end for

Table 1.1: DQN Algorithm

1.3.2 DDQN: Double Deep Q-Network

Double Deep Q-Network (DDQN) is an enhancement of the Deep Q-Network (DQN) algorithm. The primary motivation behind DDQN is to address the overestimation bias inherent in the action-value estimates (Q-values) in traditional DQN. Overestimation occurs due to the use of the same network to both select and evaluate actions, which can lead to overly optimistic Q-values and, consequently, suboptimal policies.

1.3.2.1 Problem with DQN: Overestimation Bias

In the standard DQN algorithm, the target value used for updating the Q-network is computed as:

$$y_t = r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a'; \theta^-) \tag{1.21}$$

where:

- y_t is the target value for the Q-network.
- r_t is the reward received after taking action a_t in state s_t.
- γ is the discount factor, which determines the importance of future rewards.
- $Q(s_{t+1}, a'; \theta^-)$ is the Q-value for the next state s_{t+1} and action a', as estimated by the target network with parameters θ^-.

The use of the same Q-network to both select the action ($\max_{a'}$) and evaluate the Q-value of that action can lead to an overestimation of the Q-values. This occurs because the maximization step is more likely to favor overestimated values, leading to overly optimistic updates.

1.3.2.2 Solution with DDQN: Decoupling Action Selection and Evaluation

To mitigate the overestimation bias, DDQN decouples the action selection from the action evaluation in the target computation. The target value in DDQN is calculated as:

$$y_t = r_t + \gamma \cdot Q(s_{t+1}, \arg\max_{a'} Q(s_{t+1}, a'; \theta); \theta^-) \tag{1.22}$$

Here:

- The action $\arg\max_{a'} Q(s_{t+1}, a'; \theta)$ is selected using the current online Q-network $Q(s_{t+1}, a'; \theta)$ with parameters θ.
- The Q-value of this selected action is then evaluated using the target Q-network $Q(s_{t+1}, \cdot; \theta^-)$.

This separation ensures that the evaluation of the action's value is more accurate, which reduces the likelihood of overestimation. The steps of the DDQN algorithm are outlined in Table 1.2.

1.3.2.3 DDQN Algorithm

Algorithm 2 DDQN Algorithm

Initialize the online Q-network Q with random weights θ_0
Initialize the target Q-network Q_{target} with weights $\theta_0^- = \theta_0$
Initialize replay memory $\mathcal{D}$
for each episode **do**
  Initialize state s_0
  for each step t in the episode **do**
    With probability ε, select a random action a_t
    Otherwise, select $a_t = \arg\max_a Q(s_t, a; \theta)$
    Execute action a_t and observe reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in replay memory $\mathcal{D}$
    Sample a random minibatch of transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{D}$
    for each transition i in the minibatch **do**
      Select action $a'_i = \arg\max_{a'} Q(s_{i+1}, a'; \theta)$
      Evaluate the action's value using the target network: $y_i = r_i + \gamma \cdot Q(s_{i+1}, a'_i; \theta^-)$
      Update the online network by minimizing the loss:

$$L_i(\theta) = \left(y_i - Q(s_i, a_i; \theta)\right)^2$$

    end for
    Every C steps, update the target network: $\theta^- \leftarrow \theta$
    Decay ε according to the epsilon-greedy schedule
  end for
end for

Table 1.2: DDQN Algorithm

1.3.2.4 Key Components and Equations

- **Target Value Calculation :**

  In DDQN, the target value y_t is calculated by first selecting the action that maximizes the Q-value using the online network:

$$a_t^* = \arg\max_{a'} Q(s_{t+1}, a'; \theta) \tag{1.23}$$

  Then, the Q-value of this action is evaluated using the target network:

$$y_t = r_t + \gamma \cdot Q(s_{t+1}, a_t^*; \theta^-) \tag{1.24}$$

- **Loss Function :**

  The online Q-network is updated by minimizing the loss function $L(\theta)$, which is the mean squared error between the predicted Q-value and the target value:

$$L(\theta) = \mathbb{E}\left[(y_t - Q(s_t, a_t; \theta))^2\right] \tag{1.25}$$

- **Epsilon-Greedy Policy**

  The action selection follows an epsilon-greedy policy, where the action is chosen randomly with probability ε, and greedily (i.e., selecting the action that maximizes the Q-value) with probability $1 - \varepsilon$.

1.3.2.5 Target Network Update

The target network's weights θ^- are periodically updated by copying the weights from the online network:

$$\theta^- \leftarrow \theta$$

This update is performed every C steps to stabilize the learning process.

1.3.2.6 Benefits of DDQN

The primary advantage of DDQN is that it reduces the overestimation bias present in standard DQN, leading to more stable and reliable learning. This is particularly important in environments where the overestimation can cause significant degradation in performance.

1.3.3 3DQN: Dueling Double Deep Q-Network

The 3DQN (Dueling Double Deep Q-Network) algorithm is an extension of the Deep Q-Network (DQN) algorithm that combines two main improvements: the Double DQN algorithm to correct the overestimation bias in Q-values, and the dueling network architecture to better decompose state values and action advantages. These two techniques work together to enhance exploration efficiency and learning stability, particularly in continuous state space environments where some actions may be less relevant [8].

1.3.3.1 Motivation and Problem Statement

In many environments, the possible actions in certain states have little or no effect on the future outcome. In such cases, evaluating the Q-value for each state-action pair, as done in standard DQN, can be inefficient. For example, in a given state, several actions may lead to similar results, making the individual estimation of each action's value unnecessary. This leads to inefficient exploration, where the algorithm takes a long time to learn an optimal policy.

1.3.3.2 Dueling Network Architecture

The dueling network architecture introduced in 3DQN aims to address this inefficiency by decomposing the state value $V(s)$ and the action advantage $A(s, a)$. The network is designed to separately estimate:

- The value of being in a state s, denoted $V(s)$.

- The relative advantage of taking an action a in that state, denoted $A(s, a)$, which measures the difference between the action's value and the average value of all possible actions in that state.

The Q-function is then computed by combining these two streams:

$$Q(s, a) = V(s) + \left(A(s, a) - \frac{1}{|A|} \sum_{a'} A(s, a') \right) \tag{1.26}$$

Where:

- $Q(s, a)$ is the value of taking action a in state s.
- $V(s)$ is the value of the state s.
- $A(s, a)$ is the advantage of taking action a in state s.
- $\frac{1}{|A|} \sum_{a'} A(s, a')$ is the average advantage over all possible actions a', subtracted to normalize the advantages.

This architecture allows the network to learn more efficiently by focusing on estimating state values and identifying actions that provide significant advantages, reducing computational effort for actions that have little impact.

1.3.3.3 Double DQN

The Double DQN algorithm, integrated into 3DQN, aims to correct the overestimation bias in Q-values. This bias occurs when the same network is used to select the action and evaluate its value, which can lead to optimistic overestimation of Q-values. To address this issue, Double DQN separates the action selection from its evaluation:

$$y_t = r_t + \gamma \cdot Q(s_{t+1}, \arg\max_{a'} Q(s_{t+1}, a'; \theta); \theta^-) \tag{1.27}$$

Here:

- The action $\arg\max_{a'} Q(s_{t+1}, a'; \theta)$ is selected using the online network with the current parameters θ.
- The value of this action is then evaluated using the target network with parameters θ^-.

This method reduces the risk of overestimation by avoiding the use of the same network for both operations, which stabilizes learning and leads to a more accurate policy.

1.3.3.4 3DQN Algorithm

The 3DQN algorithm combines these two techniques to improve the efficiency and accuracy of learning. Here is the pseudo code of the algorithme:

Algorithm 3 3DQN Algorithm

Initialize the online Q-network Q with random weights θ_0

Initialize the target Q-network Q_{target} with weights $\theta_0^- = \theta_0$

Initialize replay memory $\mathcal{D}$

for each episode **do**

  Initialize state s_0

  for each step t in the episode **do**

    Select action a_t using an epsilon-greedy policy:

    - With probability ε, select a random action
    - Otherwise, select $a_t = \arg\max_a Q(s_t, a; \theta)$

    Execute action a_t and observe reward r_t and next state s_{t+1}

    Store the transition (s_t, a_t, r_t, s_{t+1}) in replay memory $\mathcal{D}$

    Sample a minibatch of transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{D}$

    for each transition i in the minibatch **do**

      Select action $a'_i = \arg\max_{a'} Q(s_{i+1}, a'; \theta)$

      Compute the target value $y_i = r_i + \gamma \cdot Q(s_{i+1}, a'_i; \theta^-)$

      Update the online network by minimizing the loss:

$$L_i(\theta) = (y_i - Q(s_i, a_i; \theta))^2$$

    end for

    Every C steps, update the target network: $\theta^- \leftarrow \theta$

    Decay ε according to a decay schedule

  end for

end for

Table 1.3: 3DQN Algorithm

1.3.3.5 Benefits of the 3DQN Algorithm

The 3DQN algorithm offers several benefits:

- Reduction of Overestimation Bias: By integrating Double DQN, the algorithm reduces the overestimation bias in Q-values, leading to more stable learning.
- Exploration Efficiency: The dueling architecture allows for more efficient exploration by focusing learning efforts on the most significant states and actions.
- Faster Learning: By avoiding repetitive evaluation of irrelevant actions, the algorithm converges more quickly to an optimal policy, especially in complex environments.

The 3DQN algorithm is a significant improvement over standard DQN, combining the advantages of Double DQN and the dueling network architecture to offer more efficient, stable, and faster learning. This combination overcomes some of the inherent limitations of DQN, particularly in environments where distinguishing between states and actions is crucial for optimal decision-making.

1.3.4 Per3DQN: 3DQN with Prioritized Experience Replay

The Per3DQN (Prioritized Experience Replay with Dueling Double Deep Q-Network) algorithm is an enhancement of the 3DQN algorithm. It introduces a prioritized experience replay mechanism, which improves learning efficiency by ensuring that more informative experiences are replayed more frequently. This mechanism is combined with the benefits of the dueling network architecture and Double DQN, making Per3DQN a powerful tool for reinforcement learning in complex environments.

1.3.4.1 Motivation and Problem Statement

In standard DQN, experiences (state transitions) are stored in a replay memory and sampled uniformly during training. However, not all experiences are equally useful for

learning. Some transitions have a higher impact on the learning process, particularly those with high temporal difference (TD) errors. The challenge with uniform sampling is that it treats all transitions as equally important, leading to inefficient learning where less informative transitions are replayed as frequently as more critical ones.

1.3.4.2 Prioritized Experience Replay

The Per3DQN algorithm addresses this inefficiency by introducing a prioritized experience replay mechanism. Instead of sampling experiences uniformly, transitions are prioritized based on their TD error, $|\delta|$. The TD error is a measure of how surprising or unexpected a transition is, which correlates with how much the Q-network can learn from it. The higher the TD error, the more significant the learning potential, and thus, the higher the priority assigned to that transition [9].

- **Priority Calculation**

  The priority p_i of each transition i is calculated using its TD error $|\delta_i|$:

  $$p_i = (|\delta_i| + \varepsilon)^\alpha \tag{1.28}$$

  where:

  - $\delta_i = r_i + \gamma \cdot Q(s_{i+1}, \arg\max_{a'} Q(s_{i+1}, a'; \theta); \theta^-) - Q(s_i, a_i; \theta)$ is the TD error for transition i.
  - ε is a small positive constant that ensures all transitions have a non-zero probability of being sampled.
  - α is a hyperparameter that controls the degree of prioritization, with $\alpha = 0$ corresponding to uniform sampling and $\alpha = 1$ to full prioritization.

- **Sum Tree Data Structure**

  To efficiently manage and sample from the prioritized transitions, Per3DQN uses a sum tree data structure. The sum tree allows for quick sampling of transitions

based on their priority values, enabling efficient selection of experiences with high learning potential.

The sum tree is a binary tree where each leaf node corresponds to a transition's priority, and each parent node contains the sum of its children's priorities. Sampling a transition involves traversing the tree based on a random value sampled between 0 and the sum of all priorities, allowing transitions with higher priorities to be selected more frequently.

1.3.4.3 Importance Sampling

While prioritized replay improves learning efficiency, it introduces a bias because the sampling distribution no longer matches the distribution under which the original policy was trained. To correct for this bias, importance sampling weights are applied during the Q-network updates:

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)} \right)^{\beta} \tag{1.29}$$

where:

- $P(i)$ is the probability of sampling transition i.
- N is the total number of transitions in the replay memory.
- β is a hyperparameter that controls the amount of correction applied, with $\beta = 0$ applying no correction and $\beta = 1$ applying full correction. β is typically annealed from an initial value to 1 over the course of training.

The loss function is then adjusted using these importance sampling weights:

$$L(\theta) = \frac{1}{M} \sum_{i=1}^{M} w_i \cdot (y_i - Q(s_i, a_i; \theta))^2 \tag{1.30}$$

where M is the minibatch size, and y_i is the target value computed similarly to Double DQN.

1.3.4.4 Per3DQN Algorithm

The Per3DQN algorithm combines these components with the 3DQN framework. Here is the pseudocode of the Per3DQN algorithm:

Algorithm 4 Per3DQN Algorithm

Initialize the online dueling Q-network Q with random weights θ_0

Initialize the target dueling Q-network Q_{target} with weights $\theta_0^- = \theta_0$

Initialize prioritized replay memory with capacity N

Initialize the sum tree data structure

for each episode **do**

  Initialize state s_0

  for each step t in the episode **do**

    Select action a_t using an epsilon-greedy policy:

    - With probability ε, select a random action
    - Otherwise, select $a_t = \arg\max_a Q(s_t, a; \theta)$

    Execute action a_t, observe reward r_t and next state s_{t+1}

    Compute TD error δ_t and priority $p_t = (|\delta_t| + \varepsilon)^\alpha$

    Store transition $(s_t, a_t, r_t, s_{t+1}, p_t)$ in prioritized replay memory

    Sample a minibatch of transitions from prioritized replay memory using sum tree

    for each sampled transition i in the minibatch **do**

      Compute importance sampling weight $w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta$

      Compute target $y_i = r_i + \gamma \cdot Q(s_{i+1}, \arg\max_{a'} Q(s_{i+1}, a'; \theta); \theta^-)$

      Update online network by minimizing the weighted loss:

$$L_i(\theta) = w_i \cdot (y_i - Q(s_i, a_i; \theta))^2$$

    end for

    Every C steps, update the target network: $\theta^- \leftarrow \theta$

    Decay ε according to a decay schedule

  end for

end for

Table 1.4: Per3DQN Algorithm

1.3.4.5 Benefits of the Per3DQN Algorithm

The Per3DQN algorithm offers several key benefits:

- **Increased Learning Efficiency**: By prioritizing transitions with higher learning potential, the algorithm focuses on the most informative experiences, leading to faster convergence.
- **Bias Correction**: The use of importance sampling weights corrects the bias introduced by prioritized replay, ensuring that the learning process remains stable and accurate.
- **Improved Exploration**: The combination of prioritized experience replay with the dueling network architecture and Double DQN enhances the algorithm's ability to explore and exploit effectively in complex environments.

1.4 MetaNet

The MetaNet model, developed by Messmer and Papageorgiou in the early 1990s [10], is a macroscopic traffic flow model designed to simulate the dynamics of traffic on large-scale motorway networks. The model captures the evolution of traffic flow, density, and speed over time and space using a set of differential equations based on fundamental traffic flow theory [11]. These equations are instrumental in predicting traffic states and assessing the impact of various control measures such as ramp metering and variable speed limits [12].

1.4.1 Key Components of the MetaNet Model

1. **Density Update Equation:**

   The density update equation models the temporal change in vehicle density on each road segment, accounting for the inflow and outflow of vehicles [10].

$$\rho_i(t+1) = \rho_i(t) + \frac{T}{L_i}\left(q_{i-1}(t) - q_i(t)\right) \tag{1.31}$$

where:

- $\rho_i(t)$ is the density of vehicles on segment i at time t.
- T is the time step size.
- L_i is the length of segment i.
- $q_{i-1}(t)$ is the flow into segment i from the previous segment.
- $q_i(t)$ is the flow out of segment i.

2. **Flow Update Equation:**

   The flow update equation relates the traffic flow on a segment to the vehicle density and speed [11].

$$q_i(t) = \rho_i(t) \cdot v_i(t) \tag{1.32}$$

where:

- $q_i(t)$ is the traffic flow on segment i.
- $\rho_i(t)$ is the density of vehicles on segment i.
- $v_i(t)$ is the speed of vehicles on segment i.

3. **Speed Update Equation:**

   The speed update equation models the change in vehicle speed, considering the current density and adjustments towards a desired speed [12].

$$v_i(t+1) = v_i(t) + \frac{T}{\alpha}\left(v_{i,\text{desired}}(\rho_i(t)) - v_i(t)\right) \tag{1.33}$$

where:

- $v_i(t)$ is the speed on segment i at time t.
- T is the time step size.
- α is a parameter influencing the rate of speed adjustment.
- $v_{i,\text{desired}}(\rho_i(t))$ is the desired speed as a function of density $\rho_i(t)$.

1.5 Related Works

Ramp metering is a critical component of modern traffic management systems, designed to regulate the flow of vehicles entering highways to optimize traffic conditions, prevent congestion, and enhance the overall efficiency of road networks. As urban areas expand and vehicle numbers increase, the importance of effective traffic management solutions has become paramount. Ramp metering plays a pivotal role in smoothing traffic flow, reducing bottlenecks, and maintaining safe driving conditions by controlling the rate at which vehicles merge onto busy highways. The development of ramp metering strategies reflects the growing complexity of traffic systems and the escalating demands placed on them.

The evolution of these strategies began with relatively simple, rule-based methods and has progressed to more sophisticated, data-driven approaches. Traditional strategies, such as demand-capacity control and the ALINEA algorithm, provided the foundation for managing traffic at specific freeway entry points. However, these methods often struggled to adapt to the dynamic and unpredictable nature of real-world traffic, where conditions can change rapidly due to various factors, such as accidents, weather changes, or fluctuations in traffic volume. This chapter traces the progression of ramp metering strategies from these early approaches to the most recent advances, including the use of reinforcement learning (RL) and multi-agent systems.

1.5.1 Evolution of Ramp Metering Strategies in Traffic Management

1.5.1.1 Early Ramp Metering Approaches

In 1967, Wattleworth [13] introduced foundational ramp metering concepts, including the demand-capacity and occupancy-capacity approaches. These methods were among the first systematic attempts to balance the inflow of vehicles with the available road capacity, aiming to prevent congestion and maintain smoother traffic flow on freeways. Wattleworth's work marked a significant advancement in traffic management, providing a structured approach to addressing congestion. However, these strategies were relatively simplistic, reactive rather than proactive, and did not fully account for the dynamic nature of traffic flow, limiting their effectiveness in more complex scenarios.

In 1991, Papageorgiou et al. [14] developed the ALINEA method, which became a cornerstone in the field of ramp metering. ALINEA, a feedback-based control strategy, dynamically adjusted ramp metering rates based on the occupancy levels observed at downstream bottlenecks. This method aimed to maintain freeway occupancy at an optimal level, thereby maximizing throughput and minimizing congestion risks. The simplicity and effectiveness of ALINEA led to its widespread implementation, but its reliance on precise parameter tuning and its localized approach limited its applicability in more complex traffic scenarios. To address these limitations, the PI-ALINEA strategy was introduced, incorporating a proportional-integral (PI) controller into the original framework. This enhancement allowed for more dynamic adjustments, considering both current occupancy and traffic flow trends over time, thereby improving the system's responsiveness to varying traffic conditions. Despite these advancements, the localized nature of ALINEA and PI-ALINEA meant that while they were effective at specific bottlenecks, they were not as successful in managing network-wide traffic dynamics, where a more holistic approach is necessary.

1.5.1.2 Advanced Control Strategies

By 2004, the need for more comprehensive traffic management solutions led to the development of the Advanced Motorway Optimal Control (AMOC) strategy by Kotsialos and Papageorgiou [15]. AMOC represents a significant evolution in ramp metering by adopting a network-wide, optimal control approach. This strategy continuously optimizes traffic flow across a freeway network by periodically updating control strategies based on real-time traffic data. The implementation of AMOC on the A10 ring road in Amsterdam demonstrated its effectiveness, particularly during peak traffic periods, where it significantly reduced overall travel times and delays. However, the success of AMOC is heavily reliant on the availability of high-quality, real-time traffic data, and the approach requires substantial computational resources, which can be a barrier to broader implementation.

In 2006 [16], Kerner introduced the ANCONA strategy, grounded in his three-phase traffic theory, which categorizes traffic into free flow, synchronized flow, and wide moving jams. ANCONA focuses on maintaining synchronized flow conditions to manage and spatially confine congestion, thereby preventing the formation of wide moving jams that can lead to severe congestion. The strategy's emphasis on managing traffic flow phases was a novel approach at the time and proved effective in reducing the frequency and severity of wide moving jams. However, the complexity of implementing ANCONA, particularly its need for precise calibration and deep understanding of traffic dynamics, posed challenges in real-world applications. In 2011 [17], Srivastava developed the Stratified Zone Metering (SZM) algorithm, which was designed to address the challenges of freeway congestion more effectively by organizing freeway segments into distinct zones, each monitored for traffic density. The SZM algorithm, implemented by the Minnesota Department of Transportation, tailored ramp metering rates to the specific conditions of each segment, allowing for a more nuanced and targeted control of traffic. This approach proved particularly effective in managing congestion in areas prone to frequent bottlenecks. However, its success depends on the availability of detailed, accurate traffic data and the precise calibration of the system, which can be challenging in highly variable

traffic environments.

1.5.1.3 Reinforcement Learning-Based Approaches

The limitations of traditional control strategies in adapting to the dynamic and unpredictable nature of traffic flow led to the exploration of more advanced, data-driven approaches, particularly those leveraging reinforcement learning (RL). In 2014, Fares and Gomaa [18] introduced the concept of Multi-Agent Reinforcement Learning (MARL) for ramp metering. MARL involves multiple RL agents, each responsible for controlling a specific ramp within a network. These agents operate either cooperatively or competitively to optimize traffic flow across the entire network. This decentralized approach allows each agent to make decisions based on local observations while contributing to the overall traffic management goals. However, the coordination and communication between agents pose significant challenges, especially in large and complex networks.

Building on this foundation, in 2020, Tan and colleagues [19] emphasized the importance of communication protocols and shared reward structures within MARL systems. Their work demonstrated that when agents share information and rewards, they can optimize not only local traffic conditions but also contribute to the efficiency of the entire network. However, as the complexity of the traffic network increases, ensuring effective coordination among agents becomes increasingly difficult, and scalability remains a significant challenge. In 2021 [20] , researchers applied Q-learning, a model-free RL algorithm, to ramp metering. Q-learning enables systems to learn optimal traffic management policies through direct interaction with the traffic environment, allowing them to adapt to changing conditions without relying on pre-defined models. This approach marked a significant departure from traditional methods, offering the potential for more adaptive and responsive traffic management. However, Q-learning faces challenges related to the curse of dimensionality, where the vast number of possible states and actions can make the learning process slow and computationally expensive.

1.5.1.4 Recent Innovations and Hybrid Approaches

In 2022, Yu Han and colleagues [20] made significant contributions to the field by advancing several innovative approaches to ramp metering. They expanded the application of Q-learning to both local and coordinated ramp metering scenarios, demonstrating its adaptability in complex and dynamic traffic environments. Their research also explored the potential of Deep Reinforcement Learning (DRL), which integrates the capabilities of deep learning with RL to handle large and complex state-action spaces using neural networks. DRL showed promise in managing intricate traffic scenarios, but it also presented challenges related to computational demands and the risk of overfitting.

Han and his team [20] further introduced a hybrid approach by combining traditional physics-based traffic flow models with RL, known as physics-informed reinforcement learning. This method leveraged the strengths of both approaches, where the RL agent was trained using both historical traffic data and synthetic data generated from macroscopic traffic models. This hybrid approach helped anchor the RL agent's learning process in real-world traffic dynamics, leading to more reliable and effective ramp metering strategies. However, the success of this approach is highly dependent on the accuracy of the underlying traffic models and the integration of different data sources, which can be both complex and resource-intensive.

Lastly, Han and colleagues [20] proposed a decentralized and coordinated control system that utilizes multiple RL agents to manage complex freeway networks more efficiently. This system divides the network into smaller subsystems, each controlled by an RL agent operating independently based on local traffic conditions. The coordination among these agents, achieved through a hierarchical control structure, allowed for greater flexibility and scalability in traffic management. However, ensuring effective communication and alignment among agents remains a significant challenge, particularly in large and dynamic traffic networks.

1.6 Analysis and Discussion

The preceding sections have provided a comprehensive overview of the evolution of ramp metering strategies, from early rule-based methods to advanced reinforcement learning and hybrid approaches. Each of these strategies has contributed uniquely to the development of traffic management systems, addressing specific challenges related to congestion, traffic flow optimization, and the adaptability of control mechanisms to dynamic traffic conditions.

To facilitate a clear understanding of the progression and relative strengths of these approaches, a comparison table is presented below. This table systematically contrasts the key methodologies discussed, focusing on aspects such as the underlying control strategy, the adaptability to varying traffic conditions, the complexity of implementation, and the overall impact on traffic management efficiency. By examining these dimensions, the table highlights the advancements made over time and the ongoing challenges that remain in the field of ramp metering.

Table 1.5: Comparison of Ramp Metering Strategies

Strategy	Control Method	Adaptability to Traffic Conditions	Complexity of Implementation
Demand-Capacity Control [13]	Rule-based, balances inflow with capacity	Low, reactive and not adaptive to dynamic changes	Low, simple implementation but limited effectiveness in complex scenarios
ALINEA [14]	Feedback-based, adjusts metering rate based on downstream occupancy	Moderate, effective for specific bottlenecks but limited in network-wide scenarios	Moderate, requires precise tuning and is localized
PI-ALINEA	Proportional-Integral controller added to ALINEA	Moderate, more responsive to traffic trends over time	Moderate, enhanced over ALINEA but still localized
AMOC [15]	Network-wide optimal control using real-time data	High, continuously updates strategies based on real-time conditions	High, requires substantial computational resources and real-time data
ANCONA [16]	Phase-based control focused on maintaining synchronized flow	Moderate, effective in preventing wide moving jams	High, complex implementation with a need for precise calibration
Stratified Zone Metering (SZM) [17]	Zone-based, adjusts metering rates for specific freeway segments	High, tailored control for specific congestion-prone areas	High, requires detailed traffic data and precise calibration
Multi-Agent Reinforcement Learning (MARL) [18]	Decentralized, RL agents control individual ramps	High, agents adapt to local conditions while optimizing network-wide flow	High, complex coordination and communication between agents
Q-Learning [20]	Model-free RL, learns optimal policies through interaction with traffic environment	High, adapts to changing conditions without pre-defined models	High, faces challenges related to the curse of dimensionality
Deep Reinforcement Learning (DRL) [20]	Combines deep learning with RL to manage complex state-action spaces	Very High, capable of handling intricate traffic scenarios	Very High, computationally demanding and risks of overfitting
Physics-Informed RL [20]	Hybrid, combines traditional traffic models with RL	High, grounded in real-world dynamics with enhanced adaptability	High, success depends on the accuracy of traffic models and data integration
Decentralized Coordinated Control [20]	Multi-agent RL with hierarchical control structure	Very High, flexible and scalable for complex networks	High, challenges in ensuring effective communication among agents

1.7 Conclusion

In summary, this chapter has connected the fundamental theories of reinforcement learning with their application in modern ramp metering strategies. By first establishing a solid understanding of RL and DRL, the chapter provided a foundation for exploring more advanced traffic management systems. The shift from early, rule-based methods to more sophisticated, data-driven approaches reflects the increasing complexity of traffic networks and the need for smarter management systems.

The review of recent advancements shows how combining traditional traffic models with RL techniques can lead to more effective and scalable solutions. However, these methods also come with challenges, such as the need for better scalability, coordination, and computational efficiency.

The insights gained from this review pave the way for the research in the following chapters, which will focus on addressing the key challenges identified here. The upcoming research will explore new approaches to improving the scalability, efficiency, and adaptability of traffic management systems, pushing the boundaries of what is possible in modern urban environments.

Proposed Approach

2.1 Introduction

The purpose of this chapter is to outline the research design, methods, and procedures used to address the research objectives of this thesis. The methodology adopted in this study was carefully chosen to ensure a systematic approach for analyzing and solving the research problem. Given the complexity of traffic management and the challenges associated with ramp metering optimization, this chapter details the rationale behind the selection of specific methods and techniques, as well as the process through which related and exited methods collected, processed, and analyzed.

This chapter is structured to provide a clear and comprehensive explanation of the steps undertaken during the research. It begins by describing the overall research framework, followed by an in-depth discussion of the models, algorithms, and tools used. The decision-making process for choosing particular methodologies, including reinforcement learning and traditional traffic management strategies, is also explained. Additionally, the chapter addresses the implementation environment, data sources, and validation techniques applied to ensure the reliability and accuracy of the results.

The methods employed in this study aim to provide a robust foundation for answering the research questions, advancing the field of traffic optimization, and contributing to the broader understanding of adaptive traffic control systems. Through this systematic and structured approach, the study aspires to produce meaningful, reproducible, and applicable results that can inform future research and practice in traffic management.

2.2 Methodology

The methodology follows a structured approach to solving the research problem, beginning with the Problem Understanding phase and concluding with Results Analysis. Each step builds on the previous one, ensuring a systematic and comprehensive exploration of the research questions. The process is divided into eight key stages, which are organized in a circular or iterative manner, suggesting that insights gained during later stages may loop back to earlier phases for refinement or adjustment. The following figure shows the flow of the steps :

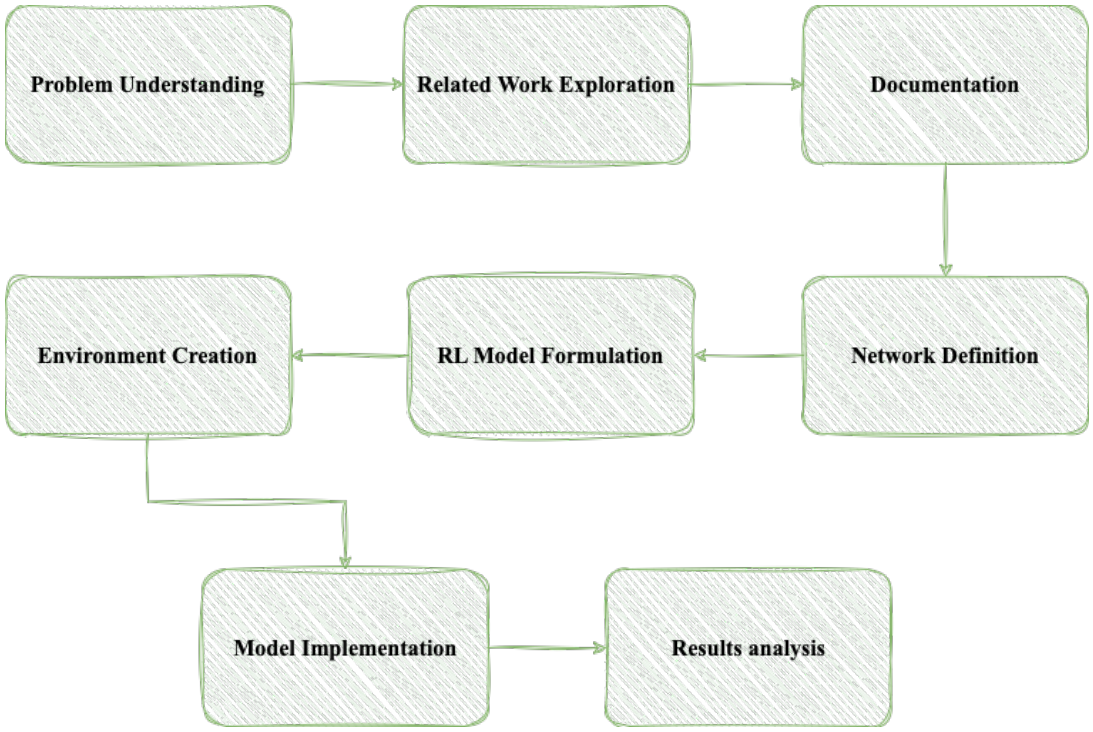


Figure 2.1: Methodology

- **Problem Understanding**: The first step involves identifying and clearly defining the problem at hand. This is essential for setting the direction of the research and ensuring that the objectives are well understood before proceeding with more technical aspects.
- **Exploring Related Work**: After defining the problem, the next step is to explore

the existing literature, frameworks, and methodologies related to the research topic. This helps to identify gaps in the current knowledge, as well as potential strategies and algorithms that could be applied to the problem.

- **Documentation**: Throughout the research process, detailed documentation is maintained to capture key decisions, progress, and findings. This step ensures that the research is transparent, reproducible, and provides a strong foundation for further investigation.

- **Network Definition**: In this stage, the traffic network is defined, which is a critical step in implementing and testing various models. This involves specifying the road layout, ramp locations, traffic flows, and any other relevant parameters.

- **Reinforcement Learning Model Formulation**: This phase involves the design and development of the RL model that will be applied to the traffic network. It includes defining the state space, actions, rewards, and training the model to optimize traffic flow through ramp metering.

- **Environment Creation**: The RL model requires a simulated environment in which it can be trained and tested. This stage involves creating or adapting a simulation platform that accurately reflects the traffic conditions and challenges outlined in the network definition phase.

- **Model Implementation**: Once the environment and RL model are ready, the models are implemented and tested. This step involves coding, testing, and refining the models to ensure they perform as expected.

- **Results Analysis**: The final stage involves analyzing the outcomes of the model's performance. This includes evaluating whether the model achieved the desired objectives, analyzing metrics such as traffic flow improvements, and identifying areas where the model can be further refined.

2.2.1 Problem Understanding

The problem of ramp metering in freeways focuses on managing the flow of vehicles entering highways to alleviate congestion and improve traffic conditions. The challenge lies in optimizing the timing of vehicle entry from ramps to minimize traffic bottlenecks while maintaining smooth flow on the mainline freeway. To fully understand this problem, we began by reviewing the fundamental principles of traffic dynamics, freeway infrastructure, and metering strategies. We analyzed the impact of ramp metering on traffic congestion, travel time, and overall efficiency of freeway systems. In-depth discussions with domain experts, coupled with a review of practical implementations, allowed us to identify the key variables and constraints influencing ramp metering decisions.

2.2.2 Exploring Related Work

To build a solid foundation for the research, we explored extensive related work in the field of traffic control and ramp metering. This included reviewing various ramp metering algorithms such as ALINEA, HERO, and other adaptive strategies. The focus was on understanding how different approaches optimize freeway performance under varying traffic conditions, and how machine learning and reinforcement learning methods have recently been applied to ramp metering. By examining existing literature, we identified gaps in current methodologies, particularly with respect to the scalability and real-time adaptability of these systems.

2.2.3 Documentation

As part of the implementation phase, we documented our work using the SUMO (Simulation of Urban MObility) traffic simulation environment and the TraCI (Traffic Control Interface) protocol. SUMO was used to model and simulate traffic behavior on freeway networks, while TraCI allowed us to interact with the simulation in real time to adjust ramp metering controls. Extensive documentation efforts were made to understand the full capabilities of SUMO and TraCI, including how to configure simulations, implement

custom ramp metering functions to manipulate the on-ramps, traffic lights and vehicles and then extract performance metrics for evaluation. This documentation was critical for ensuring a smooth and replicable experimentation process.

2.2.4 Network Description

The network implemented for this research is inspired by a ramp metering system studied in a previous work that utilizes a real-world freeway segment for testing purposes. Specifically, the reference network is based on a stretch of the Bruce Highway near Brisbane, Australia, as presented in the study [20]. This urban freeway experiences traffic patterns largely driven by commuters heading to and from the city center. The original network includes a section of approximately 5 km, characterized by multiple on- and off-ramps and a known bottleneck due to lane-drop situations and high demand on the ramps. In the reference study, the coordinated ramp metering strategy HERO was tested, and the traffic dynamics were simulated using SUMO, an open-source microscopic simulation tool, calibrated with traffic data from loop detectors.

For this research, the network configuration is represented as a simplified version of the Bruce Highway's ramp metering setup (Illustrated in the Figure 2.2). This simulation was conducted using SUMO with traffic flows managed by the TraCI API. The network includes three lanes with specific 3 on-ramp entries and traffic signals at each ramp entry to control the flow, thereby mimicking a real-world ramp metering scenario as shown in Figure 2.3.

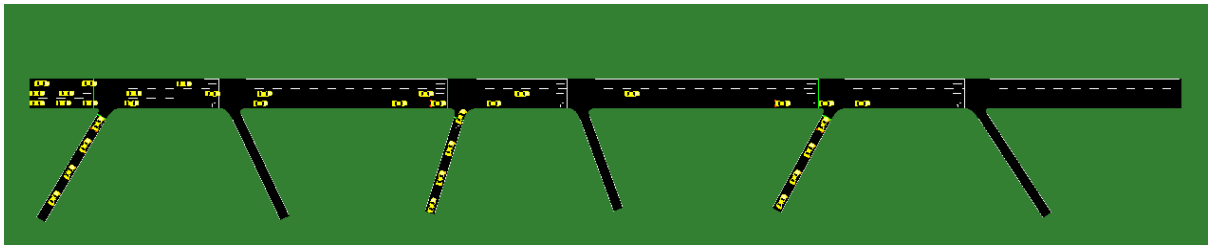


Figure 2.2: The Network used for the study

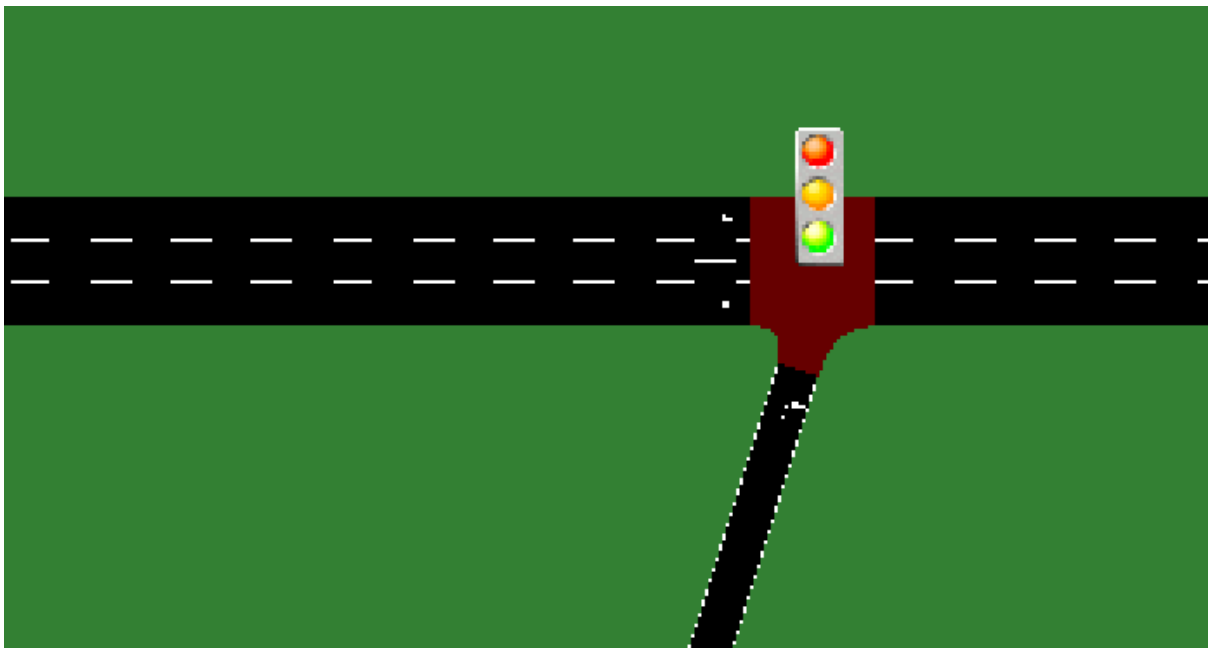


Figure 2.3: Traffic lights on the On-ramps

The network was described in the Markup Language (XML) file representing a freeway simulation created for use in SUMO. This network consists of several key components, including edges (representing roads or segments), lanes, and junctions. Each segment of the road is defined by its speed limit, length, and shape, indicating how vehicles should move through the network. Traffic lights and priority junctions control vehicle movement at intersections.

- **Edges**: Represent segments of roads or freeway sections, connecting different junctions. Each edge has multiple lanes with specified speeds, lengths, and geometric shapes (indicating the curvature or straight paths).
- **Lanes**: Each edge consists of multiple lanes, each with its speed and length attributes. Lanes guide the movement of vehicles and are responsible for determining how vehicles traverse each road segment.
- **Junctions**: Junctions connect various edges and determine how vehicles move from one edge to another. The network includes both priority junctions (where certain roads have the right of way) and traffic light-controlled junctions. Junctions are

defined with traffic light logic to manage vehicle flow. For example, junctions like J1, J8, and J10 are controlled by traffic lights with predefined signal phases (green, yellow, red).

- **Ramps**: The network includes specific ramp entries that connect smaller roads or freeway on-ramps to the main freeway. Ramps such as ramp12, ramp14, and ramp16 merge into the main freeway lanes at designated junctions controlled by traffic lights.

- **Traffic Light Logic**: Traffic lights at specific junctions (e.g., J1, J8, J10) operate on fixed schedules, cycling through phases to manage the flow of vehicles, controlling when cars can enter the freeway or continue through a junction.

2.2.5 Reinforcement Learning Model Formulation

2.2.5.1 Solution Overview

In our approach to optimizing freeway traffic flow, we address the challenge of controlling vehicle entry rates at on-ramps to minimize congestion while maximizing vehicle speeds on the freeway. This solution involves a **coordinated ramp metering control** system, operating dynamically across three lanes and three on-ramps. Each ramp is equipped with traffic lights that cycle through three distinct phases: green (G), yellow (Y), and red (R). The green phase allows vehicles to merge from the ramp onto the freeway, potentially increasing overall flow but also introducing the risk of congestion. The yellow phase signals a transition period to slow down the merging process, reducing the risk of sudden bottlenecks. Finally, the red phase halts vehicles at the on-ramp, giving priority to freeway traffic and preventing excessive merging that could lead to congestion.

The following figures show the different traffic light phases and their impact on vehicle flow during the ramp metering process. Figure 2.4 illustrates the green phase, where vehicles are allowed to merge onto the freeway. Figure 2.5 shows the yellow phase, a transition period aimed at safely controlling the flow of merging vehicles. Finally, Figure 2.6

presents the red phase, which holds back vehicles at the on-ramp, ensuring smoother traffic flow on the freeway. This coordinated ramp metering control system ensures that vehicle entry rates are dynamically adjusted based on real-time traffic conditions to optimize both the freeway and on-ramp flows.

These phases are dynamically adjusted based on real-time traffic conditions to optimize the flow of vehicles on both the freeway and the on-ramps.

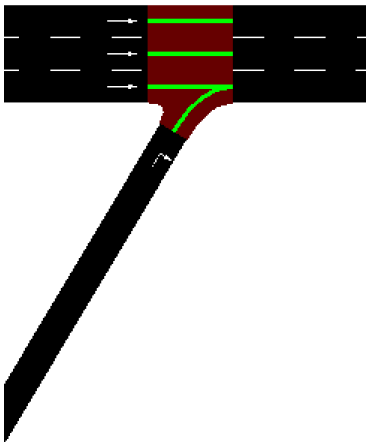


Figure 2.4: Green Light Phase

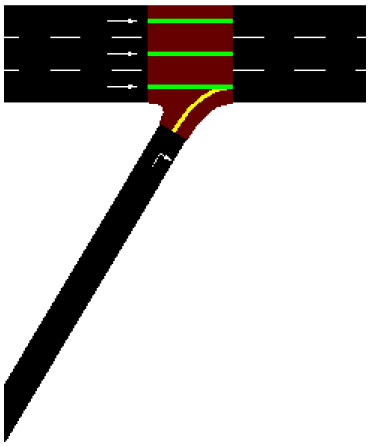


Figure 2.5: Yellow Light Phase

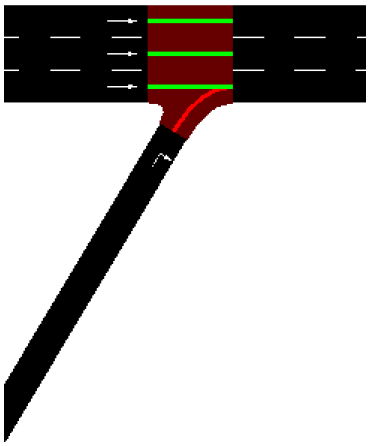


Figure 2.6: Red Light Phase

The objective is to regulate the flow of vehicles from the ramps onto the freeway in real time, thereby optimizing traffic flow, reducing congestion, and minimizing vehicle travel times. Figure 3.2 illustrates the network structure used for this solution.

2.2.5.1.1 Double Deep Q-Learning in Ramp Metering

For this solution, The Double Deep Q-Learning (DDQN) is employed to address the intricate challenge of coordinated ramp metering control. The primary objective is to optimize traffic flow on freeways by managing the rate at which vehicles merge from ramps, thereby reducing congestion and enhancing overall efficiency.

This is accomplished by deploying multiple agents, each responsible for controlling

individual ramps, and coordinating their actions to ensure system-wide optimization. DDQN is an advancement over traditional Deep Q-Learning (DQN) that effectively addresses the overestimation bias inherent in DQN.

In this framework, each agent learns to minimize local congestion while simultaneously considering the overall performance of the network. Each ramp is managed by a dedicated agent, which serves as an independent controller tasked with determining the optimal metering rate. Although these agents operate independently, the solution necessitates coordination among them to optimize traffic flow at the network level.

This coordination is facilitated through a shared reward function designed to reflect the system's overall performance. By optimizing this shared reward function, agents are incentivized to work collaboratively, reducing congestion not only at their respective ramps but throughout the entire freeway network.

2.2.5.2 Model Formulation

In this model, the goal is to optimize freeway traffic management by controlling the rate of vehicles entering from on-ramps. To achieve this, the model relies on a well-structured formulation involving three essential components: the state space, the action space, and the reward function. These components interact dynamically to allow for an adaptive traffic control mechanism, which minimizes congestion while maximizing overall traffic efficiency.

2.2.5.2.1 State Space

The state space represents the current traffic conditions on the freeway and at the ramps. Each variable in the state space encapsulates a critical aspect of the system's performance, providing the necessary information for making decisions about traffic control. The key variables are:

- **Traffic Density** (ρ): This variable captures the number of vehicles per unit length on the freeway. It is crucial because high densities indicate potential congestion, while low densities reflect smooth traffic flow. By monitoring traffic density, the

model can adjust ramp metering rates to prevent congestion buildup.

$$\rho = \frac{\text{Number of vehicles on segment}}{\text{Length of the segment}} = \frac{n_v}{L} \tag{2.1}$$

- **Queue Length** (Q_r): This variable represents the number of vehicles waiting at the on-ramp r. High queue lengths can lead to ramp spillback, which can block nearby intersections and worsen overall traffic conditions. Controlling queue lengths helps ensure that ramp operations remain efficient without causing excessive delays.

$$Q_r = \sum_{i=1}^{N_r} \delta_i \tag{2.2}$$

where N_r is the total number of vehicles in the on-ramp r, and δ_i is the indicator function that equals 1 if vehicle i is queuing and 0 otherwise.

- **Flow Rate** (f_r): This variable measures the number of vehicles merging from the ramp onto the freeway. Flow rate is critical for maintaining optimal traffic movement. A balanced flow prevents the freeway from becoming overwhelmed by ramp vehicles, while also ensuring that vehicles at the ramp are not unduly delayed.

$$f_r = \frac{\Delta n_r}{\Delta t} \tag{2.3}$$

where Δn_r is the number of vehicles entering the freeway from ramp r over the time interval Δt.

- **Average Vehicle Speed** (v): The average speed of vehicles on the freeway is an indicator of overall traffic conditions. Reduced speed is often a sign of congestion, and maintaining a high average speed is a key objective in optimizing freeway performance.

$$v = \frac{\sum_{i=1}^{n_v} v_i}{n_v} \tag{2.4}$$

where v_i is the speed of vehicle i on the freeway.

- **Red Light Duration** (τ_r): This variable tracks how long ramp r has been subject to a red signal. The duration of the red light affects the queue length and helps to balance the rate at which vehicles are allowed to merge onto the freeway.

Thus, the state space $\mathcal{S}$ at any time t is represented as:

$$\mathcal{S}(t) = (\rho(t), Q_r(t), f_r(t), v(t), \tau_r(t))$$

Together, these variables provide a comprehensive representation of the system's state at any given time, enabling informed decision-making regarding ramp control [21, 22].

2.2.5.2.2 Action space

The action space $\mathcal{A}$ consists of two key components:

1. **Activation Decision**:

$$a_{r,t} \in \{0, 1\}$$

where $a_{r,t} = 1$ means that ramp r is activated at time t, and $a_{r,t} = 0$ means that the ramp is not activated.

2. **Metering Rate**: m_r for the activated ramps:

$$m_r = \frac{\Delta n_r}{\Delta t} \quad \text{for active ramps}$$

The metering rate is subject to upper and lower bounds:

$$m_{\min} \leq m_r \leq m_{\max}$$

where $m_{\min}$ and $m_{\max}$ are the minimum and maximum allowable metering rates, respectively [23, 24].

2.2.5.2.3 Operational Mechanism

The system operates in two stages :

1. **Activation Decision**: The decision to activate ramp r at time t is determined by analyzing traffic conditions. The ramp is activated if the density in the station downstream of the access $\rho(t)$, queue length $Q_r(t)$, or flow rate $f_r(t)$ are less than the predefined thresholds:

$$\rho(t) < \rho_{\text{threshold}} \quad \text{or} \quad Q_r(t) < Q_{\text{threshold}} \quad \text{or} \quad f_r(t) < f_{\text{threshold}}$$

The density in the station downstream of the ramp is shown in the following figure :

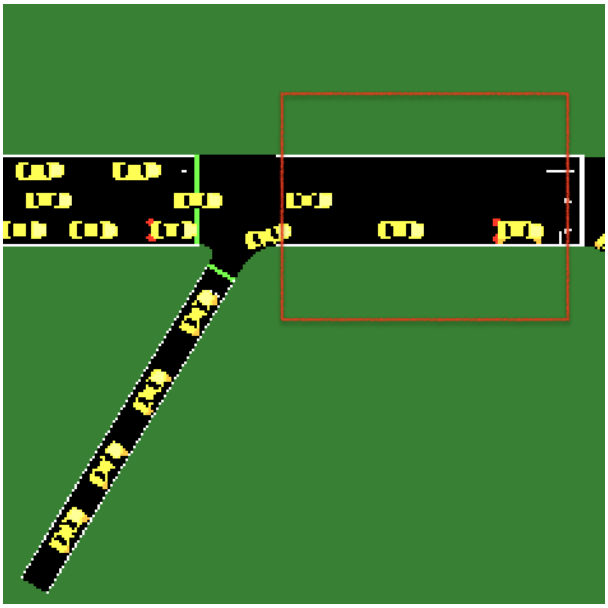


Figure 2.7: The station downstream the ramp

2. **Rate Setting for Activated Ramps**: Once ramps are activated, the metering rate m_r is set based on traffic density, flow rate, and queue length to ensure efficient traffic flow and avoid queue spillback.

A reinforcement learning (RL) agent is used to optimize the metering rates m_r. The RL agent selects the metering rate that maximizes the long-term reward R_t [14, 20, 25].

2.2.5.2.4 Reward Function

The **reward function** evaluates the outcomes of the actions taken by the model, providing feedback on how well the system is performing. It is designed to optimize multiple objectives [3, 20, 26]:

- **Delay Reduction**: The reward function incentivizes actions that minimize the delay experienced by vehicles on the freeway. Delay is calculated as the difference between the current vehicle speeds and the maximum allowable speed. Minimizing this delay is crucial for maintaining efficient traffic flow.

$$R_{\text{delay}} = -\sum_{i=1}^{n_v} \left(1 - \frac{v_i}{v_{\max}}\right) \tag{2.5}$$

- **Queue Length Minimization**: The model is also rewarded for reducing the queue lengths at the ramps. Long queues can lead to ramp spillback and disruption of nearby roads, so minimizing queues is essential for keeping the entire traffic network functional.

$$R_{\text{queue}} = -\sum_{r=1}^{R} Q_r(t) \tag{2.6}$$

- **Flow Maximization**: To ensure that vehicles continue to move efficiently from ramps to the freeway, the reward function encourages a high flow rate, allowing the system to discharge ramp vehicles effectively without overwhelming the freeway.

$$R_{\text{flow}} = \sum_{r=1}^{R} f_r(t) \tag{2.7}$$

- **Collision Penalty**: Safety is a crucial concern, and the reward function includes a penalty for vehicle collisions. This ensures that the system prioritizes not only efficiency but also safe traffic conditions.

$$P_{\text{collision}} = -\lambda_c \times C \tag{2.8}$$

where C is the number of collisions and λ_c is a constant penalty factor for each collision.

Thus, the total reward is a weighted sum of these components:

$$R_t = w_1 \times R_{\text{delay}} + w_2 \times R_{\text{queue}} + w_3 \times R_{\text{flow}} + P_{\text{collision}} \tag{2.9}$$

where w_1, w_2, w_3 are weighting factors that balance the trade-offs between delay reduction, queue minimization, and flow maximization.

In summary, the state space captures real-time traffic conditions, the action space defines the possible decisions the system can make, and the reward function guides the model by providing feedback based on the effectiveness of the actions taken. Together, these components enable the system to dynamically and adaptively manage freeway traffic, ensuring smoother flow, reduced congestion, and enhanced efficiency. This model allows for continuous learning and improvement, helping it respond to ever-changing traffic conditions in real time.

2.2.6 Real-time applicability

The applicability of the ramp metering model in real-time decision-making is critical for effectively managing traffic flow and mitigating congestion on highways. The model, based on reinforcement learning techniques, ensures that decisions are made swiftly enough to adapt to changing traffic conditions in real time. To verify the real-time applicability, the decision-making time can be calculated by measuring how long it takes for the model to process the input state and compute an action. This is done by profiling the system using the following equation for decision time:

$$T_{\text{decision}} = T_{\text{end}} - T_{\text{start}}, \tag{2.10}$$

where T_{start} is the timestamp at the start of the state observation, and T_{end} is the timestamp after the action is predicted. The total time, T_{decision}, must remain below a

certain threshold to ensure real-time performance, typically below 100 milliseconds for traffic systems.

The decision time involves multiple interactions between the agent and the environment. During this process, the RL agent observes the current state s_t, selects an action a_t based on the learned policy, and updates its parameters to improve performance. The decision-making process includes evaluating the expected future rewards and optimizing the agent's actions accordingly. The time required to process the state, compute the optimal action, and update the model's parameters directly influences the overall decision time, which is crucial for real-time performance.

According to the profiling of similar models, such as in lane-changing decisions in autonomous vehicles, decision times as low as 2.63 milliseconds have been achieved [27]. This demonstrates that reinforcement learning models for ramp metering can operate well within real-time constraints, ensuring rapid and adaptive responses to fluctuating traffic conditions. Therefore, the ramp metering model is computationally efficient, making it suitable for real-world implementation where timely decisions are critical to maintaining traffic flow and safety.

2.2.7 SUMO Environment

The SUMO environment provided in the code is designed to simulate traffic dynamics, with a particular focus on traffic light and ramp metering management. The `SumoEnv` class initializes the environment by loading necessary configuration and network files, including the road network and traffic light control configuration. SUMO is used as a microscopic simulation tool to model the movement of vehicles and their interaction with the traffic management system.

2.2.7.1 Key Functions and Workflow

Some functions are used directly from the DQN-frameworQ Framework: The project is available on GitHub: https://github.com/romainducrocq/DQN-frameworQ.

- **Initialization:** The `SumoEnv` class initializes the SUMO network using `.net.xml` files. It sets up the traffic light IDs and their corresponding logic. Additional parameters such as `gui`, `log`, and randomization settings are defined to manage the interface and the simulation's experimental conditions.

- **Simulation Control:** The `start()` and `simulation_step()` functions manage the flow of the simulation. These functions are responsible for starting the simulation and advancing the time steps during which vehicles move, and traffic lights change according to predefined control logic. The simulation can be reset using `simulation_reset()` to start the experiment afresh.

- **Traffic Flow and Signal Control:** Traffic light logic and signal timing are managed by `tl_logic` and `tl_signals`. These parameters control the green, yellow, and red phases for each traffic light. Functions like `gen_flow_logic()`, `find_ramp_edges()`, and `find_edges_after_ramps()` identify critical points within the network, such as ramps, where traffic flow may need special attention or control.

- **Reward System:** In the context of reinforcement learning, the reward function `rew()` calculates performance metrics based on factors such as total travel time, queue lengths at ramps, and vehicle flow rates. The system assigns penalties for conditions like long queue lengths or traffic jams and rewards for improving traffic flow efficiency.

- **Traffic Data Collection:** The `obs()` function collects observations related to traffic density, queue lengths, and flow rates at different points in the network. This data dynamically represents the traffic conditions and is crucial for simulation and analysis. The `get_ramp_lanes()` function identifies lanes associated with on-ramps, enabling specific control logic for traffic entering the freeway.

2.2.7.2 Usage in the Reinforcement Learning Framework

The environment is designed to work in tandem with a controller, such as the `RLController`. The controller provides actions, such as changing traffic light phases, and receives observations, including queue lengths or traffic densities. This dynamic interaction forms the foundation for training the RL models that aim to optimize traffic control strategies over time.

2.2.8 MetaNet Environment

The MetaNet environment is a macroscopic traffic simulation model that simulates traffic dynamics on freeways. Unlike microscopic models like SUMO, which simulate individual vehicles, MetaNet focuses on aggregated traffic flow at a higher level. The road network is divided into segments, and each segment has key traffic properties such as *density*, *speed*, and *flow*. These variables are updated iteratively during each time step of the simulation, based on traffic flow models such as the fundamental speed-density relationship.

2.2.8.1 Network Creation

In the MetaNet environment, the road network is divided into road segments. The key components of the network include:

- **Length**: The length of each road segment.
- **Number of Segments (nx)**: The total number of road segments in the network.
- **Segment Size (dx)**: The spatial resolution, or the length of each individual segment.
- **Time Step (dt)**: The time resolution that defines how frequently traffic variables are updated.

Each segment is associated with traffic parameters such as *density*, *flow*, and *speed*. These parameters are initialized using the speed-density relationship, and the MetaNet traffic model equations are used to update them iteratively.

2.2.8.2 Traffic Dynamics

The traffic conditions are updated at each time step using key equations:

- **Density Update**: The change in vehicle density depends on the inflow and outflow of vehicles from the segment.
- **Speed Update**: Speed is updated based on traffic density, desired speed, speed limits, and current traffic conditions.
- **Flow Calculation**: Flow is the product of speed and density within each segment, representing the total number of vehicles passing through a road segment per unit time.

The MetaNet model captures the interactions between traffic density, speed, and flow, simulating how congestion propagates through the road network.

2.2.8.3 Control System (Ramp Metering)

The main control system in MetaNet is *ramp metering*, which regulates the flow of vehicles onto the freeway from on-ramps. The key components of the control system include:

- **Ramp Metering Control**: The control adjusts the rate at which vehicles enter the freeway from ramps, preventing congestion on the mainline.
- **Ramp Queue Dynamics**: Vehicles arriving at an on-ramp may queue up if ramp metering is in place. The queue length is updated based on the arrival rate and metered entry rate.
- **Control Actions**: The RL agent selects control actions, such as ramp metering rates, which regulate the inflow of vehicles onto the freeway.

2.2.8.4 Reinforcement Learning Framework

The MetaNet environment is used in conjunction with a reinforcement learning (RL) agent to optimize traffic conditions. The framework consists of:

- **State Representation**: The traffic state is represented by density, speed, and flow in each segment, as well as queue lengths at ramps.
- **Actions**: The RL agent controls the metering rates at on-ramps.
- **Reward**: The reward is designed to optimize traffic flow, minimize congestion, and reduce queue lengths. It is based on factors like total travel time, average speed, and flow.

The RL agent learns an optimal control policy by interacting with the environment, adjusting ramp metering rates, and improving its policy over time.

2.2.9 Hyperparameter Configuration

The training of the reinforcement learning agent is controlled by several hyperparameters that guide both the learning process and training stability. Below is a summary of the key hyperparameters used:

- **GPU Usage** : Specifies if the GPU device is used for training.
- **Number of Environments** (`'n_env' = 1`): Number of parallel environments. With 1 environment, no multi-processing is applied.
- **Learning Rate** (`'lr' = 1e-4`): Controls how much model weights are updated with each gradient step. A lower value ensures stable convergence.
- **Discount Factor** (`'gamma' = 0.99`): Defines the importance of future rewards, balancing immediate and long-term rewards.
- **Epsilon (Exploration-Exploitation Trade-off)**:
  - `'eps_start' = 1.0`: Initial exploration rate.
  - `'eps_min' = 0.01`: Minimum exploration rate.
  - `'eps_dec' = 2e6`: Steps over which epsilon decays.

  - `'eps_dec_exp' = True`: Exponential decay is used for epsilon.

- **Batch Size** (`'bs' = 32`): Number of samples in each training batch.
- **Replay Memory**:
  - `'min_mem' = 100000`: Minimum memory size before training begins.
  - `'max_mem' = 1000000`: Maximum memory buffer size.
- **Target Network Updates**:
  - `'target_update_freq' = 30000`: Steps between hard target network updates.
  - `'target_soft_update' = True`: Enables soft updates.
  - `'target_soft_update_tau' = 1e-03`: Soft update rate.
- **Save and Log Frequency**:
  - `'save_freq' = 10000`: Steps between model checkpoint saves.
  - `'log_freq' = 1000`: Steps between logging training metrics.
- **Episode Time Limit** (`'max_episode_steps' = 1000`): Limits the number of steps in each episode.
- **Algorithm** : Specifies the algorithm used.

This configuration fine-tunes the learning process, ensuring a balance between exploration and exploitation while stabilizing the training and managing computational resources.

2.2.10 Neural network Architecture

The proposed neural network consists of a series of convolutional layers followed by fully connected layers, optimized using the Adam optimizer. The structure is described in detail below, including the number of layers and nodes.

2.2.10.1 Convolutional Layers (CNN)

The convolutional part of the network is designed to extract meaningful features from the input data through two convolutional layers. Each convolutional layer is followed by a non-linear activation function:

- **Layer 1 (Convolutional)**:
  - Input channels: C
  - Output channels (filters): 16
  - Kernel size: 1×1
  - Stride: 1
  - **Activation Function**: ELU (Exponential Linear Unit)
- **Layer 2 (Convolutional)**:
  - Input channels: 16
  - Output channels (filters): 32
  - Kernel size: 1×1
  - Stride: 1
  - **Activation Function**: ELU

The ELU (Exponential Linear Unit) is used as the activation function after each convolutional layer to introduce non-linearity and ensure smoother convergence. The ELU function is defined as:

$$\text{ELU}(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{if } x \leq 0 \end{cases}$$

Where α is a hyperparameter that controls the saturation of negative inputs.

2.2.10.2 Flattening Layer

After the convolutional layers, the output feature maps are flattened into a one-dimensional vector for input into the fully connected layers. Let the flattened dimension be denoted as n_{flatten}.

2.2.10.3 Fully Connected Layers (FC)

Two fully connected layers are employed to further process the features extracted by the convolutional layers. The structure is as follows:

- **Layer 3 (Fully Connected)**:
  - Input size: n_{flatten} (from the flattened output of the convolutional layers)
  - Output size (nodes): 128
  - **Activation Function**: ELU
- **Layer 4 (Fully Connected)**:
  - Input size: 128
  - Output size (nodes): 64
  - **Activation Function**: ELU

The fully connected layers progressively reduce the dimensionality of the feature space, applying the ELU activation function after each transformation to maintain non-linearity and improve learning capability.

2.2.10.4 Optimization and Loss Function

The model is trained using the **Adam optimizer**, a popular choice for deep learning models because it adapts the learning rate during training based on the first and second moments of the gradient. The loss function is **Smooth L1 Loss**, which balances

robustness and precision by being less sensitive to outliers than traditional L2 loss. The Smooth L1 loss function is defined as:

$$\text{SmoothL1}(x) = \begin{cases} 0.5x^2, & \text{if } |x| < 1 \\ |x| - 0.5, & \text{otherwise} \end{cases}$$

2.2.11 Training Process

The training process is fundamental in reinforcing the learning capabilities of the agent as it interacts with the environment. The process, as implemented in `train.py`, can be described as follows:

2.2.11.1 Agent and Environment Initialization:

The training process begins with the initialization of the agent and its interaction environment. The environment, constructed using the `CustomEnvWrapper`[4] and `CustomEnv` classes, defines the state space, action space, and transition dynamics. The agent is instantiated with key parameters such as the learning rate (α), discount factor (γ), epsilon values for exploration, neural network configurations, and memory buffer sizes. These parameters govern the agent's learning behavior, enabling it to explore the environment and learn from experience effectively.

2.2.11.2 Replay Memory Buffer:

To enhance learning efficiency, a replay memory buffer is employed to store past experiences in the form of state-action-reward-next state tuples. This buffer allows the agent to sample and learn from a diverse set of experiences rather than relying on sequential data, which may introduce temporal correlations. The buffer ensures that the agent's learning process is more stable and efficient by enabling experience replay.

2.2.11.3 Training Loop:

The agent's training loop is an iterative process where it interacts with the environment over many episodes. During each step, the agent observes the current state of the environment and selects an action according to its policy. After executing the action, the environment returns the next state, a reward, and information indicating whether the episode has terminated. The agent then computes the temporal difference (TD) error, which measures the difference between the predicted and target Q-values. Using gradient descent, the agent updates its neural network parameters to minimize this error and improve its policy. Periodically, a target network is updated to stabilize the learning process by providing more consistent target values.

2.2.11.4 Logging and Model Saving:

Throughout the training process, key metrics and performance data are logged at predefined intervals. This enables continuous monitoring of the agent's progress and performance. Additionally, model checkpoints are saved to ensure the trained model can be resumed or further trained, preventing data loss in case of interruptions.

2.2.12 Observation Process

For the SUMO environment, the observation process, implemented in `observe.py`, governs how the agent interacts with the environment during the evaluation phase. This process allows for the testing and assessment of the agent's learned policy.

During the observation process, important information is logged, including the agent's actions, the rewards received, and episode statistics. This logged data allows for the evaluation and analysis of the agent's performance, enabling a deeper understanding of how well the agent has learned to optimize its behavior in the environment. The collected data is stored in CSV files, which ensures ease of access and facilitates further analysis using data processing tools.

2.3 Conclusion

In this chapter, we outlined the research methodology employed to address the research objectives of this thesis. We adopted a systematic approach, starting with the problem definition of ramp metering and exploring existing literature to identify gaps and inform the selection of suitable methods. Through a structured methodology involving reinforcement learning models, traffic simulation environments (SUMO and MetaNet), and real-time data analysis, we designed a comprehensive framework to optimize traffic flow and minimize congestion on freeways.

The formulation of the reinforcement learning model, including its state space, action space, and reward function, enabled a dynamic and adaptive ramp metering control system. The integration of traffic simulation environments provided the necessary infrastructure to test, evaluate, and refine the models in realistic traffic scenarios. The use of both microscopic (SUMO) and macroscopic (MetaNet) simulation tools further allowed for thorough experimentation and performance validation.

This methodology, through its iterative process and robust foundation in both traditional traffic management strategies and advanced reinforcement learning techniques, sets the groundwork for significant advancements in traffic optimization. The chapter provides a clear pathway from problem understanding to solution implementation, contributing to the growing body of research in adaptive traffic control systems and offering insights for future work in this field.

Experimental Studies and Results

3.1 Introduction

This chapter presents the experimental setup, tools, and methodologies employed in the development of the reinforcement learning models for optimizing traffic flow. We begin by detailing the working environment, tools, and libraries used to implement and train the models. This includes the choice of the Visual Studio Code IDE, the SUMO traffic simulation tool, and key Python libraries such as PyTorch and Gym, which were integral to the development process.

The experiments were conducted by simulating real-world traffic conditions using SUMO and evaluating the performance of reinforcement learning agents through interaction with the environment via the TRACI API. We assess the effectiveness of the DDQN and ALINEA agents in various traffic scenarios, using a variety of performance metrics, including rewards, episode lengths, and queue lengths. The results obtained from these experiments provide insight into how each model handles traffic management and congestion reduction.

This chapter also analyzes the performance of the reinforcement learning models and offers a detailed comparison of the results between the two agents. The outcomes highlight the strengths and limitations of each approach in optimizing traffic flow.

3.2 Tools Presentation

3.2.1 Working environmnent

3.2.1.1 Visual Studio Code

For this thesis, the Visual Studio Code IDE was selected as the primary development environment. VSCode is widely used for Python development due to its extensive support for Python through extensions and built-in functionalities like IntelliSense, linting, debugging, and Jupyter Notebook integration. Its popularity stems from its lightweight nature, cross-platform compatibility, and flexibility in managing Python environments, making it an ideal tool for both machine learning and reinforcement learning tasks [28, 29, 30].

In addition to Python development, VSCode integrates with libraries such as Tensorboard for monitoring machine learning models and supports cloud deployment with Azure, enabling easy scaling of reinforcement learning models [28]. Its multi-panel views, terminal integration, and keyboard shortcuts for frequent tasks improve productivity, especially for managing large-scale reinforcement learning projects [30].

3.2.1.2 SUMO (Simulation of Urban MObility)

SUMO was used as the traffic simulation tool which is an open-source traffic simulation suite designed to study urban mobility patterns. In this thesis, SUMO was responsible for simulating complex traffic scenarios, which were then used to train reinforcement learning models for traffic management [31].

3.2.2 Programming Languages

3.2.2.1 Python Programming Language

The programming language used in this thesis is Python, which is one of the most widely used programming languages for machine learning, deep learning, and reinforcement learning. Python's simplicity, combined with its extensive libraries and strong support

for scientific computing, make it an ideal choice for AI development. With powerful libraries like PyTorch, Python allows for the development of flexible and dynamic reinforcement learning models. PyTorch was particularly helpful due to its ease of debugging and dynamic computation graph [32].

3.2.2.2 Gym Library and gym.spaces

The OpenAI Gym library was used to define the environments for training the reinforcement learning agents. Specifically, the 'gym.spaces' module provided the tools necessary to define action and observation spaces in the simulation. These spaces allow for efficient representation of the agent's states and actions, ensuring that the reinforcement learning agent can effectively learn and interact with its environment [33].

3.2.2.3 PyTorch

PyTorch was the primary machine learning library used for developing the neural networks and implementing the reinforcement learning algorithms. PyTorch's dynamic computation graph and easy-to-use API make it a popular choice for reinforcement learning. The 'torch' module, which offers a variety of utilities for tensor manipulation and deep learning, was crucial in building and optimizing the models used in this thesis [32, 34].

3.2.2.4 TRACI

The TRACI API was used to control the SUMO simulation in real-time and extract relevant information. TRACI allows reinforcement learning agents to interact with the SUMO simulation by sending commands and receiving feedback, making it possible to evaluate the impact of different actions on traffic flow. This real-time interaction was crucial for the agent's learning process [31, 34].

3.3 Experimental Results

This section presents the key results obtained from both the SUMO and MetaNet environments, comparing the performance of the DDQN agent and the ALINEA agent. We evaluate and analyze multiple metrics, including rewards, episode length, ramp queue length, and collisions. These results provide insights into how each agent optimizes traffic flow, manages congestion, and ensures overall system efficiency. The visualizations highlight the comparative strengths and weaknesses of each approach, allowing for a detailed assessment of their effectiveness in traffic management scenarios.

3.3.1 SUMO Environment

For this part, we evaluated the performance of the DDQN and ALINEA agents in managing ramp metering and optimizing traffic flow using all the mentioned performance metrics to assess the effectiveness of both agents within the SUMO simulation.

3.3.1.1 Rewards

The Figure 3.1 plots of the Average reward of both DDQN and ALINEA agents over steps during the training phase.

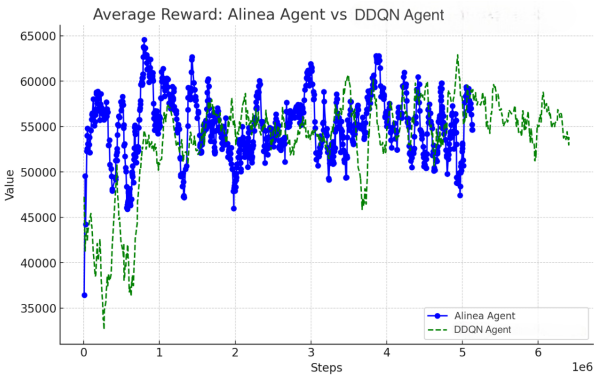


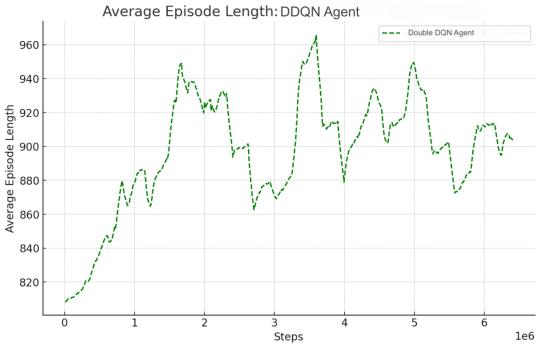
Figure 3.1: Average Reward for both DDQN and ALINEA

The figure shows that the Double DQN Agent has high initial fluctuations but stabilizes with higher rewards over time. In contrast, the ALINEA Agent demonstrates more

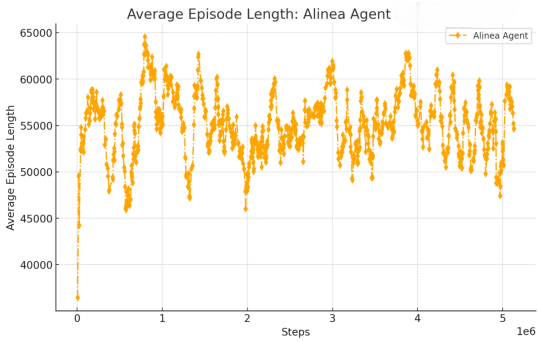
stable behavior but achieves lower overall rewards throughout the training process.

3.3.1.2 Episode Length

The Figure 3.2a and Figure 3.2b plots of the Episode Length over steps of both DDQN and ALINEA agents consecutively during the training phase.



(a) Average Episode Length for DDQN



(b) Average Episode Length ALINEA Agent

Figure 3.2: Comparison of Average Episode Length between DDQN and ALINEA

The DDQN Agent shows high initial variability in episode lengths but stabilizes as the policy improves. In contrast, the ALINEA Agent maintains consistent, shorter episode lengths throughout training, indicating more stable performance.

3.3.1.3 Episodes

The Figure 3.3 plots of the Episode over steps for both DDQN and ALINEA agents.

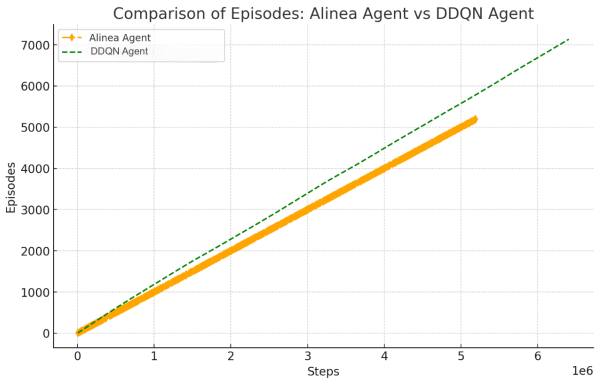


Figure 3.3: Comparison of Episodes: DDQN Agent vs ALINEA Agent

The DDQN Agent completes more episodes during the learning phase and stabilizes afterward. While, the ALINEA Agent completes fewer episodes but shows more stability throughout training.

3.3.1.4 Collisions

The plot in Figure 3.4 illustrates the variation of collision number over steps for both the DDQN and ALINEA agents.

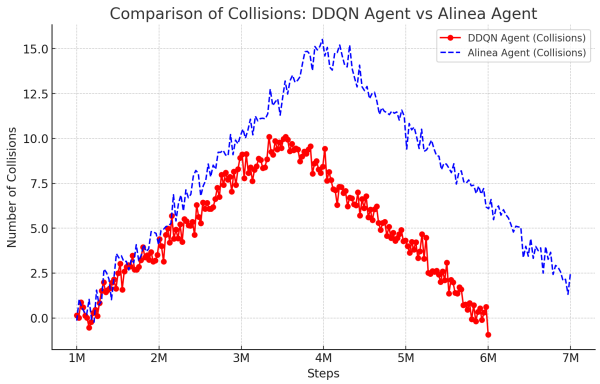


Figure 3.4: Collisions comparison : DDQN vs ALINEA

The DDQN Agent initially experiences a high number of collisions, which decreases as the policy improves. In contrast, the ALINEA Agent maintains lower and more consistent collision management throughout training.

3.3.1.5 Ramp queue length

The plot in Figure 3.5 illustrates the ramp queue length changes over steps for both the DDQN and ALINEA agents.

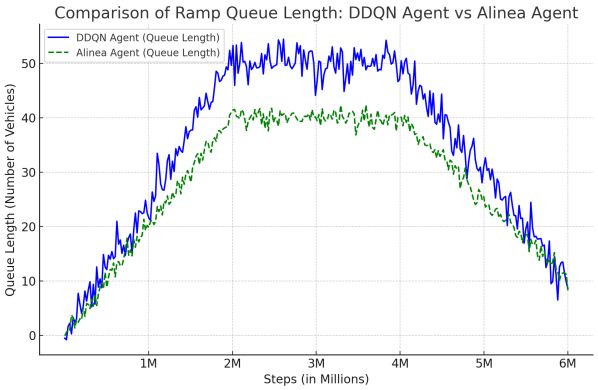


Figure 3.5: Comparison of Ramp Queue Length: DDQN Agent vs Alinea Agent

The DDQN agent experiences a steady increase in ramp queue length during training, peaking around 4 million steps before decreasing. In contrast, the ALINEA agent maintains a more stable and consistently lower queue length throughout the training process.

3.3.2 For MetaNet Environment

In this section, we evaluated the performance of the DDQN and ALINEA agents within the MetaNet environment. Using the same performance metrics—rewards, episode lengths, queue lengths, and collisions—we assess how each agent optimizes traffic management in a macroscopic traffic simulation model. The results highlight the effectiveness of both agents in controlling traffic flow and reducing congestion in this environment.

3.3.2.1 Rewards

The figure Figure 3.6 illustrates the comparison between the DDQN and ALINEA agents in terms of the variation in rewards over steps during the training phase.

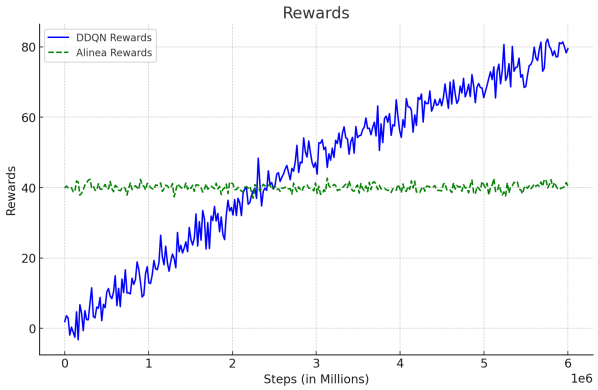


Figure 3.6: Reward comparison between DDQN and ALINEA

The DDQN agent shows a steady increase in rewards as training progresses, reflecting improved performance. The ALINEA agent, however, demonstrates stable rewards throughout, indicating consistent but lower reward accumulation.

3.3.2.2 Episode length

The Figure 3.7 illustrates the variation in episode length over steps during the training phase for both the DDQN and ALINEA.

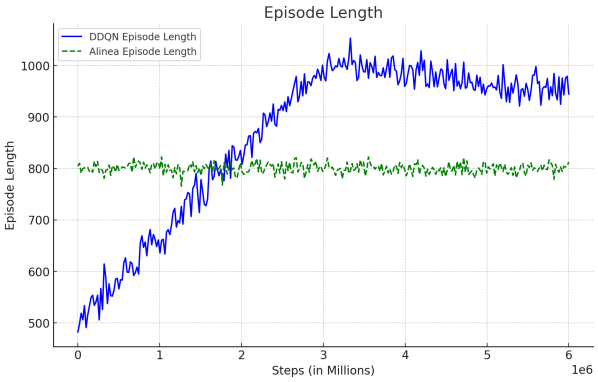


Figure 3.7: Episode length comparison between DDQN and ALINEA

The DDQN agent's episode length increases steadily during training, indicating longer interactions with the environment and then starts stabilizing. The ALINEA agent keeps a more stable episode length across training, with minimal fluctuations.

3.3.2.3 Episodes

The Figure 3.8 illustrates the variation in episodes over steps for both DDQN and ALINEA.

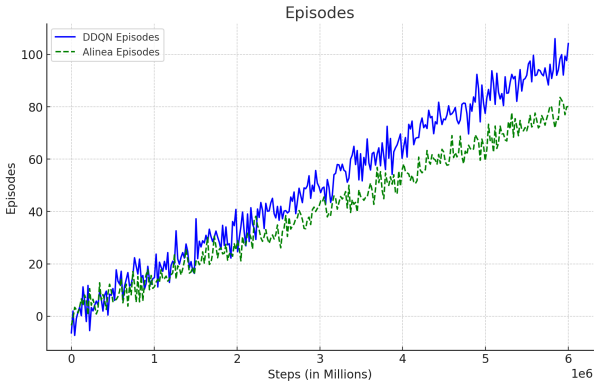


Figure 3.8: Episodes comparison between DDQN and ALINEA

The DDQN agent completes more episodes over time, particularly in the later stages of training. The ALINEA agent completes fewer episodes, reflecting its longer and more stable interactions with the environment.

3.3.2.4 Collisions

The Figure 3.9 shows the variation of collisions number over steps during the training phase for both the DDQN and ALINEA.

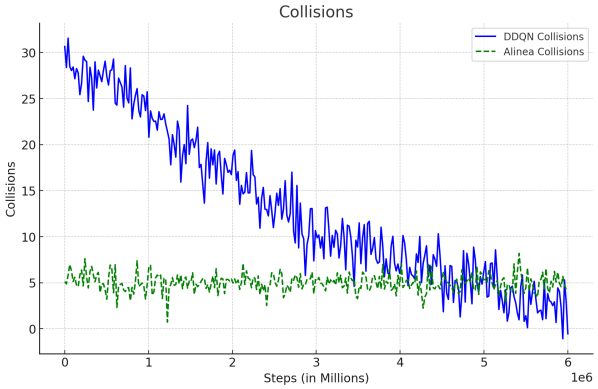


Figure 3.9: Collisions comparison between DDQN and ALINEA

The DDQN agent shows a high number of collisions initially, which gradually decreases as training progresses. The ALINEA agent maintains consistently lower collision rates throughout the process.

3.3.2.5 Ramp queue length

The plot in Figure 3.10 illustrates the changes of ramp queue length over steps for both the DDQN and ALINEA.

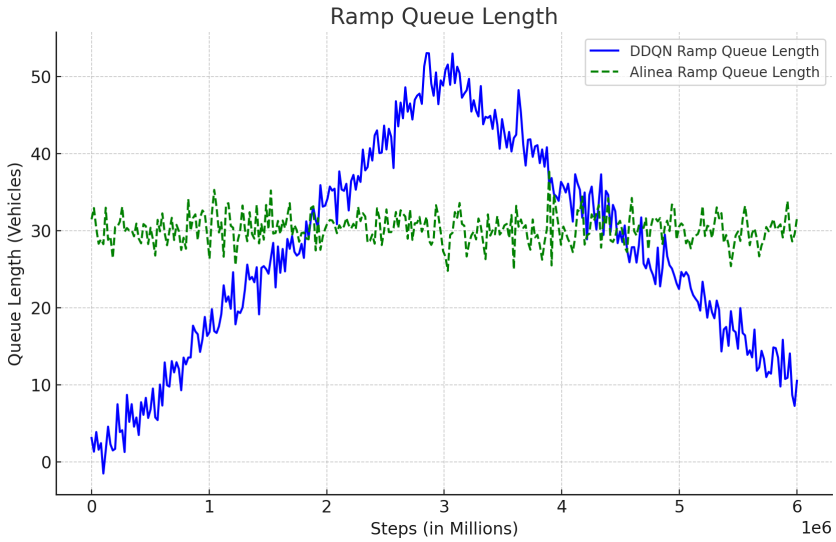


Figure 3.10: Ramp queue length comparison between DDQN and ALINEA

The DDQN agent experiences a significant increase in queue length during the learning phase, peaking at around 4 million steps and then decreasing over time. In contrast, the ALINEA agent maintains more stable and lower queue lengths throughout training.

3.4 Comparison and Discussion

In this section, we analyze the performance of the DDQN and ALINEA agents across two different environments: SUMO and METANET. Each agent's performance is evaluated based on key factors such as rewards, episode length, number of episodes, collisions, and ramp queue length. The DDQN agent, being a learning-based approach, exhibits higher

variability, especially in the learning phase, while ALINEA's feedback-driven control results in more stable and predictable behavior. The following table presents a detailed comparison of the agents' performance in both environments:

Factor	**SUMO (DDQN)**	**SUMO (Alinea)**	**METANET (DDQN)**	**METANET (Alinea)**
Rewards	High initial fluctuations, stabilizes with higher rewards over time	Stable, lower overall rewards	Similar pattern, takes longer to stabilize due to complexity	Stable, lower overall rewards, similar to SUMO
Episode Length	Initially variable, stabilizes as policy improves	Consistent and shorter episode lengths	More fluctuations due to complexity, takes longer to stabilize	Consistent and shorter, similar to SUMO
Episodes	More episodes during learning phase, stabilizes after learning	Fewer episodes, stable	Higher episodes during learning due to complexity	Fewer episodes, stable
Collisions	High initially, decreases as policy improves	Lower, consistent collision management	Similar pattern, takes longer to reduce due to complexity	Low and consistent, similar to SUMO
Ramp Queue Length	Fluctuates during learning phase, decreases over time	Consistent and lower queue length	More fluctuations, longer learning phase before decreasing	Consistent and stable, similar to SUMO

Table 3.1: Comparison of DDQN and Alinea in SUMO and METANET environments

So we can conclude as an overall comparison that :

- **DDQN**: The learning-based DDQN agent shows a clear pattern of high variability during the learning phase in both SUMO and METANET, with longer learning times and higher fluctuations in the more complex METANET environment. Over time, the agent learns to optimize rewards, reduce collisions, and stabilize queue lengths.

- **ALINEA**: ALINEA, by contrast, shows stable and predictable behavior across both environments, with fewer fluctuations and generally more conservative performance in terms of rewards and queue management. It is well-suited for environments where stability and predictability are key, but it may not achieve the peak performance of DDQN after its learning phase.

3.5 Conclusion

The experimental studies presented in this chapter demonstrate the application of reinforcement learning techniques to traffic management, specifically through the use of DDQN and ALINEA agents. By leveraging tools such as SUMO for traffic simulation, PyTorch for model implementation, and the TRACI API for real-time interaction, we successfully evaluated the performance of these agents under various traffic conditions.

The experiments show that both agents can effectively manage traffic flow, with each method offering distinct advantages. The DDQN agent, with its capacity for learning and adaptation, excelled in dynamic traffic scenarios, while the ALINEA agent, based on a more traditional control method, showed reliable performance in more predictable settings. Metrics such as rewards, episode lengths, and queue lengths provided a clear understanding of how each agent performs and where improvements can be made.

These results highlight how reinforcement learning can improve traffic management, providing a strong foundation for future improvements and the wider use of smart traffic control systems in real-world situations.

General Conclusion

This thesis has presented a comprehensive study on the optimization of traffic management using reinforcement learning (RL) techniques, specifically focusing on the application of ramp metering. Through a systematic approach, we explored both traditional traffic control strategies and advanced RL models to address the growing challenges of freeway congestion and traffic flow efficiency. The research utilized advanced traffic simulation tools, such as SUMO and MetaNet, to simulate real-world traffic scenarios, enabling the testing and validation of the proposed solutions in both microscopic and macroscopic environments.

The key contribution of this work lies in the formulation of a reinforcement learning framework that adapts to real-time traffic conditions, dynamically adjusting ramp metering rates to optimize vehicle flow and minimize congestion. By designing a state space that captures critical traffic parameters, defining an action space for control decisions, and developing a reward function that aligns with traffic efficiency goals, the proposed RL model offers a scalable and robust solution for adaptive traffic management.

In particular, the integration of classical control strategies like ALINEA with reinforcement learning allowed us to combine the strengths of established traffic control algorithms with the adaptability of modern machine learning methods. The experimental results demonstrated that RL-based approaches could outperform traditional methods by offering more flexible and responsive traffic management solutions.

This work has established a solid foundation for further research in the domain of intelligent transportation systems. Future work could explore more sophisticated RL algorithms, multi-agent systems, or the inclusion of additional real-time data sources such as connected vehicle technology. Additionally, real-world implementations of these strategies could provide valuable insights into their practical applicability.

In conclusion, this thesis has made significant progress in advancing the field of traffic optimization through the use of reinforcement learning, contributing to a deeper under-

standing of how intelligent control systems can enhance urban mobility and reduce traffic congestion. The findings of this research offer promising directions for future developments in adaptive traffic management systems.

Bibliography

[1] Faris B. Mismar, Jinseok Choi, and Brian L. Evans. A framework for automated cellular network tuning with reinforcement learning. *IEEE Transactions on Communications*, 67(10):7152–7167, 2019. doi: 10.1109/TCOMM.2019.2926715. URL https://www.researchgate.net/publication/327045314_A_Framework_for_Automated_Cellular_Network_Tuning_With_Reinforcement_Learning. Accessed: 2024-08-24.

[2] Phil Tabor. Modern reinforcement learning: Deep q learning in pytorch, 2020. URL https://www.udemy.com/course/deep-q-learning-from-paper-to-code. Udemy Course.

[3] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

[4] Romain Ducrocq. Deep reinforcement learning for traffic signal control in partial detection environments. Master's thesis, Université Gustave Eiffel, Marne-la-Vallée, France, 2021. Master Thesis.

[5] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4): 279–292, 1992.

[6] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017. URL https://arxiv.org/abs/1710.02298.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King,

Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. doi: 10.1038/nature14236. URL https://www.nature.com/articles/nature14236.

[8] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML-16)*. PMLR, 2016. URL https://arxiv.org/abs/1511.06581.

[9] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, 2016. URL https://arxiv.org/abs/1511.05952.

[10] A. Messmer and M. Papageorgiou. Metanet: A macroscopic simulation program for motorway networks. In *Mathematics and Computers in Simulation*, volume 31, pages 389–395. Elsevier, 1990.

[11] M. Papageorgiou, C. Diakaki, V. Dinopoulou, A. Kotsialos, and Y. Wang. Review of road traffic control strategies. *Proceedings of the IEEE*, 91(12):2043–2067, 2012.

[12] A. Kotsialos, M. Papageorgiou, F. Middelham, and Y. Wang. Traffic flow modeling of large-scale motorway networks using the macroscopic modeling tool metanet. *IEEE Transactions on Control Systems Technology*, 10(1):90–99, 2002.

[13] Joseph A. Wattleworth. Peak-period analysis and control of a freeway system. *Highway Research Record*, 157:1–25, 1967.

[14] M. Papageorgiou, H. Hadj-Salem, and J.-M. Blosseville. Alinea: A local feedback control law for on-ramp metering. *Transp. Res. Rec.*, 1320(1):58–67, 1991.

[15] Athanasios Kotsialos and Markos Papageorgiou. Efficiency and robustness of motorway network traffic control in presence of random incidents. *Transportation Research Part C: Emerging Technologies*, 12(6):431–454, 2004.

[16] Boris S. Kerner. *Probabilistic features of synchronized flow in traffic networks*. Springer, 2006.

[17] Alok Srivastava. *Development of Stratified Zone Metering Algorithm for Minnesota Freeways*. PhD thesis, University of Minnesota, 2011.

[18] Ahmed Fares and Walid Gomaa. Multi-agent reinforcement learning for intelligent traffic signal control. In *2014 IEEE 17th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1550–1555. IEEE, 2014.

[19] Jiyuan Tan, Qianqian Qiu, and Weiwei Guo. Coordinated ramp metering control based on multi-agent reinforcement learning. In *2020 35th Youth Academic Annual Conference of Chinese Association of Automation (YAC)*. IEEE, 2020.

[20] Yu Han, Meng Wang, Linghui Li, Claudio Roncoli, Jinda Gao, and Pan Liu. A physics-informed reinforcement learning-based strategy for local and coordinated ramp metering. *Transportation Research Part C*, 2022.

[21] Roberto Horowitz, Gabriel Gomes, Andreas Kotsialos, et al. Optimal ramp metering using advanced traffic state estimation. *Transportation Research Part C: Emerging Technologies*, 14(6):479–495, 2006.

[22] Andreas Kotsialos and Markos Papageorgiou. Freeway ramp metering: An overview. *IEEE Transactions on Intelligent Transportation Systems*, 5(4):271–281, 2004.

[23] M.J. Lighthill and G.B. Whitham. On kinematic waves ii: A theory of traffic flow on long crowded roads. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 229(1178):317–345, 1955.

[24] Tom Bellemans, Bart De Schutter, and Bart De Moor. Model predictive control for ramp metering of motorway traffic: A case study. *Control Engineering Practice*, 14 (7):757–767, 2006.

[25] Gabriel Gomes, Roberto Horowitz, et al. Optimal freeway ramp metering using the extended kalman filter. *Transportation Research Part C: Emerging Technologies*, 11 (3-4):223–240, 2003.

[26] Gail A. Carman, Qi Luo, Gabriel Gomes, et al. A reinforcement learning approach to traffic management in urban road networks. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[27] Shurong Li, Chong Wei, and Ying Wang. Combining decision making and trajectory planning for lane changing using deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 23(9):16110–16120, 2022. doi: 10.1109/TITS.2022.3148085. URL https://ieeexplore.ieee.org/document/9712364.

[28] Microsoft. Pytorch development in visual studio code. https://learn.microsoft.com/en-us/shows/docs-ai/pytorch-development-in-visual-studio-code, 2024.

[29] DataCamp. Setting up vscode for python: A complete guide. https://www.datacamp.com, 2023.

[30] Real Python. Advanced visual studio code for python developers. https://realpython.com, 2023.

[31] SUMO Project. Traci python api - sumo. https://sumo.dlr.de/docs/TraCI.html, 2024.

[32] PyTorch. Reinforcement learning (ppo) with torchrl tutorial. https://pytorch.org/tutorials/intermediate/torchrl_tutorial.html, 2024.

[33] OpenAI. Spaces - gym documentation. https://www.gymlibrary.dev, 2023.

[34] PyTorch. Reinforcement learning with torchrl. https://pytorch.org/tutorials/intermediate/torchrl_tutorial.html, 2024.

Appendix

.1 ALINEA Agent

The **ALINEA Agent** in the context of reinforcement learning can be viewed as an adaptive traffic control agent that combines classical traffic control strategies with reinforcement learning. Specifically, it implements the ALINEA feedback control algorithm to regulate ramp metering on highways. The goal of the ALINEA Agent is to maintain the desired traffic flow or density by adjusting the metering rate at the highway on-ramps based on real-time traffic conditions [14, 20, 22].

.1.1 ALINEA Control Law

The ALINEA Agent is based on a simple linear control law derived from classical control theory. The metering rate r is adjusted based on the difference between the target traffic density q_{target} and the observed traffic density q_{current} at the on-ramp. The control law is expressed as:

$$r_{\text{new}} = r_{\text{old}} + K \cdot (q_{\text{target}} - q_{\text{current}}) \tag{1}$$

Where:

- r_{new} is the updated metering rate.
- r_{old} is the current metering rate.
- K is the gain factor that controls the sensitivity of the adjustment.
- q_{target} is the desired traffic density.
- q_{current} is the actual traffic density.

The ALINEA agent adjusts the metering rate in real time to ensure that traffic density stays close to the target q_{target}, thus preventing traffic congestion.

.1.2 Components of the ALINEA Agent

The ALINEA Agent encapsulates the following key components:

- **Rate Control**: The agent controls the metering rate r at the highway on-ramp and updates it dynamically based on traffic conditions.
- **Feedback Mechanism**: The agent receives real-time traffic density data and compares the current density with the target density to compute the new metering rate.
- **Update Process**: The agent updates the metering rate at each time step using the ALINEA control law.

.1.3 Learning Process

Although the ALINEA Agent follows a classical control approach, it can be integrated with reinforcement learning to enhance its adaptability to varying traffic conditions over time. The agent learns to adjust its parameters (e.g., gain K and target density q_{target}) to optimize traffic flow based on experience. The learning process involves the following steps:

1. **Feedback Observation**: The agent observes the current traffic density q_{current} at the on-ramp.
2. **Action Selection**: Based on the difference between q_{current} and q_{target}, the agent computes the new metering rate r_{new} using the control law (3.10).
3. **Update Process**: The metering rate is updated in the environment to control the flow of vehicles on the highway.